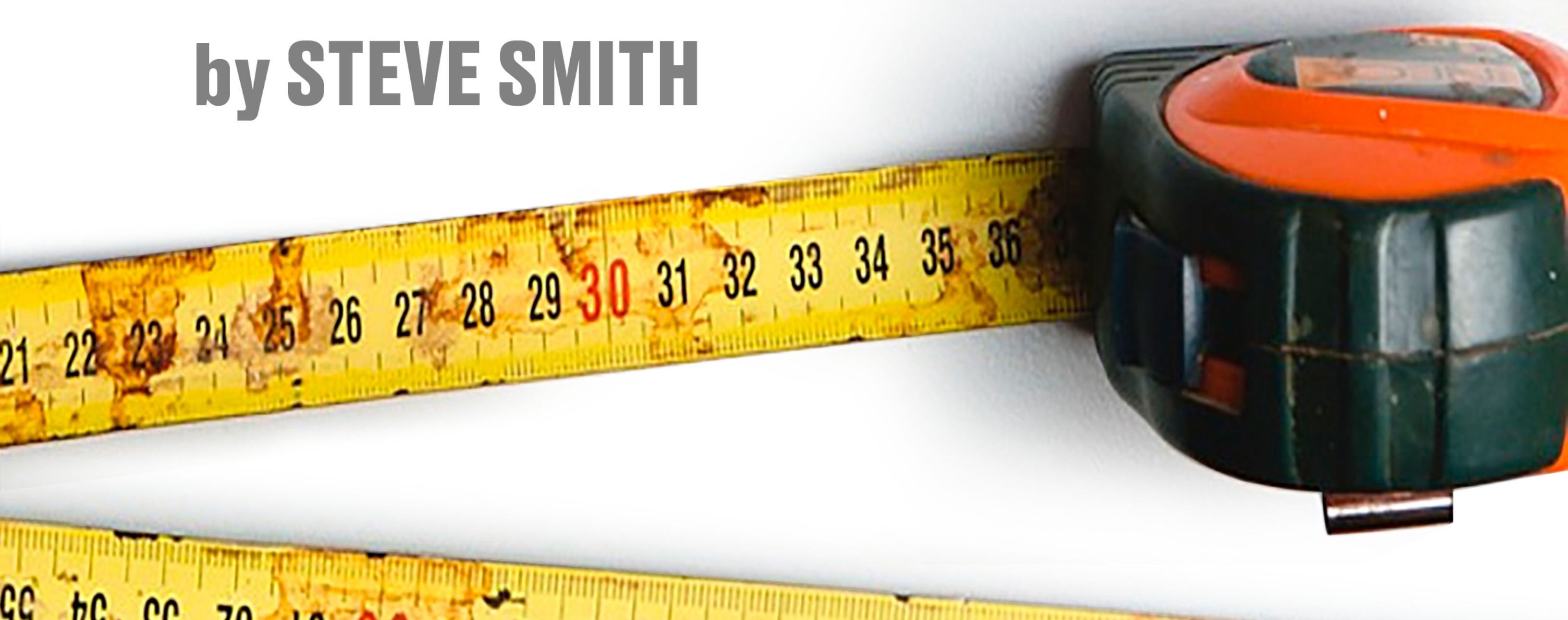
The **what, why,** and **how** of measuring Continuous Delivery

MEASURING CONTINUOUS DELIVERY

by STEVE SMITH



Measuring Continuous Delivery

The what, why, and how of measuring Continuous Delivery

Steve Smith

This book is for sale at http://leanpub.com/measuringcontinuousdelivery

This version was published on 2020-10-31

ISBN 978-1-5272-7743-4

Tweet This Book!

Please help Steve Smith by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just bought Measuring Continuous Delivery by @SteveSmithCD!
https://leanpub.com/measuringcontinuousdelivery

The suggested hashtag for this book is #MeasuringContinuousDelivery.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#MeasuringContinuousDelivery

To Rupa, Amara, and Serena

Contents

Preface . . . . **i**
Who should read this book i
In this book i
Audiobook ii
How to contact me ii
Acknowledgements iii

About Steve Smith . . . . **iv**

1. Adopting Continuous Delivery **1**
Introduction 1
Know Discontinuous Delivery 2
Understand Optimising For Robustness 3
Learn Continuous Delivery 8
Use the Improvement Kata 10
Measure Continuous Delivery 15

Preface

How do you measure IT performance? How do you decide which Continuous Delivery practices to try in your organisation? How do you know if your adoption of Continuous Delivery is headed towards success?

Continuous Delivery is a set of holistic principles and practices to reduce time to market and provide an organisation with a strategic competitive advantage, but adoption is invariably a challenging and time-consuming journey. Before adoption, the current time to market and desired time to market are often unknown, which makes alignment and collaboration between individuals, teams, and departments difficult. During adoption practices, techniques, and tools are often introduced without acceptance criteria, with makes it hard to assess and learn from the impact of changes.

In this book, I will show you how to measure your Continuous Delivery journey, so you can understand how fast you are travelling, and how fast you need to go.

Who should read this book

I wrote this book for executives, managers, leaders, and practitioners who wish to adopt Continuous Delivery. I want to help people who are stuck on one or more of the following questions:

- What would Continuous Delivery look like for my organisation?
- Where should I start to try and make improvements?
- How do I know if an improvement has had a positive impact?

Over the past 10 years I've worked on Continuous Delivery in financial, media, governmental, and retail organisations. I've used delivery metrics to steer large scale Continuous Delivery programmes, including 10 teams at a media organisation for 3 years and 60 teams in a UK government department for 2.5 years. I will show how automating a collection of holistic metrics can provide powerful insights into Continuous Delivery.

In this book

This book describes the what, why, and how of measuring Continuous Delivery.

It contains a high-level discussion of Continuous Delivery principles, practices, and techniques. It is assumed the reader has previously read *Continuous Delivery*[1] by Dave Farley and Jez Humble, and

[1] *Continuous Delivery* by Dave Farley and Jez Humble (2010)

is at least aware of other Continuous Delivery literature. There is little advice on specific tools, as there are many good tools available in the Continuous Delivery space and there is never a single, perfect tool.

Chapter 1 introduces the main theme of the book — how to facilitate a successful adoption of Continuous Delivery, by combining the Improvement Kata with stability and throughput measurements. Chapter 2 outlines the theory behind measures, metrics, and indicators of stability and throughput. Chapters 3 to 7 are a deep dive into deployment stability, deployment throughput, build stability, build throughput, and mainline throughput indicators. Chapter 8 shows how to use those indicators to power the Improvement Kata, and implement Continuous Delivery.

Throughout this book, the following sidebars are used:

A ToyVille episode — part of a book long story of a fictional organisation trying to adopt Continuous Delivery

A tip — something to try when you are adopting Continuous Delivery in your own organisation

A client experience — a brief, real world experience from my work with different clients

Audiobook

An audiobook narrated by James Cooper is available on Audible[2]. A workbook of all the book charts and images is available in the Audible customer library after purchase. Alternatively, it can be downloaded for free from my website books page[3].

