

# Tema 6

## *DevOps*

### **6.1 Introducción**

### **6.2 Las Tres Vías**

#### **6.2.1 Primera Vía: Acelerar el flujo**

#### **6.2.2 Segunda Vía: Retroalimentación continua**

#### **6.2.3 Tercera Vía: Experimentación continua**

### **6.3 Practicando la Primera Vía**

#### **6.3.1 Infraestructura es código en la pipeline**

#### **6.3.2 Pruebas automáticas y CI**

#### **6.3.3 CD**

#### **6.3.4 Arquitectura y Despliegues**

### **6.4 Estado Actual**

### **6.5 Casos de Estudio**

Pablo.Bermejo@uclm.es

## **6.1 Introducción**

## **6.2 Las Tres Vías**

**6.2.1 Primera Vía: Acelerar el flujo**

**6.2.2 Segunda Vía: Retroalimentación continua**

**6.2.3 Tercera Vía: Experimentación continua**

## **6.3 Practicando la Primera Vía**

**6.3.1 Infraestructura es código en la pipeline**

**6.3.2 Pruebas automáticas y CI**

**6.3.3 CD**

**6.3.4 Arquitectura y Despliegues**

## **6.4 Estado Actual**

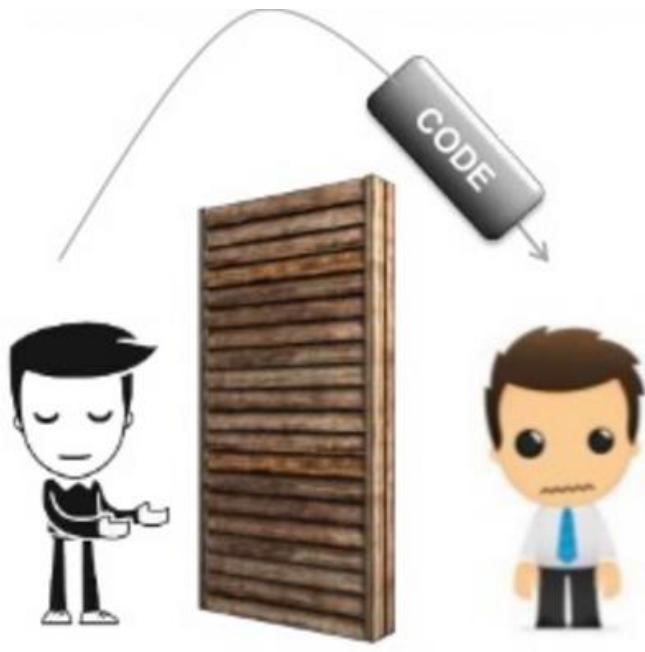
## **6.5 Casos de Estudio**

## 6.1 DevOps - Introducción

---

- Durante la década de 2000 las metodologías ágiles fueron adoptadas por los desarrolladores de manera exponencial, siendo Scrum la metodología mayoritariamente aplicada a partir de 2010.
- En estas décadas también resurgió la filosofía **Lean** (flujo continuo, Lead Time, aseguramiento de la calidad desde el principio, confianza dentro del equipo,...)
- Los **rápidos incrementos en la funcionalidad** del software en iteraciones cortas fue una gran evolución para los equipos de desarrolladores: autoorganizativos (Scrum), pruebas e integración continua (XP), vision del trabajo en progreso (Kanban)...
- Pero... no todo es(era) programar...

# 6.1 DevOps - Introducción



Development

Operations

¿Código funcionando en servidores finales?  
*No me quedan..., no se levantan...*

¿Problemas de seguridad?

¿Nuevos bugs en nuevos entornos?  
*Configuración manual...*

Cuando una release finalmente se despliega,  
Devs ya ha enviado dos builds más!

Si falla algo (resolverlo podía ser >1mes),  
¿La culpa es de Dev o de Ops?

## 6.1 DevOps - Introducción



# 6.1 DevOps - Introducción

## Métricas de interés de DEVs

Deployment frequency



Agility performance indicators

Change lead time



## Métricas de interés de Ops

Change fail rate



Mean time to detect & repair



Microsoft

Reliability performance indicators

# 6.1 DevOps - Introducción

- En una realidad en la que un despliegue tenía que ser pre-acordado, normalmente para un Viernes por la tarde, y su frecuencia media era entre 2 y 12 meses, en 2009 se dio el ultimo gran salto en el mundo de IT hasta la fecha:



John Allspaw  
y  
Paul Hammond

O'reilly Velocity 2009 - 10+ Deploys Per Day: Dev and  
Ops Cooperation at Flickr  
2010 - Web Operations (book)

<https://www.youtube.com/watch?v=LdOe18KhtT4>

# 6.1 DevOps - Introducción

- Pusieron nombre a lo que todos pensaban:

DEVS



OPS

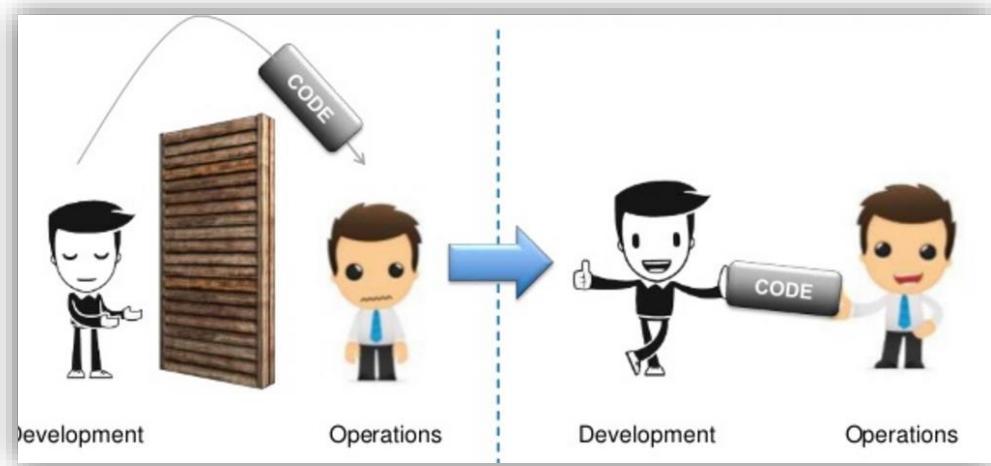


Little bit weird  
Sits closer to the boss  
Thinks too hard

Pulls levers & turns knobs  
Easily excited  
Yells a lot in emergencies

# 6.1 DevOps - Introducción

- Y contaron que si una empresa conseguía algo como esto:

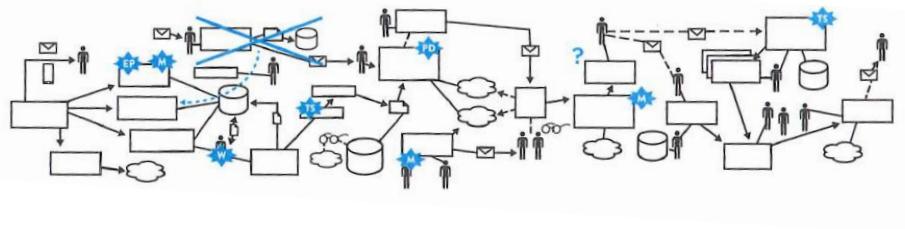


DevOpsDays 2009 – Patric Debois

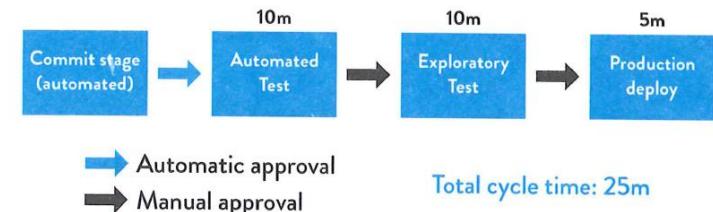


# 6.1 DevOps - Introducción

- Se podrían alcanzar **10 despliegues por día**, como ya estaban haciendo en Flickr.
- En esta última década se han definido mejores prácticas para alcanzar la cultura o mentalidad DevOps.
- 10 despliegues por día ya no parece tanto... Una empresa, start-up, unicornio o caballo que evoluciona a DevOps puede hacer 1 despliegue en minutos.



Flujo del valor tecnológico en una empresa con lead time de despliegue de 3 meses.

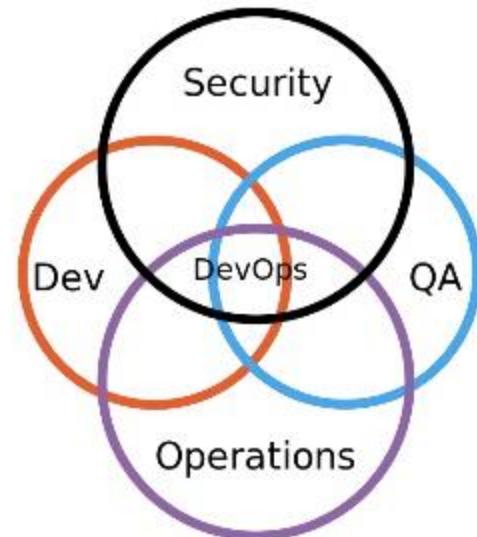


Flujo del valor tecnológico en una empresa con lead time de minutos.

*The DevOps Handbook Gene Kim, Jez Humble, et al.*

# 6.1 DevOps - Introducción

- DevOps:
  - No es una tecnología concreta
  - No es una metodología concreta



*“Una mezcla de patrones con la intención de mejorar la colaboración entre Desarrollo y Operaciones. DevOps comparte tanto objetivos e incentivos como procesos y herramientas.”*

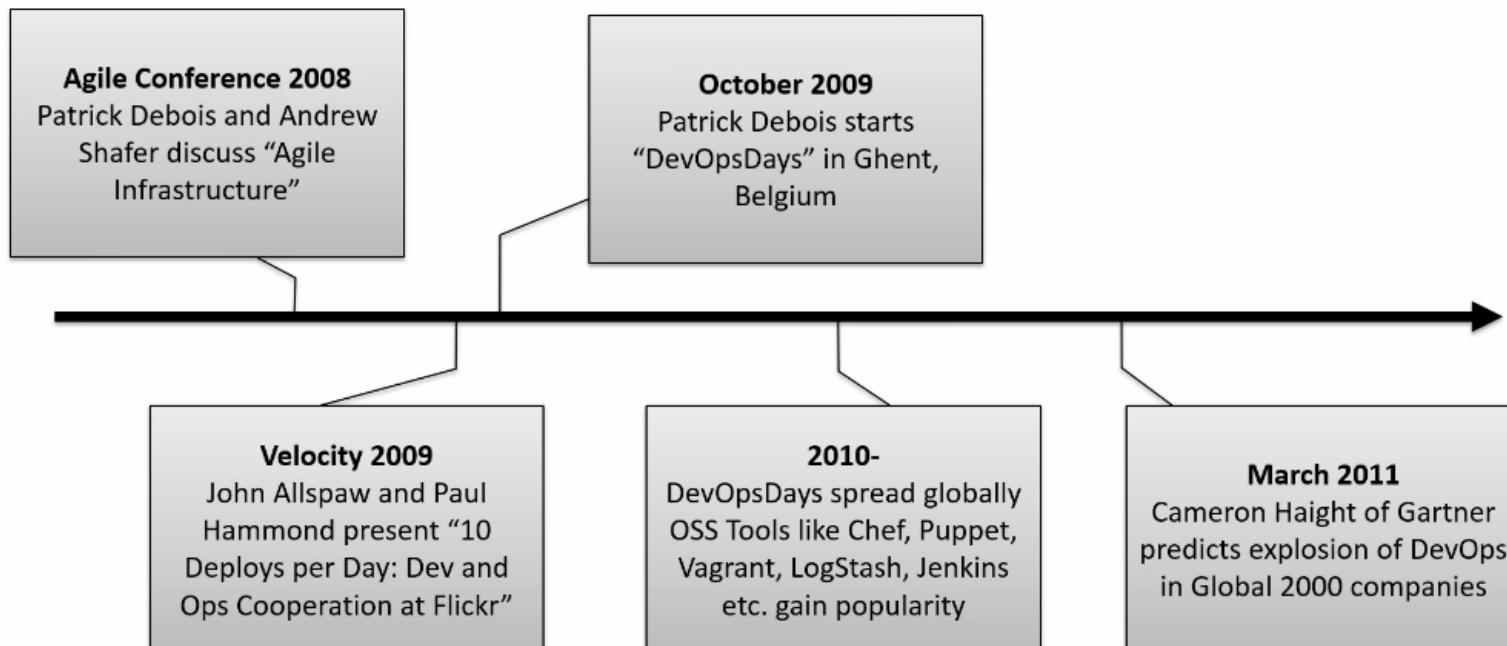
*Michael Huttermann - DevOps for Developers*

*“Un movimiento de personas que se preocupan por desarrollar y operar sistemas a escala de forma confiable, segura y con altas prestaciones.”*

*Jez Humble - [www.thoughtworks.com](http://www.thoughtworks.com)*

# 6.1 DevOps - Introducción

## History of DevOps



# 6.1 DevOps - Introducción

- Facebook, Netflix, Flickr, Amazon, Twitter,... han adoptado DevOps. Algunos piensan que DevOps solo funciona en estas empresas gracias a su estela mágica y que ya son triunfadores (unicornios).
- Pero por supuesto es útil para cualquier caballo:



- Y start-ups:



Porque DevOps reduce el tiempo de despliegue, la tasa de fallos, aumenta la satisfacción del cliente, la calidad, la seguridad,...

¿Cómo?

¿Por qué?

Vamos a verlo...

