The Lean-Agile eBook Practitioner Series

GETTING STARTED WITH KANBAN

A GUIDE FOR IMPLEMENTING LEAN & AGILE COLLABORATION ACROSS THE ENTIRE ENTERPRISE

gearstream

Brad Murphy, CEO
brad.murphy@gearstream.com

FOREWORD

Most CEOs dream of a future where all employees, teams, and departments work in complete synchronicity. In this dream, a giant lever on human capital acts as a secret weapon in an age of global competition, hypercompetitive markets, and continuous commoditization of products and services.

CEO surveys rank innovation and human capital among the primary corner office concerns, demonstrating that reality is far from this ideal scenario.

How people organize and collaborate to get things done is the essence of business. When people are motivated and empowered to work together in high-performing teams, traditional rules of business are suspended, and magic happens.

While the Agile movement has had much to say about enabling these kinds of hyperperforming teams so desperately needed today, it has also been responsible for promoting ideals that seem elusive to many firms. The fifth principle of the Agile manifesto suggests that you should "build projects around motivated individuals" by "[giving] them the environment and support they need, and trust them to get the job done."

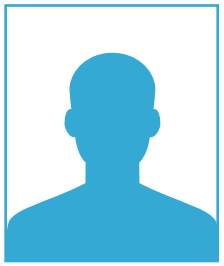
While the premise of this philosophy is spot-on, most Agile collaboration models depend heavily on the genius of teams, which is more of an ideal than a strategy you can scale.

So, how does Kanban succeed at scale where other methods have failed?

Fundamental to Kanban is the principle of extreme visualization within and across teams. When actualized, all teams are on the same page, understanding the relative volume, priority, and pacing of work items across their entire collaborative work community.

The effective application of Kanban principles and practices results in teams that deliver disproportionately better results with respect to rates of innovation, output, productivity, and predictability.

Kanban may well represent our best hope for bringing true agility to business execution—and it may well be that elusive blueprint for true strategic advantage.



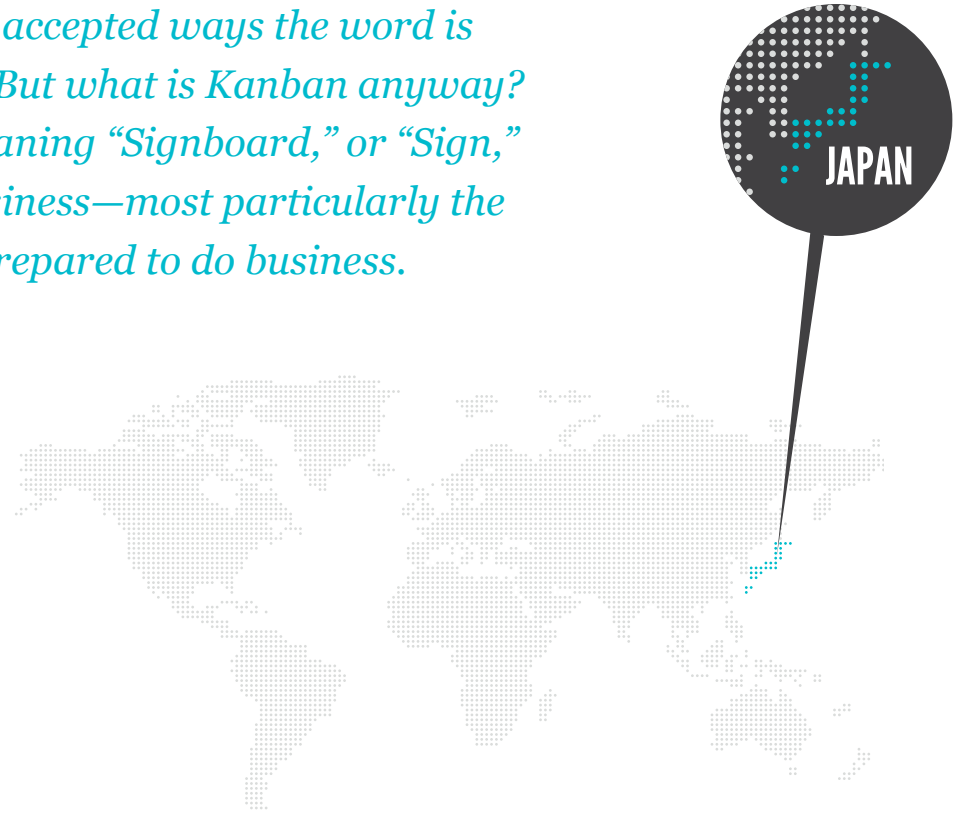
By Brad Murphy
Founder/CEO Gear Stream, Inc.

CONTENTS

gearstream

1. AN INTRODUCTION TO KANBAN

Kanban . . . such an odd name, including the variety of accepted ways the word is pronounced: "con-bon," "can-ban," or even "con-ban." But what is Kanban anyway? The meaning is derived from an old Japanese term meaning "Signboard," or "Sign," which in ancient times was usually associated with business—most particularly the notice "Open for Business," which implied being fully prepared to do business.



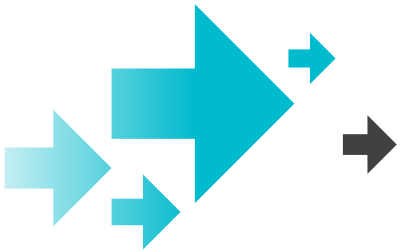
The word Kanban in contemporary usage still makes the same claim but is now most commonly associated with a system of visually controlling and balancing production first implemented by Toyota almost a half century ago. Toyota credits Kanban and its impact on their production line effectiveness as the catalyst for consistently outperforming bigger, more established, and better capitalized competitors, including Ford and General Motors. Although Toyota may be the most cited example of Kanban in industry, the concept has expanded significantly and is now synonymous with the implementation of Lean Principles of production. This eBook, however, was written specifically to discuss how Kanban has been successfully applied in recent years to help improve the unique challenges of teams and entire organizations engaged in the more complex and unpredictable world of software and product development.

Kanban can also be defined as a tool to drive forward continuous improvement and evolutionary change in your organization. Are you searching for just such a system? If so and if you are interested in implementing Lean Principles in a more direct way than traditional Agile methods, then Kanban's consistent focus on flow and context may be precisely what you and your business need.

Although change in business, as in everything, is both necessary and inescapable, every company can control how it responds to change. A company's goal should always be to identify and make necessary changes in a controlled and coherent way. Kanban offers a clear system for managing this change.

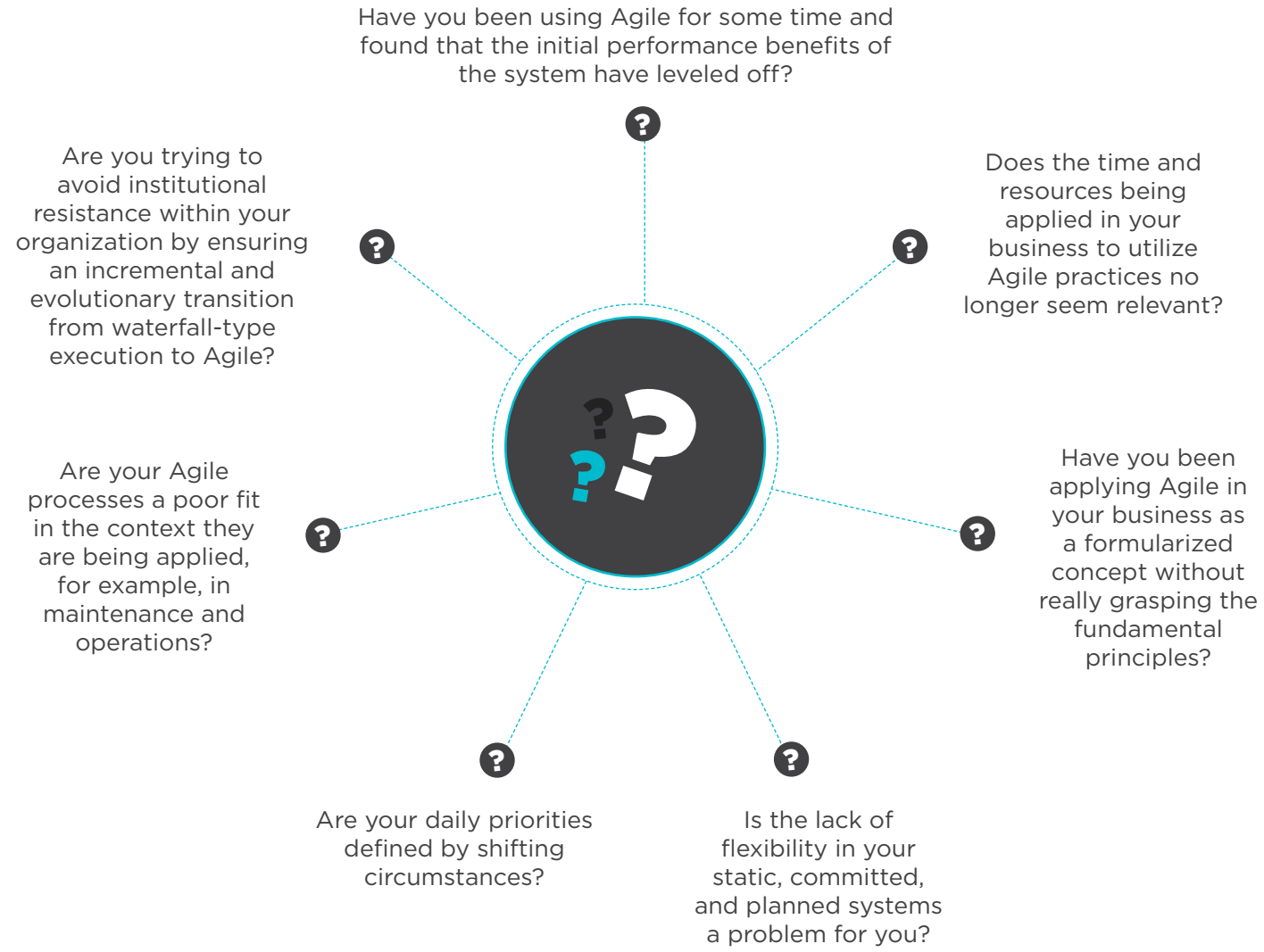
Kanban will not tell you what change is needed—numerous systems have been configured to give you that information—but it can help you move forward with necessary change while still retaining overall control over the process.

It has been our experience that incremental, continuous improvements produce better and more sustainable results than radical transformation, and this is precisely Kanban's goal—to implement an incremental and evolutionary process that improves the survivability of a system in a changing environment. With Kanban you will learn how to control changes in your systems rather than allowing haphazard change to control you and sometimes even override the basic systems of your organization.



DO I NEED KANBAN?

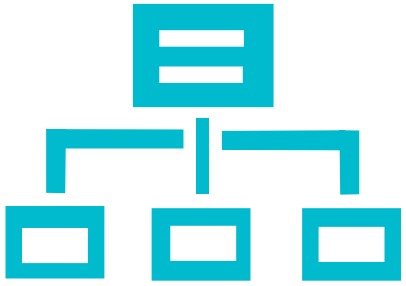
Have you been challenged to successfully scale Agile in your organization?



If the answer is yes to any of the above questions, then keep reading, because this book is for you! The principles of Kanban are increasingly being used to complement, improve, and optimize a company's commitment to strategy.

gearstream

LET'S UNDERSTAND THE KEY PRINCIPLES OF KANBAN.



Kanban is a flexible and at times fluid concept that is applicable and adaptable to just about every individual approach to change management, but certain key principles almost always apply:

- You should clearly visualize each step in your production process
  - From concept to tangible, it is vital that you have a clear map of the journey.
- Control your Work In Progress (WIP)
  - Clear limits must be in place for the amount of work undertaken at each point.
- Your route map must be clearly understood
  - If all points of the production map are clearly understood, the odds of everyone arriving at the same time are much improved.
- Keep a handle on the flow of production
  - Position yourself to be able to make informed decisions and visualize consequences of change.
- Stay alert for any opportunity for improvement
  - Ensure that a workforce culture is in place where continuous improvement is possible, desirable, and expected.

Sound familiar? If you've worked with Lean systems in the past, much of the following will resonate with your experience.

One of the principal achievements of Kanban has been to serve as a catalyst for the introduction of Lean ideas into software delivery systems at all stages.

gearstream

WHAT DOES KANBAN LOOK LIKE?

Note that no WIP limit is assigned to the last column. In this case, the team in question has opted for a weekly release cadence, which means that all completed work is released once a week (Tuesday at 3 p.m.).

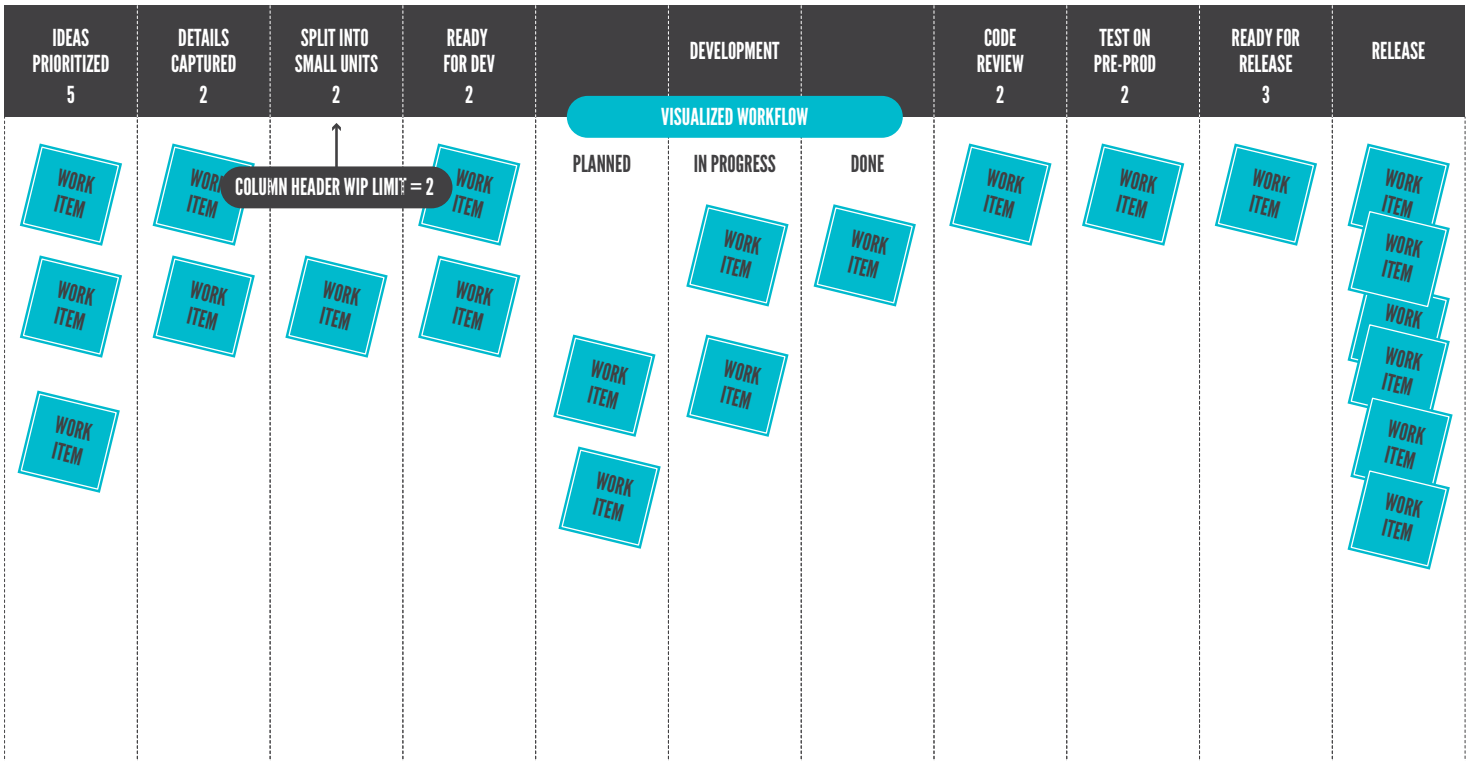


Figure 3 Kanban Stages
This board demonstrates very clearly the first four key principles of Kanban: workflow visualization is achieved in the column headers, while Work in Progress limits are represented. Policies are clearly defined and flow monitored.

The visualized flow and WIP limits that are outlined on this Kanban board help teams to avoid backlogs of more work than they have the capacity to handle. This is key to Kanban and is referred to as a Kanban Pull System—a simple analogy that is intended to illustrate a degree of inherent tension in a successful production system. In this example, only a certain number of maximum concurrent units of work are allowable, requiring teams to complete existing work before new work can begin. Work items are pulled through the system based on capacity similar to air being sucked into a vacuum rather than pushing work items through the system based on forecasts or demand, which, using the same analogy, would be producing pressure, which, in turn, creates resistance. This simple process of managing workflow is transformational, tending to have a profound impact on corporate productivity, quality, and predictability.

In this book, you will learn more about how and why these mechanisms work and how to take maximum advantage of them in your organization.

WHAT IS THE FIRST STEP IN IMPLEMENTING KANBAN?

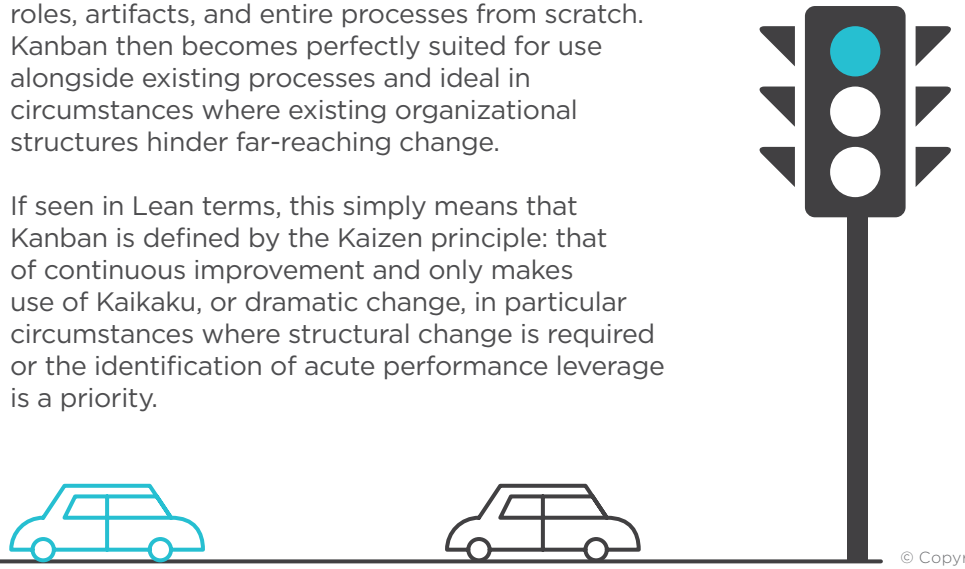
First, recognize that Kanban has a different approach to change management than most other Agile methods.