How to contact me

I love feedback. Please let me know what you think of this book on Twitter at @SteveSmith_Tech[4].

[2]www.audible.co.uk
[3]https://www.stevesmith.tech/books
[4]https://twitter.com/SteveSmith_Tech

Acknowledgements

No person is an island. I want to thank:

- Dave Farley and Jez Humble for their ongoing leadership in the Continuous Delivery community, and writing *Continuous Delivery* in the first place
- Phil Parker, Thierry de Pauw, and Wouter Lagerweij for their book reviews
- Emily Bache, Michiel Rook, and Troy Magennis for their reader feedback
- Dave Farley, Nik Zarza, and Alun Coppack for their support for Continuous Delivery at LMAX, Sky Network Services, and Equal Experts respectively over the past 10 years
- Dave Farley, Rob Sloss, Charles Kubicek, the PIPELINE team, and many others in the Continuous Delivery community for the sharing of ideas

Finally, I want to thank my wife Rupa, and my daughters Amara and Serena.

About Steve Smith

Steve Smith is an IT consultant specialising in Digital Transformation, Continuous Delivery, and Operability. Since 2008, Steve has worked on transformation transformation programmes, in leadership and hands-on roles for public and private sector organisations.

Those roles include creating a deployment pipeline for 8 teams/70 microliths in a media company with £14Bpa revenue, building an operability toolchain for 60 teams/600 microservices in a UK government department with £500Bpa revenue, and creating a Continuous Delivery & Operability strategy for 30 teams/100 microservices in a high street retailer with £2Bpa website revenue.

Steve is a Principal Consultant and Operability Practice Lead at Equal Experts[5], a worldwide network of technology consultants that specialises in solving complex challenges for enterprise organisations.

Steve is the author of *Measuring Continuous Delivery*[6], plus a co-author of *A Children's A to Z of Continuous Delivery*[7] and *Build Quality In*[8]. Steve blogs at stevesmith.tech[9], chats on Twitter at @SteveSmith_Tech[10], and is a regular conference speaker.

[5]https://www.equalexperts.com
[6]https://www.leanpub.com/measuringcontinuousdelivery
[7]https//www.leanpub.com/achildrensatozofcontinuousdelivery
[8]https//www.leanpub.com/buildqualityin
[9]https://www.stevesmith.tech
[10]https://www.twitter.com/SteveSmith_Tech

1. Adopting Continuous Delivery

Introduction

In 2003, Nicholas Carr proclaimed "*IT doesn't matter*"[11] in the Harvard Business Review. Carr argued that as a ubiquitous commodity IT was merely an infrastructural technology, as opposed to a proprietary technology of strategic value. He concluded that IT was simply a cost of doing business, and could not provide an organisation with a sustained competitive advantage.

Carr's advice dovetails with the traditional approach to IT, which is best described as IT as a Cost Centre[12]. In this paradigm, IT is deemed incapable of revenue creation. An organisation will have a standalone IT department, and it will be designated as a cost centre with an annual budget and plan. The annual plan will consist of static programmes and projects, each with its scope, resources, and deadlines agreed in advance. Overall, there will be a constant emphasis on maximising resource utilisation by keeping people busy, and on minimising costs by means such as outsourcing work to third-party suppliers.

Carr was only partially correct in 2003, and since then history has not been kind. IT is an infrastructural technology when an organisation uses it for utility business operations, such as employee payroll. However, IT services can also create proprietary business differentiators, such as on-demand video streaming. These IT services can set an organisation apart from its competition, and become revenue generators of strategic importance[13].

Carr failed to predict the rise of Agile development[14], Lean product development[15], and in particular Cloud computing[16], which has commoditised many lower-order technology functions. These advancements have been significant contributors to the modern approach to IT, which can be called IT as a Business Differentiator[17]. In this paradigm, IT is considered a direct or indirect revenue creator. An organisation will have an IT department integrated into its business verticals, with a rolling budget and plan. The rolling plan will consist of dynamic product areas with scope, resources, and deadlines constantly refined by feedback. Overall, there will be a focus on optimising the flow of work by eliminating handovers with cross-functional teams, reducing inventory with work in progress limits, and removing wasteful activities.

[11] *IT Doesn't Matter* by Nicholas Carr (2003)
[12] *IT as a Cost Centre is when an organisation has a 'functionally segregated IT department, that is accounted for as a cost centre'.
[13] *Utility vs. Strategic Dichotomy* by Martin Fowler describes how utility and strategic IT differ by underlying business function.
[14] Agile development is "an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organising cross-functional teams and their customer(s)/end users(s)"
[15] Lean product development is 'a Lean approach to meet the challenges of product development'.
[16] Cloud Computing is 'an IT paradigm that enables ubiquitous access to shared pools of configurable system resources and higher-level services that can be rapidly provisioned'.
[17] *IT as a Business Differentiator is when 'IT staff work in one or more product departments, which are accounted for as profit centres'.

The disruptive impact of IT as a Business Differentiator on global commerce was summarised by Marc Andreessen in 2011, when he declared "*software is eating the world*"[18] in the Wall Street Journal. Andreessen asserted the world is in a profound economic and technological shift due to a software revolution. In 2013, this was correlated by the Deloitte Shift Index[19] reporting a decline in average life expectancy for Fortune 500 organisations from 75 to 15 years.

The long-term survival of any organisation is now dependent on its ability to use technology to discover business differentiators and increase revenues. Every week of delay between having an idea and launching it to customers can mean millions of dollars lost in opportunity costs. IT matters.

Know Discontinuous Delivery

An organisation will have one or more value streams. A value stream is a sequence of activities that converts business ideas into value-adding changes for customers. An activity will consist of one or more tasks, which could be manual, semi-automated, or automated. The first part of a value stream will be design and development activities, which are inherently non-deterministic and highly variable. The second part will be build, testing, and operational activities, which should be as deterministic and invariable as possible. The latter part is known as the technology value stream.

When an organisation lacks the necessary stability and speed in its technology value streams to satisfy customer demand, it is in a state of Discontinuous Delivery[20]. Discontinuous Delivery is usually caused by one or more of the following:

- *Lack of urgency*. An organisation might be complacent about its competitors and market position, or unwilling to jeopardise its market position with the launch of a new product[21].
- *Detrimental culture*. An organisation might have a power-oriented culture that discourages cooperation between teams, or a rule-oriented culture that promotes modest cooperation with narrow responsibilities[22].
- *Asinine use of projects*. An organisation might be beholden to an annual budgeting and planning process, with fixed amounts of time and money allocated to deliver large batches of features to long-term, unrealistic deadlines[23].
- *Unsuitable team structures*. An organisation might be arranged by functional orientation, with work split between siloed, specialised teams merely responsible for pre-planned tasks devoid of context or purpose[24].

The perennial challenge is how to achieve a speed of delivery that overcomes Discontinuous Delivery, while simultaneously achieving sufficient reliability to protect daily business operations.