## **6.1 Introducción**

## **6.2 Las Tres Vías**

**6.2.1 Primera Vía: Acelerar el flujo**

**6.2.2 Segunda Vía: Retroalimentación continua**

**6.2.3 Tercera Vía: Experimentación continua**

## **6.3 Practicando la Primera Vía**

**6.3.1 Pipeline**

**6.3.2 Pruebas automáticas y CI**

**6.3.3 CD**

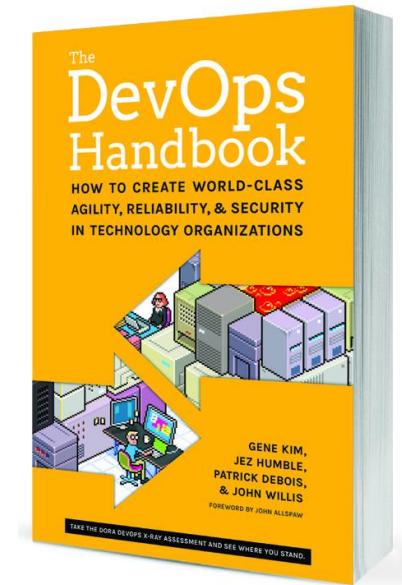
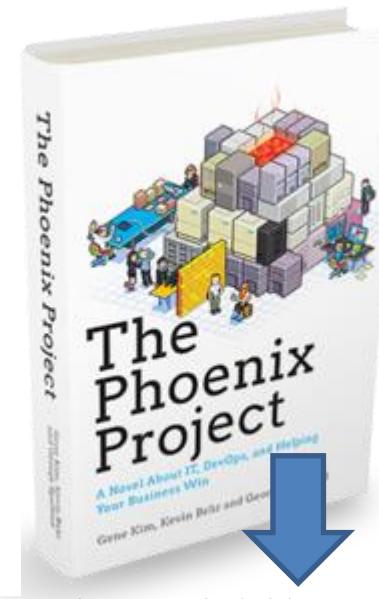
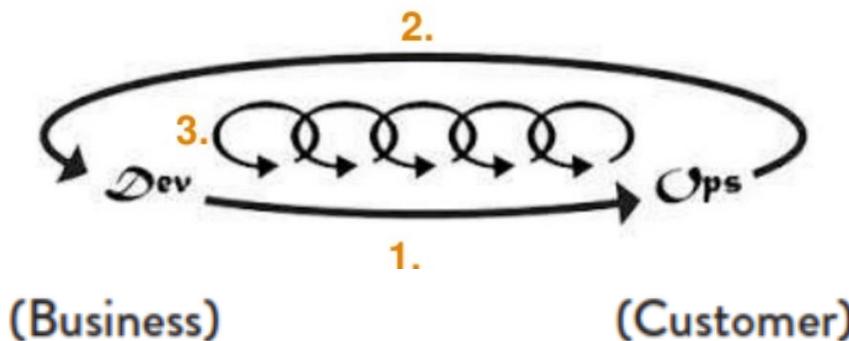
**6.3.4 Arquitectura y Despliegues**

## **6.4 Estado Actual**

## **6.5 Casos de Estudio**

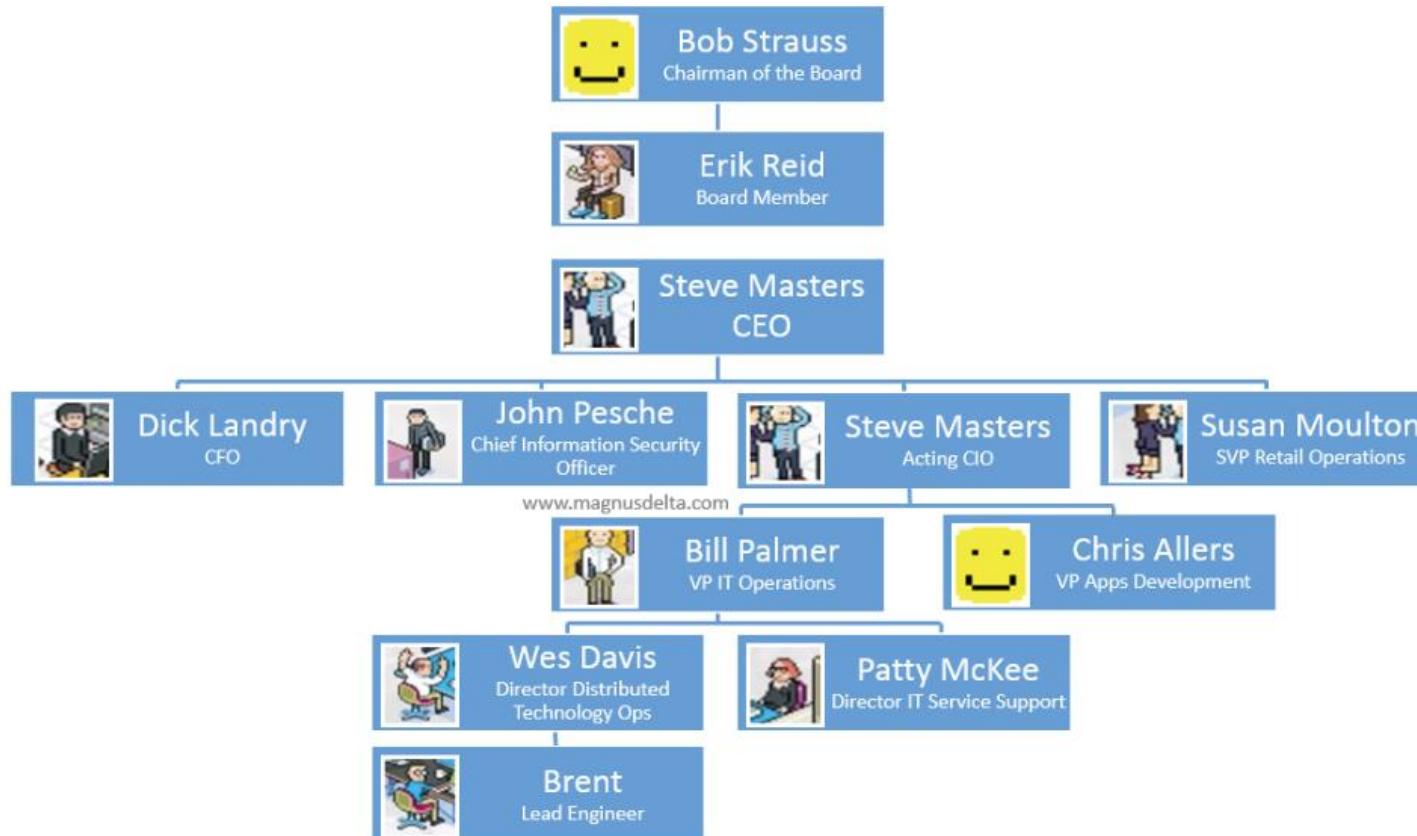
## 6.2 Las Tres Vías

- **Las Tres Vías** (The Three Ways) describen los principios y filosofía que enmarcan los procesos y prácticas de DevOps.
- Gene Kim, uno de los mayores comunicadores sobre la cultura DevOps, es el autor de estos principios.



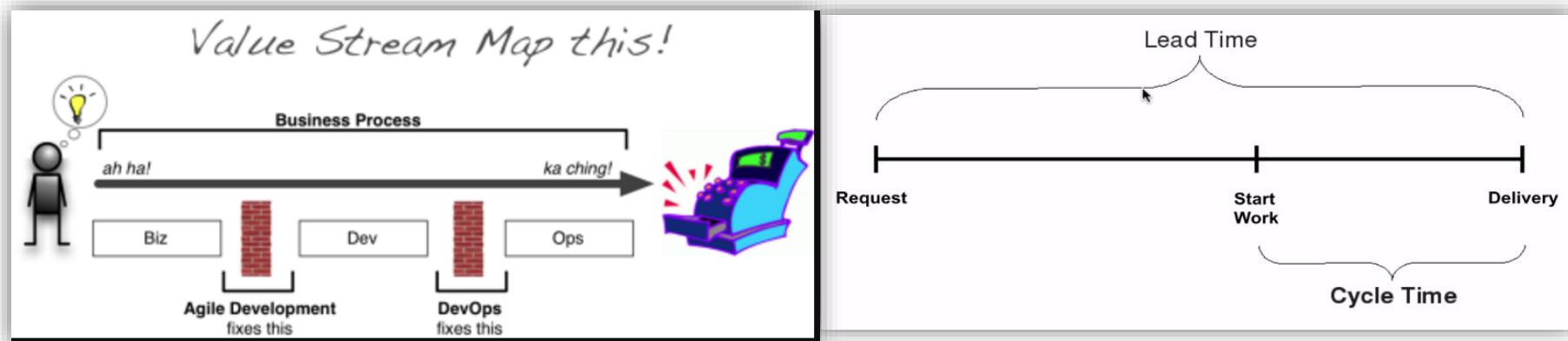
4 ejemplares en la biblioteca del campus.  
0.75 extra sobre tu peor nota (o nota final) al leerlo y  
DevOps      presentar un resumen

- Organigrama de los personajes de Phoenix Project en  
<https://www.magnusdelta.com/blog/2017/9/16/the-phoenix-project-summary>



## 6.2 Las Tres Vías

- Cada vía indica cómo actuar sobre el flujo del valor tecnológico.
- El **flujo del valor** conlleva:
  - Crear una idea
  - Añadir trabajo al backlog.
  - Crear una historia de usuario o característica.
  - Implementar.
  - Check-in en control de versiones.
  - Desplegar en producción.
  - Validar la experiencia del cliente.



## **6.1 Introducción**

## **6.2 Las Tres Vías**

### **6.2.1 Primera Vía: Acelerar el flujo**

**6.2.2 Segunda Vía: Retroalimentación continua**

**6.2.3 Tercera Vía: Experimentación continua**

## **6.3 Practicando la Primera Vía**

**6.3.1 Infraestructura es código en la pipeline**

**6.3.2 Pruebas automáticas y CI**

**6.3.3 CD**

**6.3.4 Arquitectura y Despliegues**

## **6.4 Estado Actual**

## **6.5 Casos de Estudio**

## 6.2 Las Tres Vías

### La Primera Vía: Acelerar el flujo hacia la derecha

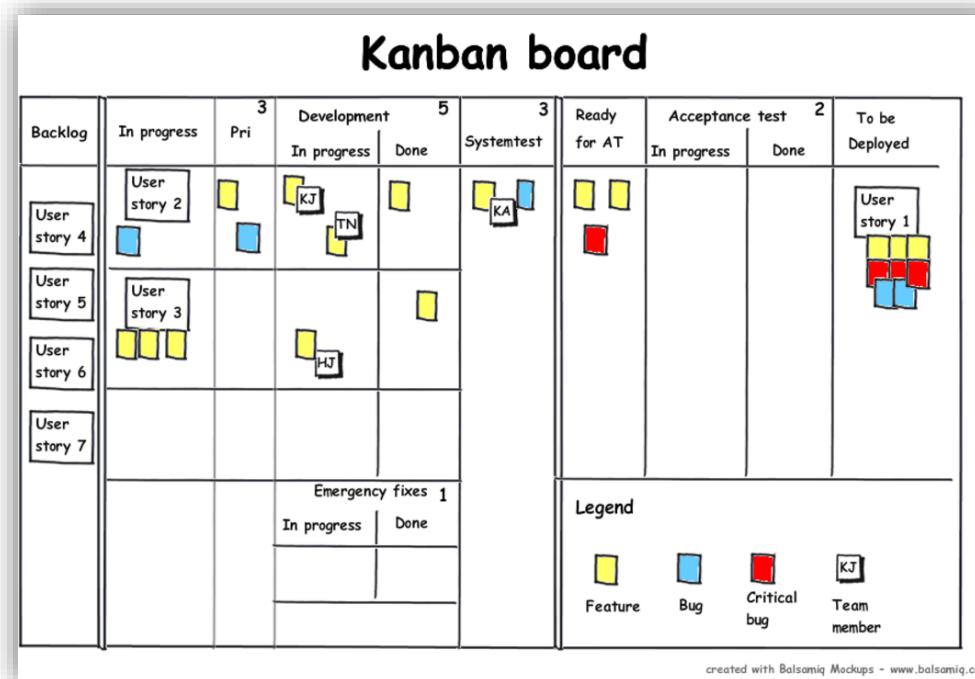


- Conseguir que el flujo de trabajo avance de manera rápida y suave de Desarrollo a Operaciones para así entregar **valor a los clientes** cuanto antes.
- Ya no nos centramos en objetivos locales (tasa de desarrollo de características, ratios de pruebas con éxito, disponibilidad de los recursos en Operaciones,...).
- A partir de Lean, se identifican las siguientes **prácticas para aumentar el flujo** de valor tecnológico de Dev hacia Ops:
  1. Hacer que el trabajo sea visible.
  2. Limitar el WIP.
  3. Reducir el tamaño de los lotes (*batch*) de características.
  4. Reducir el número de traspasos (*handoffs*) entre departamentos o centros de trabajo.
  5. Identificar continuamente nuestras restricciones o cuellos de botella.
  6. Eliminar el sobrante (*waste*) del flujo.

## 6.2 Las Tres Vías – Primera

### 1. Hacer que el trabajo sea visible

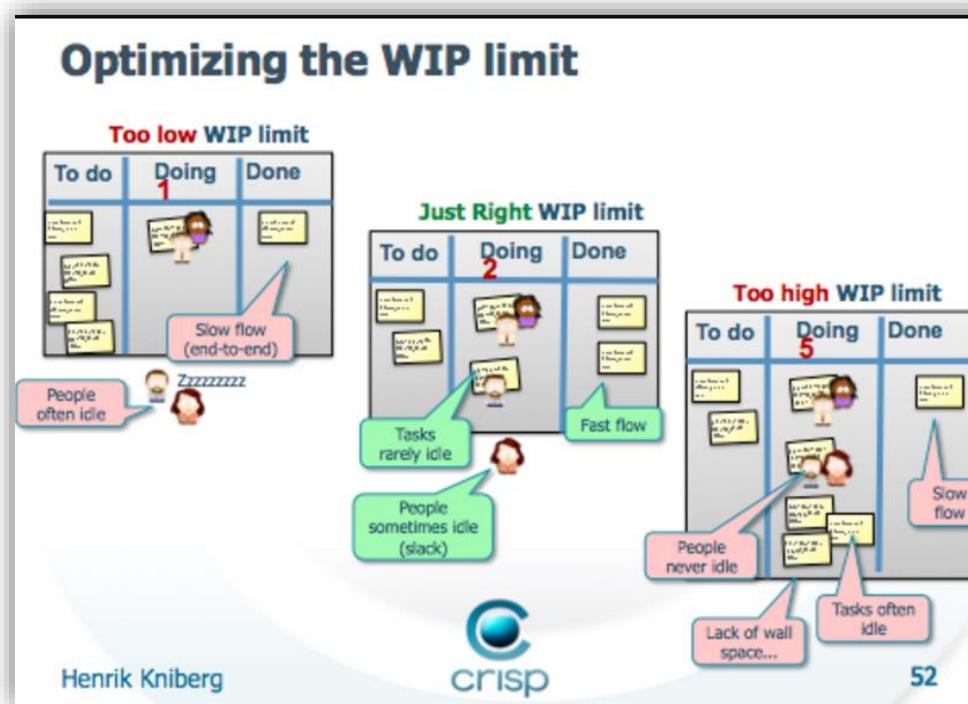
- El movimiento o flujo del trabajo tecnológico no es visible, como lo puede ser en cadenas de montaje, transporte, etc...
- Conviene utilizar **tableros** como Kanban o planificación de sprints, mostrando no solo la parte de Desarrollo sino todo el *pipeline*.
- El trabajo no está hecho hasta que la característica o historia no está al servicio del cliente en Ops.