FIRST, IT IS IMPORTANT TO RECOGNIZE THAT THE APPROACH KANBAN OFFERS TO THE PROBLEM OF CHANGE MANAGEMENT IS VERY DIFFERENT FROM THAT OFFERED BY MOST OTHER AGILE METHODS.

Kanban is a concept built on the principle of evolutionary or incremental change.The first step toward harnessing the power of Kanban is to fully understand how your current software delivery system works, which is achieved by mapping your value stream. Once you have clearly visualized, measured, and managed your flow, you will be able to begin making changes.

Changes in Kanban are incremental. You will identify the largest bottleneck in the process and work to relieve it. Other methods, such as Scrum, often start out by completely redefining roles, artifacts, and entire processes from scratch. Kanban then becomes perfectly suited for use alongside existing processes and ideal in circumstances where existing organizational structures hinder far-reaching change.

If seen in Lean terms, this simply means that Kanban is defined by the Kaizen principle: that of continuous improvement and only makes use of Kaikaku, or dramatic change, in particular circumstances where structural change is required or the identification of acute performance leverage is a priority.



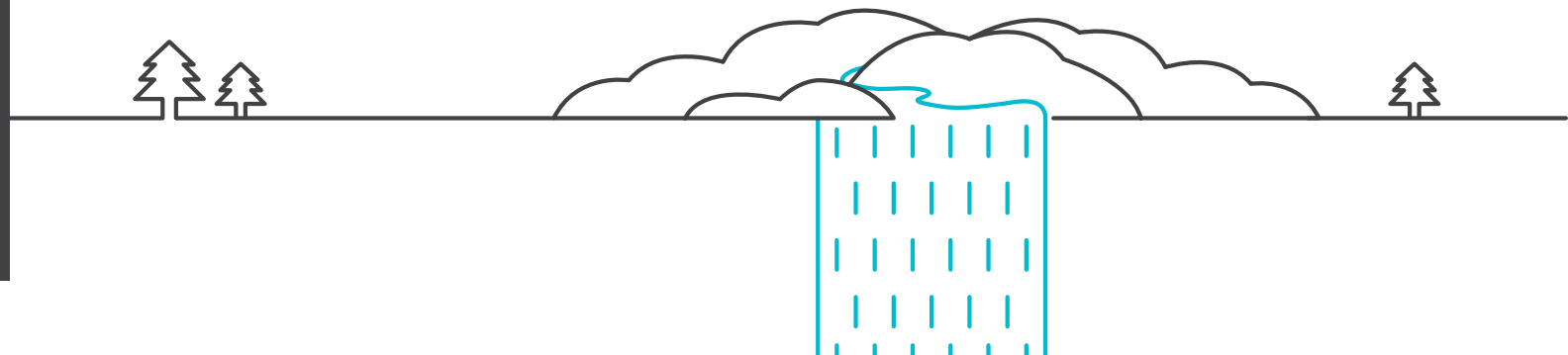
WHERE CAN I MAKE USE OF KANBAN?

At Gear Stream, we have introduced the concept of Kanban to many teams and companies, and while Kanban may appear to apply mainly to maintenance and operations teams, it has also proved its value in the field of product and software development, including stakeholders from Marketing, Product Management, Software Development Operations, and Governance.

We have also found that teams and organizations working in a waterfall environment often find the evolutionary principle of Kanban a more useful catalyst for sustaining an incremental transition to Agile software and product development.

When faced with the prospect of putting any new system into place, it is both natural and healthy to be skeptical, but it is also important for skeptics to be informed by fact and not myth.

Many objections to Kanban certainly fall into the realm of myth instead of rational, fact-based reality.



LET'S EXPLORE SOME OF THE KANBAN MYTHS

	Myth	Fact
	Kanban only works with small, uniform tasks, like those in operation and maintenance.	Kanban has been proven to be as effective in software development as it is for operations and maintenance teams.
	Kanban and Scrum are incompatible.	Not true. Kanban and Scrum work well together, and none of the principles of the former restrict or inhibit the latter. Kanban's focus on making change more visible and optimizing flow can be easily leveraged by teams using Scrum.
	Kanban is subject to Parkinson's Law: "Work expands so as to fill the time available for its completion."	Kanban keeps a tight focus and work item flow through: • Rigid cadences • Acute visualization • Measurement of cycle time • Tight feedback loops
	Timeboxes are not used by Kanban teams.	Timeboxes are not obligatory with Kanban, but the principle can certainly help to optimize quality, flow, and feedback. Kanban teams most commonly use fixed but separate cadences of planning, review, and releases, which removes the need for a conventional iteration model while keeping the value intact.
	Estimates are not used by Kanban teams.	Again, with Kanban, estimates are not obligatory, but using them when appropriate is certainly prudent. Estimates are an excellent way of deepening knowledge in the team, and some degree of preliminary sizing is very useful in optimal portfolio management, prioritization, and alignment. Work that is more sensitive to cost/benefit analysis or due date performance may also benefit from estimation.
	Kanban is an improvement over Scrum/XP/Crystal/FDD or other similar systems.	Kanban is primarily the yeast in the fermentation of change, and therefore it requires a starting point, or a brew. So while using Kanban will be beneficial to most projects, it does not have to be an alternative to Scrum. In fact, in most instances, Scrum offers an excellent starting point for the adoption of Kanban.

Despite widely available literature and case studies within the software and product development community, Kanban is frequently misunderstood as being incompatible with Agile values and methods, including Scrum and XP. Such misconceptions include the following:

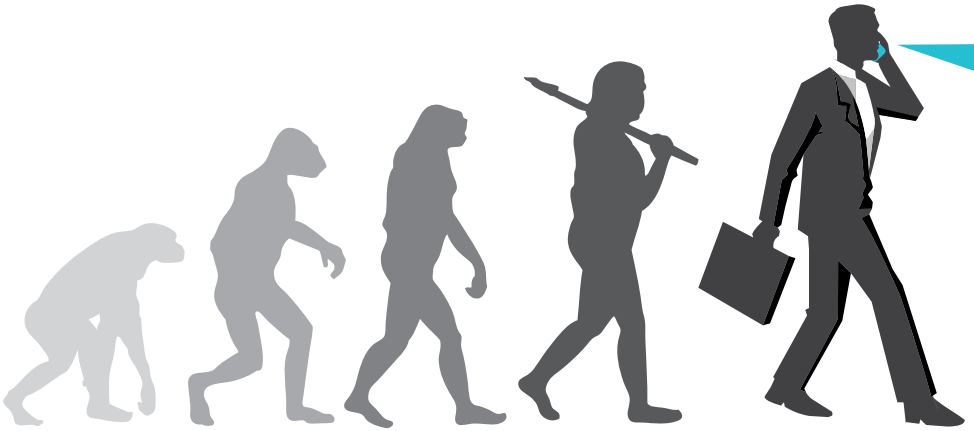
1. The concept of a change method is poorly understood by many Agile practitioners, leading to invalid comparisons with prescriptive methods like Scrum and XP.

2. The word "Kanban" is strongly linked with production systems—thanks mainly to its use by Toyota. Accordingly, many people feel that Kanban isn't appropriate for software development and IT operations.

WHAT IS A CHANGE METHOD?

The natural tendency for most people is to try to relate Kanban to a method they already know and understand, but such an approach can be difficult because methods such as Scrum and XP tend to be highly prescriptive. As a change method, Kanban has very few descriptive parts telling you how to work and which roles to fill.

Kanban is about using Lean principles to optimize existing processes in an evolutionary way. The result is that many Kanban projects show the same emergent practices, which leads some people to think of a distinct Kanban method. In fact, Kanban is not a prescriptive method at all. Trying to compare Kanban directly with Scrum, XP, Crystal, FDD, or any other prescriptive method is inherently flawed.



Kanban is designed to control change within these methods in an incremental and evolutionary way. It is complementary and compatible with most prescriptive methods. When applying Kanban alongside these methods, the result is a more Lean version of the original prescriptive method.

Kanban in Software Development

The concept of Kanban came of age in the Lean production systems of the Toyota Corporation, where these Kanban pull systems helped drive change. In software development, Kanban builds on a broad set of Lean principles. Those who have worked extensively with Lean in the past may still find this a new approach.

Most projects, whether Agile or non-Agile, can make positive use of the Kanban principles set forth in this book to drive change and continuous improvement, such as limiting death marches, increasing market responsiveness, and reducing costs.

GETTING STARTED

Our hope is that this primer will help you transition more quickly (and less painfully) into adopting the key Kanban principles. It is also worth bearing in mind that this book is not intended to be a checklist for you to blindly follow but rather a way for you to learn so that you can move forward.

Each step in the book begins with an explanation of The Why, focusing on why a particular step is important. We will explore the underlying principles of Kanban in these sections.

The second portion of each step describes The How, which presents specific directions on implementing these principles to create a Kanban process.

As you read through these sections, remember that The Why is the more important of the two sections. Even though you may not fully grasp The Why right away, however, then The How can often help solidify the concept with more concrete examples.

As you progress, you may feel that the principles when applied to your organization result in a different process from that illustrated in the book. If so, this is actually perfectly fine! It means you're taking ownership of Kanban principles and personalizing them to suit your particular needs. Congratulations!

On that positive note, let's now move on to how we can apply these principles in practice.

2. MAKING YOUR WORKFLOW VISIBLE

Know Thyself.
- Ancient Greek aphorism

gearstream

THE WHY -

UNDERSTANDING YOUR SOFTWARE DELIVERY SYSTEM

The first step in making informed decisions is to visualize the current workflow process so that you can fully understand the existing situation. You may begin to identify steps that are inefficient, redundant, or awkward . . . possibly even archaic or just plain crazy. You might be tempted to change these right away.

Resist this temptation!

Changing too quickly will certainly be counterproductive. People will feel threatened and will either be silent or try to shut you down. Also bear in mind that such changes will not actually result in the optimization you're looking for.

Your responsibility right now is not to create an ideal process but to visually capture and understand your current process. The key is to try to map your entire software delivery workflow and not simply focus on the development portion.

This visualization, however, does not have to contain each detailed step. Ideally, you should keep the stages fairly broad. Later, when you begin looking to remove bottlenecks, is when it may be beneficial to provide a more detailed breakdown.

Although visualizing the workflow might seem to be fairly straightforward, it can be surprisingly difficult. In knowledge work, some tasks often remain hidden, possibly part of an ill-defined "tribal knowledge" that has never been well documented. Remember, though, that the more difficult this task proves to be, the more important it is.

Change can be hard for any organization to embrace, so you will likely encounter some internal resistance (even though you are not yet actually changing anything). Part of the problem is that people may not want to admit their own ignorance of processes. Perhaps they are aware they are doing things they shouldn't be doing. Maybe they are nervous that their superiors will discover how things really work. These problems need to be fixed. It should be fully understood that neither blame nor disapproval will be assigned to anyone honestly displaying the current status.

If visualizing gets complicated, then you need to stop or at least slow down. Do not proceed before you can effectively visualize all the work that you do.

You can only manage the work that you can see.

You have almost no chance of making informed decisions if important information is hidden from you and some tasks are completed outside your workflow system.

Key Visualization Points

- Map out the process from conception to completion

- No changes. (Not yet!)

- Do not proceed until this is finished

Keep an eye open for these more holistic benefits while going through this process:

- Focus on thinking globally: It becomes visible exactly how your work affects others and vice versa

- Each person knows precisely what is going on, and no information is hidden or withheld

- You will naturally start to question why you are doing things the way you are. (More on that later. Remember: Don't start changing things yet!)

THE HOW -

CREATING A WORKFLOW VISUALIZATION

Not surprisingly, many different approaches to visualizing workflows have been used, but the Lean theory of Value Stream Maps (VSM) is one of the most popular. At its most basic, a VSM is a visualization of the stages work passes through—in the case of industry, from raw material to finished product, or in the case of software development, from a working concept through to a fully functional release.

As with many Lean concepts, however, VSMs were originally developed within production systems, so the transition to knowledge work is not always trouble free. Since knowledge work is not linear, some Agile enthusiasts have designed specific techniques to handle nonlinear work, e.g., Knowledge Creation Networks.

For now, we will focus on the VSM technique because it's simpler and extremely helpful for clear visualization of workflows. As you begin to visualize, however, you should explore other options to find what best suits your context.

The key here is that each stage in a knowledge work VSM is centered on the principal method of information arrival, e.g., a stage called Test, which includes much more than simply testing and could include inter alia, repair, discussion, refactoring, and the updating of established criterion. We will, however, define the entire stage as "Test." The area between stages where there is no specific information is considered "wait time." A real project example is shown in Figure 3.

Initially, you should concentrate on a fairly high-level view of the process, limiting the number of stages. Too many stages tend to transfer the focus from the big picture to the mechanics. Keep it simple for the time being.

Even a bare-bones VSM can trigger improvement ideas.

- Can we improve the five-month wait time between Specification and Implementation?
- Do we really need to wait two weeks to move on to the Test stage?

Remember: Focus on clearly understanding how the current system works and not on actually fixing things.

With a VSM created, you can then progress to creating a Kanban board. Each stage from the VSM translates into a column in the Kanban board, such as the one you see below. These columns are used to track and manage the specific work allowed in each stage, which is the heart of the Kanban system.

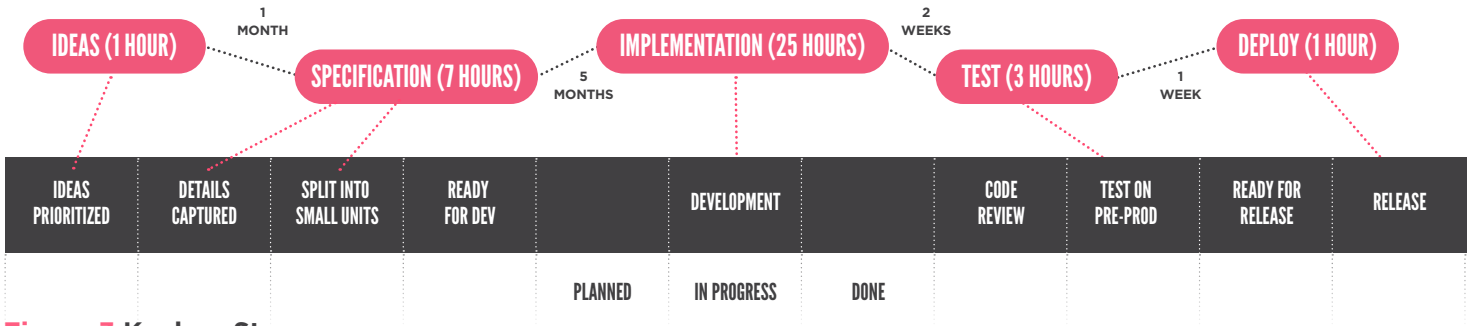
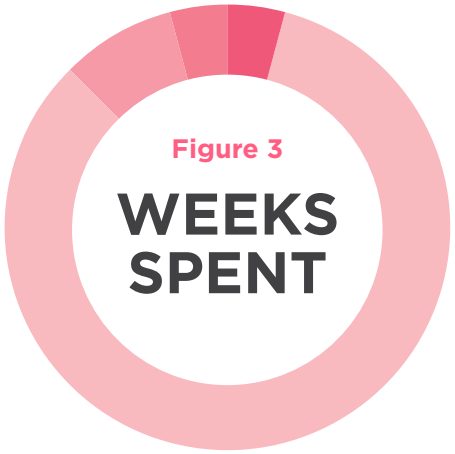


Figure 3 Kanban Stages
The VSM at the top translates into the Kanban stages beneath.



Kanban boards can be generated and illustrated either electronically or simply by using the low-tech option of a whiteboard. We recommend a whiteboard to begin with unless you are working remotely as part of a dispersed team. Having the board right in front of you, with the tactile element of moving colored sticky notes, makes the process more compelling than simply gazing at columns on a computer screen. As team development increases and as it becomes necessary to acquire more data, an electronic version of your Kanban board might prove to be more practical, but for the time being stick to the whiteboard.

Usually, at least two types of stages will initially be identifiable:

- Activity stages—Performance of active work on a product
- Buffer stages—Work waiting to be released, developed, or otherwise moved

A basic board may look something like the example illustrated below in Figure 4. Here we have broken the Implementation stage down into both Development and Code review stages. The development stage has been further broken down into planned, in progress, and done stages.

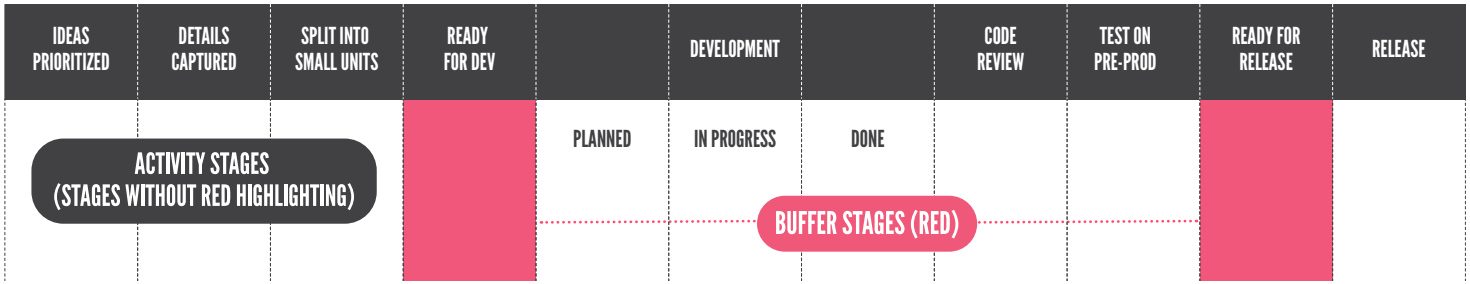


Figure 4 Activity vs. Buffer Stages
A Kanban board consists of Activity Stages, where work is performed, and Buffer Stages, where work is waiting to move.