[18] *Why Software is Eating the World* by Marc Andreessen (2011).
[19] *Deloitte Shift Index* by John Hagel and John Seely Brown (2013).
[20] Discontinuous Delivery is 'when an organisation has a value stream that lacks the stability and speed required to satisfy business demand'.
[21] *The Innovator's Dilemma* by Clayton Christiensen (1997) hypothesises that organisations are often reluctant to invest in new product development if it threatens the market for an existing product.
[22] *A Typology of Organisational Cultures* by Ron Westrum (2004) categorises culture as power-, rule-, or performance-oriented.
[23] *The Principles of Product Development Flow* by Don Reinertsen (2009) explains why projects are so unsuitable for product development.
[24] *Extreme Programming Explained* by Kent Beck *et al* (2004) explains why specialised functional teams are so inappropriate for software.

Unreliable IT services leave business operations vulnerable to IT outages, and in unfavourable market conditions the cost of downtime can be ruinous. However, all too often organisations use a reliability strategy that is itself a cause of Discontinuous Delivery.

Understand Optimising For Robustness

When an organisation has IT as a Cost Centre, its reliability strategy will be to Optimise For Robustness[25]. It will try to maximise the robustness of its IT services, by implicitly prioritising a higher Mean Time Between Failures (MTBF)[26] over a lower Mean Time To Repair (MTTR)[27]. The production environment will be thought of as a failure-free place, in which services are homogeneous processes with predictable interactions in repeatable conditions. Failures will be believed to be caused by isolated, faulty changes and therefore considered preventable. This is all based on the belief that a production environment is a complicated system[28], in which expert knowledge can predict the cause and effect of events.

Usually, Optimising For Robustness involves a reliance on the following risk management activities in technology value streams:

- *End-To-End Testing*[29] to verify the functionality of a new service version against its unowned dependent services.
- *Change Advisory Boards (CABs)*[30] to assess, prioritise, and approve the deployment of new service versions into production.
- *Change Freezes*[31] to restrict the deployment of new service versions for a period of time due to market conditions.

Other practices such as mandatory code reviews, manual regression testing, segregation of duties, and uptime incentives might also be used.

For example, consider the fictional organisation ToyVille.

[25]Optimise For Robustness is ‘maximising the ability of IT services to resist change without adapting their initial stable configuration’.

[26]Mean Time Between Failures is ‘the predicted elapsed time between inherent failures of a mechanical system’.

[27]Mean Time To Repair is ‘the average time required to repair a failed component or device’.

[28]A Complicated System is the Cynefin domain of known unknowns, in which experts can rationally make decisions.

[29]End-To-End Testing is ‘testing conducted on a complete, integrated system to evaluate the system’s compliance with its specified requirements’.

[30]A Change Advisory Board ‘delivers support to a change management team by approving requested changes and assisting in the assessment and prioritization of changes’.

[31]A Change Freeze is ‘a point in time in the development process after which the rules for making changes to the source code or related resources become more strict’.

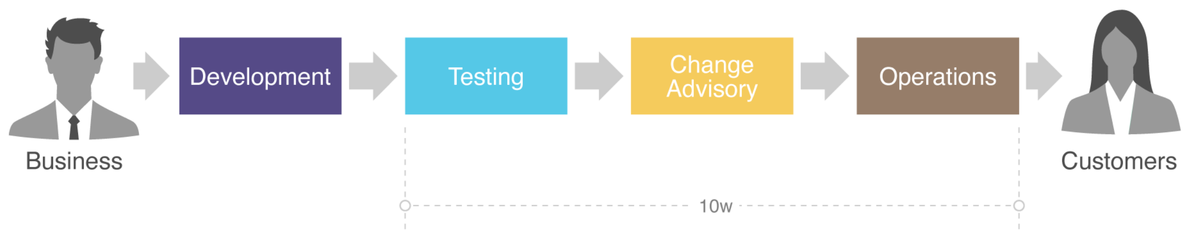
ToyVille: a troubled toy store

ToyVille is a toy store that sells its products in-store and online. Historically, ToyVille executives and managers have designated IT as a cost centre, and thought of website reliability as zero production failures.

ToyVille has sluggish online sales growth of 5% per annum, while its competitors are enjoying double digit growth. Limited customer uptake for new website features is blamed on poor product/market fit and a slow, error-prone release process.

ToyVille has multiple development teams working in two-week iterations. There is a production launch every three months, after ten weeks of preparation. Each release goes through End-To-End Testing with the testing team, a Change Advisory Board with the change approval team, and operational configuration with the operations team prior to a production launch. Furthermore, each year there are four-week Change Freezes in July and December for summer clearance and Christmas respectively.

TOYVILLE — LEAD TIME



Recognise the costs and theatre

Robustness is a desirable capability of an IT service, but to Optimise For Robustness is to invariably spend too much time for too little risk reduction. The risk management activities used will be far more costly and less valuable than expected. End-To-End Testing incurs long test execution times and significant maintenance time, and test coverage will be low[32]. CABs involve extensive documentation and slow approval times, and deployments can still fail[33]. Change Freezes cause significant opportunity costs, and production failures can still occur[34].

In addition, these activities are risk management theatre[35]. They are based on the misguided assumption that preventative controls on everyone will prevent anyone from making a mistake. Ironically, they actually increase risk by ensuring a large batch size and a sizeable amount of

[32]*End-To-End Testing Considered Harmful* by the author explains the attractiveness and fundamental flaws of End-To-End Testing.
[33]*Continuous Delivery and ITIL Change Management* by Jez Humble details the inherent problems with CABs.
[34]*Code Freezes Don't Prevent Outages* by Baron Schwarz outlines the folly of trying to prevent production changes for a set time period.
[35]Risk management theatre is 'commonly-encountered control apparatus, imposed in a top-down way, which makes life painful for the innocent but can be circumvented by the guilty'.

requirements/technology changes per service version. They impede knowledge sharing, restrict situational awareness, and create enormous opportunity costs.

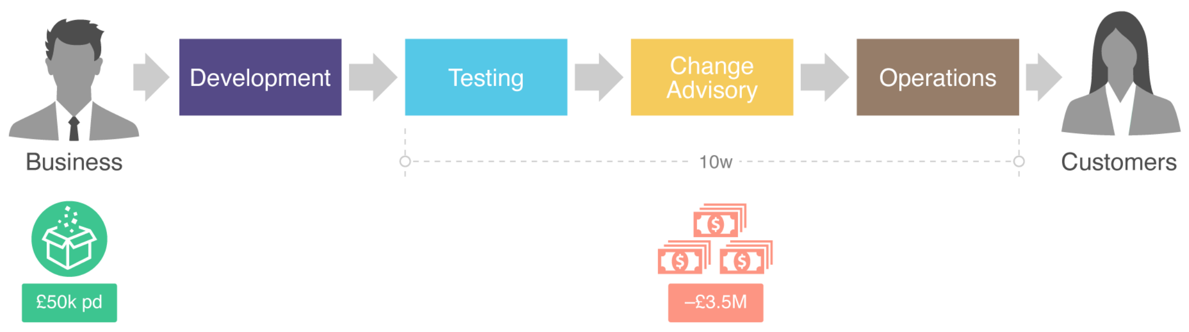
Optimising For Robustness constrains the stability and throughput of IT delivery such that business demand cannot be satisfied. It is a significant contributor to Discontinuous Delivery.