## 6.2 Las Tres Vías – Primera

### 2. Limitar el WIP (work in progress).

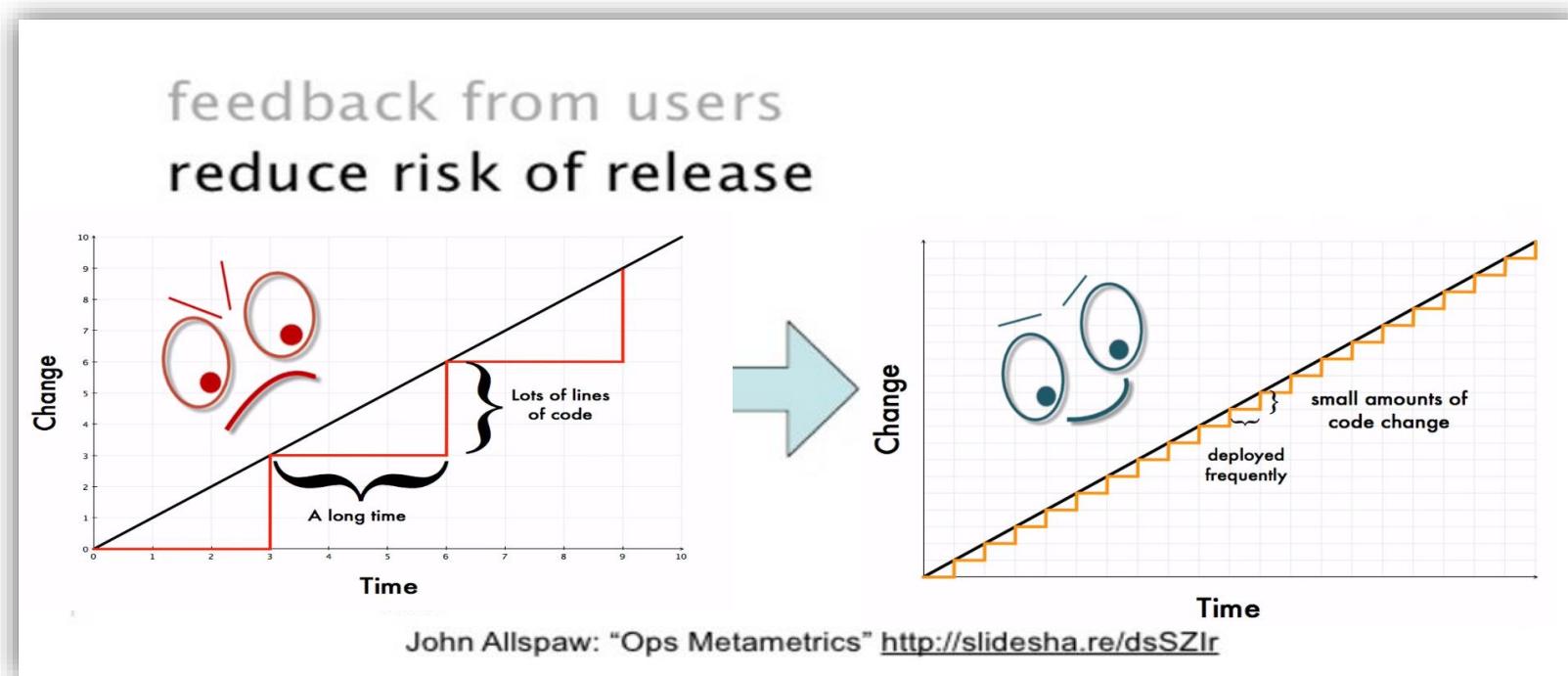
- Debemos tener en cuenta que el tiempo para realizar una tarea, por simple que sea, aumenta cuando se está trabajando en modo **multitarea (alto WIP)**.
- Usando el tablero que hace visible nuestro trabajo, podemos limitar el WIP limitando el número de características en progreso (*Doing*). 3 es un valor asumible.
- ‘*Stop Starting. Start Finishing*’.



## 6.2 Las Tres Vías – Primera

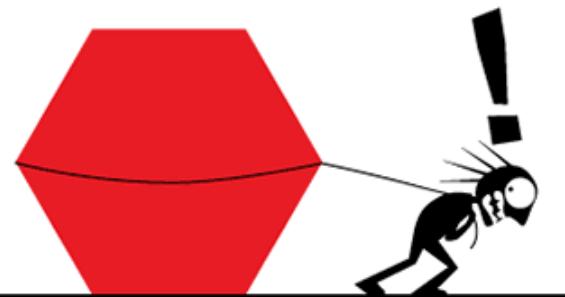
### 3. Reducir el tamaño de los lotes (batch size)

- Hacer que nuestros lotes tengan **pocas características** reduce el WIP y reduce el tiempo de detección de errores, reduciendo así riesgos y aumentando la calidad.
- **“flujo de 1 pieza”**: cuando cada cambio o incremento es automáticamente integrado, testeado y desplegado en producción.



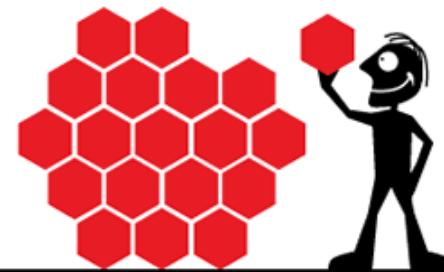
## 6.2 Las Tres Vías – Primera

THE WATERFALL PROCESS



*'This project has got so big,  
I'm not sure I'll be able to deliver it!'*

THE AGILE PROCESS



*'It's so much better delivering this  
project in bite-sized sections'*

## 6.2 Las Tres Vías – Primera

### 4. Reducir el número de traspasos (handoffs)

- Transmitir código por el flujo de valor puede suponer trabajo de múltiples departamentos y todo tipo de comunicaciones: pruebas de integración, creación de entornos, redes, almacenamiento, seguridad, peticiones, emails, coordinar, resolver conflictos de intereses,...
- **Cada paso puede suponer una cola de espera** que aumenta nuestro *lead time*.
- Realizar el mayor esfuerzo para reducir el número de traspasos y **automatizar** todo lo posible.



<https://www.youtube.com/watch?v=Dr67i5SdXiM>

<https://twitter.com/paberlo/status/1275031412001906690>

## 6.2 Las Tres Vías – Primera

### 5. Identificar continuamente el cuello de botella (constraints)

- “Siempre hay 1 y solo 1 cuello de botella; cualquier mejora que no se realice sobre esa limitación es una ilusión” – *Beyond the Goal*.
- A medida que se adopta DevOps, el cuello de botella suele ir apareciendo **en este orden**:
  - Creación de entornos
  - Despliegue del Código
  - Tests
  - Protocolos estrictos con puertas manuales (falta de autonomía).
- Nuestro cuello de botella final será Desarrollo o el Product Owner. A partir de aquí todo dependerá de la productividad del equipo Dev.

## 6.2 Las Tres Vías – Primera

### 6. Eliminar el sobrante (waste)

- **Sobrante o Desperdicios** es cualquier elemento que no añade valor y retrasa la llegada de éste al cliente.
- Trabajo sin acabar, demasiados procesos, demasiadas características, cambiar mucho entre tareas, errores, trabajo manual, héroes (se les encarga demasiado (imposible) trabajo constantemente).

Wastes of Manufacturing	Wastes of Software Development
Inventory	Partially work done
Extra processing	Paperwork or excess documentation
Overproduction	Extra features
Transportation	Building the wrong thing
Waiting	Waiting for the information
Motion	Task switching & Motion
Defects	Defects

Reference: M & T Poppendieck, [Lean Software Development](#), 2003 : Chapter 1

## **6.1 Introducción**

## **6.2 Las Tres Vías**

**6.2.1 Primera Vía: Acelerar el flujo**

**6.2.2 Segunda Vía: Retroalimentación continua**

**6.2.3 Tercera Vía: Experimentación continua**

## **6.3 Practicando la Primera Vía**

**6.3.1 Infraestructura es código en la pipeline**

**6.3.2 Pruebas automáticas y CI**

**6.3.3 CD**

**6.3.4 Arquitectura y Despliegues**

## **6.4 Estado Actual**

## **6.5 Casos de Estudio**

## 6.2 Las Tres Vías

### La Segunda Vía: Retroalimentación continua



- **Retroalimentación** rápida y continua de derecha a izquierda, desde cualquier fase del flujo de valor. Esto permite:
  - 1) Detectar problemas antes, mientras son más pequeños y fáciles de arreglar.
  - 2) Solucionar los problemas al instante.
  - 3) Crear una cultura de aprendizaje para no volver a cometer los mismos errores.

## 6.2 Las Tres Vías – Segunda

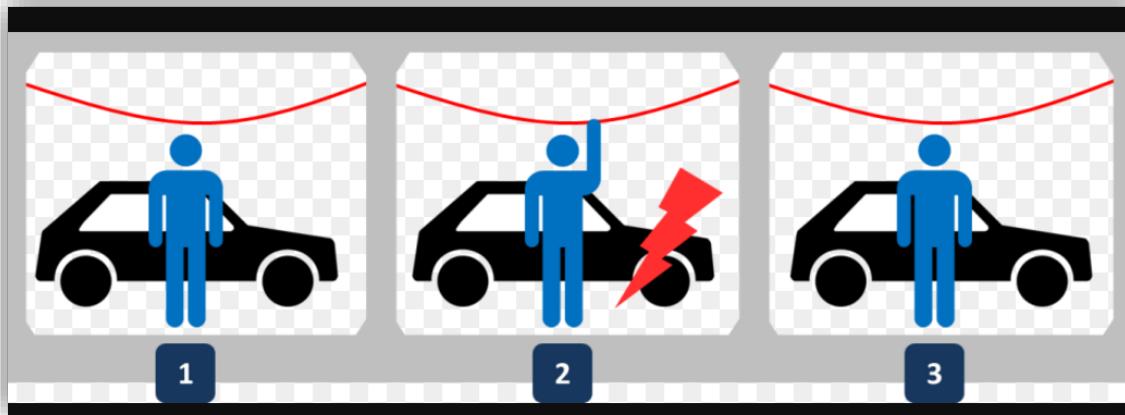
### 1. Detectar problemas antes

- En un proyecto con ciclo de cascada, se puede programar durante un año y no ver los errores en fase de pruebas o en despliegue con el cliente.
- Es mejor enviar retroalimentación desde todas las fases del flujo: Dev, QA, Information Security y Ops → automatizar builds, integraciones, pruebas y publicar los resultados.

### 2. Solucionar problemas al instante

- Cuando aparece un problema se debe movilizar a quien haga falta para resolverlo cuanto antes, en vez de programarlo para cuando haya tiempo (*cable de Andon*).
- Se previene que el problema pase a otras fases donde su coste es mayor.
- Se aprende del problema para que no vuelva a pasar.
- '*Let's make sure this does not happen again*' – The Phoenix Project (una y otra vez)

## 6.2 Las Tres Vías – Segunda



El cable de Andon  
en Toyota

<http://www.allaboutlean.com>

### 3. Cultura de aprendizaje

- Se debe crear una cultura en la que los **errores** no se castigan, sino que se tratan como una **oportunidad** de aprendizaje.
- *'It's impossible for a developer to learn anything when someone yells at them for something they broke six months ago'* – Gary Gruver.
- La responsabilidad de un trabajador ya no reside solo en su puesto de trabajo, sino en todo el flujo del valor. Todos los equipos **se ayudan y tienen cierta idea** del trabajo de todos.

## **6.1 Introducción**

## **6.2 Las Tres Vías**

### **6.2.1 Primera Vía: Acelerar el flujo**

### **6.2.2 Segunda Vía: Retroalimentación continua**

### **6.2.3 Tercera Vía: Experimentación continua**

## **6.3 Practicando la Primera Vía**

### **6.3.1 Infraestructura es código en la pipeline**

### **6.3.2 Pruebas automáticas y CI**

### **6.3.3 CD**

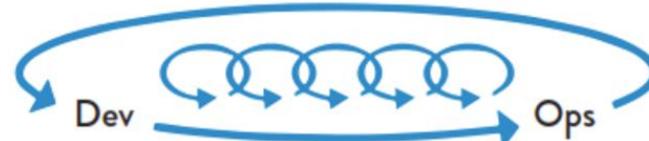
### **6.3.4 Arquitectura y Despliegues**

## **6.4 Estado Actual**

## **6.5 Casos de Estudio**

## 6.2 Las Tres Vías - Tercera

### La Tercera Vía: Experimentación y aprendizaje continuo



- Crear una cultura de alta confianza que nos permita/anime a **tomar riesgos y experimentar continuamente**, aplicando un método científico para mejorar procesos y desarrollo, aprendiendo de nuestros éxitos y fallos.
- **Reservar tiempo** a diario para:
  - Reducir la deuda tecnológica.
  - Arreglar fallos.
  - Refactorizar y mejorar nuestro código y entornos.
  - Añadir nuevas características y liberarlas solo para un subconjunto de usuarios, tomar métricas al respecto (**feature toggle/feature flag**)

## 6.2 Las Tres Vías - Tercera

- Todas nuestras **mejoras** deben convertirse en **conocimiento global**:
  - Informes en repositorio con búsqueda por palabras clave.
  - **Compartir el Código**: en vez de documentos detallando parámetros, compartir bibliotecas y configuraciones que contengan el mejor conocimiento colectivo y fáciles de utilizar mediante scripts.
- Programar ejercicios donde creamos **fallos a gran escala** (apagar clusters, desconectar bases de datos, hacer check-in de Código erróneo,...), para comprobar nuestra capacidad de recuperación/resistencia.

Netflix Chaos Monkey

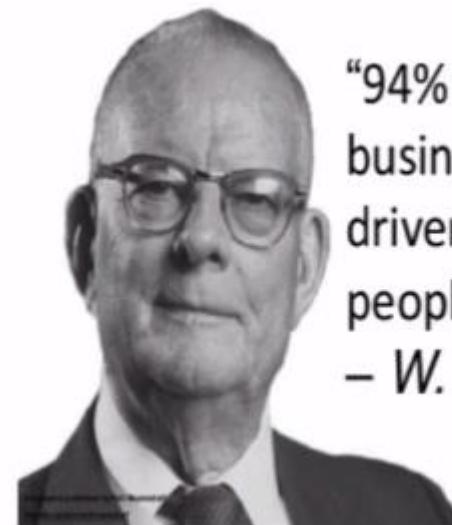
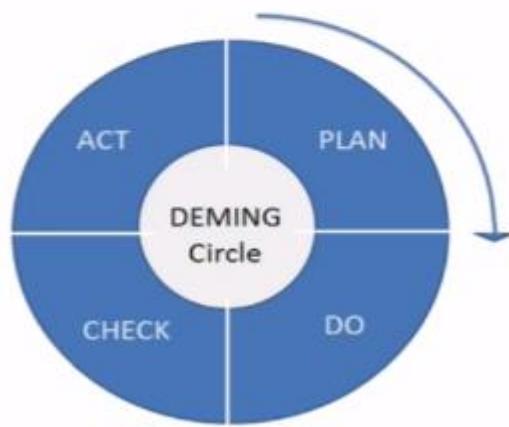
*Netflix lanza de manera regular su Chaos Monkey, el cual aleatoriamente elimina procesos y servidores de producción.*



- Netflix Simian Army
  - Chaos Monkey (Hosts)
  - Chaos Gorilla (Data Center)
  - Latency Monkey (Inject Latency)
  - Conformity Monkey (Best Practice)
  - Security Monkey (Security Violations)

## 6.2 Las Tres Vías - Tercera

- Dr Deming



“94% of problems in business are systems driven and only 6% are people driven.”  
– *W. Edwards Deming*

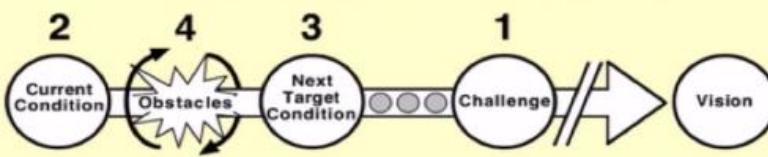
“Learning is not compulsory... neither is survival.” — Dr. W. Edward Deming

## 6.2 Las Tres Vías - Tercera

### THE IMPROVEMENT KATA PATTERN

The **Improvement Kata** is a 4-step pattern that includes practice routines in order to make striving and scientific working a daily habit

- 1 In consideration of the overarching direction or **challenge**...
- 2 Grasp the **current condition**.
- 3 Define the next **target condition**.
- 4 Move toward that target condition iteratively via experiments, which uncovers the **obstacles** that need to be worked on.