Work items typically enter a Kanban board from the backlog in the "Ideas Prioritized" stage and exit as working software in the "Release" column. Notice that we have not changed anything; you might still follow a strict Scrum implementation here.

Your process has simply been captured in a visual context, which then allows us to apply Kanban change management principles.

3. CONSTRAIN WORK IN PROGRESS (WIP)

It is the quality of our work which will please God and not the quantity.
- Mahatma Gandhi

THE WHY -

UNDERSTANDING WIP

Once you have succeeded in visualizing your workflow and understand exactly how your process operates, you are ready to begin the crucial next step in building a Kanban system: placing limitations on the WIP allowed at each stage in the system.

At first glance, limiting WIP probably seems wholly counterintuitive. Our culture places a premium on utilizing every resource to its fullest, so putting limits on the amount of work that can be performed simply feels wrong.

The key idea here is that work items not being worked on do not generate any business value. It simply means that resources are being wasted.

To understand why WIP limits are not only sensible but also crucial, we need to examine Little's Law, a formula that applies to any queue system regardless of specific details:

Little's Law

$$\text{Cycle Time} = \frac{\text{WIP}}{\text{Throughput per unit of time}}$$

Little's Law Definitions

Cycle Time: The total time taken for a work item to transit the system, from the moment of implementation to the precise point that it is working in production.

WIP: The amount of Work In Progress in the system. This again depends on your situation, possibly including all uncompleted items or only considering items selected for implementation.

Throughput per unit of time: The average number of items produced in a specific time period. (This is usually referred to as "velocity" in Scrum.)

THE WHY -

UNDERSTANDING WIP

Little's Law

(IN ACTION)

Let's assume that you have the following:

WIP = 100 user stories in progress

Throughput = 2 user stories per week

According to Little's Law, we get:

Cycle Time = 100 / 2 = 50 weeks

This means that the Cycle Time to clear out all of this WIP is going to be 50 weeks, or roughly one year.

gearstream

Remember that we want to have increased flow, which means a low Cycle Time.

If your goal is to reduce Cycle Time, Little's Law makes it clear that there are really only two ways to do this:

1. **Reducing WIP** + 2. **Increasing throughput**

Care to guess which one is easier?

It's far simpler to impose WIP limits on your process than to try to force the system (or the employees) to increase the throughput, especially while retaining a high level of quality in the final product.

Your intuition at this point might be telling you that limiting the WIP means that you are getting less work done, but that is definitely not the case. In the earlier example, a WIP of 100 user stories takes 50 weeks to complete. If you cut the WIP down to 50 user stories, it will take only 25 weeks to complete. Over the following 25 weeks, they'll complete another 50 user stories. It will still take 50 weeks to complete work on 100 user stories, but the Cycle Time has been cut in half, flow has increased, and at any given time fewer pieces of work are slowing down the system.

This drop in WIP is about more than just expedience. Actually, it can have a major positive impact on the defect rate. Evidence and common sense are in accord on this one:

The more things that you're working on at once, the more likely you are to make mistakes.

This relationship can be difficult to quantify, but attempts have shown that the relationship is most often nonlinear. In other words, a given increase in WIP causes a disproportionately larger increase in defects! If this correlation is true, then decreasing the total WIP quantity would disproportionately reduce the number of defects. If the WIP is cut in half, then the defects should be cut by more than half.

This alone is a clear indication of why it's best to keep clear limits on WIP at any given time.

Key WIP Limit Points

- A continuous flow of work is represented by low Cycle Time.
- Cycle Time can be decreased by limiting WIP.
- Greater WIP results in an even greater defect rate.

THE HOW -

SETTING WIP LIMITS

Now that you understand why the WIP limits are so important, there are two steps to actually determining how to do it. First, you have to incorporate the limits into your Kanban board, and then you have to decide what those limits should be.

Visualizing WIP Limits

No specific rules govern how you visualize WIP on a Kanban board, so it depends very much on how you set up your process visualization. Two common ways are shown in Figures 5 and 6.

Figure 5 uses boxes that can contain only one WIP item, which, when the box is empty, offers an obvious indication at what point a "permit" is available to start/pull new work into the box.

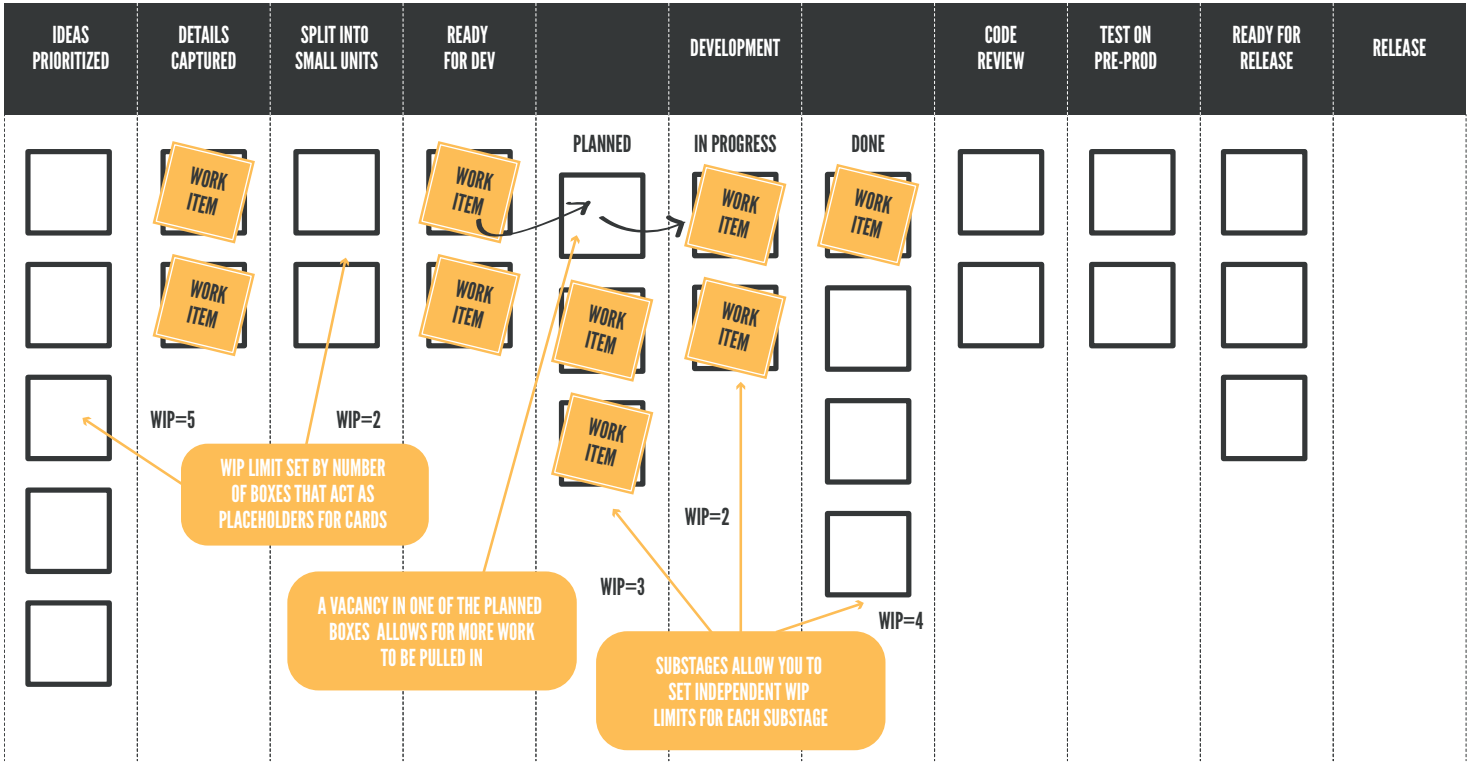


Figure 5 Kanban Boxes
Each box represents an opening for a piece of work.
The notes attached represent the work items that fill the openings.

THE HOW -

SETTING WIP LIMITS

In Figure 6, the WIP limits are written in the column headers. The Development stage is split into three substages and it may be helpful to define WIP limits for each of these stages, depending on your context. This can give you additional insight into how your system is working and is the common way of doing it in IT.

Figure 6 WIP Limits
The WIP limits can also be written explicitly in the stage headers.

If you have worked with Lean manufacturing in the past, you might feel more comfortable with the idea of the physical plastic cards used in those systems. That more accurately approximates the first example from Figure 5.

Finding the right WIP limits

Once you have established how to visualize the WIP limits, the next step is the tricky business of establishing what numbers to use. Many different theories have been proposed on how best to achieve this, and we will do our best to present some of these ideas.

Remember that the WIP limits are really just the answer to the following question:

How many items will we allow in each stage of our board at any given time?

Remember that Kanban allows for incremental, evolutionary change, so these numbers will not be set in stone. If you find that they are too high or too low, they can be modified later on to suit your needs and keep a smooth flow running through the system.

Individual teams have to enforce the system, so it's a good idea to let the team guide its own WIP limits. If a team of six decides that it is a good idea that no developer should work on a user story single-handedly, then the WIP limit would be three. At any given time, the six-person team will be working on three user stories, and it will be the team's responsibility to enforce this.

Consider two different approaches that a team might use in setting their WIP limits:

The Conservative Approach: Set WIP limits just loose enough for your current workflow to continue unhindered; then identify your bottleneck and adjust one limit at a time.

The Radical Approach: Set WIP limits on activity columns tighter than you expect your system to be able to handle and buffer each stage; then observe where work builds up, and gradually loosen until work flows through the system.

Both approaches require some experience, so don't expect to get it right the first time. (This, by the way, is an excellent rule of thumb for every new technique suggested in this book.) There is no final consensus as to which one is better, but setting the limits with your policies in mind seems to work in both circumstances.

If you really have absolutely no idea where to start, then the running joke among insiders is that the universal WIP limit is five, which is as good a place to start as any.

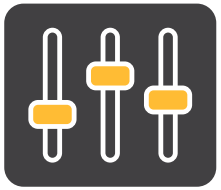
THE HOW -

SETTING WIP LIMITS

Buffer Stages

Monitoring the capacity of such buffer stages as "Ready for Development" is helpful in evaluating your WIP limits. Some guidelines for adjusting the size of these buffer stages include the following:

- If a buffer stage is empty only once a year, the WIP limit is too large. For 364 days, you are working with a larger buffer than needed.
- If a buffer stage is emptied on a daily basis, it is too small. Every day you are in danger of running out of work.



Policies for Change

Going into the process, you should expect the WIP limits to change, which is crucial. Your initial limits are just best guesses made when you had the least amount of information available. As you gain more information, limits should be adjusted continuously as you optimize the work.

The goal of Kanban is to facilitate continuous improvement!

If, after three months of operation, you find that you are still working with your original starting limits, it can be assumed that you have likely missed some opportunities for improvement.

Overly tight limits can have the effect of blocking the flow, which has the additional negative effect of idleness in the line or that the limits will simply be ignored. Broad limits, on the other hand, will tend to increase cycle time, rendering work items idle for longer than necessary.

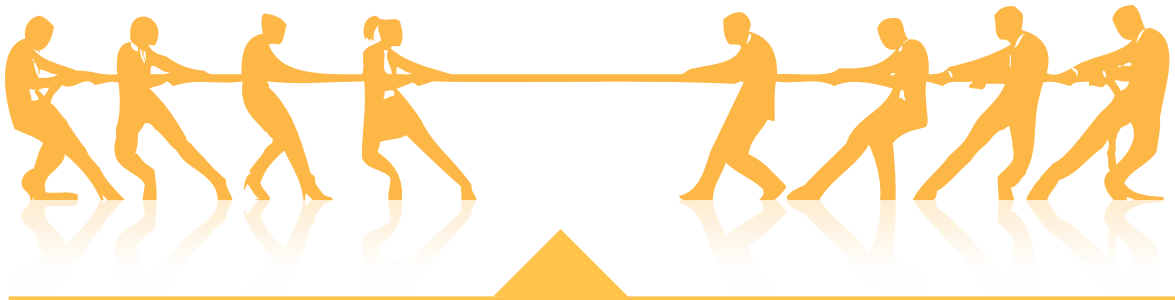
It is important to set an explicit policy on how decisions to change the WIP limits will be made. To maximize learning, the whole team should be involved in this process. If everyone on the team gets the chance to voice their opinion and understand the decision, it becomes a learning point rather than just a matter of flow.

Adapting to a Pull System

Correct WIP limits mean that your system can only work to capacity.
To get permission to start something new, you need to finish the work you have.

While this might point might seem trivial, it is actually the key concept and an extremely useful vehicle in your journey toward a more efficient, sustainable, and predictable software delivery system.

A good metaphor to illustrate the different effects of push and pull is to imagine a chain link or a rope or even a fishing line. So long as tension is applied, the system works. This is the pull, but when you start to push, control is lost. The system loses cohesion, and the fish jumps off the hook.



You will become frustrated when you embark on new work that you feel is the correct course of action, but permission is not forthcoming simply because capacity is not available. Such a development is a clear indication that you have encountered an impediment to flow.

Figure 7 Pull vs. Push
What you must not do, though, is respond negatively. Instead, consider this obstacle as an opportunity for improvement!

4. DEFINE AND MAKE QUALITY POLICIES EXPLICIT

For you to sleep well at night, the aesthetic, the quality, has to be carried all the way through.
- Steve Jobs

gearstream

THE WHY -

WHAT IS QUALITY?

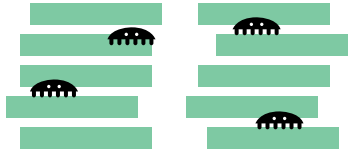
Steve Jobs was obsessive about many things, but probably his biggest obsession was his maniacal focus on the quality of the customer experience when interacting with Apple software and products.

Achieving this kind of outcome doesn't happen by accident. It's the result of intentionally engineering quality into every single aspect of design, implementation, and delivery.

Lean values and principles incorporate the concept of "quality built in." Quality, though, can often mean different things to different people. It may mean following intent or guidelines or structural coherence. However you define it in your context, maintaining a high level of quality is key.

Why is quality so important? While quality impacts the degree to which your customers and users will enjoy the benefits of the new software or product you're building, there is something more fundamental that should guide our emphasis on quality. In the simplest terms:

Poor quality results in more work.



If an error occurs or a product is defective, the likely result is additional work further down the road. The nature of the poor quality does not really matter so much. It might be some counterintuitive process within the system that limits the user's ability to complete a task, or it might be a bug that completely inhibits the workflow. Each of these puts stress on the system and creates an unnecessary cycle of waste. Having clear quality policies is crucial to eliminating, or at least reducing, this wasted time and energy.

In Lean circles, when a product is poorly designed at inception, all activities and rework related to that failure are better known as failure demand: a demand for work that is caused by a failure. In contrast, a software company is ideally in the business of responding to value demand, which is the demand for work because of the value obtained.

As negative as it might sound, there are certainly some cases when having some failure demand is acceptable . . . at least from the developer's point of view.

THE WHY -

UNDERSTANDING QUALITY

For example, some complex problems are expensive to uncover. It can be genuinely cheaper and more efficient to release the software to elicit actual feedback from real customers using the system. Yes, fixing the problem after release might be expensive, but it could still be cheaper than it would have been to have discovered this through up-front testing.

Ideally, these cases are a clear business decision made with as much knowledge as we can reasonably have at the time.

But let's be honest. Failure demand is most often simply caused by an immature process. It is a mistake . . . and an expensive one!

To understand how expensive failure demand can be, let's examine what happens when a user named Jim is unable to complete his task thanks to a bug in our system.

The first thing he will probably do is submit a bug report or make contact with first-level support for help with his problem. It is probably fair to assume that the information Jim provides will not be particularly accurate or even complete.

If so, then the first-level supporter will pass defective information to the developer who will then be confused, ultimately solving a completely different problem that actually might not have been a problem at all—very possibly compounding the original problem with the introduction of a brand-new bug!

Therefore, an incomplete understanding of the original bug might well introduce a secondary bug that will then require the development of a complex SQL script to restore data to a clean state. Meanwhile, Jim is delayed in his task with the possible result that another bug report is submitted to first-level support, and the whole cycle starts again. Clearly, such a situation could spiral completely out of control, and it would not be unusual for time and resource costs to range from 100 to 1,000 times in excess of what it would have taken to address the bug in the preliminary stages.

Thus, Lean strongly emphasizes a fail-safe delivery system, a process known as Poka Yoke—achieved in production systems by carefully following standards and checklists after completing a task.

In certain instances, photocells are used to determine if a specific tool has been used the correct number of times or if each part required in production has been removed from the heap.

Although this might appear on the surface to be a somewhat inhumane working environment, workers in the Lean context rarely see themselves as automatons responding numbly to standards and checklists but rather as skilled operators working constantly to improve their systems by evolving new and improved concepts. Toyota has been able to implement one million new improvements every year, based largely on the input from these experts.

Key Quality Points

- Problems with quality generate waste and extra work.
- A failure can take up to 1,000 times more resources to fix than to avoid in the first place.
- Explicit standards and checklists provide the metrics for ensuring quality.
- Expert operators are crucial for ensuring that improvements are made to increase quality.

THE HOW -

VISUALIZING QUALITY POLICIES

No doubt you already have quality assurance (QA) policies in place, so that is our starting point. In Agile circles, this is called "Definition of Done." The next step is to add these policies explicitly to the Kanban board. Your Kanban board should then begin to look something like that featured in Figure 8.