ToyVille: the big launch opportunity cost

In a bid to improve online sales, ToyVille development teams work on a project to simplify the checkout user journey. The value of the project is estimated as £50K per day in new revenue. While the project is in progress, the executive team realise their ten-week lead time to production will create an estimated opportunity cost of £3.5 million. In addition, if a Change Freeze is encountered that cost would rise to £4.9 million.

IT managers reassure the executive team the End-To-End Testing will eliminate as many defects as possible, and will not overrun.

TOYVILLE — OPPORTUNITY COST



Accept the constancy of failure

Optimising For Robustness will inevitably lead to an over-investment in pre-production risk management, and a corresponding underinvestment in delivery and operability. Symptoms of this underinvestment include:

- *Stagnant requirements*. Operational requirements are deprioritised indefinitely.
- *Manual deployments*. New service versions are released in a multi-step process performed by multiple individuals and/or teams.
- *Configuration drift*. Configuration changes for service instances are inconsistent.
- *Snowflake infrastructure*. Environments are manually created and maintained in an unreproducible state.

- *Inadequate telemetry*[36]. Logs and metrics are scarce, anomaly detection and alerting are manual, and user analytics lack insights.
- *Fragile architecture*. Services are coupled, service instances are stateful, failures are uncontained, and load vulnerabilities exist.
- *Insufficient training*. Operators are not given the necessary coaching, education, or guidance.

This underinvestment results in unstable deployments and an inoperable production environment, which makes it difficult for operators to update and maintain IT services in a safe and reliable condition. This will be deemed acceptable, as production failures are expected to be rare.

Unfortunately, a production environment of running IT services is not a complicated system. It is an intractable mass of heterogeneous processes, with unpredictable interactions occurring in unrepeatable conditions. It is a complex system[37] of emergent behaviours, in which the cause and effect of an event can only be perceived in retrospect. Furthermore, as Richard Cook explains in *How Complex Systems Fail*[38] "the complexity of these systems makes it impossible for them to run without multiple flaws being present". A production environment is always changing, always in possession of multiple faults, and always in a state of near-failure.

A failure occurs when multiple faults unexpectedly coalesce such that one or more business operations cannot succeed. It will create a revenue cost expressed as a function of cost per unit time and duration, and the impact can be considerable. The sunk cost incurred until failure detection can be high, as unimplemented operational requirements and inadequate telemetry will restrict situational awareness. The opportunity cost until failure resolution can also be high, as snowflake infrastructure and a fragile architecture will increase failure blast radius. In addition, the loss of customer confidence and increased failure demand will create further opportunity costs.

The assumption that failures are caused by individuals encourages an attitude Sidney Dekker calls the Bad Apple Theory[39], in which the production environment is considered absolutely reliable bar the actions of a few unreliable employees. When a failure occurs, the combination of the Bad Apple Theory and hindsight bias[40] will produce an oppressive culture of naming, blaming, and shaming the individuals involved. This will discourage knowledge sharing and collaboration.

See the countermeasure

The inherent costs of Optimising For Robustness mean features will take weeks or months to move through a technology value stream, but lead times need to be much faster to remediate failures.

When a failure is discovered, people will share a sense of urgency that a sunk cost has been incurred, and that an immediate fix is required to minimise the opportunity cost duration. As a result, Dual Value Streams[41] will be used as a countermeasure to long lead times. An organisation will operate

[36]*The DevOps Handbook* by Patrick Debois *et al* defines telemetry as logging, monitoring, anomaly detection, alerting, and analytics.
[37]A Complex System is the Cynefin domain of unknown unknowns, in which there is constant flux.
[38]*How Complex Systems Fail* by Richard Cook (1998).
[39]*The Field Guide To Understanding Human Error* by Sidney Dekker (2006) defines the Bad Apple Theory as 'you believe your system is basically safe if it were not for those few unreliable people in it'.
[40]Hindsight bias is 'the inclination, after an event has occurred, to see the event as having been predictable'.
[41]Dual Value Streams are 'a feature value stream with a lead time of months, and a fix value stream with a lead time of days'.

a feature value stream and a fix value stream for each of its technology value streams. The feature value stream will retain all the advertised pre-production risk management activities, while the fix value stream will be a truncated variant that sacrifices those activities for speed. This allows production fixes to be released in hours or days, instead of weeks or months.

Dual Value Streams highlight the potential for delivery throughput improvements. They show what is possible within an organisation when people share a sense of urgency, and omit pre-production risk management activities in favour of fast customer feedback.