*The Improvement Kata pattern is a way of achieving things that you don't know how you are going to achieve*

© Mike Rother

TOYOTA KATA

Source: Mike Rother - Toyota Kata

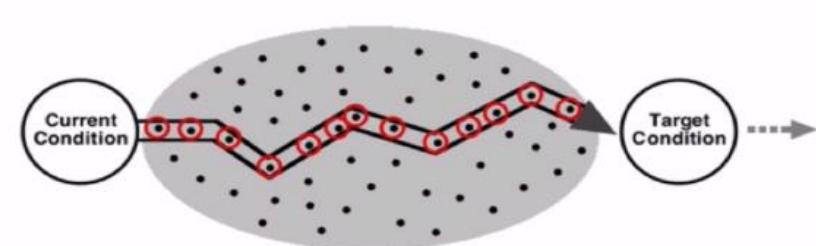


Diagram by Mr. Jeff Uitenbroek

© Mike Rother

TOYOTA KATA

Source: Mike Rother - Toyota Kata

### Los 10 puntos del Espíritu Kaizen

1. Abandonar las ideas fijas, plantearse el estado actual de las cosas.
2. EN LUGAR DE EXPLICAR LO QUE NO SE PUEDE HACER, REFLEXIONAR CÓMO HACERLO.
3. Realizar inmediatamente las buenas propuestas de mejora.
4. No buscar la perfección desde el principio, comenzar a caminar para conseguir el 60% desde ahora.
5. Corregir un error inmediatamente e in situ
6. Encontrar las ideas en la dificultad
7. Buscar la causa real, respetar los 5 "por qué" y después buscar la solución.
8. Tener en cuenta las ideas de 10 personas en lugar de esperar la idea genial de una sola.
9. PROBAR Y DESPUÉS VALIDAR.

**10. LA MEJORA ES INFINITA.**

ARCOS

## **6.1 Introducción**

## **6.2 Las Tres Vías**

**6.2.1 Primera Vía: Acelerar el flujo**

**6.2.2 Segunda Vía: Retroalimentación continua**

**6.2.3 Tercera Vía: Experimentación continua**

## **6.3 Practicando la Primera Vía**

**6.3.1 Infraestructura es código en la pipeline**

**6.3.2 Pruebas automáticas y CI**

**6.3.3 CD**

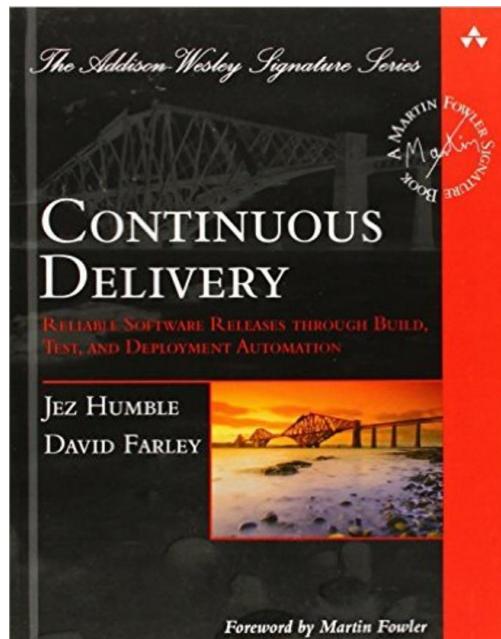
**6.3.4 Arquitectura y Despliegues**

## **6.4 Estado Actual**

## **6.5 Casos de Estudio**

## 6.3 – Practicando la Primera Vía

- El conjunto de prácticas tecnológicas llevadas a cabo para cumplir con los principios de la Primera Vía es conocido como **entrega continua** (*continuous delivery*). Para conseguir entrega continua, cumpliremos:



1. La infraestructura de nuestro pipeline es código.
  2. Pruebas continuas (y automáticas).
  3. Integración continua (CI) (y automática).
  4. Entrega continua (continuous delivery, CD).
- A lo largo de nuestro pipeline se deben tener en cuenta los **intereses de Devs, QA, InfoSec y Operaciones en conjunto**.

## **6.1 Introducción**

## **6.2 Las Tres Vías**

**6.2.1 Primera Vía: Acelerar el flujo**

**6.2.2 Segunda Vía: Retroalimentación continua**

**6.2.3 Tercera Vía: Experimentación continua**

## **6.3 Practicando la Primera Vía**

**6.3.1 Infraestructura es código en la pipeline**

**6.3.2 Pruebas automáticas y CI**

**6.3.3 CD**

**6.3.4 Arquitectura y Despliegues**

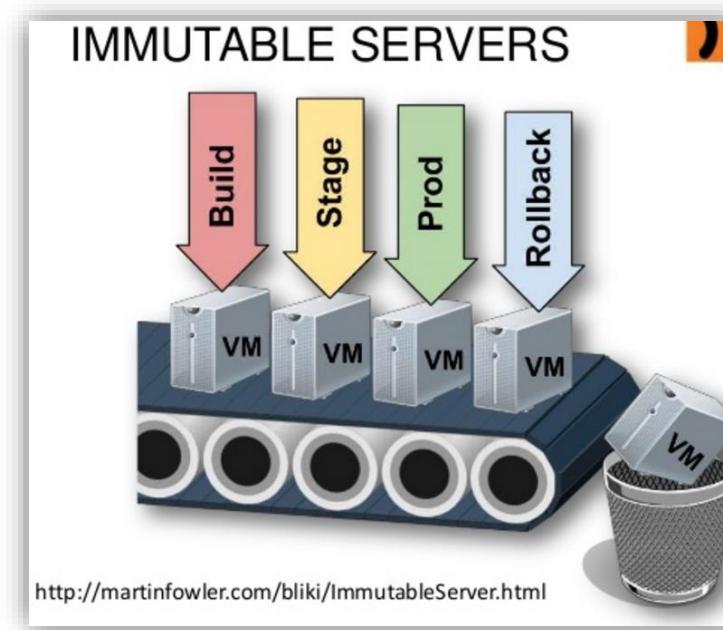
## **6.4 Estado Actual**

## **6.5 Casos de Estudio**

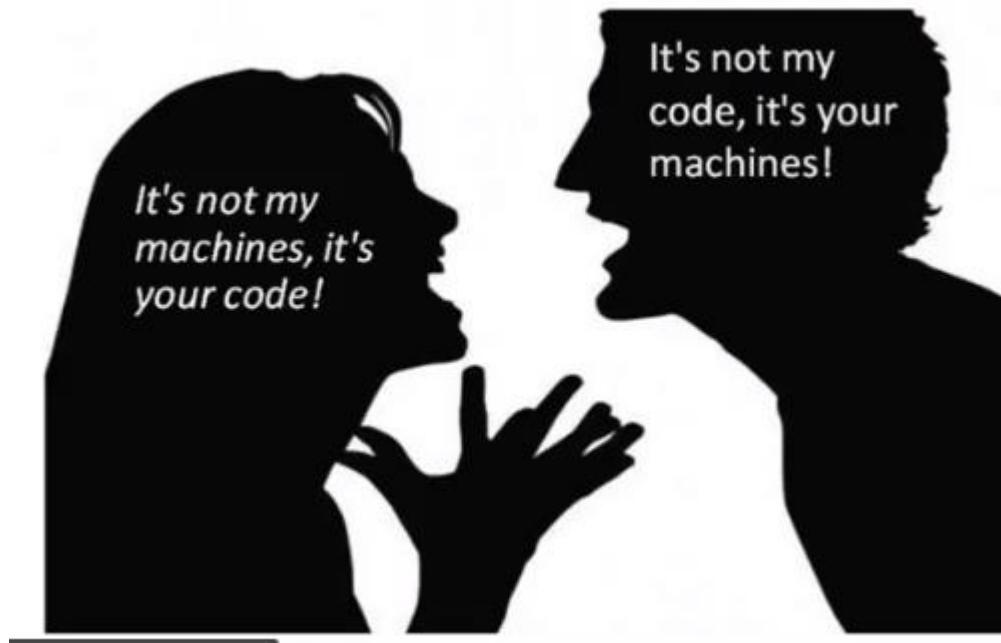
## 6.3.1 – Practicando la Primera Vía - *Pipeline*

### La infraestructura es código

- Debemos poder **generar de manera automática los entornos** de desarrollo, test y producción.
- Así, los desarrolladores pueden hacer pruebas automáticas de su código simulando las fases venideras de la *pipeline* hasta despliegue en producción.
- Los **scripts** de creación de entornos deben estar también en nuestro **repositorio** de control de versiones.



Reduciremos situaciones como ésta:



*Machine Learning to Turbo Charge the Ops Portion of DevOps*

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

## 6.3.1 –Practicando la Primera Vía - *Pipeline*

- Herramientas para mantener código, scripts y archivos de configuración en control de versiones:



- Herramientas para configurar infraestructuras lanzando scripts:



- Contenedor de entornos virtuales inmutables:



- Crear entornos en servicios en la nube:



## 6.3.1 –Practicando la Primera Vía - *Pipeline*

---

- Todo el personal involucrado en la pipeline debe tener **acceso al repositorio** en control de versiones.
- Para asegurar que podemos restablecernos de cualquier evento catastrófico, estos deben ser los elementos que contenga nuestro **repositorio de la verdad**:
  - Código y dependencias.
  - Herramientas para creación de entornos.
  - Archivos de creación de contenedores.
  - Scripts para creación y migración de bases de datos.
  - Scripts de tests automáticos.
  - Scripts de empaquetado y despliegue.
  - Todos los artefactos del Proyecto (requisitos, procedimientos,...)
  - Archivos de configuración de la nube.
  - Reglas de configuración para cortafuegos, DNS,...

## 6.3.1 –Practicando la Primera Vía - *Pipeline*

- El uso de **control de versiones** en Ops es mejor **predictor del desempeño** de una empresa que su uso en Dev → porque en Ops hay una cantidad mucho mayor de configuraciones que en Devs.
- **Mascotas Vs. Ganado** (*Bill Baker, Microsoft*): ya no hace falta sanar a nuestros servidores como si fueran mascotas hasta su muerte, ahora podemos tratarlos como Ganado: si enferma se elimina y se reconstruye rápidamente de manera automática.

**Service Model**

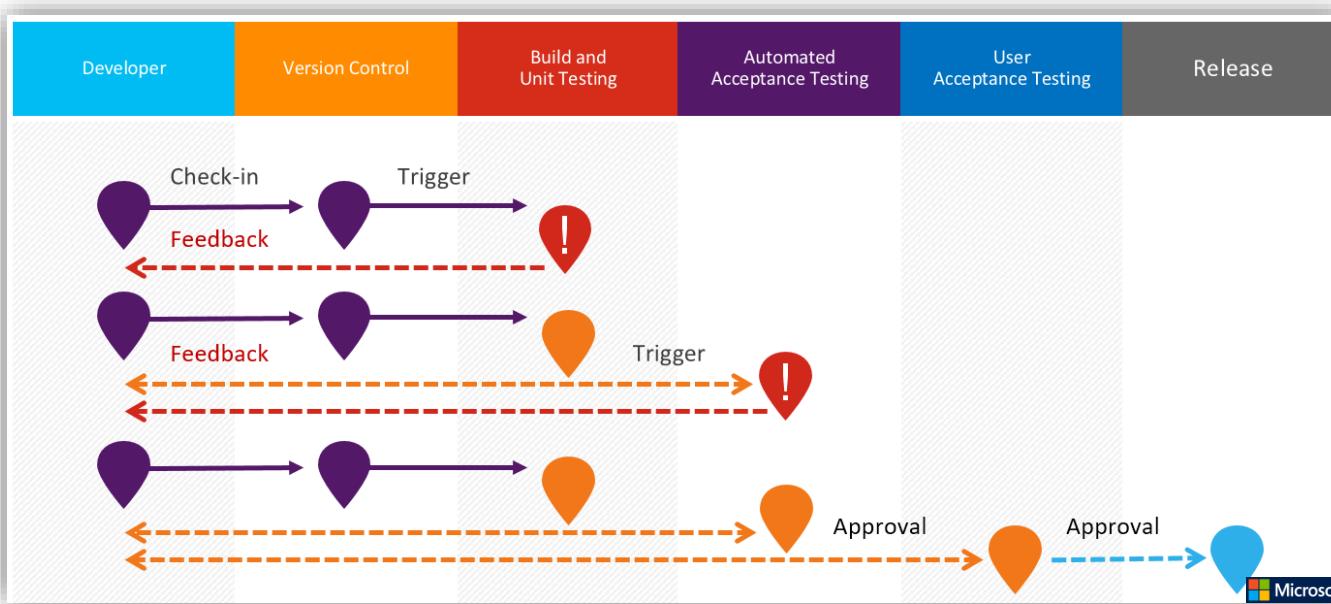
The slide features two images: a close-up of a ginger cat's face at the top, and a group of cows in a field below it. To the right of each image is a bulleted list of characteristics:

- Pets (Cat):**
  - Pets are given names like `pussinboots.cern.ch`
  - They are unique, lovingly hand raised and cared for
  - When they get ill, you nurse them back to health
- Cattle (Cows):**
  - Cattle are given numbers like `vm0042.cern.ch`
  - They are almost identical to other cattle
  - When they get ill, you get another one

*Gavin McCance, CERN*

## 6.3.1 – Practicando la Primera Vía - *Pipeline*

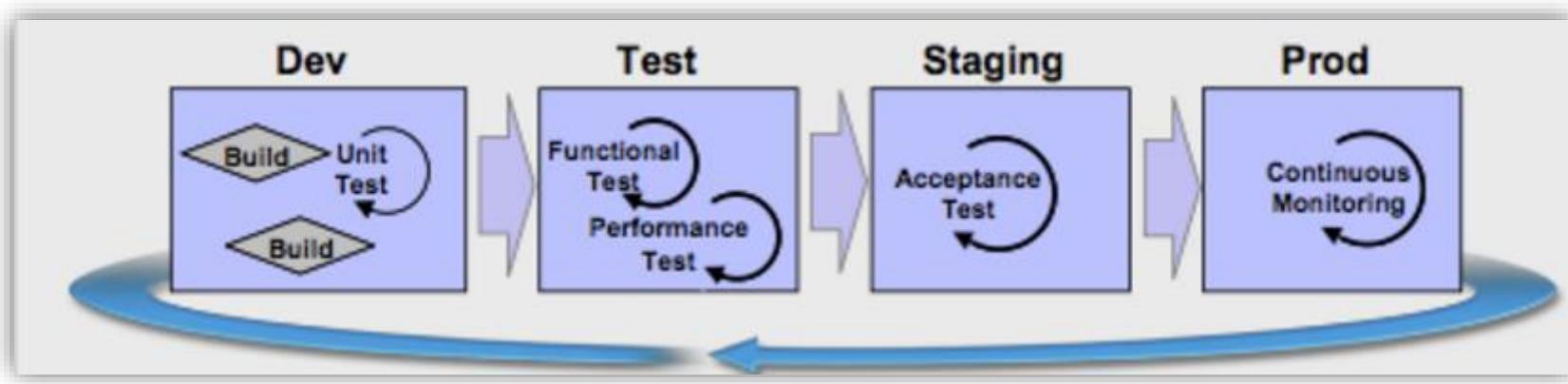
- Así, debería quedarnos una pipeline donde se pueda automatizar todo lo posible (depende de la empresa, proyecto,...), siendo el **disparador** el paso con éxito del código en la fase anterior. Ej: al hacer un build con éxito, se disparan los test unitarios.