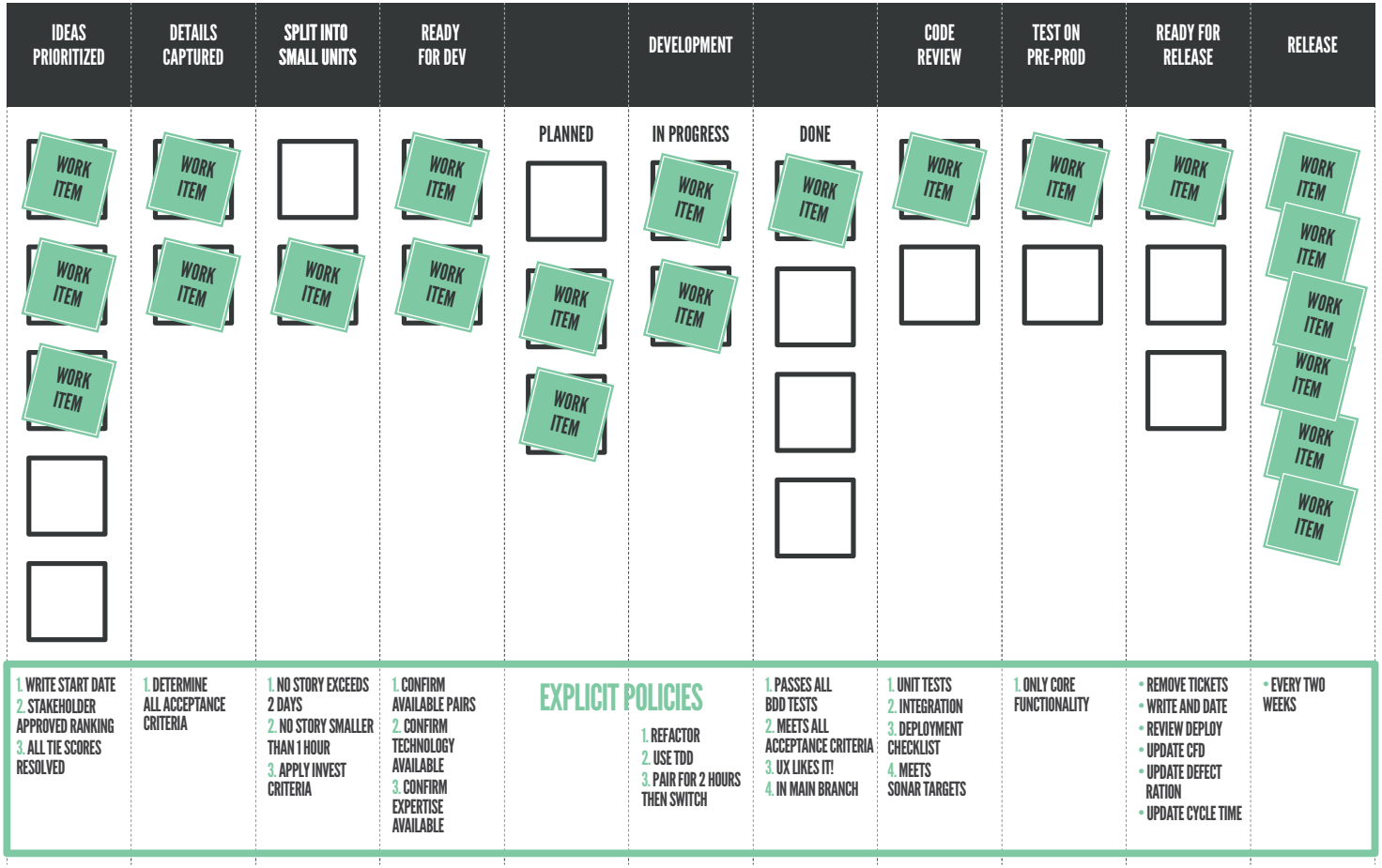


Figure 8 Explicit Policies
The policies are explicitly spelled out at the bottom of the Kanban board.

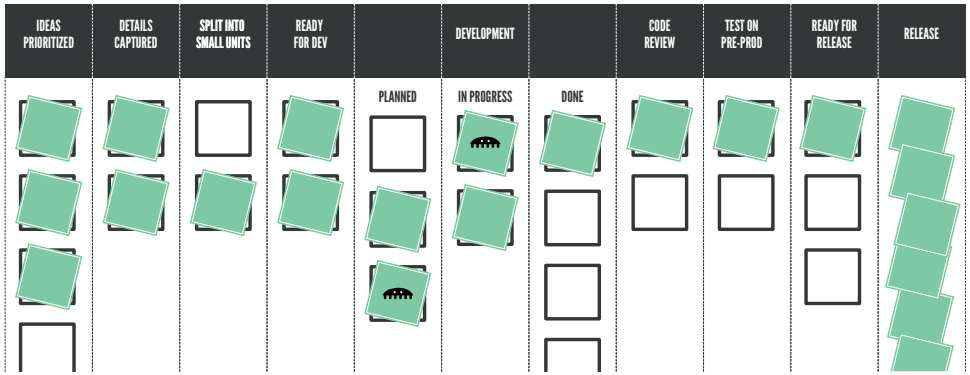
Note that policies are stated for each stage where work is being performed.

Consider the "Code Review" stage. On the surface, it might seem obvious what this QA stage should entail, but different people might make different interpretations. It is not enough to just say that someone has to review the code—the specifics of what that review entails must be absolutely explicit.

The importance of having these systems in place becomes clear when you consider the "Lean Start-up Movement," a system that presents software to production a number of times each day, with neither downtime nor high rates of defect. This is achieved by the introduction of fail-safe systems that protect against entire classes of error:

- Unit, integration, regression, and performance tests in place to validate the code.
- Key Performance Indicators trigger an alarm should the system for any reason not behave as expected.
- Automatic system rollback to previous versions as needed.

You will begin with the current procedures, but expect to add, remove, and change them as you determine fresh and improved methods of guaranteeing quality. Look at every bug as an opportunity to learn and improve your system. How did it get there, and how can similar bugs be prevented? The thought of reflecting and learning from every bug will seem intimidating at first, but it is the only way to eliminate errors. With time, and as quality improves, the process will begin to feel more and more natural.



While such policies and checklists are crucial, they are there to liberate, not constrain. You are an expert knowledge worker, not a robot, and your role is to constantly observe and try to improve the system.

5. CALIBRATE DELIVERY CADENCE

Beauty of style and harmony and grace and good rhythm depend on simplicity.
- Plato

THE WHY -

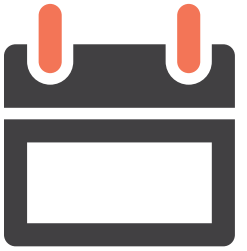
WHAT IS CADENCE?

Evaluating cadence is the next step. Finding the optimal delivery cadence is one of the most important things in Lean product development, since it helps you create essential feedback loops, reduce risk, and optimize your delivery process.

If you are already familiar with Agile systems, the cadence concept will be familiar to you too.

In simple terms, the cadence concept implies a regular interval upon which you have established that a set activity will take place.

A delivery cadence, therefore, is the schedule upon which you've agreed to deliver working software.



Cadences and Trust

One reason for cadences is that they establish a pattern within the organization.

A set cadence means that at regular intervals the team shows that it is meeting deadlines. The delivery of several small but consistent batches does more to build trust than a handful of large batches.

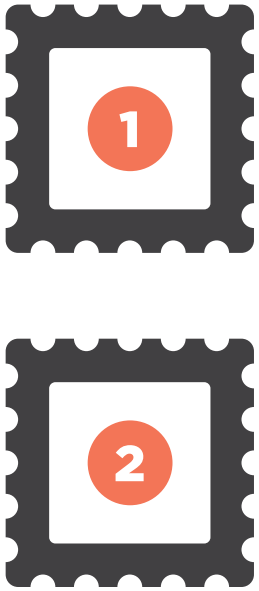
In a business setting, the relationship between dependability and trust is not limited to delivery. Imagine that you have a friend who meets with you regularly, either at a bar or going to a gym to work out, or perhaps even just occasionally calling to see how you're doing. This friend remembers to touch base with you on a regular and consistent basis. Another friend, on the other hand, rarely calls, but visits erratically, occasionally bearing gifts. Which one of these two engenders the most trust in the relationship?

Consider the "Lean Start-up Movement" discussed previously, which deploys software to production several times a day, with both low defect rates and little downtime. If you work with such a team, what level of trust would you enjoy? How much would you be able to depend on their ability to deliver on their commitments? If they make a commitment to do something, they have a track record of dependability and a great deal of credibility.

Would you not like to be a team that engenders this level of trust? Fortunately, you do not need many cadences each day to achieve this trust.

THE WHY -

WHAT IS CADENCE?



Variable Cadences

Not all activities are created equal, and the cadences associated with them will also be different.

In a Kanban system, we do not synchronize everything to the lowest common denominator. Each activity can have its own optimal cadence.

We could consider several different types of cadences:

- Planning (input) cadence
- Delivery (output) cadence
- Review/retrospective cadence
- Quality assurance cadence

For the moment, we will focus on the delivery cadence. In a pull system, the delivery cadence draws the work out of the system at a rate determined by the customer's needs, which, in turn, pulls available work through the rest of the workflow. The delivery system is crucial in this regard.

The most optimal delivery cadence, if your system can handle it, is to release every feature directly to production. In reality, though, most projects utilize two delivery cadences:

- The first cadence deploys code to a preproduction system in order to acquire preliminary reaction (internal release cadence)
- The second cadence deploys the new version to the real production environment (external release cadence)

On "Greenfield" projects in particular, these two cadences can be quite widely separated. For example, code might well be deployed and tested daily on the preproduction environment for several months before the system is ready to roll out into production.

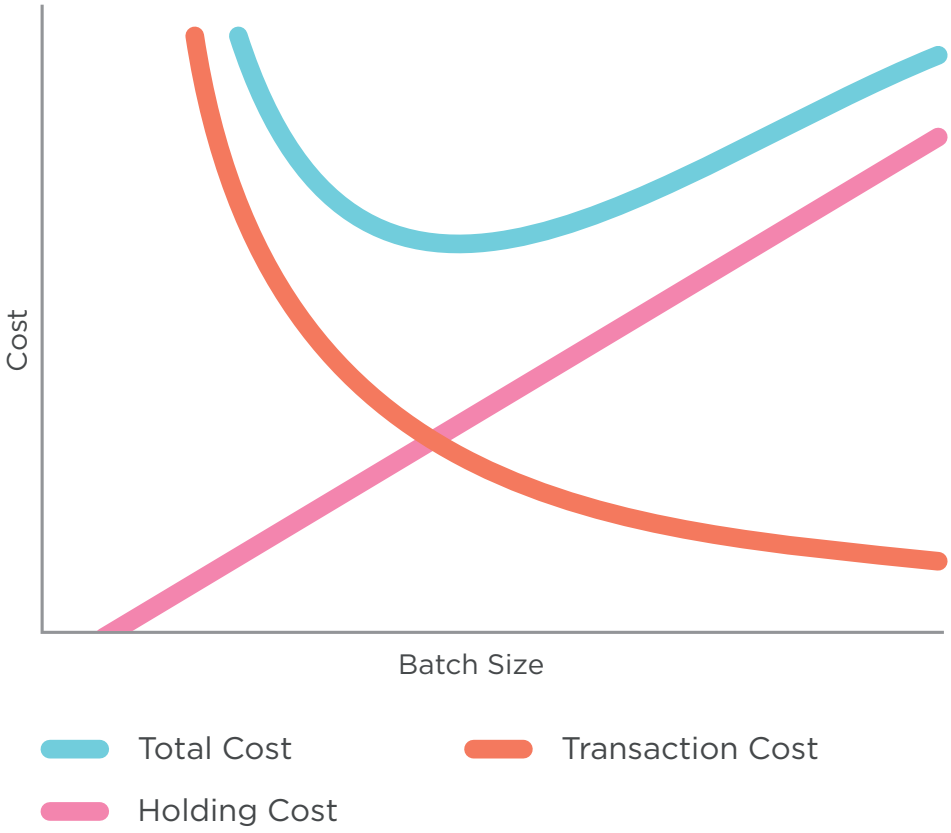


Figure 9 Batch Size Optimization
From *The Principles of Product Development Flow,* by Donald G. Reinertsen. Celeritas Publishing: 2009. Copyright 2009, Donald G. Reinertsen

Cadence and Cost

It is important to take into account the economic cost of choosing the optimal internal and external release cadence. Transaction cost is always a factor (the expense accrued when shifting your version from one environment to another) in release, and do not forget holding costs associated with waiting. This is visualized in Figure 9.

You can lower the cost per feature (transaction cost) by releasing a batch of software, but doing so means holding on to those features longer. This yields a higher holding cost associated with loss of business value, outdated feedback, poorly informed decisions, and reduced user participation.

It is clear from the figure that the "total cost" U-curve features a noticeably flat "bottom."

It is not really important for you to strike the optimal release cadence, since a 10 percent or 15 percent margin of error will still produce a good result.

Key Cadence Points

- A cadence is a regular interval where a set activity takes place.
- Regular cadences build trust.
- Each activity type may have its own cadence.
- Delivery cadence may be split between internal and external releases.
- Optimal delivery cadence is determined by balancing the holding costs with the transaction costs.

THE HOW -

DETERMINING THE CORRECT CADENCES

Once you have established the batch size that minimizes total cost, you then have to predict which cadence will most likely result in releasing batches of that size. As with every other step of the process, you may need to adjust the cadence later as you gain more information. The goal is to improve things, so don't worry too much if your initial cadence does not end up being perfect.

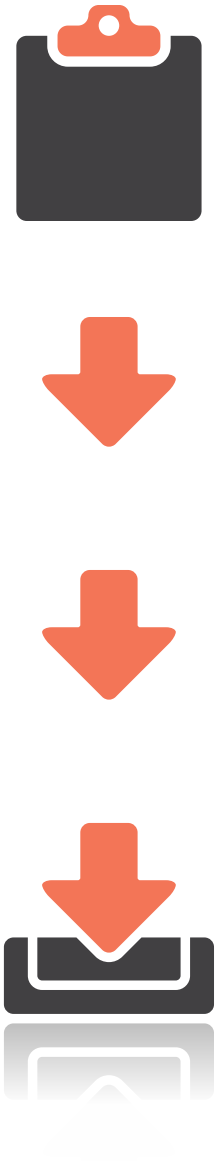
Cadence is not always considered as carefully as it should be. Though many mature Agile teams can release to a preproduction environment at the turn of a switch, they still delay collecting feedback on a specific feature from users for three weeks.

In a situation where transaction costs can be determined in single dollars, it is advisable to limit batch sizes and use short-term cadences to quickly distribute them to users.

Since users are not always available, this may not always work, but at the very least it should be considered. (Working with a demand cadence, though, as described below, means you're always delivering directly based on customer demand, so users are always available.)

Toyota has taught us yet another key point in this regard—transaction costs are always variable. What we have learned from the continuous deployment movement is that it is possible to deliver consistent versions to production fifty times daily for systems handling millions of dollars.

The only way this can be achieved is by implementing a fully automated deployment procedure alongside an entire set of unit, integration, and regression tests. This process is an investment, but it does allow work in batch sizes comprising no more than a few lines of code at a time.



Planning Cadence

So far, we've discussed the delivery cadence, but that's not the only cadence that has to be taken into account. After all, before work can be completed and released, it has to be put into the system so that it can get started.

Planning cadence determines how frequently the group will meet to plan the work that enters the Kanban system.

If time lapses between planning meetings extend for too long, then it stands to reason that much more planning becomes necessary, with the result that there tends to be less in the way of informed designs and more design in progress. It is also true that more frequent meetings, every day for example, create their own costs and inefficiencies.

Let's say you set up a weekly planning cadence. You set a weekly meeting—perhaps on Monday morning—and all the key decision-makers know what is expected of them. They appear on Monday morning ready to decide what goes into the input queue and in what order. The pull system takes care of the rest.

A common Kanban strategy is to plan on demand, which can be achieved by carefully monitoring the input queue and then arranging meetings as needed to populate the queue with new work when needed, such as when it drops below a preset number of items. On-demand methods usually work best in more advanced team environments where some prior experience in handling flow is available. It makes sense then to consider your particular circumstances very carefully before you adopt this plan as your initial strategy.

6. TRACK AND MEASURE YOUR FLOW

Progress has not followed a straight ascending line, but a spiral with rhythms of progress and retrogression, of evolution and dissolution.

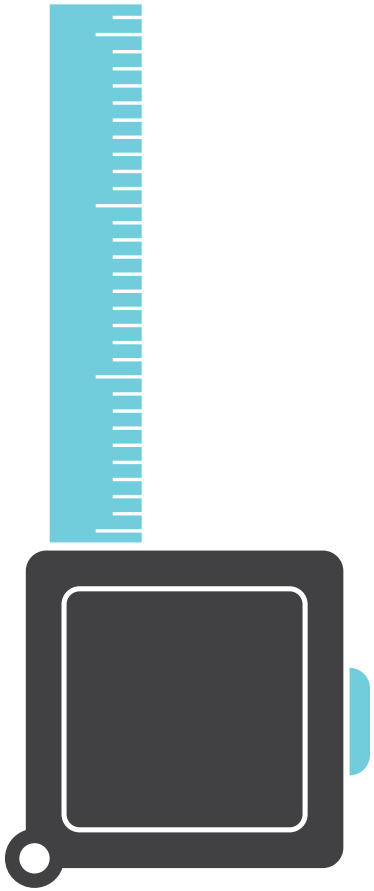
- Johann Wolfgang von Goethe

THE WHY -

WHAT ARE METRICS?

It is ironic in an industry that is based entirely on information that so many companies misapply or misunderstand the information they gain about their own work.

Software development metrics are recognized as being one of the least understood and regularly misapplied factors of software development. Quite often the result is that project managers are held accountable for aspects of development they had no control over to begin with or the establishment of fixed success criteria at the onset of a project at a point when little is known about the system under development.



Metrics Measure Capacity

When considering metrics, it is easy to forget a simple ground rule:

Your software delivery system has only a certain capacity.

If you have worked in software development for any amount of time, you will be aware that trying to pressure your system beyond its capacity will more than likely result in some combination of the following:

- Lower quality
- Unsustainable pace
- Higher maintenance costs

Worse still, the software development culture sometimes actively encourages this result. Frequently, we observe project managers celebrating that either by extended overtime or through some other commendable effort they have driven their teams to rescue a project at the eleventh hour when all seemed lost.