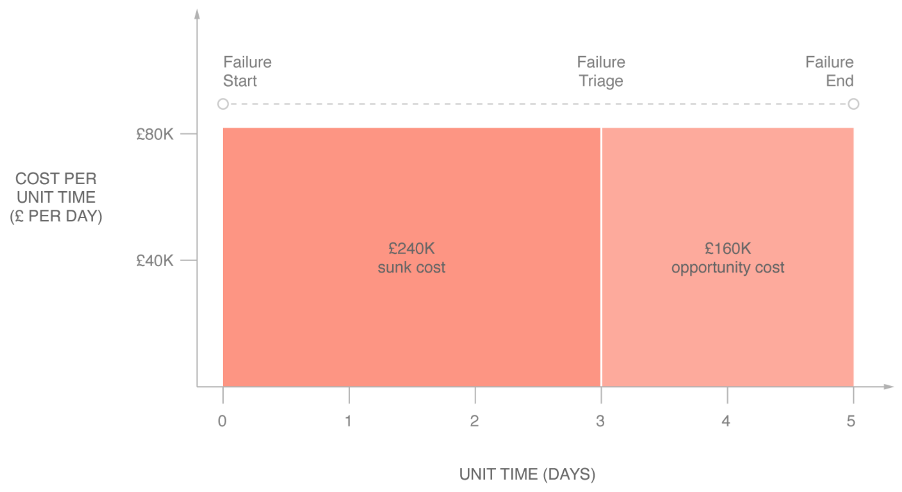
ToyVille: the big registration failure

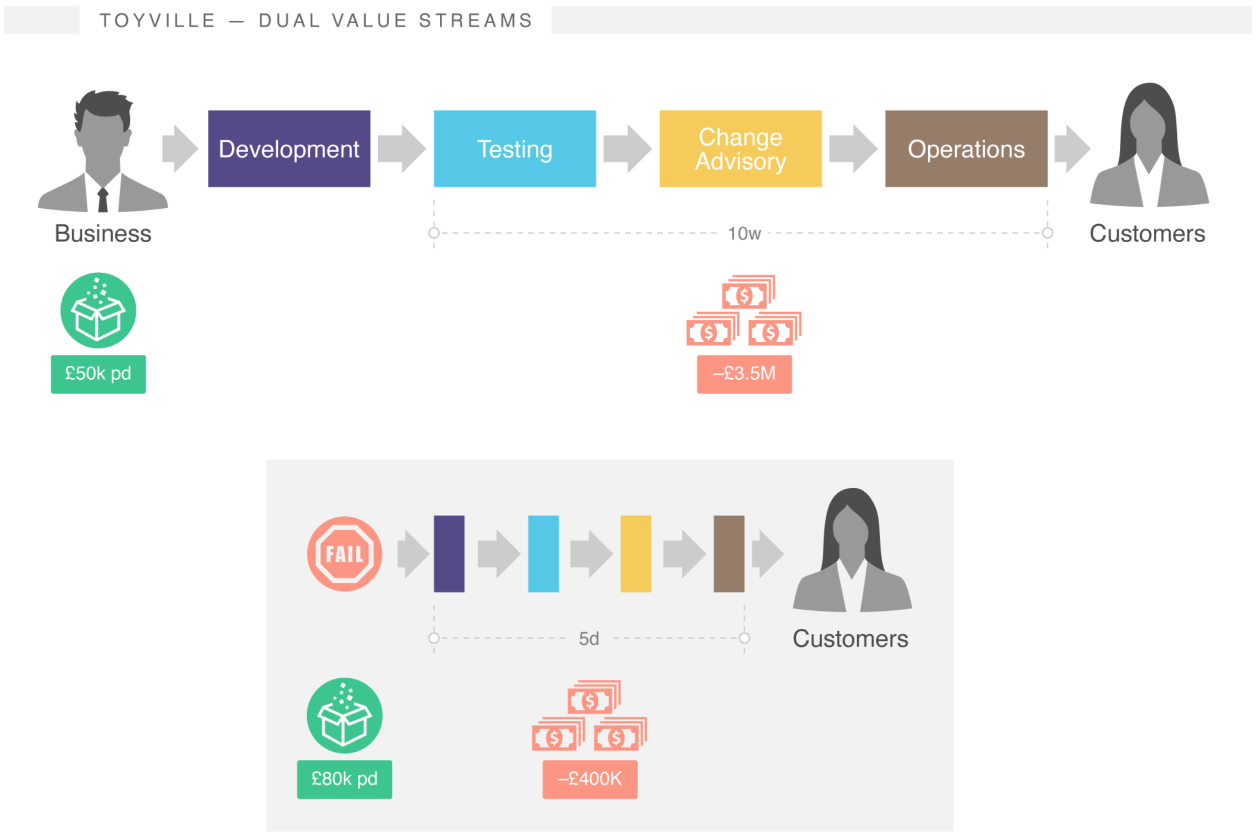
The new ToyVille checkout journey is launched, and there is a significant spike in new customers using the ToyVille website. A third-party registration service begins to suffer under load, and a majority of new customers are rejected on checkout. The failure has a static cost per day of £80K, but with no telemetry the failure is not detected for three days.

After its discovery the failure is assigned to the checkout team, who improve connection handling within a day. It is agreed in a CAB meeting that the fix can skip End-To-End Testing, and it is deployed the following day. The failure has a five-day cost of £400K, with a detection sunk cost of £240K and a remediation opportunity cost of £160K.

The development, testing, and operations teams all condemn the performance of the third-party registration service, and the supplier agrees to implement performance improvements on an existing budget. There is no post-incident review, so few lessons are learned.

TOYVILLE — REGISTRATION FAILURE COST





Learn Continuous Delivery

Continuous Delivery is a set of holistic principles and practices to improve the stability and throughput of IT delivery, first established by Dave Farley and Jez Humble in their seminal book *Continuous Delivery* in 2010. The principles of Continuous Delivery are:

Principle	Description
Repeatable, Reliable Process	Establish an automated, consistent deployment mechanism
Automate Almost Everything	Automate tasks until human decision making is required
Keep Everything In Version Control	Put all code, configuration, infrastructure, etc. in version control
Bring The Pain Forward	Increase the cadence of difficult, time-consuming tasks
Build Quality In	Find and fix defects as early as possible to minimise rework
Done Means Released	Do not mark a feature as complete until it is delivering value
Everybody Is Responsible	Ensure everyone is aligned with delivery goals
Continuous Improvement	Incrementally and iteratively improve the delivery process

Continuous Delivery has an emphasis on automating manual, repetitive tasks in order to free people up to work on higher-value problems. This is accomplished by building a pull-based automated toolchain from code commit to production, known as a deployment pipeline. In a deployment pipeline every code, configuration, data, and infrastructure change must pass a series of automated and exploratory tests in order to become available for an on-demand production deployment. This requires a substantial investment in build, test, and deployment automation.

The best practices associated with a deployment pipeline are:

Practice	Description
Build Your Binaries Once	Create immutable, environment-neutral artifacts
Deploy The Same Way	Use the same deployment mechanism from workstations to production
Smoke Test Your Deployments	Run a handful of end-to-end tests post deployment
Deploy Into Production Copy	Build production-like test environments for on-demand use
Instant Propagation	Automatically trigger automated tests in a pipeline
Stop The Line	Prioritise the remediation of failed deployments over ongoing work

Continuous Delivery requires substantial technology and organisational changes, in addition to a deployment pipeline. The recommended technology changes include:

- *Version Control*. Keep the entire system in version control to preserve change history — code, configuration, infrastructure definitions, database migration scripts, and reference data.
- *Environments*. Automate environment creation to enable self-service, on-demand provisioning of test environments and incremental deployment patterns.
- *Development*. Use Test-driven development with pair programming to build quality in, and use Trunk Based Development with Continuous Integration to ensure it is always releasable.
- *Architecture*. Establish an Evolutionary Architecture to encourage loosely-coupled, independently deployable services.
- *Testing*. Automate parallelisable acceptance tests with dynamic test data, use Smoke Testing to validate deployments, and use Exploratory Testing to uncover new information.
- *Operability*. Aggregate logs and metrics to create a centralised telemetry platform for visualising normal conditions and alerting on abnormal conditions.

The recommended organisational changes include:

- *Batch size*. Reduce the number of features per release to decrease variability, feedback delays, risk, overheads, inefficiencies, and lead time via Little's Law[42].
- *Management*. Devolve decision making to the employees closest to the work to empower better choices in software design, testing practices etc.
- *Culture*. Grow a performance-oriented culture of cooperation and collaboration to increase information flow between teams.
- *Responsibilities*. Change responsibilities for testing and on-call production support to encourage a shared sense of accountability.

[42]Little's Law was stated by John Little as 'the long-term average number of customers in a stable system is equal to the long-term average effective arrival rate multiplied by the average time a customer spends in the system'. It guarantees a long-term smaller batch size at a stable completion rate will shorten time to market.