- Cuando se requiera aprobación de una persona para pasar a la siguiente fase, se conoce como “**Puerta manual**”

## 6.3.1 – Practicando la Primera Vía - *Pipeline*

- En los **diferentes entornos** se testeaa el código de diferente manera:



## **6.1 Introducción**

## **6.2 Las Tres Vías**

**6.2.1 Primera Vía: Acelerar el flujo**

**6.2.2 Segunda Vía: Retroalimentación continua**

**6.2.3 Tercera Vía: Experimentación continua**

## **6.3 Practicando la Primera Vía**

**6.3.1 Infraestructura es código en la pipeline**

**6.3.2 Pruebas automáticas y CI**

**6.3.3 CD**

**6.3.4 Arquitectura y Despliegues**

## **6.4 Estado Actual**

## **6.5 Casos de Estudio**

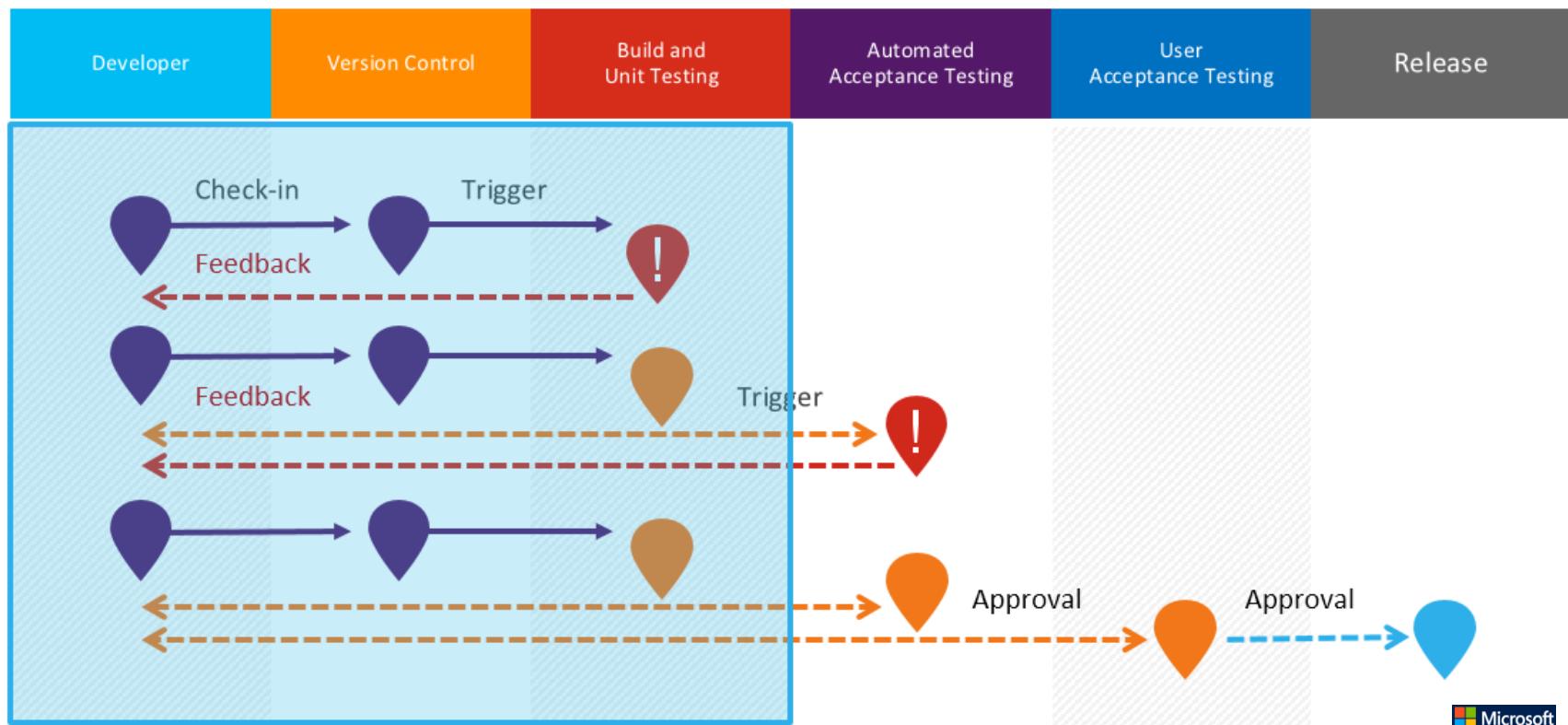
## 6.3.2 – Practicando la Primera Vía – *Pruebas continuas y CI*

*En 2005, cada despliegue a Google.com era tremadamente problemático, apareciendo conflictos de cambios entre equipos y bugs. Desde 2013, cada push lanza automáticamente miles de tests. Esto, junto a integración continua, permite trabajar a miles de pequeños equipos en Google sobre un repositorio compartido.*

- **Pruebas continuas:**
  - Ejecutar tests de manera repetida en entornos que simulen el de producción.
  - Los tests unitarios pueden realizarse de manera rápida y automática.
- **Integración Continua (CI):**
  - Code + tests unitarios en local → commit & push → build + tests unitarios

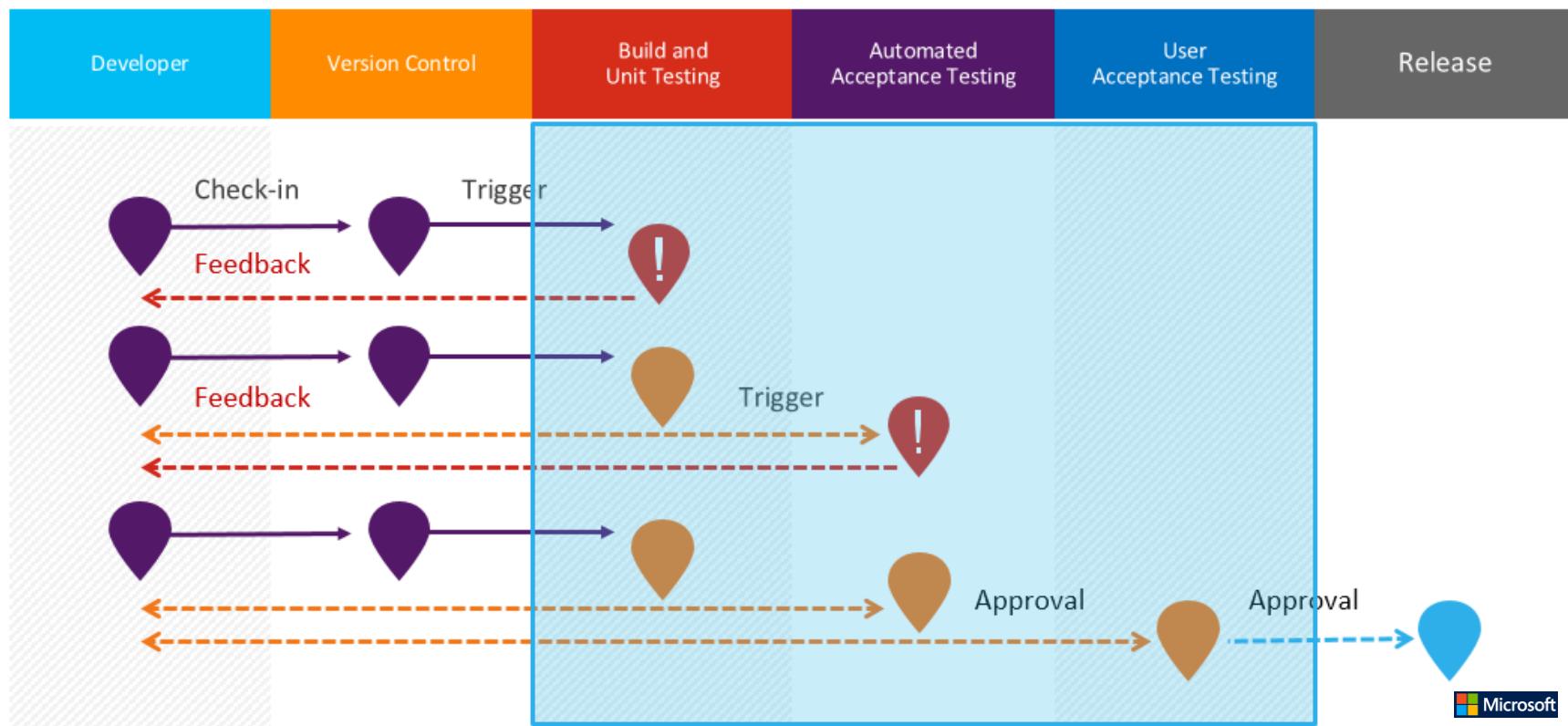
## 6.3.2 – Practicando la Primera Vía – *Pruebas continuas y CI*

### Continuous integration



## 6.3.2 – Practicando la Primera Vía – *Pruebas continuas y CI*

### Continuous testing



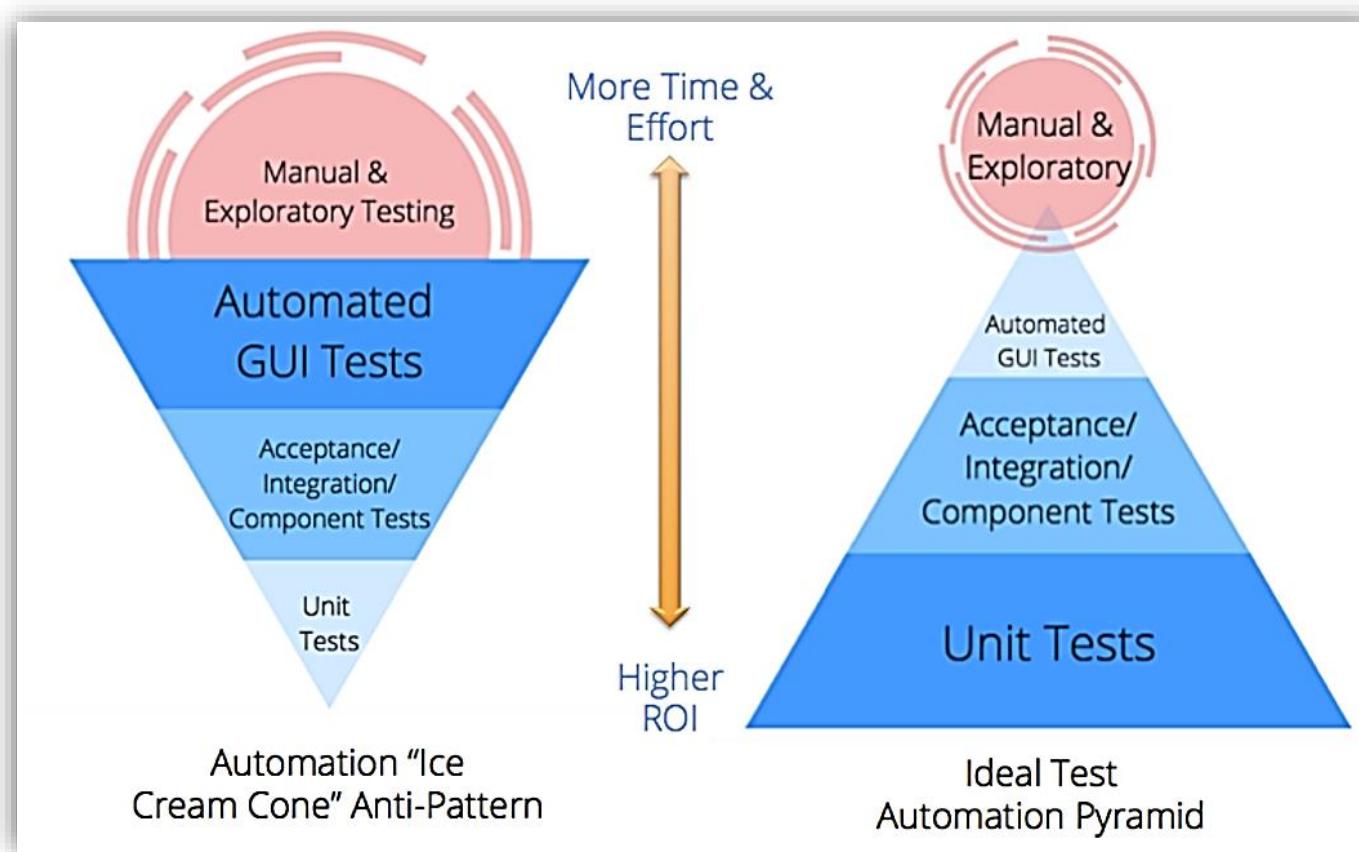
## 6.3.2 – Practicando la Primera Vía – *Pruebas continuas y CI*

Hay varios **tipos de pruebas**, no todas pueden ejecutarse de manera automática.

- **Tests Unitarios**: prueban un método, clase o función aislados del resto.
- **Tests de aceptación**: prueban que la aplicación realiza lo que el cliente quiere, no que haga lo que el programador piensa que debería.
- **Tests de integración**: prueba entre módulos de la aplicación, o con otras aplicaciones, servicios, bases de datos, ...
- **Tests de interfaz realizados por el usuario**: intentar que falle la aplicación.
- Cuando se detecte un error más allá de los tests unitarios, hay que **crear un test unitario capaz de detectarlo** para que no vuelva a extenderse a fases más avanzadas.
- Automatizar todas las pruebas posibles.
- Kent Beck (XP): escribir tests automáticos antes de escribir Código (TDD).

## 6.3.2 – Practicando la Primera Vía – *Pruebas continuas y CI*

- Pirámides de pruebas automáticas (*Martin Fowler – TestyPyramid*): lo ideal es encontrar la **mayoría de errores con tests unitarios** ya que son menos costosos en dinero, tiempo de ejecución y tiempo de vuelta atrás.



## 6.3.2 – Practicando la Primera Vía – *Pruebas continuas y CI*

- Para asegurar el éxito de las pruebas y CI:
  - Reducir al máximo el tamaño de los *batch* de características..
  - **Desarrollo basado en trunk y puertas:** como mucho abrir una rama para hacer un cambio y se lanzan los tests automáticamente para aceptar el cambio en trunk, con puerta manual o automática.
- **Si alguien rompe** el código o entornos de la *pipeline*: cable de Anton!
- El pipeline debe contener siempre código en **estado desplegable**
- “*Sin pruebas automáticas, CI sería la manera más rápida de obtener una pila de basura que nunca compile o ejecute correctamente.*” – Gary Gruver (HP Laserjet Firmware)
- **Hecho (Done):** “*Tras cada intervalo de desarrollo, el código debe estar integrado, probado y funcionando en entornos que simulen producción, y dicho Código debe haber sido añadido al trunk principal mediante un proceso de 1-click, y validado por tests automáticos.*”

Y cuando por fin conseguimos que nuestro pipeline de CI ejecute con éxito...