And although we should always recognize and reward achievement, successful software development is not built on the principle of a last-minute fix but rather as a healthy and sustainable process. The alternatives, notwithstanding the heroics, are simply not cost-effective. A good rule of thumb in such situations is provided by Ken Beck: “If you have a problem that requires more than one week of overtime, you have a problem that should not be fixed by working overtime anyway.”

THE WHY -

WHAT ARE METRICS?

It is also worth avoiding wishful thinking from the top down as a template for success or by juxtaposing the success of one group as the standard to be applied to another, notwithstanding that each group has a different skillset, varying weaknesses, and task assignments. Another simple rule to remember when considering your capacity:

Your system cannot exceed its proven capacity.

A Fresh Start

When starting on a Greenfield project, it is fair to say that your system will have no proven capacity at all. It stands to reason then that in the beginning you will need to rely on educated guesswork regarding the capacity and capability of your system. The key point here is to carefully track progress from the onset to prove or even disprove your initial assumptions.

Bear in mind that from the moment the project actually starts, the standard of capacity assessment ceases to be based on a series of educated guesses but instead relies on definitive information.

It is important to frame your plan not as a criterion for success but as a tool for alignment and also to measure your flow to determine whether or not you are still in alignment. What we ideally want is a software delivery system that is stable and predictable to be able to form a basis for informed decisions on deadlines, dependencies, staffing, scope, and budget.

Key Metric Points

- Metrics are measurements, not success criteria.
- A system has only a set capacity.
- Surpassing capacity leads to defects and higher costs.
- Metrics allow you to understand your system's actual capacity.

THE HOW -

MAKING FLOW MEASUREMENTS

A question to consider now is how do I measure flow? Many ways are available, but perhaps an even more important initial question should be taken into account:

Do you intend to act on the information you acquire?

If not, then it is probably fair to assume that you are wasting your time.

> ***If you do not intend to implement change based on a chosen metric, then don't bother measuring anything at all.***

Chances are you already have some metrics in place. Below are four common metrics followed by explanations of their use:

Cumulative Flow Diagrams (CFD)

Cycle Time

Defect Rate

Blocked Items

CUMULATIVE FLOW DIAGRAMS - (CFD)

Cumulative flow diagrams (CFD)

Cumulative flow diagrams display the current amount of work in your system for each stage over time, as shown in Figure 10, which allows for an easy visual grasp of how work is distributed at any given time.

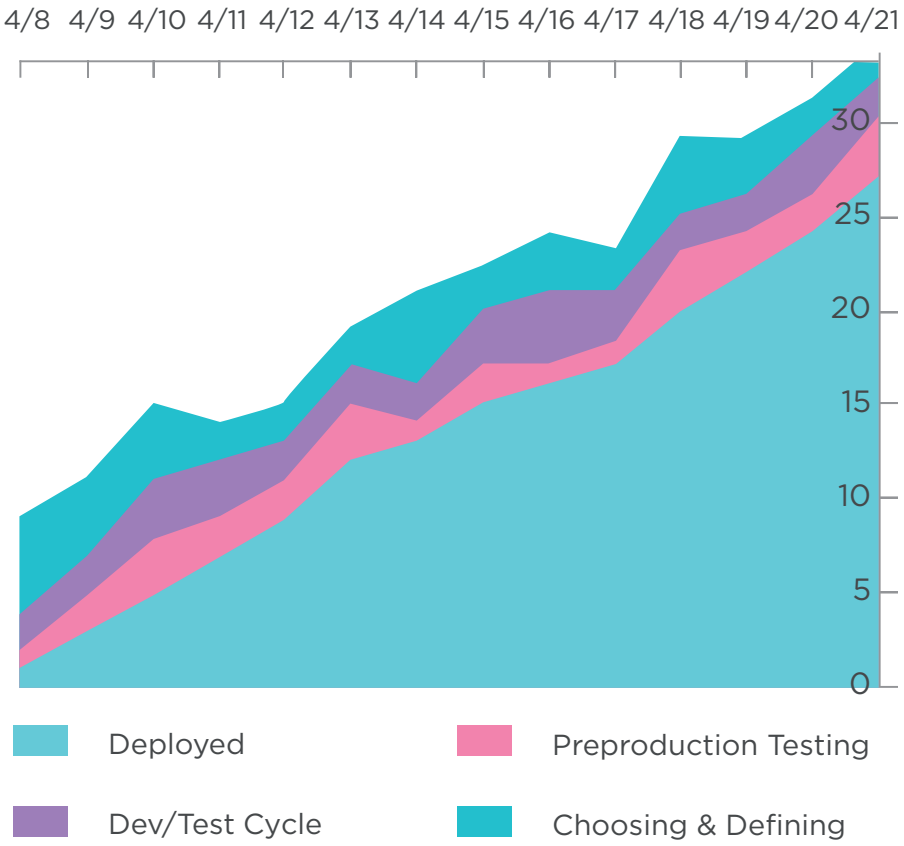


Figure 10 Cumulative Flow Diagram Example

Agile teams may be familiar with "burn down charts," but lately more mature Agile teams have been switching over to CFDs because they are easier to update and give better insight into the project's status.

If you're using burn down charts, you may want to switch.

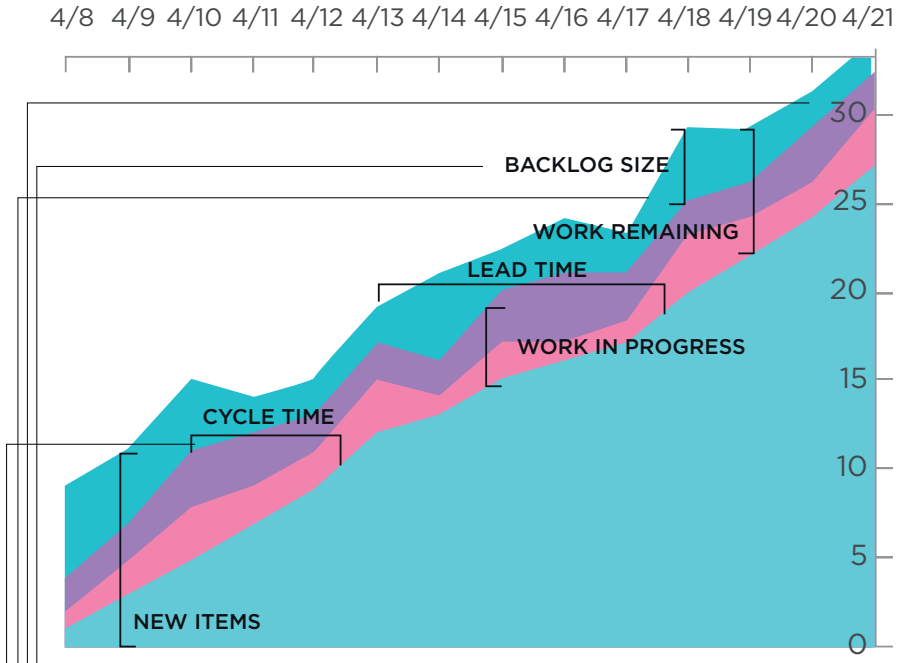


Figure 11 **Interpreting a Cumulative Flow Diagram**

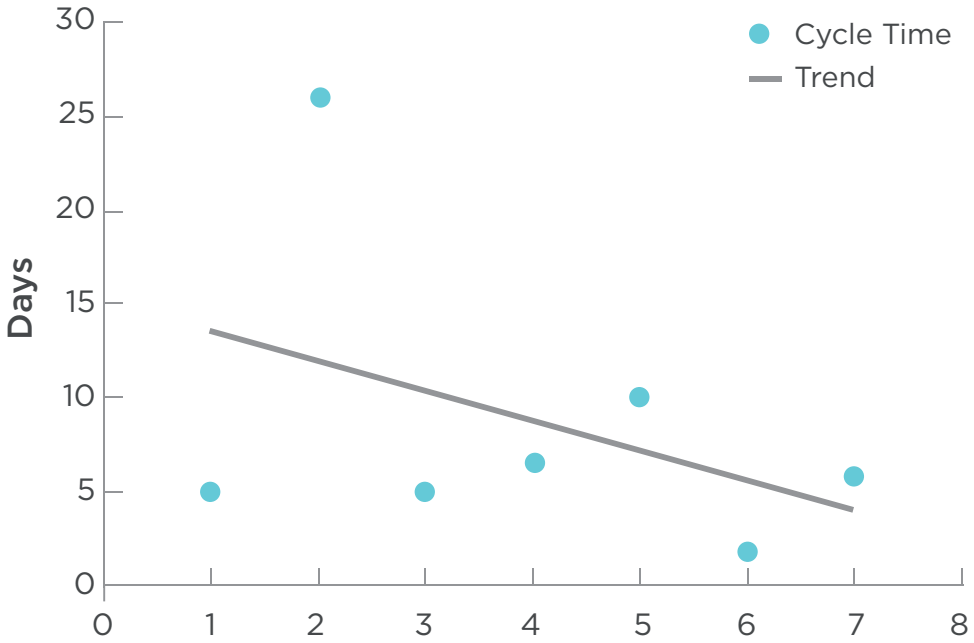


Figure 12 **Cycle Time Diagram Example**

Interpreting the CFD

Your velocity over time is defined by the "Done" area, while WIP is indicated by the space between this line and the "Backlog" line.

- A sign that a bottleneck is occurring would be an increase in the width of a portion of the WIP area.
- A clear sign that work is exceeding capacity in your system is if the pitch of the "Backlog" area is steeper than the "Done" area.
- Your current best estimate of a final release date would be estimating the point at which the gradients of "Backlog" and "Done" intersect.
- You can also determine from the diagram your Average Cycle Time and Quantity in Queue.

Learning to interpret a CFD is not difficult, and Figure 11 offers a very good illustration of how different aspects of the graph can be read.

Cycle Time

Although the Cumulative Flow Diagram is useful for indicating average cycle times, averages can sometimes be deceptive, and a higher level of predictability is possible by keeping track of individual Cycle Times. Accordingly, a visual representation of Cycle Time is useful in establishing the reliability of your system and offering the opportunity to more precisely meet customer demand.
Tracking Cycle Time is also simpler than updating the CFD, since only two pieces of information recorded about each piece of work are required:

- Record the date work began on an item.
- Plot the number of days required to complete the work.

As described previously, these data entry steps must be explicitly part of your Kanban board so that there's no chance they'll be overlooked in the process. The result will be a Cycle Time diagram much like Figure 12.

RATES OF DEFECT

Although this point has been emphasized already, it can never be overstated just how extraordinarily expensive defects can be. Keeping a clear handle on both the rate of defect and the incidence of bugs in your system is a simple method of ensuring that quality issues do not become a serious problem.

Notwithstanding the valuable information that can be generated regarding the status of your project, surprisingly few organizations make use of rates of defect as a KPI (Key Performance Indicator). Possible indications include the following:

- If new defects are occurring, could it be that you have relaxed some of your QA policies?
- What was the effect of a high level of bugs in week twenty on cycle time?
- When the number of bugs increased, what impact did this have on the cumulative flow diagram?

In general, keeping a close eye on trends is better than reaching conclusions based on individual data sets. After all, one bad week might simply be a coincidental. Figure 13 is illustrative of a Defect Rate diagram.

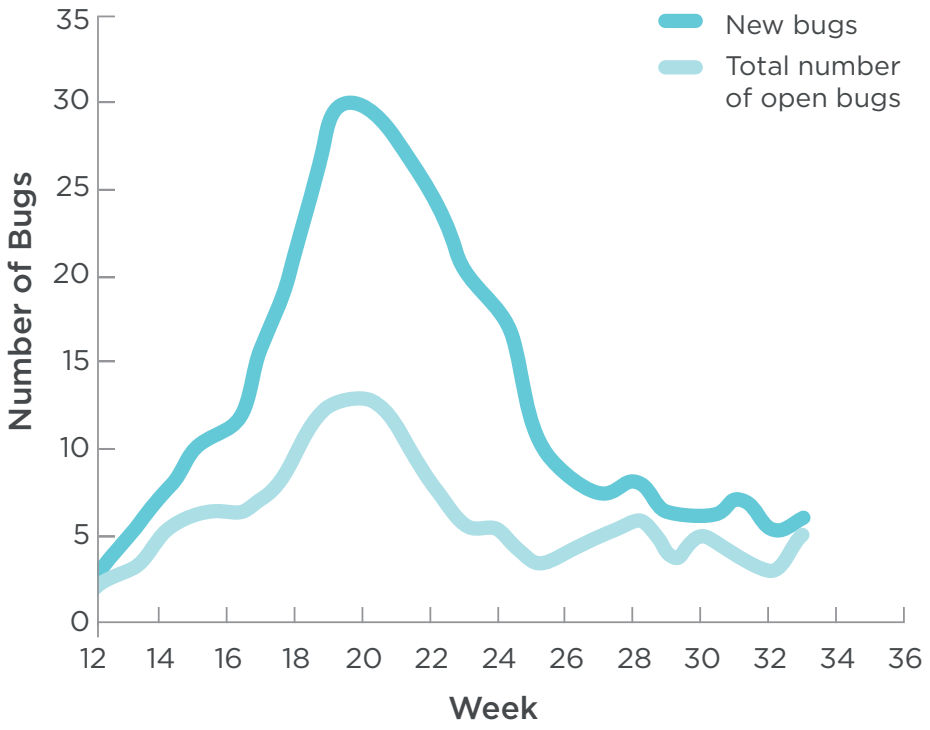


Figure 13 **Defect Rate Diagram Example**

It is usually a good policy to aim at keeping the total number of bugs between zero and twenty. As the list increases, so time is drawn away from actually fixing bugs to administrative tasks, such as:

- Checking for double entries
- Eliminating outdated issues
- Rechecking bugs that have already been fixed

In addition, as the list approaches fifty or a hundred, demand for more reports and tracking increases. Before you know it, you will be swamped by bug management boards and weekly bug meetings, which all tend to absorb time and resources. Even superficial bugs demand attention and take time to administer, so try to avoid the trap of permitting even fifty to occur. **Select a policy and stay with it.**

EXAMINING BLOCKED ITEMS

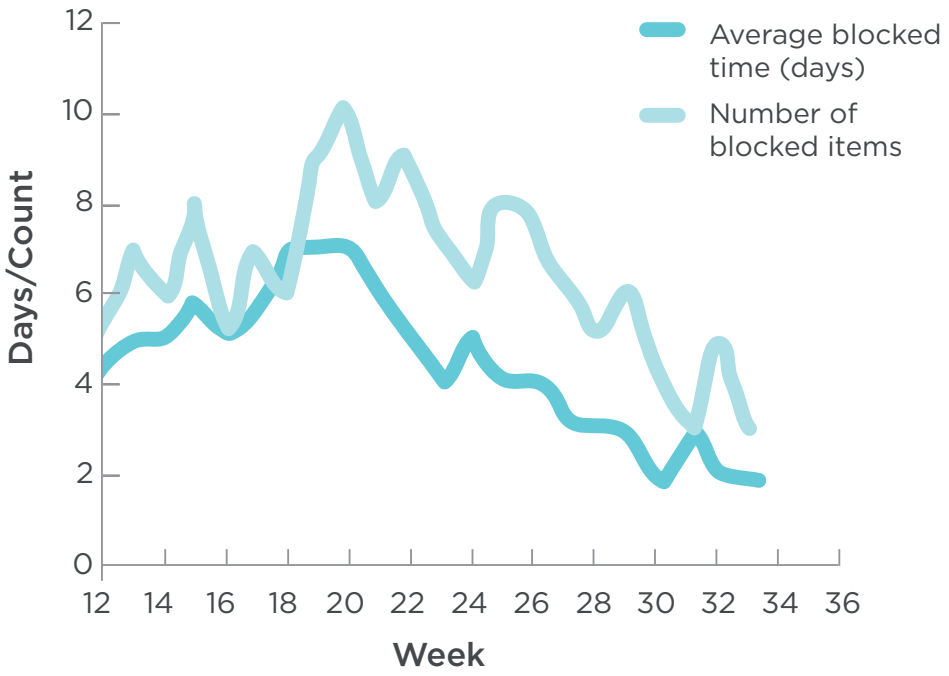
If you haven't caught on to this idea yet, regular flow is important for a system's ability to act predictably. Individual processes become less effective as flow is inhibited. Despite the goal of flow, the reality is that items will become blocked for one reason or another.

If you have not grasped the idea yet, remember that regular flow is vitally important for a system's capacity to behave with predictability. Individual processes become less effective as flow is inhibited. It stands to reason that despite the goal of flow the reality is that blockages do occur for one reason or another.

The previous metrics may give clues about blocked items (they will show up on CFDs and the Cycle Time diagrams), but it's useful to explicitly and visually track blocked items. You will be able to glimpse the team's ability to handle issues blocking one or more features in the system.

Some companies even use this as the leading Key Performance Indicator.

Blocked items have serious long-term effects on the system, so a team's ability to quickly solve issues says a lot about the team's performance and effectiveness.



Blocked items should always be visible on the board, and tracking the status over time is usually a good way of knowing whether or not the team is moving in the right direction. Figure 14 shows an example of a Blocked Items diagram.

Figure 14 Blocked Items Diagram Example

BLOCKED ITEMS

Attaching a pink sticker to any errant feature has become the standard method of visualizing blocked items. Upon this is written details regarding the dates and time that the blockage occurred. This is illustrated in Figure 15, which indicates a board where a pink sticker marks a blocked item.