- *Risk*. Use continuous code reviews via pair programming and traceability of production changes to replace risk management theatre with adaptive risk assessment.
- *Skills*. Invest in training courses for all employees to ease the introduction of new practices such as infrastructure automation, database telemetry, etc.
- *Structures*. Convert siloed functional teams into cross-functional teams and align architecture with Conway's Law[43] to enable faster delivery through the organisation.

These technology and organisational changes are challenging, but it is their application to the unique circumstances and constraints of an organisation that makes Continuous Delivery so difficult. An organisation is a complex adaptive system in which behaviours emerge from unpredictable interactions, the cause and effect of an event can only be understood in retrospect, and every individual has limited information. Adopting Continuous Delivery means implementing technology and organisational changes in the highly uncertain conditions of a human-centric system.

Continuous Delivery has a dynamic success threshold that is dependent on business needs. An organisation is in a state of Continuous Delivery when its IT services have the stability and throughput required to satisfy product demand. The use of smaller releases creates more opportunities for success, as it results in higher quality, fewer risks, and faster feedback on business outcomes during product development. An organisation with fewer, shallower defects and a faster time to market can minimise failure demand[44] and trial new ideas faster than its competitors, profiting from a First Mover Advantage[45] when they are successful.

The benefits of Continuous Delivery have been validated by the annual *State Of DevOps Reports*[46] and in *Accelerate*[47], by Dr Nicole Forsgren *et al.* The *State Of DevOps Reports* are a set of scientific studies of IT performance, based on surveys of 23,000 people working at 2,000 organisations worldwide. *Accelerate* describes the startling results:

- High performance IT leads to simultaneous improvements in the stability and throughput of IT delivery, without trade-offs.
- High performance IT means an organisation is twice as likely to exceed profitability, market share, and productivity goals.
- Implementing Continuous Delivery results in high performance IT.
- Implementing Continuous Delivery also results in less rework, an improved organisational culture, reduced team burnout, and increased job satisfaction.

Use the Improvement Kata

The adoption of Continuous Delivery should begin with the use of a Continuous Improvement framework, to provide guidelines on experimentation and problem solving in situations where there is uncertainty and incomplete information.

[43]Conway's Law was stated by Mel Conway as 'any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure' and suggests the design of a organisation influences the design of its products.

[44]Failure demand is 'demand caused by a failure to do something, or something right for the customer'.

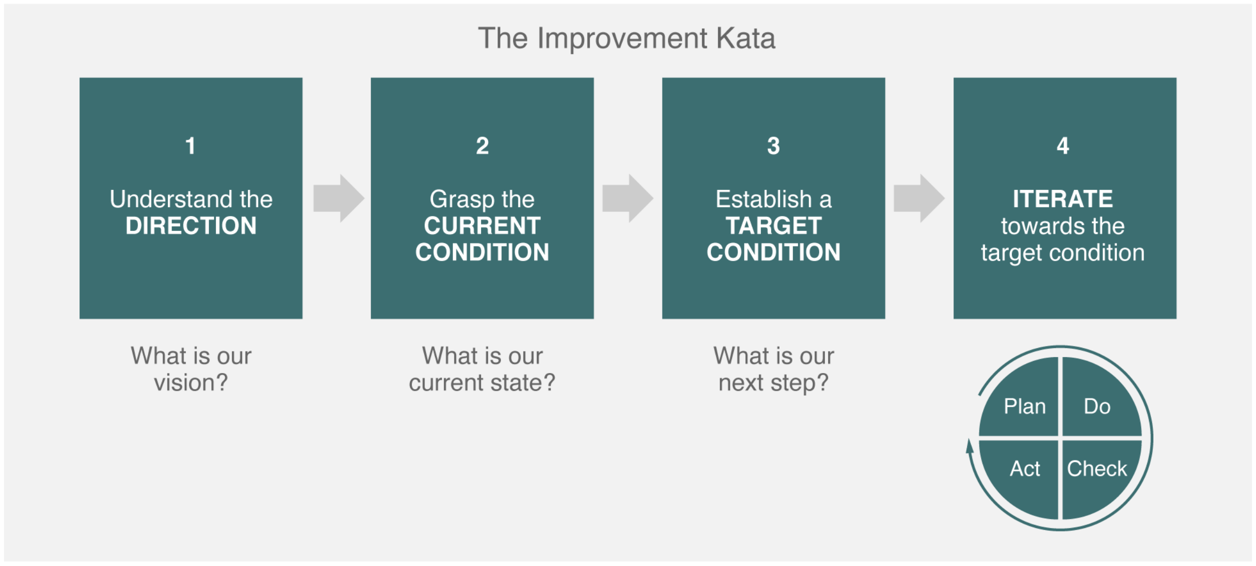
[45]First Mover Advantage is 'the advantage gained by the initial significant occupant of a market segment'.

[46]The *State Of DevOps Reports* *2014*, *2015*, *2016*, and *2017* by Dr Nicole Forsgren *et al* (2014-2017).

[47]*Accelerate* by Dr Nicole Forsgren *et al* (2018).

The Improvement Kata as defined by Mike Rother in *Toyota Kata*[48] is a Continuous Improvement framework by Toyota. A kata is a teaching method based on repetition of proven techniques, and the Improvement Kata encourages Continuous Improvement by creating a regular cycle of iterative, incremental improvements around the existing working practices of an organisation.

THE IMPROVEMENT KATA


The Improvement Kata
1 Understand the DIRECTION
2 Grasp the CURRENT CONDITION
3 Establish a TARGET CONDITION
4 ITERATE towards the target condition
What is our vision?
What is our current state?
What is our next step?
Plan
Do
Act
Check

The Improvement Kata involves repeated cycles of a four-step improvement process:

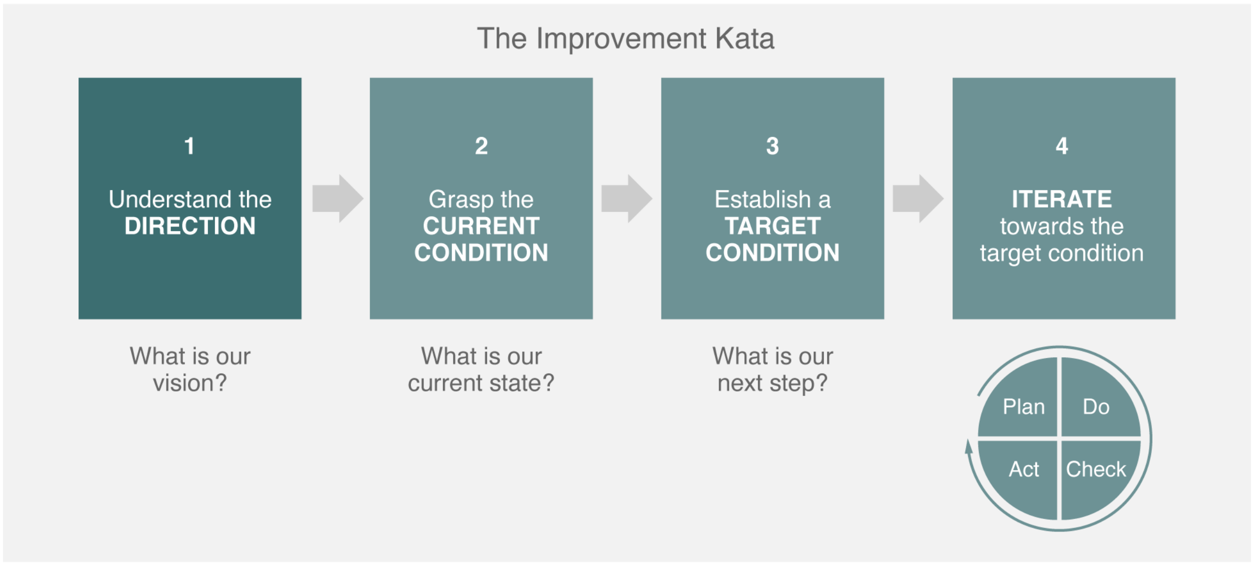
1. *Understand the Direction*. Define a vision of success to advertise the goal of the organisation and how people can contribute towards it.
2. *Grasp the Current Condition*. Analyse the available data to determine progress made in the last improvement cycle, and overall progress made towards the goal.
3. *Establish a Target Condition*. Create the next improvement cycle, with success criteria and time horizon used to create an intermediary milestone of value.
4. *Iterate towards the Target Condition*. Use the Plan-Do-Check-Act Cycle[49] to run experiments within an improvement cycle, working towards the Target Condition.

The Improvement Kata is a great fit for a Continuous Delivery adoption programme. The emphasis on regular, small improvements is perfectly aligned with the ethos of Continuous Delivery.

[48] *Toyota Kata* by Mike Rother (2009).
[49] Plan-Do-Check-Act is an iterative four-step method of continuous improvement invented by Walter Shewhart, and popularised by Dr W Edwards Deming

Create a vision of success

THE DIRECTION FOR AN IMPROVEMENT KATA


The Improvement Kata
1 Understand the DIRECTION
What is our vision?
2 Grasp the CURRENT CONDITION
What is our current state?
3 Establish a TARGET CONDITION
What is our next step?
4 ITERATE towards the target condition
Plan
Do
Act
Check

Once an organisation has the Improvement Kata in place its business stakeholders need to meet and agree on a direction for the entire organisation. This should be an ambitious vision of success that describes how Continuous Delivery will enable future economic prosperity, and the outcome of that meeting should be the publication of an agreed vision of success to all employees.

A vision of success should:

- Convey a desirable, attainable future founded upon Continuous Delivery.
- Provide guidelines for decision making when implementing changes.
- Create a sense of urgency that can be understood by everyone.

A shared sense of urgency is of critical importance. In *The Corporate Culture Survival Guide*[50], Edgar Schein describes how a crisis in an organisation can bring people together, and foment change. A good vision of success will have an urgency that stimulates an internal crisis, increasing learning anxiety and survival anxiety amongst employees. An organisation should build commitment to its vision of success by reducing Learning Anxiety via training courses, coaching, communities of practice, and positive incentives. These initial steps will encourage employees to actively participate in implementing Continuous Delivery.

Once an organisation has published its vision of success, it should be cascaded down for teams to apply the Improvement Kata to their own work. Each team needs to iteratively experiment with technology and organisational changes to move the team and the entire organisation incrementally towards Continuous Delivery.

[50] *The Corporate Culture Survival Guide* by Edgar Schein (2009).

ToyVille: a vision of success

Fearing for the future of their organisation, the ToyVille executive team recognise the need for dramatic change in their product and technology capabilities. They agree teams should investigate the benefits of Continuous Delivery, and they create a vision of success:

To avoid becoming obsolete, ToyVille will achieve 15% online sales growth within two years thanks to a step change in customer experience. This will be powered by a rapid, reliable technology launch cycle of one week with no more than 1% failures.

This means the ToyVille Improvement Kata starts as:

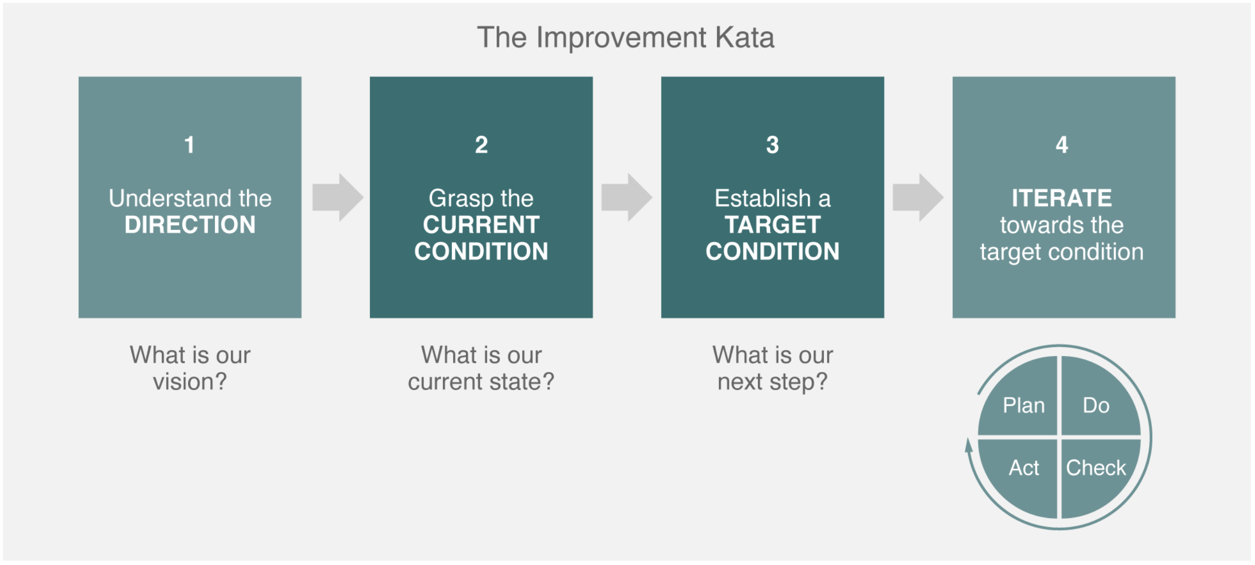
Delivery Improvement	**Current Condition**	**Target Condition**	**Direction**
Online sales growth	?	?	**15%**
Launch cycle	?	?	**1 week**
Launch failures	?	?	**1%**

The ToyVille executives communicate this vision to all their employees via email, online video presentations, and town hall meetings at different offices. Continuous Delivery training programmes are created, communities of practice are formed, and employees are encouraged to contribute to the adoption programme.

A product leadership team is formed to investigate the product/market fit of upcoming features, and a delivery enablement team is created to guide the necessary experimental changes through the development, testing, and operations teams.