<https://twitter.com/olemoudi/status/1255206061159714819>

## **6.1 Introducción**

## **6.2 Las Tres Vías**

**6.2.1 Primera Vía: Acelerar el flujo**

**6.2.2 Segunda Vía: Retroalimentación continua**

**6.2.3 Tercera Vía: Experimentación continua**

## **6.3 Practicando la Primera Vía**

**6.3.1 Infraestructura es código en la pipeline**

**6.3.2 Pruebas automáticas y CI**

**6.3.3 CD**

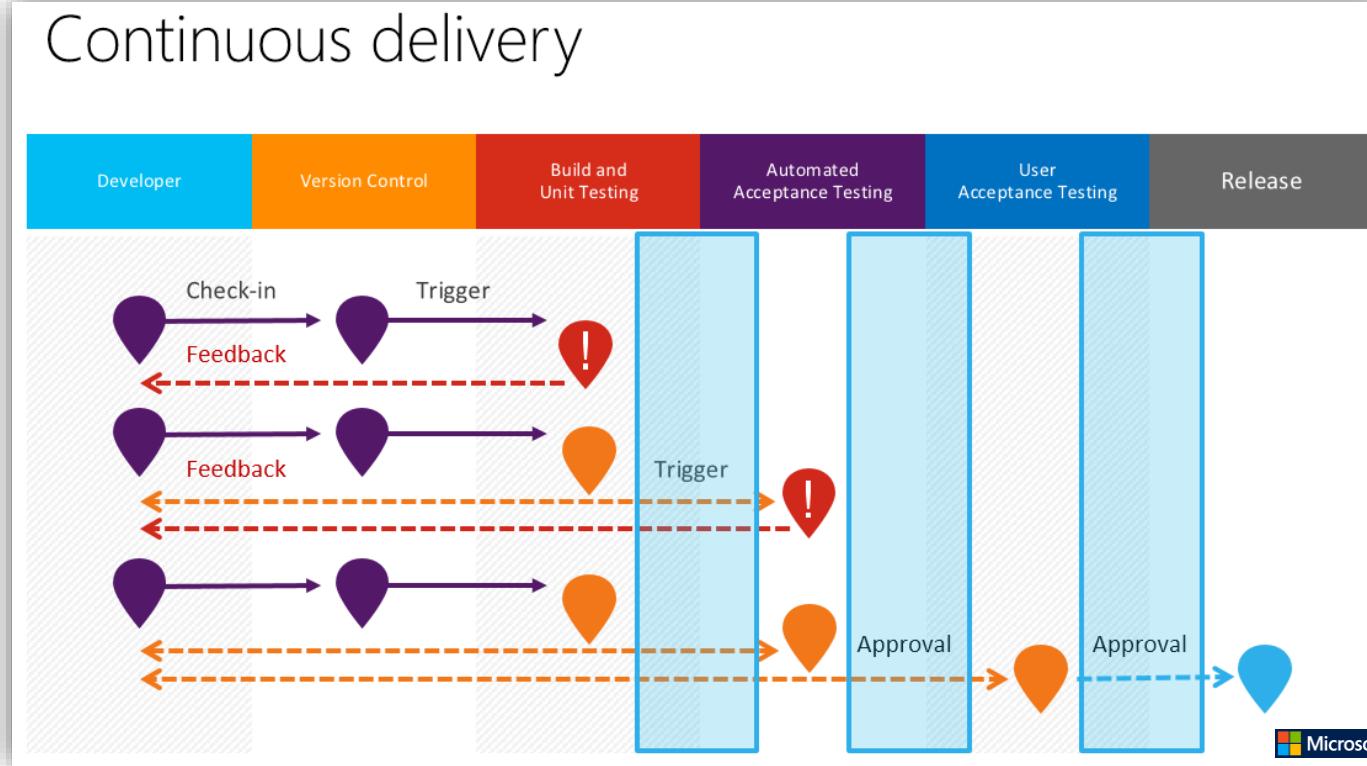
**6.3.4 Arquitectura y Despliegues**

## **6.4 Estado Actual**

## **6.5 Casos de Estudio**

### 6.3.3 – Practicando la Primera Vía – CD

- **Entrega Continua (CD):** desarrollo de software en ciclos cortos, con pruebas automáticas y actualizando Producción, de manera que el código pueda ser liberado (*released*) en cualquier momento.
- Si el Código pasa por toda la pipeline con éxito de manera regular (incluso con puertas automáticas), entonces es **Despliegue Continuo** (Continuous Deployment).



## 6.3.3 – Practicando la Primera Vía – CD

- **Comutación de características (feature toggle/flag):**



Feature Flags

- Funcionalidad integrada y entregada en producción, que puede dejarse escondida o activada según se deseé.
- Sea por testeo o por experimentación, se puede hacer disponible solo para un sector de los usuarios y así comprobar su eficacia.

*Con despliegue automático bajo demanda, la velocidad de entrega de nuevas características a los clientes es tan solo una decisión de negocio en vez de técnica.*

## 6.3.3 – Practicando la Primera Vía – CD

- Hay que tener cuidado con la gestión de los *feature flags*...

Si has descargado **la actualización Windows 10 November 2019 Update** quizás te hayas dado cuenta de que **solo ocupa... 180 KB**. ¿Cómo puede ser que una de las actualizaciones de Windows más importantes del año solo ocupe como un fichero de texto? Microsoft lo ha aclarado, y reconoce también que no ha gustado a los usuarios, así que no volverá a hacerlo en el futuro.

Lo que hace la pequeña descarga de 180 KB de Windows 10 November 2019 es *activar un switch* que desbloquea estas novedades si ya tienes la actualización de mayo en tu PC, u obliga a descargar Windows 10 May 2019, para poder activarlas. Esa es la razón de que muchos usuarios hayan detectado **actualizaciones automáticas de Windows 10**.

Microsoft reconoce que este nuevo sistema, que ha bautizado como "*programa piloto*", no ha gustado a los *Insiders* ni a los usuarios, y no lo volverá a utilizar en futuras actualizaciones de Windows 10.

<https://computerhoy.com/tutoriales/tecnologia/microsoft-windows-10-november-2019-update-solo-ocupa-180-kb-no-volvera-pasar-535497?amp>

## **6.1 Introducción**

## **6.2 Las Tres Vías**

### **6.2.1 Primera Vía: Acelerar el flujo**

### **6.2.2 Segunda Vía: Retroalimentación continua**

### **6.2.3 Tercera Vía: Experimentación continua**

## **6.3 Practicando la Primera Vía**

### **6.3.1 Infraestructura es código en la pipeline**

### **6.3.2 Pruebas automáticas y CI**

### **6.3.3 CD**

### **6.3.4 Arquitectura y Despliegues**

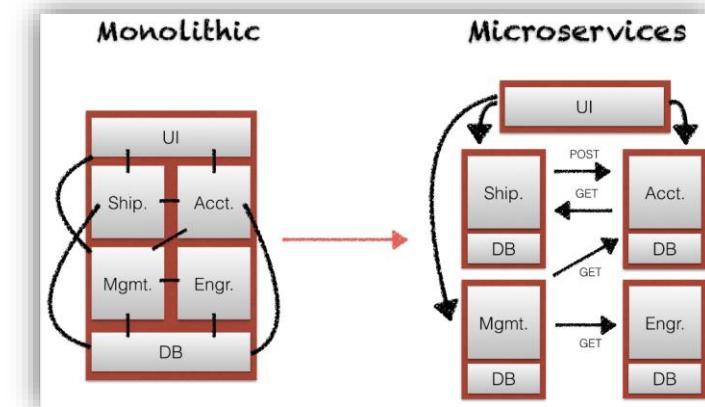
## **6.4 Estado Actual**

## **6.5 Casos de Estudio**

## 6.3.4 – Practicando la Primera Vía - Arquitectura

- Evolucionar a DevOps no significa ser escalable. Al ir añadiendo funcionalidad, solo escalar a nivel de hardware no es suficiente.
- Una start-up puede beneficiarse de una arquitectura **monolítica**, mientras que al contra con decenas de desarrolladores y un producto con muchas funcionalidades, es mejor migrar a arquitectura de **microservicios**.

	Pros	Contras
Monolítica	Simple, baja latencia	Coordinación del equipo más difícil, despliegue <i>todo o nada</i> , poco escalable
Microservicios	Unidades simples, tests y despliegues independientes, buen escalado	Muchas unidades cooperando, herramientas sofisticadas de gestión, latencias de red



## 6.3.4 – Practicando la Primera Vía - Despliegues

- **Gradual** (Rolling upgrade): un servidor tras otro. Reducción temporal de la capacidad del servicio.
- **Canario**: como el anterior, pero en subconjuntos de servidores y con varias pruebas.
- **Gradual por fases**: dividir el despliegue por grupos de usuarios.
- **Azul-Verde (Blue-Green)**: las 2 versiones de la aplicación conviven (verde, funcionando), y se comutan para hacer comprobaciones hasta que se deja la nueva.
- **Comutación de características** (feature toggle, feature flag).



## **6.1 Introducción**

## **6.2 Las Tres Vías**

**6.2.1 Primera Vía: Acelerar el flujo**

**6.2.2 Segunda Vía: Retroalimentación continua**

**6.2.3 Tercera Vía: Experimentación continua**

## **6.3 Practicando la Primera Vía**

**6.3.1 Infraestructura es código en la pipeline**

**6.3.2 Pruebas automáticas y CI**

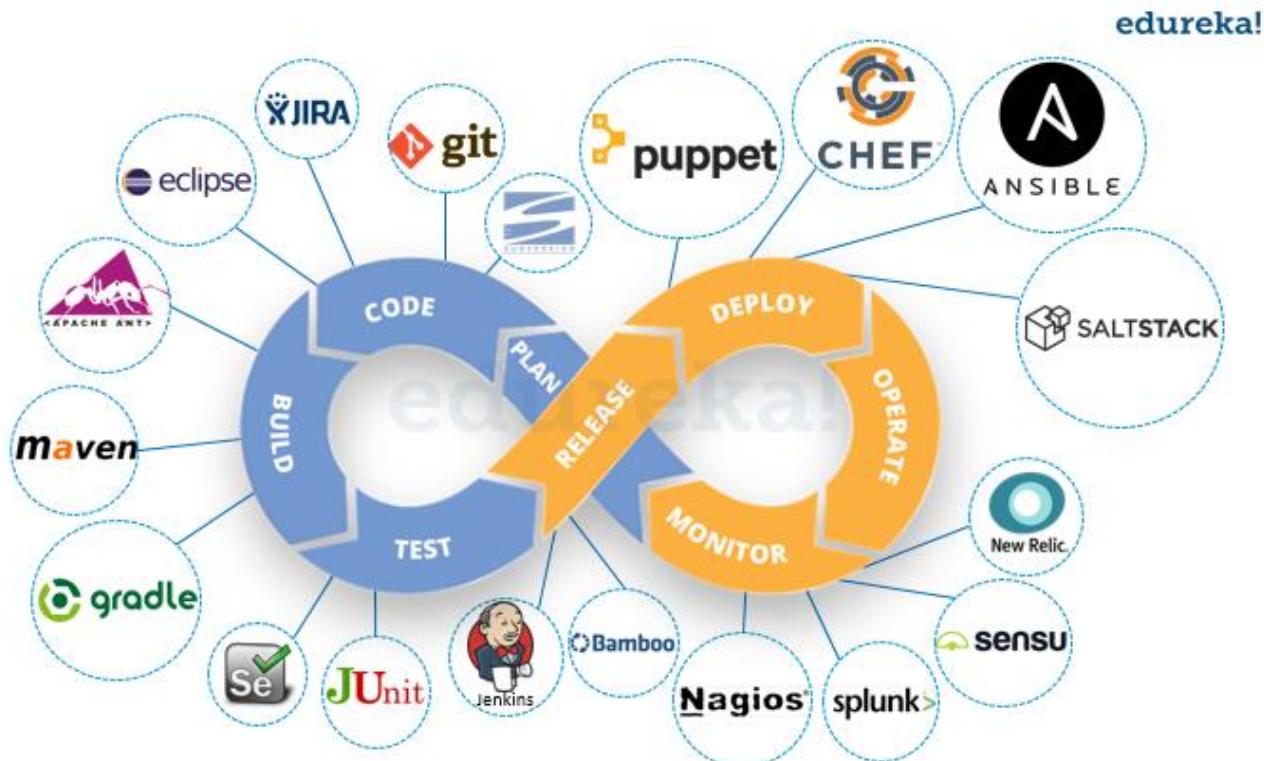
**6.3.3 CD**

**6.3.4 Arquitectura y Despliegues**

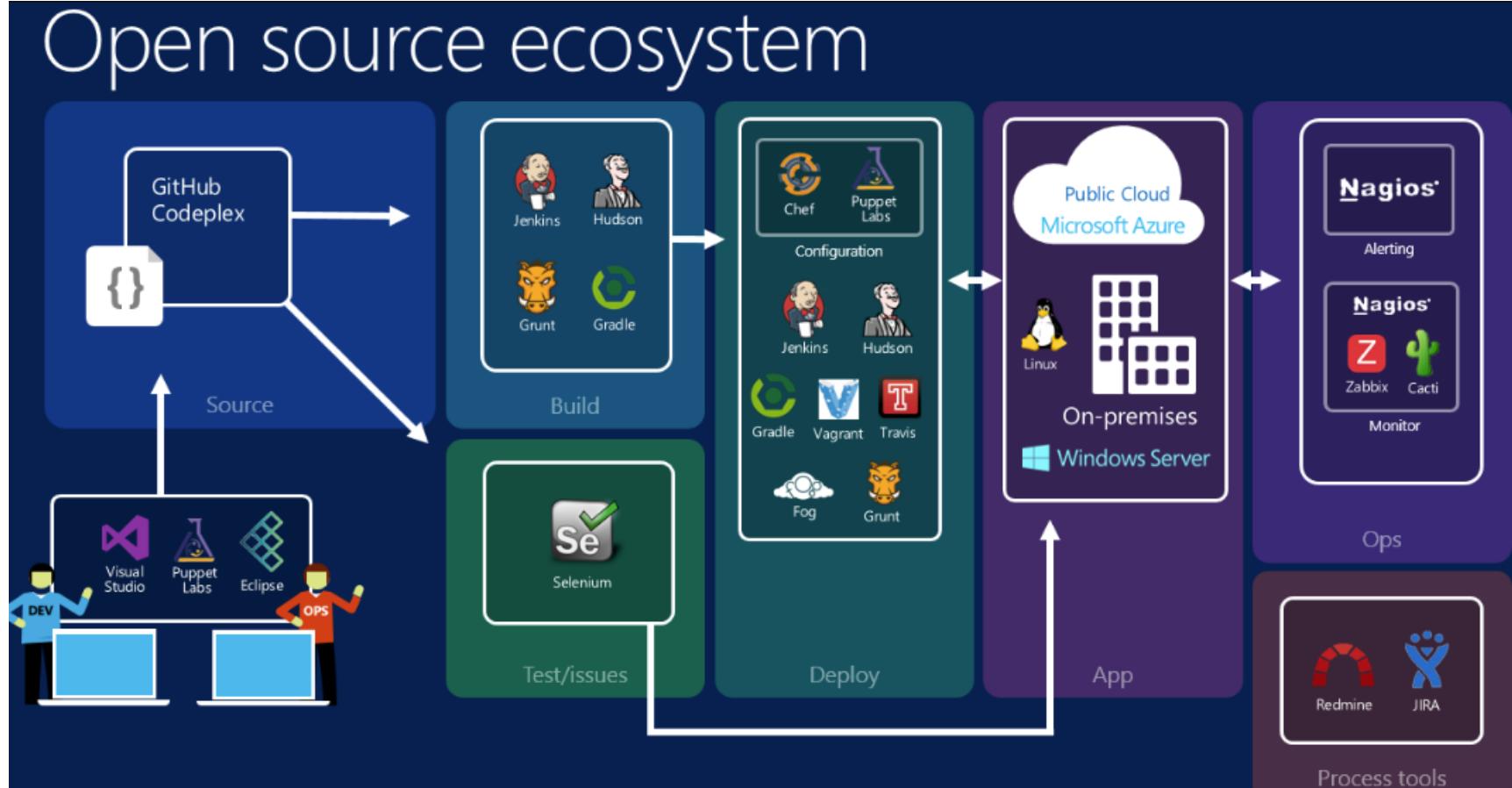
## **6.4 Estado Actual**

## **6.5 Casos de Estudio**

## 6.4 – Estado actual



## 6.4 – Estado actual



<https://www.hcltech.com>

# 6.4 – Estado actual

- Tabla periódica interactiva (v4.2 2020) de herramientas para implementar técnicas de DevOps.