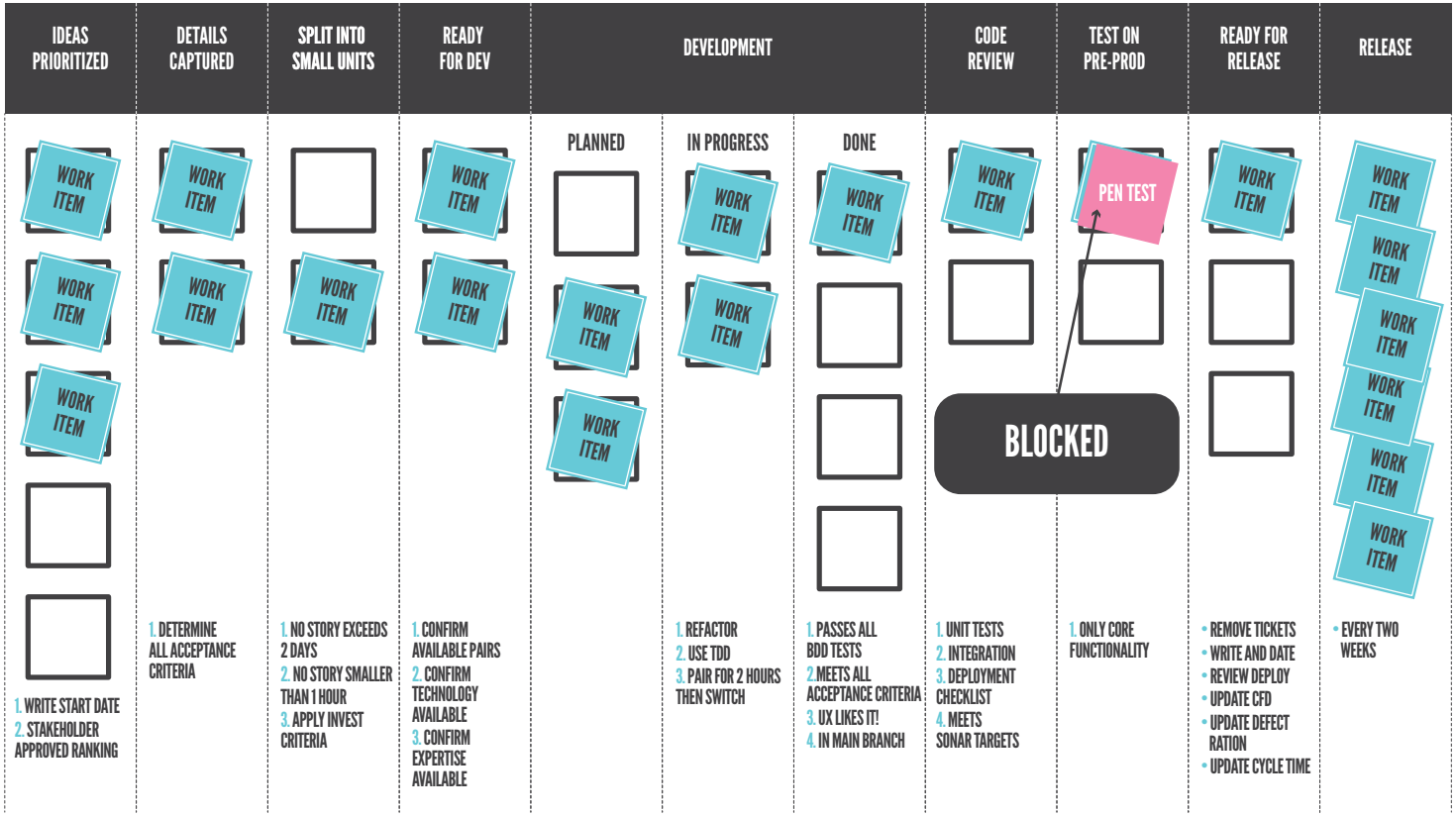


Figure 15 Blocked Items.
A pink sticker can be used to represent a blocked item.

Although this approach is hardly the best, there is nonetheless a common impulse to bunch blocked items on their own particular part of the board, and since this area is not part of the actual workflow, a tendency then develops for people to grow numb to this little corner, which then rarely gets any attention—that is until someone becomes upset and arrives on the scene demanding to know why the work was not finished several months ago.

KEEPING METRICS IN CHECK

Do not go overboard on metrics. Just as there is a limit to the amount of WIP that people can handle, there's also a limit to how much information they can process before it becomes overwhelming. When people are drowning in too much information, they start to care less.

Four diagrams of metric information is a pretty good rule of thumb to avoid overload.

Metrics should, however, be clearly posted, which could be a problem in electronic systems because there's a tendency to put the graphs on a separate sheet from the Kanban board itself. If you do use an electronic system, try to arrange the layout so that the metrics are visible on the same screen as the Kanban board.

In a physical system, this is pretty easy. For ready access, the diagrams can be conveniently placed next to the Kanban board. Figure 16 shows the four diagrams from this chapter on top of the Kanban board.

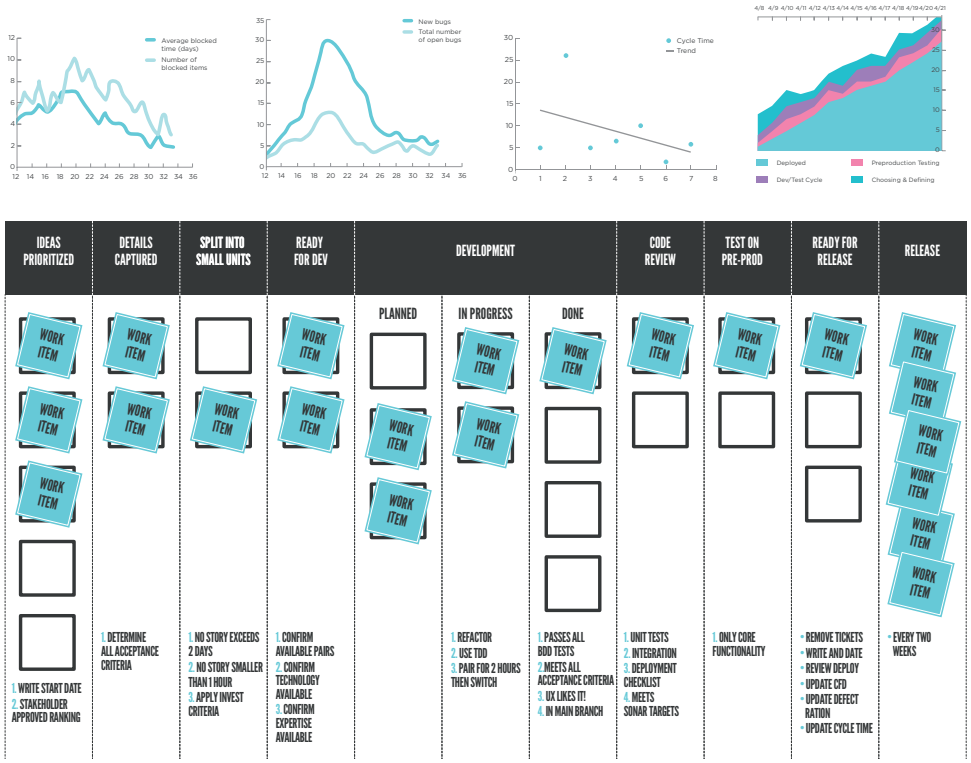


Figure 16 Kanban Board with Diagrams
Diagrams placed above the Kanban board keep metrics clear and accessible

7. PRIORITIZE WORK

The key is not to prioritize what's on your schedule, but to schedule your priorities.
- Stephen Covey

1

THE WHY -

UNDERSTANDING PRIORITIES

It may surprise some to note that all our previous chapters focused on understanding and defining your software delivery system but not actually "doing." This plan was intentional because you first need to have your system in place before you can start using it. Your highest priority should be to ensure that your software delivery system is working.

Your delivery problems should be fully addressed before you even start thinking about prioritizing other issues. The exception being ***Greenfield projects,*** *where the system is just getting started. In that case, you may want to consider prioritization even before your system is set up.*

There is no reason to stop your current method of work prioritization, so keep doing that, but be sure to also consider using the strategies outlined below when you are ready to make use of a more Lean approach to prioritization.

COD equals Cost of Delay

The crucial principle at the heart of prioritizing work is Cost of Delay (COD). In its simplest terms, COD means:

What is the cost of choosing not to work on a given item?
Calculating an exact COD is, unfortunately, very difficult in real-world product development for the following reasons:

- COD attempts to economically quantify lost opportunities.
- COD is often weighted by Cost of Implementation (COI), deadlines, time, and other factors.
- COD involves both lost revenue and cost savings lost.
- Every time you choose to work on something you are choosing to block something else.

THE WHY -

UNDERSTANDING PRIORITIES

Bearing all this in mind, it is clear that the current best guess scenario is our only option. But this approach is better than nothing, and, as is the case with everything, the more you do it, the better you will get. So, despite the imprecise nature of COD, the net result is actually very simple:

Your highest priority should be the item with the highest Cost of Delay.

A high COD means that you're losing the most money on this work because it's sitting around, so get it moving into production!

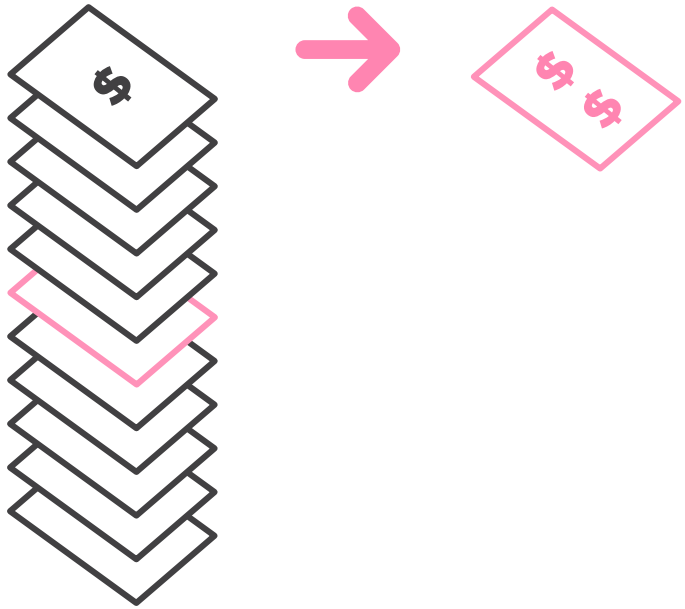
Other Prioritization Factors

Although COD is important, it is not the only factor to consider. When finally ranking the priority of items going into the input queue, the decision-makers want to be sure they are not inadvertently creating a sequence of work that will inhibit flow and create bottlenecks due to poor prioritization decisions. The following points are useful to consider:

- Risk and uncertainty. If the decision is high risk or high impact, you'll want to get as much information as you can as early as you can.
- Bare necessities: Project infrastructure, etc.
- Balance size: Mix size to keep a steady flow.
- Balance story types: Mix functional/ nonfunctional stories to ensure a steady flow of value.
- Dependencies: Handle dependencies proactively so that work does not stall.

Key Prioritization Principles

- Reliability and quality of deliveries are the first priorities.
- Cost of Delay (COD) is the cost of not working on a given item.
- High COD items should be higher priority.
- Consider the impact of the queue order on flow.



THE HOW -

VISUALIZING PRIORITY

Again, Kanban ties prioritization into a simple, explicit visual cue.

The input queue is always organized by the priorities set, so all the team has to do is pull work in from the top of the "Inbox" and they're ready to go.

No matter what system you are using, this method works. If you are working with Scrum, planning a consignment of work for the next sprint, or with a flow-based system, you will always know exactly where to pull the highest-priority work. See Figure 17 for an example.

If you are working with a 50+ item backlog, then the question will naturally arise as to how much to visualize on the board. Some teams find visualizing the complete input queue useful, while others tend to separate the list while progressively pulling the five or so most important items on the board.

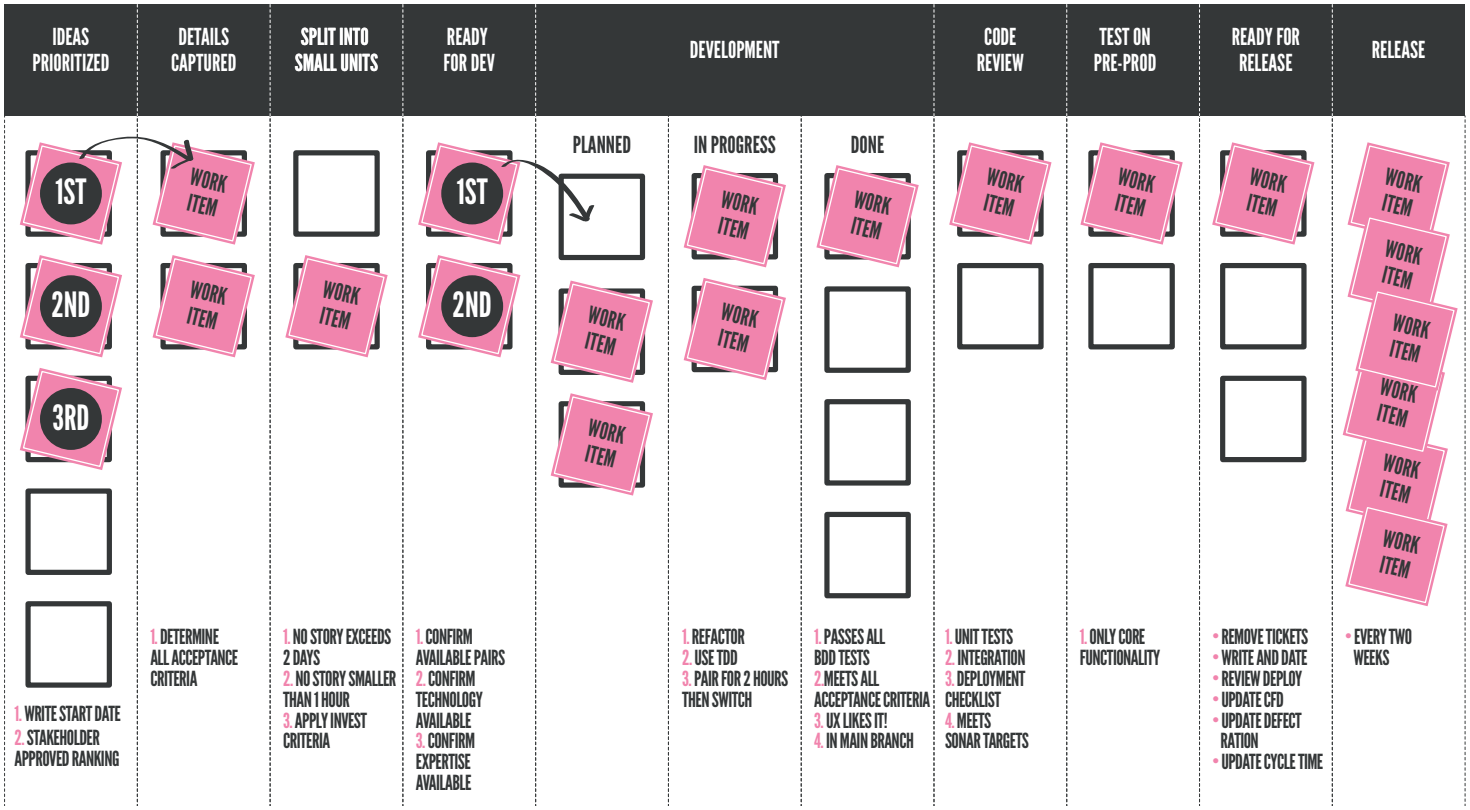


Figure 17 Kanban Priorities
Work is pulled into the next stage from the top of the queue.

It is, however, always advisable to retain an explicit WIP limit on the input queue. In this way, you can keep control. We like the comparison of a backlog to an unused attic. If you stack the attic full of everything you are reluctant to throw out, when the time comes that you actually need something up there, it will be very difficult to find it. It is important to keep the backlog clean and make sure that it is not growing uncontrollably.

8. DEFINE AND IMPLEMENT CLASSES OF SERVICE

"To raise someone's expectations then not fulfill them is worse than mediocrity."
- Seth Godin

THE WHY -

UNDERSTANDING CLASSES OF SERVICE

In our review of Kanban so far, we've treated all work as having the same general level of importance. In reality, not all work is created equal.

10,000

USERS

A case where 10,000 users are unable to access your new software product or service costs $100,000 in revenue per hour is much more important than working on a feature under development.

100,000

REVENUE PER HOUR

Not every difference is quite so clear-cut, so it is necessary to find a reasonable way of processing different types of work.

Kanban systems organize these types of work into various "Classes of Service" based on their detailed characteristics.

Among other benefits, this approach allows us to introduce into the Kanban system a way to expedite the most deadline-sensitive work over less-urgent tasks.