Plan an improvement cycle

CONDITIONS IN AN IMPROVEMENT CYCLE


The Improvement Kata
1 Understand the DIRECTION
2 Grasp the CURRENT CONDITION
3 Establish a TARGET CONDITION
4 ITERATE towards the target condition
What is our vision?
What is our current state?
What is our next step?
Plan
Do
Act
Check

A team plans an improvement cycle by evaluating its Current Condition, and applying it to the organisation-wide direction to create its next Target Condition. All available quantitative[51] and qualitative[52] data on stability and throughput from previous improvement cycles is examined to understand the current state of progress. When that snapshot is compared with the vision of success an improvement milestone can be identified, including its success criteria and time horizon.

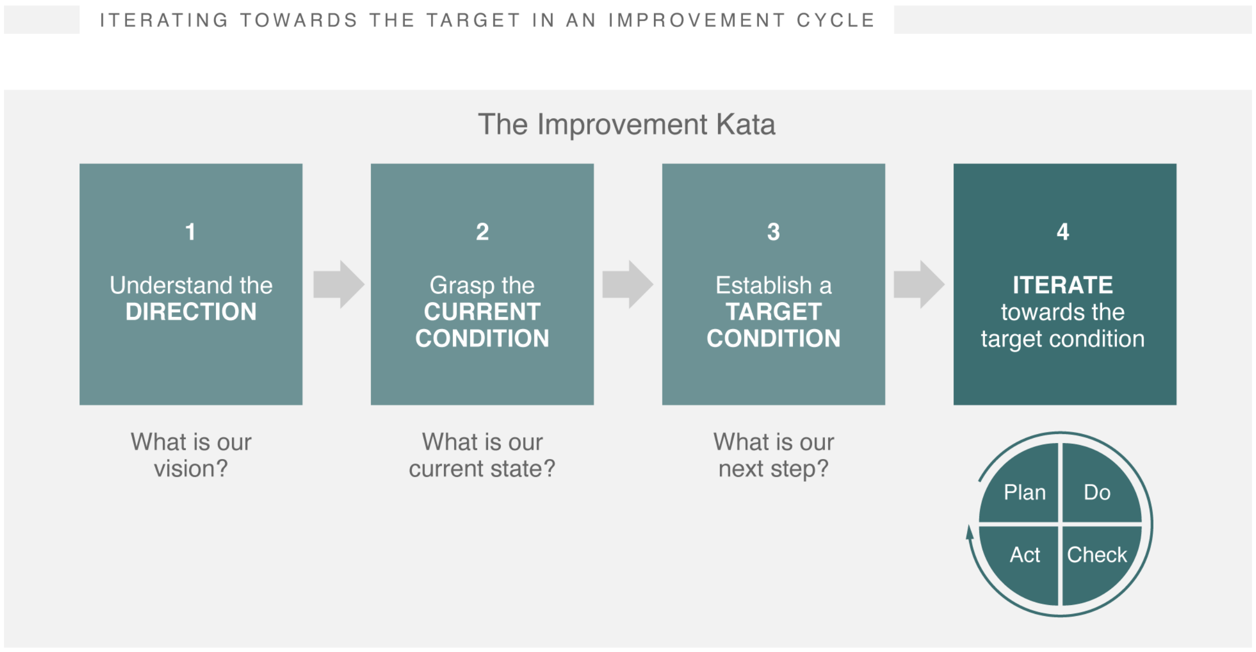
ToyVille: planning disputes

The delivery enablement team at ToyVille meets with the product leadership team, and the initial Current Condition is estimated as 5% online sales growth, a three-month launch cycle, and a 25% launch failure rate. A discussion about setting the first Target Condition proves to be more difficult, as many different opinions emerge. Some people want to improve stability, others want to focus on throughput, and there is no data to guide the conversation.

The updated ToyVille Improvement Kata is:

	Current Condition	**Target Condition**	**Direction**
Online sales growth	**5%**	?	15%
Launch cycle	**3 months**	?	1 week
Launch failures	**25%**	?	1%

Execute an improvement cycle


ITERATING TOWARDS THE TARGET IN AN IMPROVEMENT CYCLE
The Improvement Kata
1 Understand the DIRECTION
2 Grasp the CURRENT CONDITION
3 Establish a TARGET CONDITION
4 ITERATE towards the target condition
What is our vision?
What is our current state?
What is our next step?
Plan
Do
Act
Check

[51] Quantitative data is 'quantities obtained using a quantifiable measurement process'.
[52] Qualitative data is 'qualities that are descriptive, subjective or difficult to measure'.

During planning it is not possible to know how a Target Condition will be met, as a team will need to experiment with different changes amid the uncertain conditions of the organisation. A team executes an improvement cycle by repeatedly iterating on different technology and organisational changes using the Plan-Do-Check-Act Cycle:

- *Plan.* Plan a change, and state an expected improvement.
- *Do.* Execute the change, and collect data for analysis.
- *Check.* Study the results, and compare against the expected improvement.
- *Act.* Incorporate the change into the baseline if successful, or discard it.

By experimenting with different changes a team can use critical thinking to gradually solve problems, increase its knowledge, and converge on the Target Condition. The improvement cycle ends when its success criteria are met, or its time horizon expires. At that point planning for the next improvement cycle should begin immediately.

ToyVille: execution confusion

Frustrated by the lack of progress, the delivery enablement team asks the development, testing, and operations teams to try out different experimental changes while their first improvement milestones remain under discussion.

The teams are happy to start, but they do not know which changes to try in their first Plan-Do-Check-Act cycle. The development teams debate Test-driven development versus Continuous Integration, the testing team cannot decide between parallel acceptance tests or smoke tests, and the operations team is torn between automated infrastructure and a telemetry platform. Even the delivery enablement team cannot choose between creating time to market incentives or encouraging universal version control.

The lack of success criteria means the teams cannot easily know if their changes have a positive impact on their team, let alone ToyVille as a whole. Furthermore, the leadership team does not know how to compare progress made in an improvement cycle against the original vision of success.

Measure Continuous Delivery

The key to success with the Improvement Kata is to use it systematically and continuously, for all kinds of experiment. When an organisation uses the Improvement Kata to adopt Continuous Delivery, decisions need to be made on a regular basis:

- What is the current vision of success?
- Should a team focus on stability or throughput for an improvement cycle?
- What stability and/or throughput criteria should be used for an improvement milestone?

- Which technology and organisational changes should a team experiment with during an improvement cycle?
- How can the success of a technology or organisational change be confirmed?
- How can the success of an improvement milestone be confirmed?

Outcomes will inevitably be suboptimal when no information is available to reduce the uncertainty surrounding these decisions. The Improvement Kata will suffer from badly-defined Current and Target Conditions, experimental changes will fail to make a positive impact, and there will be a tangible lack of progress. Stakeholder confidence in the ongoing Continuous Delivery programme will be eroded, and Discontinuous Delivery will remain the norm.

When an organisation adopts Continuous Delivery, there needs to be a quantification of the stability and throughput of IT services. Measuring stability and throughput will uncover new information about the current constraints, enabling better decision making and giving Continuous Delivery the best possible chance of success.