**The Periodic Table of DevOps Tools (V4.2)**

The Periodic Table of DevOps Tools (V4.2) is a comprehensive guide to DevOps tools, organized into a grid where each cell represents a specific tool. The grid is divided into several sections based on tool type:

- Period 1:** Aja (Atlassian Jira Align), Daa (Digital Agility), Pv (Planview), In (Instana), Sp (Splunk), Dt (Dynatrace), Gr (Grafana).
- Period 2:** Daa (Digital Agility), Tp (Targetprocess), Br (Broadcom Rally), Dd (Datadog), Ja (JFrog Artifactory), Aws (AWS), Sl (Slack), Mt (Microsoft Teams), Rha (Red Hat Ansible), Ht (HashiCorp Terraform), Dk (Docker), Rho (Red Hat OpenShift), Lb (Liquibase), Dp (Delphix), Ud (UrbanCode Deploy), Ck (CyberArk Conjur), Hv (HashiCorp Vault), Ur (UrbanCode Release), Al (AWS Lambda), Abb (Atlassian Bitbucket).
- Period 3:** Spn (Splunk AppDynamics), Ad (AppDynamics), Snx (Sonatype Nexus), Az (Azure), Gc (Google Cloud), Ac (Atlassian Confluence), Ch (Chef), Acf (AWS CloudFormation), Ku (Kubernetes), Ak (Amazon EKS), De (Dockership Enterprise), Id (IDERA), Ha (Harness), Vc (Veracode), Sr (SonarQube), Ff (Micro Focus Fortify SCA), Azf (Azure Functions), Ci (Compuware ISPW).
- Period 4:** Dt (Dynatrace), Nr (New Relic), Dh (Docker Hub), Np (npm), Ic (IBM Cloud), So (Stack Overflow), Pu (Puppet), Hc (HashiCorp Consul), Ae (Amazon ECS), Azk (Amazon AKS), Ra (Rancher), Qt (Quest Toad), Sk (Spinnaker), Od (Octopus Deploy), Sb (Synopsys Black Duck), Cx (Checkmarx SAST), He (Heroku), Sv (Subversion).
- Period 5:** Gr (Grafana), El (Elastic ELK Stack), Yn (Yarn), Nu (NuGet), Os (OpenStack), Mm (Mattermost), Sa (Salt), Hg (HashiCorp Vagrant), Hp (HashiCorp Packer), Gk (Google GKE), Hm (Helm), Db (DBmaestro), Cfd (CloudBees Flow), Acd (AWS CodeDeploy), Sn (Short), Pbs (PortSwigger Burp Suite), Gf (Google Firebase), Cf (Cloud Foundry).
- Period 6:** Jn (Jenkins), Azc (Azure DevOps Code), Glc (GitLab CI), Tr (Travis CI), Cc (CircleCI), Mv (Maven), Ab (Atlassian Bamboo), Gd (Gradle), Acb (AWS CodeBuild), Aj (Atlassian Jira), Bi (BMC Helix BPM), At (Atlassian Jira), Sw (ServiceNow), Td (Toddesk), Pd (PagerDuty).
- Period 7:** Tt (Tricentis Tosca), Nn (Neotys NeoLoad), Se (Selenium), Ju (JUnit), Sl (Sauce Labs), Ct (Compuware Topaz), Ap (Appium), Sq (Squash TM), Cu (Cucumber), Jm (JMeter), Pa (Parasoft), Dai (Digital.ai), Tp (Tasktop), Pr (Plutora), Gl (GitLab).

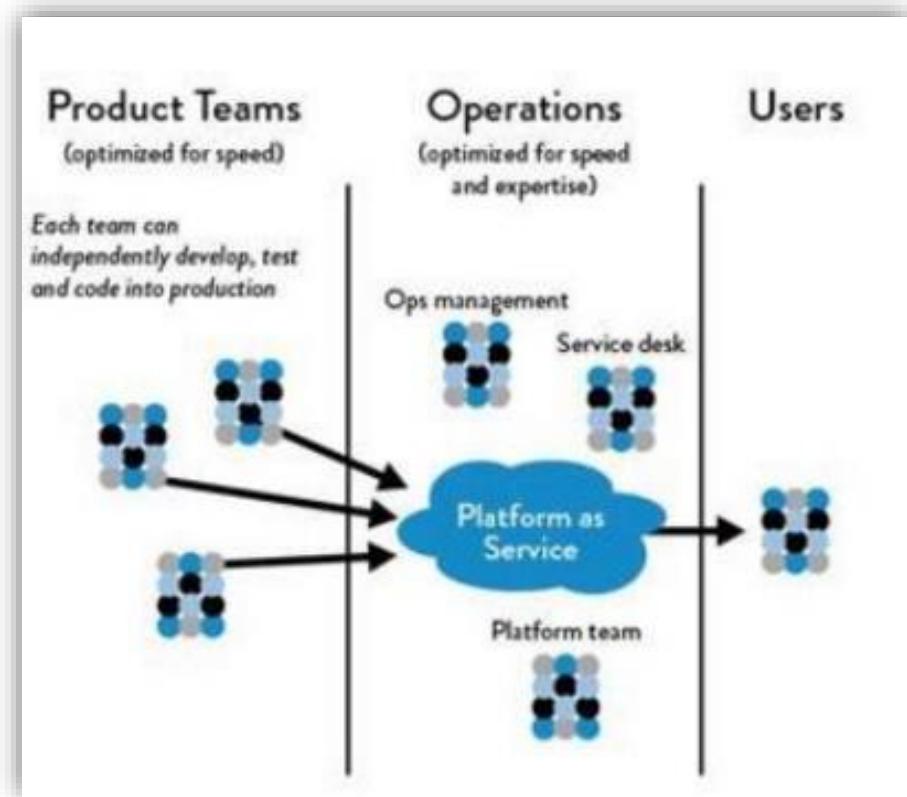
Each cell contains the name of the tool, its primary function, and its availability status (Open Source, Free, Freemium, Paid, Enterprise). The periodic table also includes a legend for tool categories and a footer with the digital.ai logo and copyright information.

<https://digital.ai/periodic-table-of-devops-tools>

## 6.4 – Estado actual

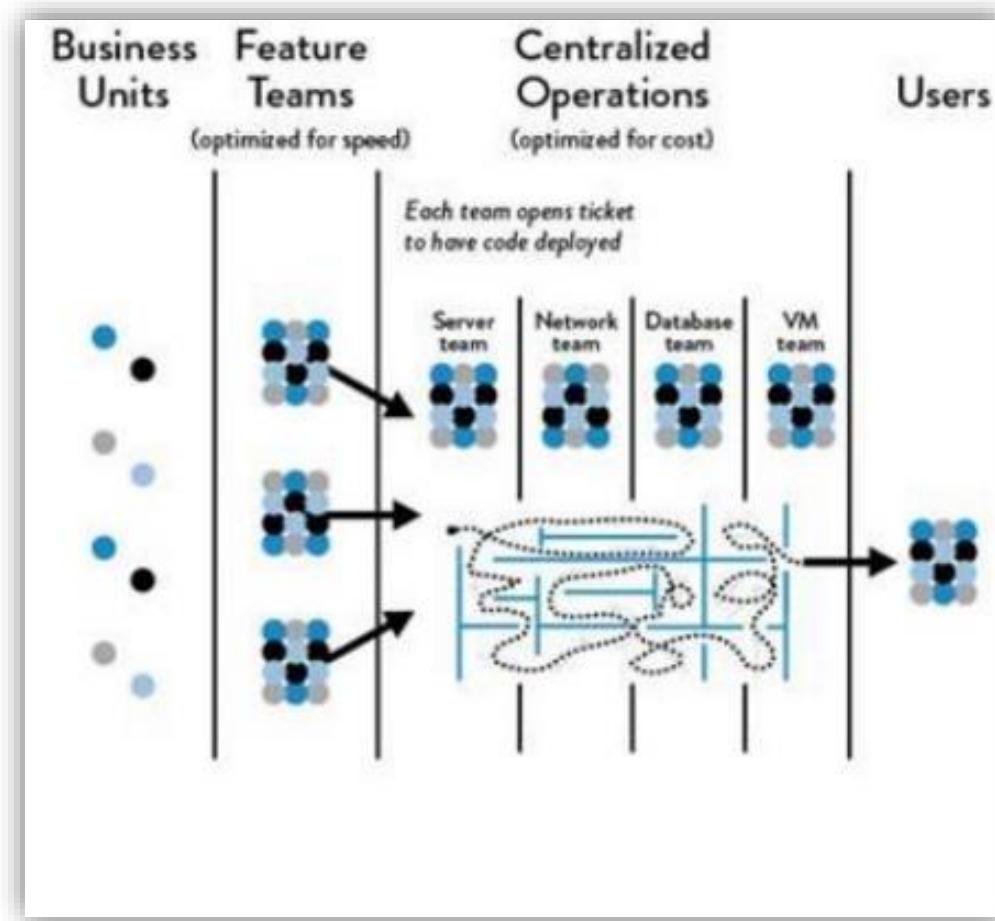
Según los arquetipos de organización empresarial vistos en el Tema 5...

- La empresa **orientada a proyectos** o mercado es el tipo de empresa que más comúnmente adopta **DevOps**, como Amazon o Netflix, donde cada equipo puede llegar a ser responsable de sus características y de mantener el servicio.
- Se integran a QA, Ops e InfoSec en un mismo equipo de desarrollo. No tienen por qué pertenecer al equipo, pero sí asistirlo de vez en cuando o de manera automática.



## 6.4 – Estado actual

- La empresa **funcional** permite el desarrollo en profundidad de habilidades, y es la organización típica que contiene a **Operaciones** en diferentes departamentos.
- En Ops hay gran cantidad de intercambios (*handoffs*), cuellos de botella, largos lead times, problemas de calidad...
- Ops sirve a cada equipo de Desarrollo, para los cuales su proyecto es el prioritario.



## 6.4 – Estado actual

- Una empresa funcional también puede adoptar con éxito DevOps, siempre que todos aquellos por los que circula el flujo de valor vean los objetivos del cliente y la empresa como objetivos globales, independientemente de a qué lugar de la empresa pertenezcan.
- Para ello, debe crearse una cultura de confianza y trabajo en grupo, además de automatizar gran parte de la pipeline.
- Google, Etsy y GitHub mantienen a Ops en una estructura funcional, con gran éxito.

*“What is decisive is not the form of your organization, but how people act and react.”*  
Mike Rother - Toyota Kata.

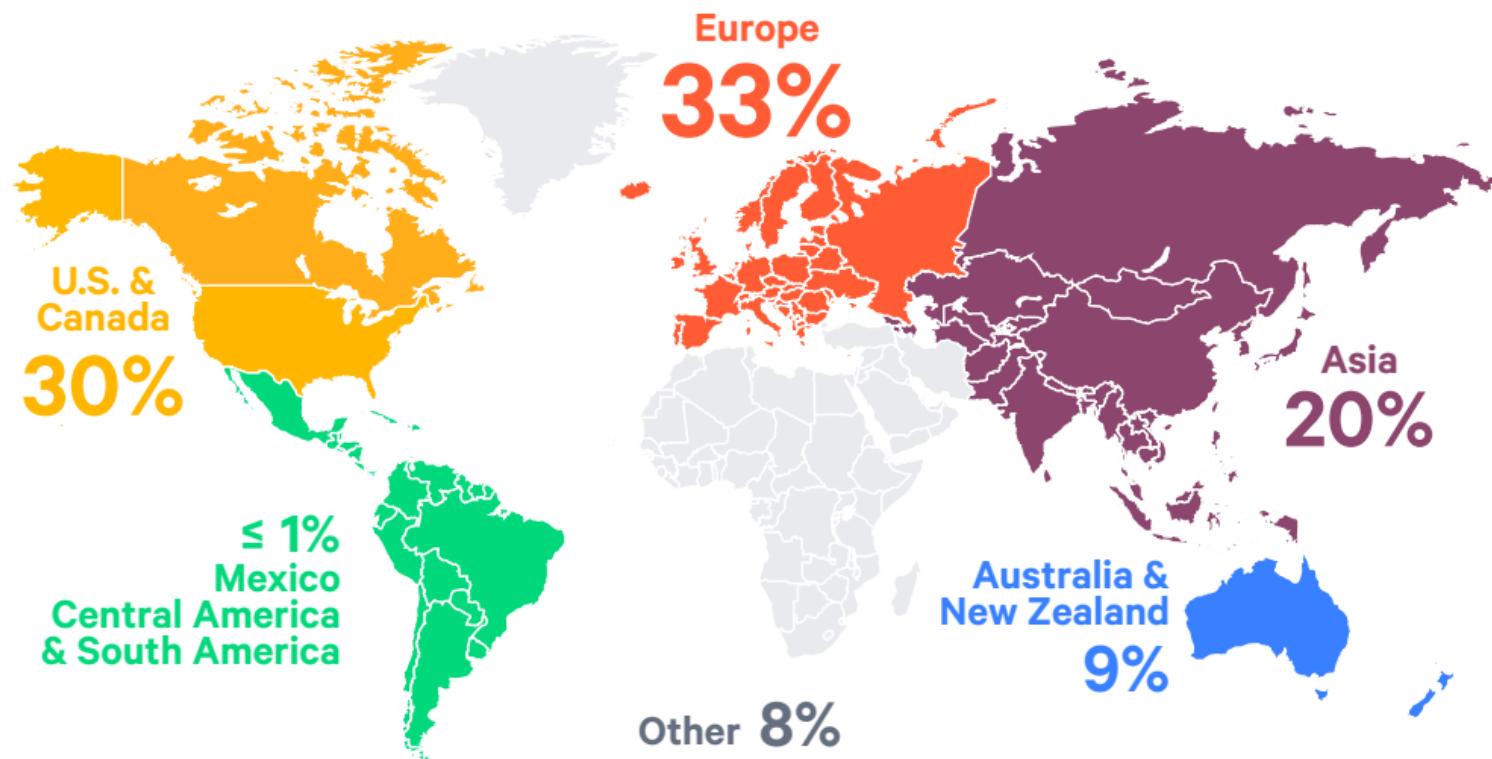
2020

# State of DevOps Report

## 6.4 – Estado actual

(2400 respuestas)

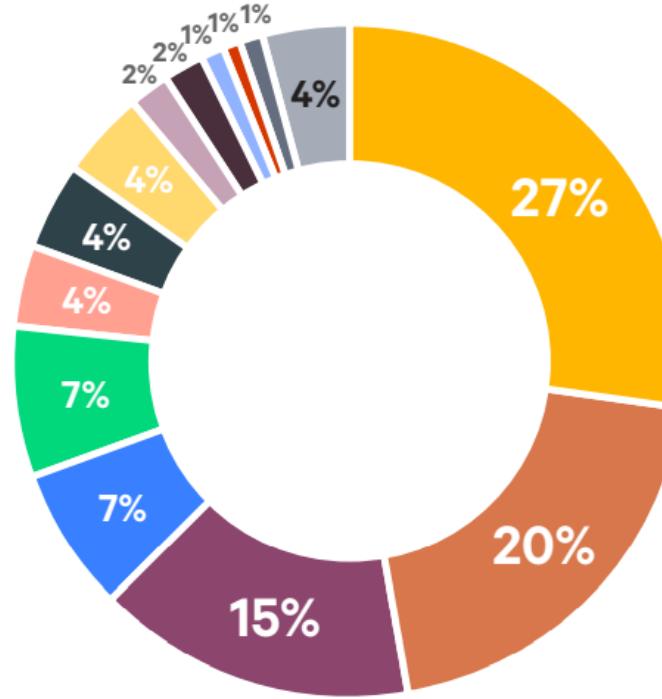
Responses by global region



## 6.4 – Estado actual

Team

- Application development
- DevOps
- Infrastructure / IT operations
- Cloud
- Information security / security
- Application security
- Site reliability engineering
- Platform engineering
- Quality assurance / quality engineering
- Compliance & audit
- Network operations
- Release engineering
- IT (help desk, AV, office support)
- Other