Key Class of Service Points

- It is necessary for some work to take precedence over other work.
- Kanban assigns work Classes of Service based on the specific characteristics.
- Crucial or deadline-sensitive work can be expedited due to its Class of Service.

gearstream

THE HOW -

DEFINE CLASSES OF SERVICE

In software development, some commonly applied methods of categorizing types of work are already in place, including the following:

- User Stories (Small, Medium, Large)
- Bugs (Cosmetic, Critical, Blocker)
- Manual Reports
- Textual Edits
- Support Tasks

Once your different work types have been defined, the next step to consider is how you will deal with these within your system. A Class of Service is simply the definition of how each work type will be handled. The simplest way to illustrate this is by example. Four classes of service have been defined in the following list:

Standard Class

- Extra cost: $0
- Work types: Cosmetic Bugs, User stories
- Special treatment: None

Priority Class

- Extra cost: $500
- Work Types: Critical bugs, High-priority user stories
- Special treatment: Takes priority at each stage

Fixed Deadline Class

- Extra cost: $0–$2,000
- Work Types: User Stories
- Special treatment: Takes priority at each stage if a deadline is deemed unsafe;
- otherwise treated as a standard class; emergency deploy if necessary

Expedite Class

- Extra cost: $3,000–$5,000
- Work Types: Blocker Bug
- Special treatment: Break WIP limits, stop existing WIP, emergency deploy

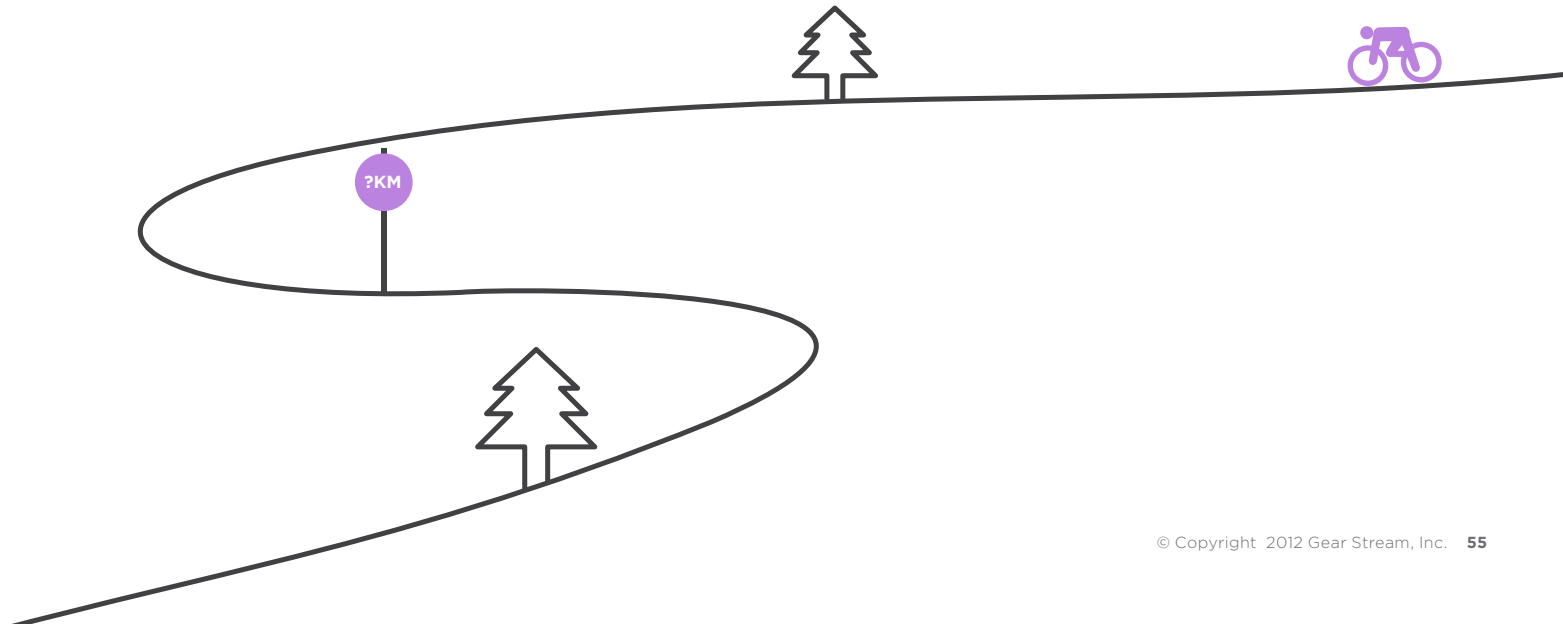
The term "special treatment" describes how this class differs from a standard work item when introduced into our software delivery system. There is always a cost associated with giving things special treatment. An expedite request will cause task switching, longer cycle times for remaining work, plus extra work if an extra deployment is needed. Though it will only be a rough guess, all classes except the standard class should have an associated extra cost.

This consequence naturally encourages an evaluation of nonstandard classes and whether or not they justify the extra trouble. To help ensure they are seldom used, or at least as seldom as possible, an emphasis on flow must be maintained. Flow measurements, in turn, allow for more informed guesswork regarding the cost of special treatment.

You can see the effect "Expedites" have on the cycle time diagram and how an emergency deployment blocks flow and consumes resources.

A good rule to consider in the matter of setting fixed limits on the number of nonstandard classes:

If everything is an "Expedite," then nothing gets expedited.



THE HOW -

HOW TO VISUALIZE CLASSES OF SERVICE

Color codes (Figure 18) and swim lanes (Figure 19) are two popular ways of displaying classes of service. Some combine the two.

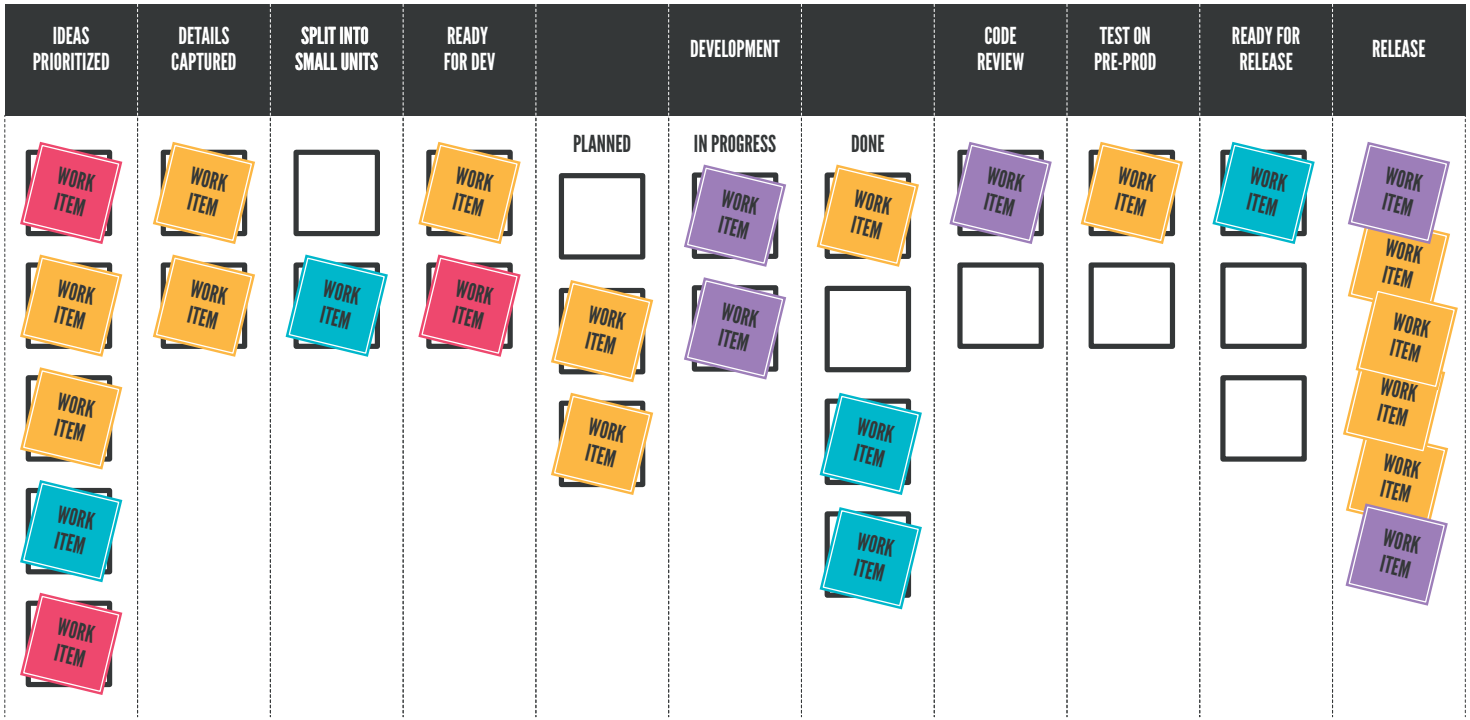


Figure 18 Color Codes
Different classes of service can be distinguished by different color codes.

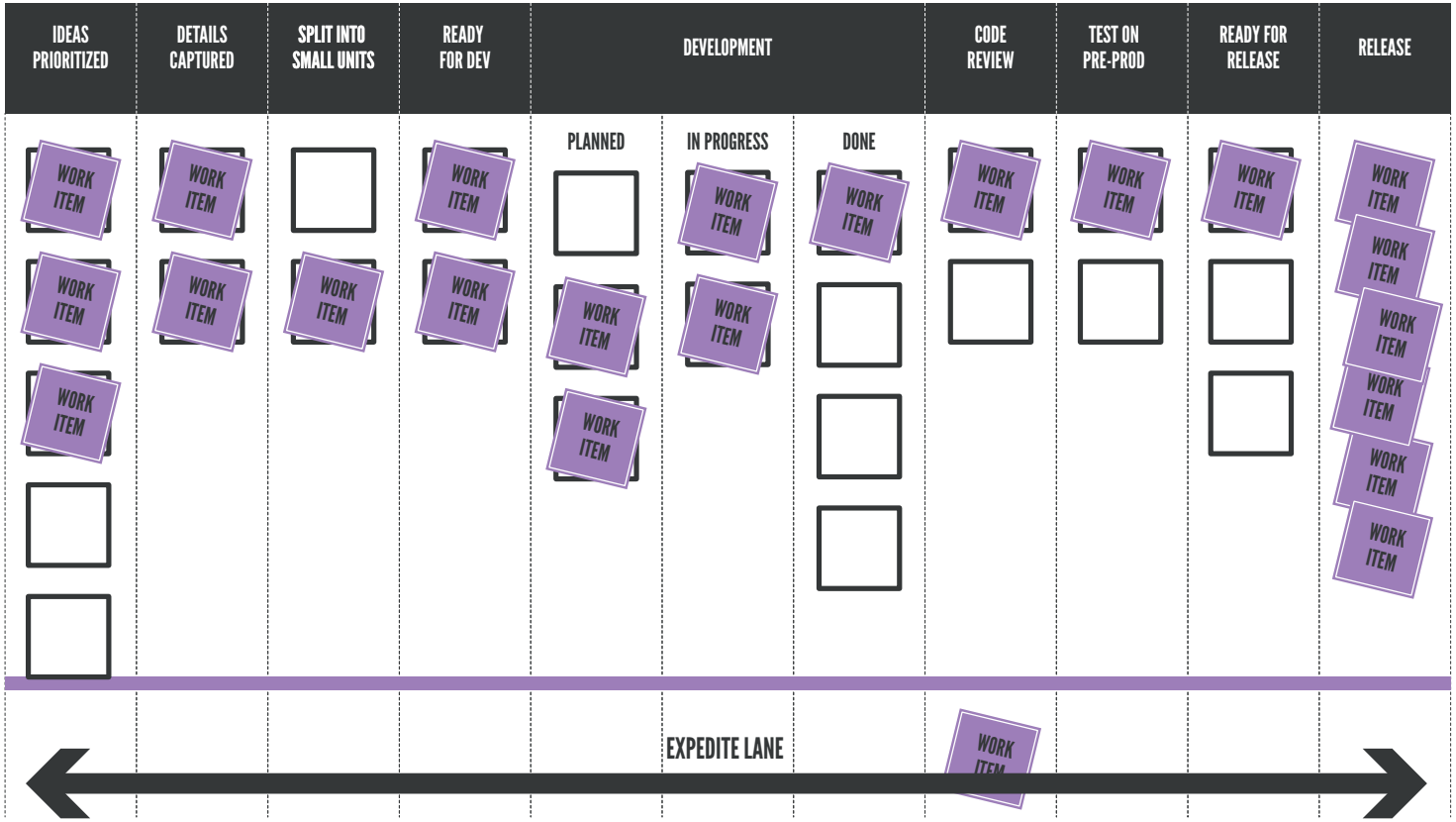


Figure 19 **Swim Lanes**
A swim lane provides a clear visual way to track work that must be expedited.

Classes of Service provide an alternative to panicking and putting out. They offer the opportunity to handle each item in a rational way according to its economic impact.

It also means we can make different promises to our customers depending on the class of service we are handling.

9. ACTIVELY MANAGE FLOW

Experience has taught me how important it is to just keep going, focusing on running fast and relaxed. Eventually it passes and the flow returns. It's part of racing.
- Frank Shorter

THE WHY -

UNDERSTANDING FLOW

Although this point has been frequently stressed in this book, it's worth repeating: flow is extremely important.

If you have been closely following the steps outlined throughout this book, then you will now be ready to take your first steps into a highly mature Agile environment. You have already visualized your entire workflow, limited WIP, established clear QA policies, and started tracking your flow.

You should now learn to read your system and take appropriate action when you see an opportunity for improvement. No concrete rules apply to this step, so the entire team should always be vigilant for opportunities to improve flow.

Remember: Optimize Flow, not Utilization

When thinking of ways to optimize flow, it's easy to think instead about utilization of resources. These two avenues of thought are not automatically equivalent. Strangely enough, there are actually times when underutilizing a resource results in greater overall efficiency.

Though it seems counterintuitive, optimizing flow instead of utilization is close to the very core of Lean.

An example of such an approach can be found in the American motor industry. Car manufacturers used to measure and reward individual productivity according to how many specific parts were produced.

The net result was that the system worked overtime stockpiling, for example, car doors, even though these items were not immediately required.

Specific machinery might have been working at full capacity although overall productivity was not improved—and what is more, inventory control and storage costs increased without a consequent increase in efficiency or profits.

Toyota turned this traditional system upside down by employing the simple matrix of matching the speed of individual part production or importation with the actual pace of vehicle assembly.

This concept is referred to as Takt Time from the German term "taktzeit," meaning literally cycle time. The economic advantages of this system are significant.

THE WHY -

UNDERSTANDING FLOW

Let's consider the units of measurements involved. If the T_a is "minutes of work per day" and T_d is "units required per day," then the Takt time is "minutes of work per unit required." For this reason, Takt time is sometimes called "cycle time." (Cycle time sometimes also refers to "operator time," which is the time it takes an operator to do a single process on a single item.)

Calculating Takt Time

There's a simple formula for calculating the Takt time, ***T:***

where T_a *is the net time available to work and* T_d *is the time demand (customer demand)*

You should identify opportunities to increase the flow of work items through your system by asking the following questions:

- Am I operating within the correct WIP limits?
- Is there a way I can make the size of user stories smaller?
- Is it possible to avoid lengthy blockages by recognizing features that have a tendency to amplify before they are introduced into the system?
- Is it possible to create a more continuous flow by leveling out the size of user stories?
- Is it possible to avoid silos and more easily reduce bottlenecks by training for flexibility?
- Are there sufficient buffers in place to deal with variation?
- Are you considering optimizing the whole instead of individual stages?

Remember that the key is to focus your attention on the flow of the end product rather than fixating on ways to speed up an individual worker or stage.

Always remember to keep your attention focused primarily on the product and not the people producing. Unfortunately, too many managers are still too concerned with urging their workforce to work faster rather than focusing on the quality of the end product!

Key Points to Manage Flow

- Improve processes to increase flow.
- Leverage expertise of entire team to identify improvement opportunities.
- Optimize flow, not utilization.

THE HOW -

TAKING CONTROL OF FLOW

Managing flow effectively is more of an art than a science, and as with any art form, it takes practice to master. The more time you spend developing an understanding of your system the greater will be the opportunities to improve flow.

Here are some sample strategies that we recommend for actively managing and improving flow. These are tactics that experienced practitioners have used in the past and found successful.

Confirm, Prioritize, Act

The Confirm, Prioritize, ACT (CPA) framework is very useful for guiding the actions of an organization. This framework challenges you to consider critical goals and values as you evaluate ways to better manage flow:

Confirm your Current Operating Reality

- Are you making progress with imperfect information?
- Are you encouraging a high-trust culture?
- Are you treating WIP as a liability rather than an asset?

Prioritize The Outcomes That You Need Most

- Value trumps flow
- Flow trumps waste elimination
- Eliminate waste to improve efficiency

You might notice that the first two bullets in the Confirm stage are fairly broad, but the others can be used to make better and more informed judgments when dealing with challenges and difficult decisions. The Prioritize Stage drives home a crucial point: Value is more important than flow.

The overall emphasis on flow may tempt you to trade value for better Cycle Time, but this is going the wrong way. Flow is important because it increases value. If you trade off value for flow, then you are working against yourself.

This is actually a common problem in Agile projects where business value (and sometimes also quality) is often sacrificed to make it fit into frozen time-boxed iterations. The focus shifts to finishing tasks rather than meeting customer demands. Flow, on the other hand, is definitely more important than waste elimination, which is why we made the earlier point about prioritizing flow over utilization.

Once you have managed to optimize for value and flow, you are ready to take a hard look at waste elimination. Always make sure that you primarily watch the product before the people.

With these Agile and Lean Decision Filters in mind, let's take a look at some core concepts for managing flow in our software delivery system.

THE HOW -

TAKING CONTROL OF FLOW

A bottleneck is a location in the system that limits production flow. To fix bottlenecks, it is important to examine the system as a whole and focus effort where it will return the most value.

Fortunately, spotting bottlenecks in a Kanban pull system is fairly easy. If you notice work accumulating at the head of the system and the workflow draining in the downstream processes, then you can reasonably assume that you have a bottleneck. It may quite literally look like a bottleneck on the board, with work flowing out faster than it is flowing in!

A common initial reaction to this predicament is to press more staff into the system, but that is not the most effective response. Humans do not scale in quite the same way as machines do, and the benefits of the added capacity you have introduced are quite often nullified by added coordination overhead and training. Use Brooks Law here as a good rule of thumb:

Adding manpower to a late software project makes it later.

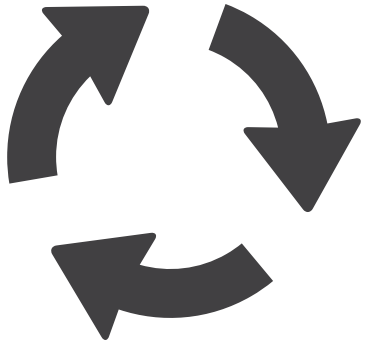
Search instead for opportunities to insulate the bottleneck from unnecessary work.

In one particular project, the product owner team proved to be the bottleneck. A significant amount of their time was spent investigating bugs and contacting users who had not been given the appropriate instructions to work in the system. Members of the development team on a rotational basis would be best suited to handling these tasks, which would have eliminated the bottleneck.

A better long-term solution would have been to establish the root cause of the issue after which it would be a matter of either improving the workflows in the system to make them easier to understand or simply offering users an improved introduction.

Removing nonvalue work is by far the most effective way to relieve a bottleneck.

A third possibility might have been to determine if work items consuming capacity had been blocked by the product owner team. If this had been the case, then the team would have been able to “swarm” on these to rectify the issue as soon as possible.



Introduce Buffers

When you have identified a bottleneck, it is sometimes appropriate to add a buffer stage in front of it. This will help to ensure that the bottleneck does not drain too quickly. For example, if you think there's a bottleneck in the "Development" stage, you could add a "Ready for Development" buffer stage.

It does require some experience to choose the appropriate size for this buffer stage, as we learned in the previous section on limiting your WIP. The buffer can empty periodically, but if this happens every couple of weeks, then you should choose either a larger buffer or determine if this is still a bottleneck at all. If the buffer clears that quickly, it is possible that the real bottleneck actually comes somewhere before Development. The buffer stage can help to act as a good diagnostic.

Planning Releases

Every software development system that we've ever been involved with has had to face the "Project" constraints of budget, time, and scope. It is dangerous and naïve to think only in terms of flow, knowing that steering group committees will expect answers about questions of time, budget, and the delivery of agreed scope.

You will need to do two things to deal with this in a constructive way:

1. Agree that scope will remain flexible.
2. Understand that modifications in scope will be carefully informed by the need to fit time and budget constraints.

Regarding schedule, budget, and scope, scope has to be the most flexible. Significant reorganization, coordination, and communication can regularly result from frequent deadline shifts. Budget increases often mean adding more people, which is seldom a viable tool to reach an approaching deadline. Adding more people is a strategy more appropriate for the longer picture. Even assuming that an increased budget does not translate into more people, it does mean forcing those people that you do have to work longer hours. Overtime in a situation of short deadlines can be a reasonable tool, but it should never become a permanent feature of your process. Overtime is used too frequently to illustrate a project manager's tool of last resort, and perhaps more accurately, to illustrate to his or her superiors that he or she is proactive. It can never be the solution to the overall problem—which the project manager is no doubt conscious of—but it does at least indicate some kind of action.

Once everyone agrees that schedule and budget are not flexible, it becomes important to really grasp what "flexible scope" actually means. This is not simply some laissez-faire approach to software development, shooting each other across cubicles with Nerf guns because you've been given free rein to goof off.

Working with flexible scope requires discipline and the ability to track progress accurately. Flexibility only works if you are making informed decisions in a constantly changing environment. This approach is a key factor in getting the best possible ROI.

THE HOW -

TAKING CONTROL OF FLOW

Let's remember one important fact:

Original complexity, cost, and business value estimates were made at a point when the least amount of information was available.

Nonetheless, deadlines and budgets were factored into these early forecasts, so it is extremely important to track progress to ensure that the project remains feasible.

Status can be tracked in a number of ways, but bearing in mind that we already have our CFD, or Cumulative Flow Diagram, in place, it makes sense to use it for this purpose too. Since nearly all project budgets are done by release, let's examine an example of that. With a budget, deadline, and initial scope established, it is easy to draw our anticipated velocity on the CFD and track progress according to that.

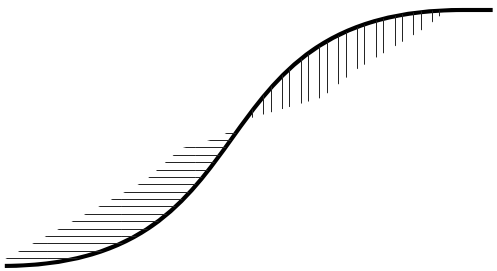
When you look at your production rates on any project, you will notice that they follow a similar pattern for most projects. They begin at a fairly slow rate of production simply because it always takes time to achieve an optimum rate. As the project progresses, however, you begin releasing features more rapidly.

The rate increases. There does come a point, though, when the rate begins to slow again, which makes absolute sense if you think about the way the project progresses. Toward the end of the project, the work that is left tends to be the more difficult elements of the project, which take the longest to complete in the first place. In addition, because far more software has been released, there are more bug fixes toward the end of development than at the beginning, which diverts resources.

For this reason, most project releases describe an S-curve. In general, deviations simply indicate that more information is available now than previously. This new information allows you to make more informed decisions (which, if the early estimates were really poor, could be an early decision to nix the project altogether).

Figure 20 illustrates an S-curve above a CFD. The S-curve shown here represents the expected velocity, a prediction made at the start of the project. The actual velocity, however, is represented by the slope of the "In Progress" work (the purple section).

In the case shown here, this graphic makes it clear that the actual velocity in the middle of the project was a lot less than what was expected (even though it started out much better than expected). This is extremely helpful to the team because it indicates a problem. The graph gives you useful information about mismatches between expectations and real outcomes, but further investigation is needed to find the cause of the mismatch. (In this case, the velocity drop was due to quality issues.)



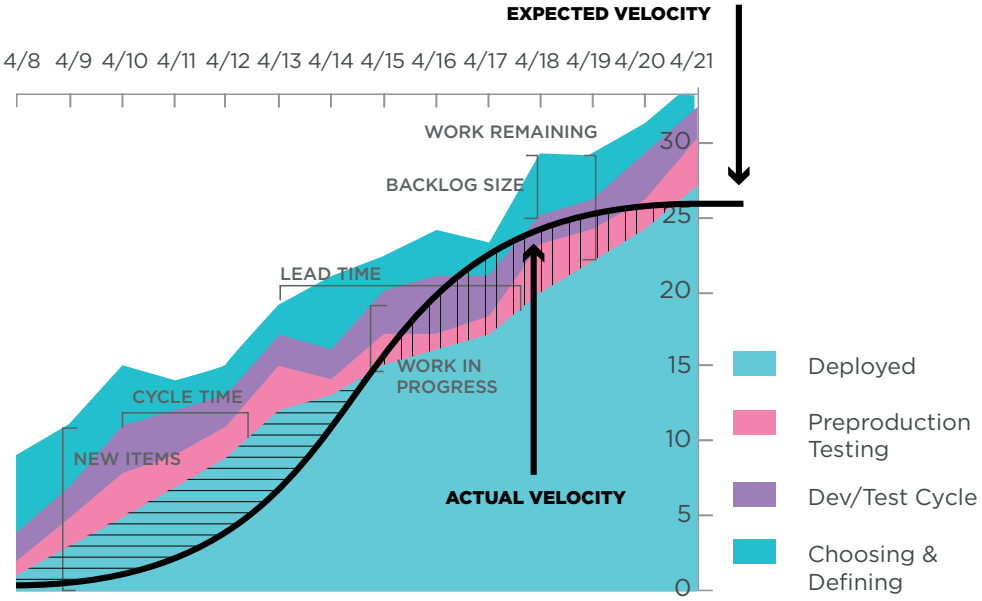


Figure 20 Release Plans Visualized on Cumulative Flow Diagram.
A CFD can allow easy comparisons between expected and actual release rates.

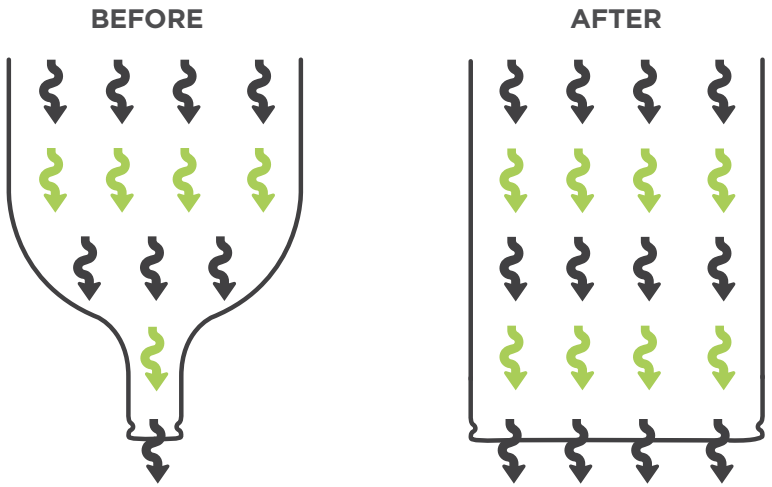


Figure 21 Relieve Bottlenecks to Improve Flow

It is Important to Experiment

Managing flow is predicated on continuous improvement. Many projects have no means of telling whether the things that were changed succeeded or failed. They are essentially randomly guessing at what will work.

Regrettably, most Agile projects are in this category. Experiments are set up using retrospectives, but follow-up, if it occurs at all, usually only includes whether or not it was carried through. Naturally, a high level of uncertainty will always be a factor in measuring software development, but the simple metrics introduced in this book often provide a clear visual indication of whether or not it worked.

If you have ever worked with the Deming circle, you will be familiar with the four steps: Plan, Do, Check, Act. Far too many systems Plan a change and then Do the change, but don't follow up with a Check and implementing any corrective actions as part of the Act step. Without these final steps, the circle is only half-complete, and we are not really able to make informed decisions moving forward.

The expected outcome should be defined before running the experiment.

As an example, let's assume that to do more up-front testing you decide to include testers in your development team. It should not be long before you see a drop in defect rates. If you don't see a change in the defect rate, then you need to modify the plan. It is clear that the experiment did not work as intended, but at least you now have more information about your system. In this case, defects do not appear to be slipping through because of a lack of testing. So you will move on to try something else.

Managing flow is deeply premised on reading your software system so that the best possible decisions in pursuit of the highest ROI can be made with what information is available.

It almost goes without saying that as you gain more information your basis for making improved decisions will also be much improved.

10. IMPLEMENT SERVICE LEVEL AGREEMENTS (SLA)

Unless both sides win, no agreement can be permanent.
- Jimmy Carter

THE WHY -

UNDERSTANDING SLAs

Until now our emphasis has been on the internal aspect of software development. The eventual goal of any software development work is to get it out into the world, to interact with others, and to generate value from the work.

That is the step we are at right now. With a stable pull system established, and using a simple set of metrics to track the system's performance, you are now poised to put into place SLAs that you can truly meet. This will help maintain the system and avoid any regression to fire fighting and chaos just at a point when the Kanban initiative has become familiar.

The more inaccurate your predictions, the more likely you are to establish SLAs that are unrealistic and unattainable. This hurts customer satisfaction and value, plus it has a negative effect on the beleaguered delivery team.

Establishing an SLA requires a prediction about the cost and time involved. Since the software delivery system is now flowing in a predictable way, you will gain predictability in the whole system, including final deliveries.

Traditional Agile approaches like Scrum put high value on predictability in terms of Sprint commitment. There is a subtle difference between these two approaches, but that difference should not be underestimated. The Agile approach is plan driven, while the Kanban approach is flow based.

Key SLA Points to Manage Flow

- The goal of a Kanban software delivery system is ultimately to provide a product to customers and other end users.
- A predictable system means predictable deliveries and higher customer satisfaction.
- Kanban's approach to predictability is flow based.

THE HOW -

ESTABLISHING THE RIGHT SLAs

In Kanban, our ability to establish SLAs is made easier by the work we did earlier when we defined the Classes of Service for work flowing thorough our software delivery life cycle. If you treat each of your Classes of Service the same way every time and measure the consequence of your improvement efforts, chances are that cycle time, quality, and cost will only improve over time, which gives you the possibility of sharing this data with your customers.

Our Classes of Service might get an SLA set up in the following way:

Standard Class

- SLA:
  - Mean: 18 days
  - 85 percent within: 21 days
  - All within: 30 days

Fixed Deadline Class

- SLA:
  - 95 percent within deadline

Expedite Class

- SLA:
  - Mean: 3 days
  - 85 percent within: 4 days
  - All within: 5 days

Priority Class

- SLA:
  - Mean: 6 days
  - 85 percent within: 11 days
  - All within: 15 days

The key at this point is that we did not simply arrive at these figures from educated guesswork but rather as a result of tracking performance. This is strictly a data-driven approach. If a demand arises for us to provide even more detailed information, we can easily adjust our metrics accordingly.

We might also want to make the SLAs even more granular. For example, if it turns out that our Standard Class work items differ a great deal in size, we have a couple of options. We could either create more Classes of Service, or we could have multiple SLAs for the Standard Class. This approach has the benefit of showing customers the direct effect based upon our data about the size of the task, as shown below:

Standard Class

- SLA 250–400 story points (Large):
  - Mean: 23 days
  - 85 percent within: 26 days
  - All within: 35 days
- SLA 150–250 story points (Medium):
  - Mean: 15 days
  - 85 percent within: 19 days
  - All within: 30 days
- SLA 10–150 story points (Small):
  - Mean: 12 days
  - 85 percent within: 13 days
  - All within: 20 days

This is extremely valuable information for many customers. It immediately helps them to prioritize, but in fact the benefits can be even more significant.

As they gain experience that these numbers hold true, this promotes a level of trust and collaboration far in excess of what most have experienced in previous projects, which is one of the biggest benefits of running Kanban with SLAs! When customers see products delivered predictably in line with agreed SLAs, it is common to see dramatic improvements to customer loyalty and higher fidelity feedback on products delivered.

To make sure everybody on the team is aware of the current SLAs and Classes of Service, most teams find it useful to post them next to the board as shown in Figure 22.

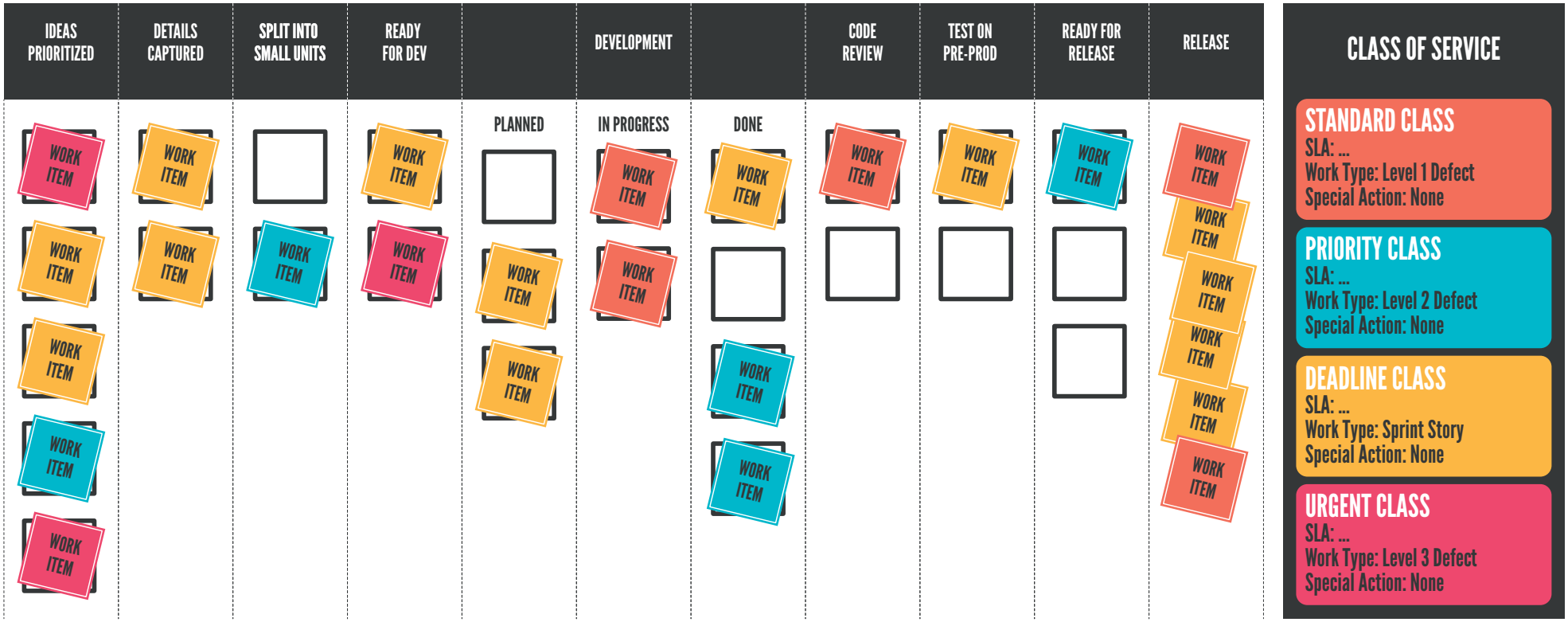


Figure 22 **Classes of Service Policies Posted Next to the Board**

11. FOSTER CONTINUOUS IMPROVEMENT

Be a yardstick of quality. Some people aren't used to an environment where excellence is expected.
- Steve Jobs

CONGRATS -

Congratulations! You have reached the big reward at the "end" of the Kanban treasure hunt:

The ongoing task of sustaining a constant cycle of improvement!

This is probably the hardest and most important element of Kanban.

Remember that

Kanban is primarily a tool for pushing forward evolutionary change.

Having progressed through the previous steps, you will find this process of continuous improvement to be quite a lot easier.

Here is our hope, which has been borne out by many Kanban transformations in the past:

That the experts in your organization will enter into an ongoing dialogue about improvement opportunities far beyond those seen in traditional software projects. These discussions will be informed by continuous visual feedback from the workflow and metrics, guided by knowledge of the explicit policies and SLAs for each Class of Service.

During this process of continuous improvement, your organization will evolve, eventually taking over the world. Or, at least, your organization can live up to its values. Or, if you are lucky, both!

Since we collect real data, Kanban also gives us the opportunity to perform and validate our experiments in a more scientific way than traditional Agile projects. Initiatives to improve cycle time should result in actual measurable effects.

This makes continuous improvement in a Kanban system much more reliable. Since we see and measure the effect of our change initiatives, we are much more likely to keep raising the bar.

For some people, working with Kanban is the first time they start to see the software delivery system as a whole. This offers an enormous insight into other people's work. How do different people depend on each other? How do various tasks connect together? Optimization discussions can expand beyond individual work silos.

IF YOU HAPPEN TO SEE A TESTER, PRODUCT OWNER, AND A DEVELOPER DISCUSSING FLOW IMPROVEMENTS NEXT TO THE KANBAN BOARD, YOU CAN BE CERTAIN THAT YOU ARE WELL UNDER WAY.

These ongoing discussions, or "spontaneous quality circles," are excellent vehicles for continuous improvement, but many Kanban teams still benefit from the use of a regular cadence of retrospectives or kaizen events, giving the team a chance to see their work from a distance. This can lead to suggestions for larger structural changes known as Kaikaku (dramatic change). A combination of ongoing quality circles and a cadence of retrospectives can be a powerful force for driving improvement.

12. THE NEXT MOVE IS YOURS

YOUR MOVE -

If you're looking for help with Kanban and Agile delivery improvements, give us a call! Gear Stream has a broad range of Lean and Agile transformation services designed to radically improve your innovation and software delivery pipeline.

Our services include on-site workshops, embedded coaching, and the world's only On Demand Agile Cloud Software Factory.

We hope this eBook has given you useful insights and inspired you to move forward on your Agile journey and to consider using Kanban on real projects in your company. We would also love to get your feedback and for you to share stories about how it might have helped you achieve better ROI for you and your customers.

Brad Murphy, Founder & CEO
Gear Stream, Inc.

Send Us Your Kanban Questions or Feedback to:
Kanban@gearstream.com
For More Information,

Call or Visit Us At:
1-800-935-1800
www.GearStream.com

gearstream