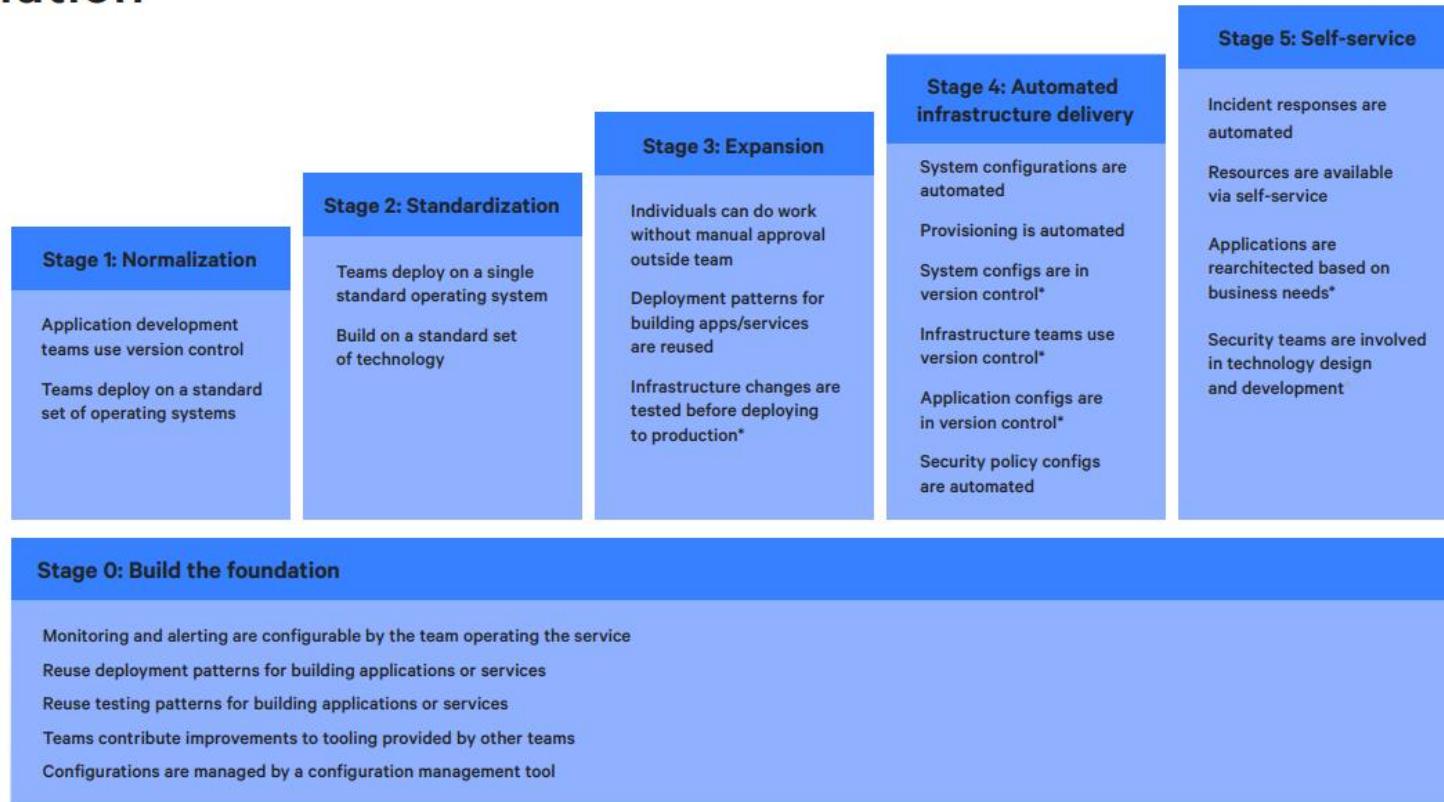
DevOps

2020

# State of DevOps Report

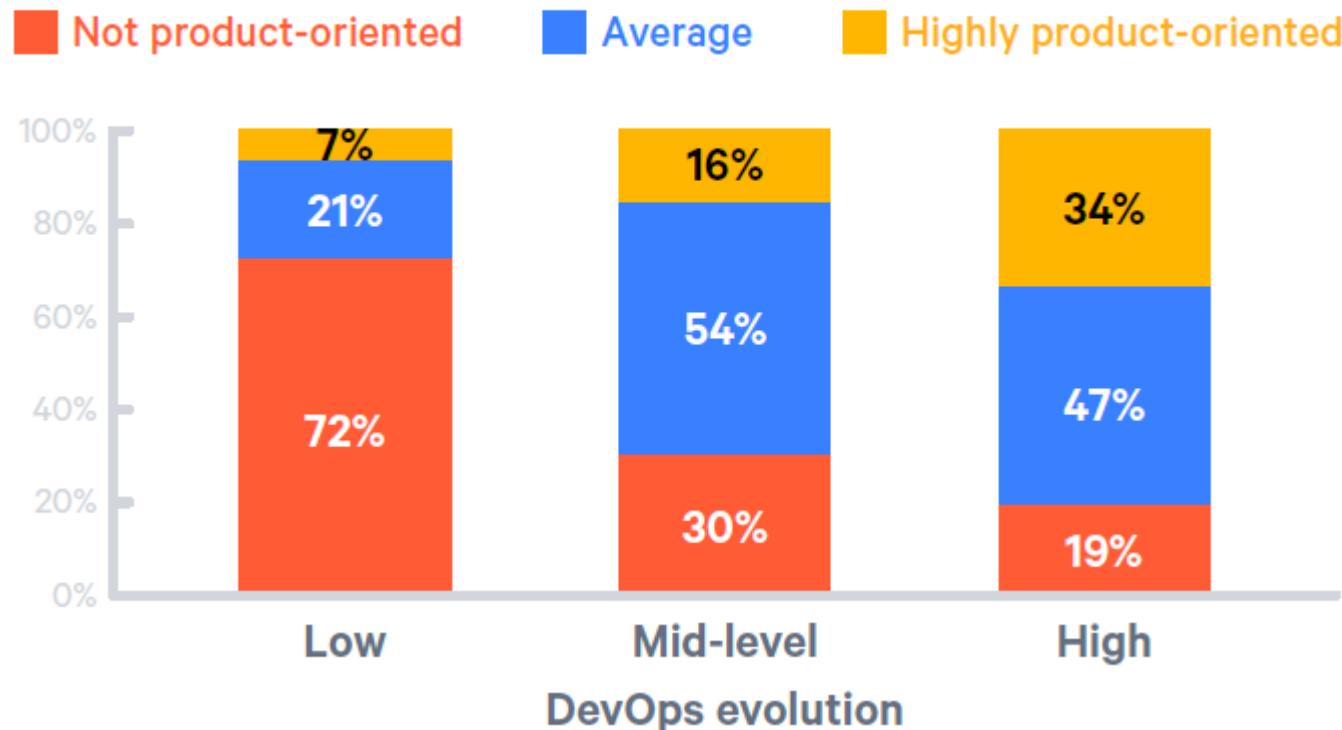
## 6.4 – Estado actual

### 5 Stages of DevOps evolution



## 6.4 – Estado actual

### DevOps evolution and platform team behavior

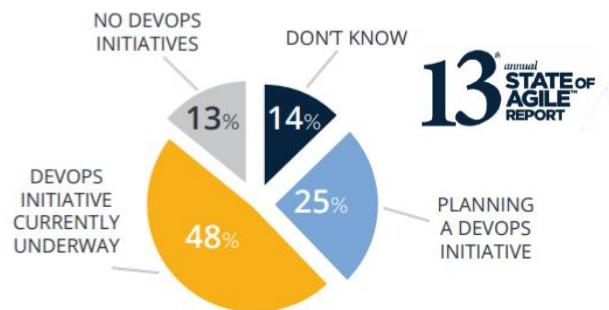
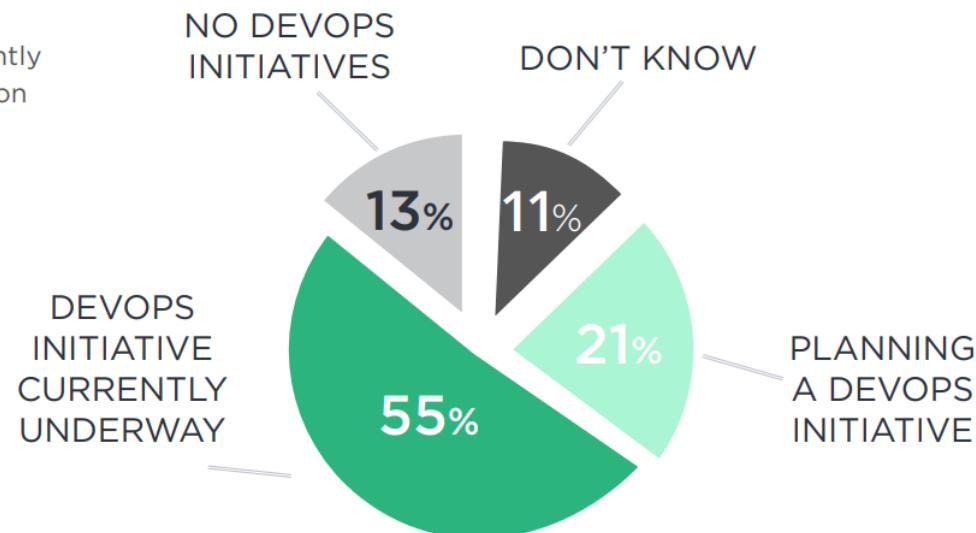


## 6.4 – Estado actual

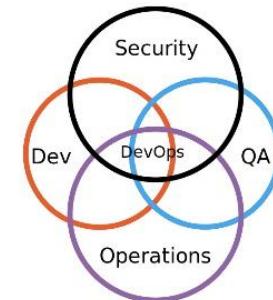
14<sup>th</sup> annual STATE OF AGILE REPORT

### DEVOPS INITIATIVES

76% of respondents stated that they currently have a DevOps initiative in their organization or are planning one in the next 12 months (compared to 73% last year).

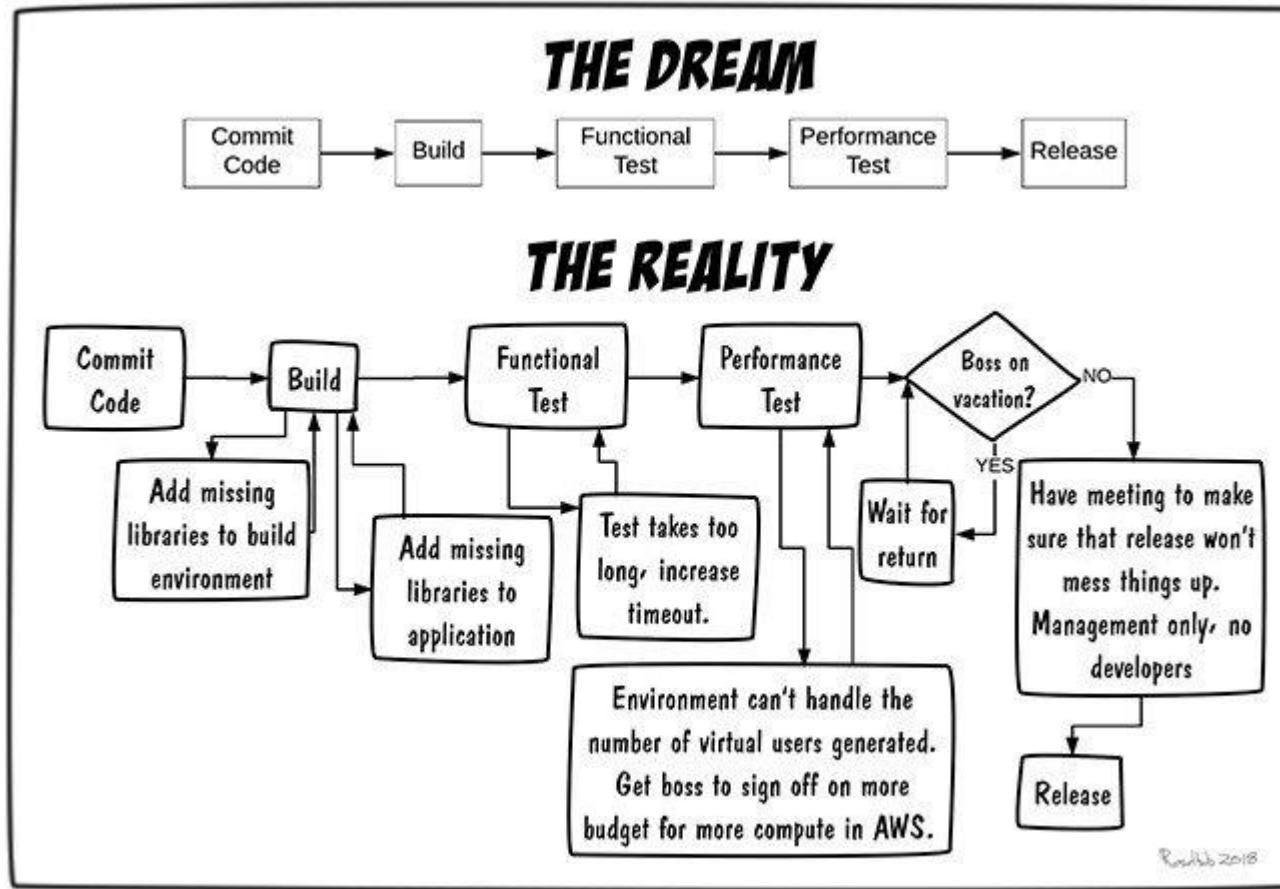


## 6.4 – Estado actual



[https://www.sonatype.com/hubfs/SON\\_Survey2018\\_final.pdf](https://www.sonatype.com/hubfs/SON_Survey2018_final.pdf)

- Mature DevOps practices were 338% more likely to integrate automated security than organizations with no DevOps practice
- 3 in 10 organizations suspected or verified breaches stemming from vulnerabilities in open source and third party components -- a 55% increase over 2017



<https://devops.com/ci-cd/>

- Podéis seguir en twitter a:
  - [@devops research](#)
  - [@ITRevBooks](#)

## **6.1 Introducción**

## **6.2 Las Tres Vías**

### **6.2.1 Primera Vía: Acelerar el flujo**

### **6.2.2 Segunda Vía: Retroalimentación continua**

### **6.2.3 Tercera Vía: Experimentación continua**

## **6.3 Practicando la Primera Vía**

### **6.3.1 Infraestructura es código en la pipeline**

### **6.3.2 Pruebas automáticas y CI**

### **6.3.3 CD**

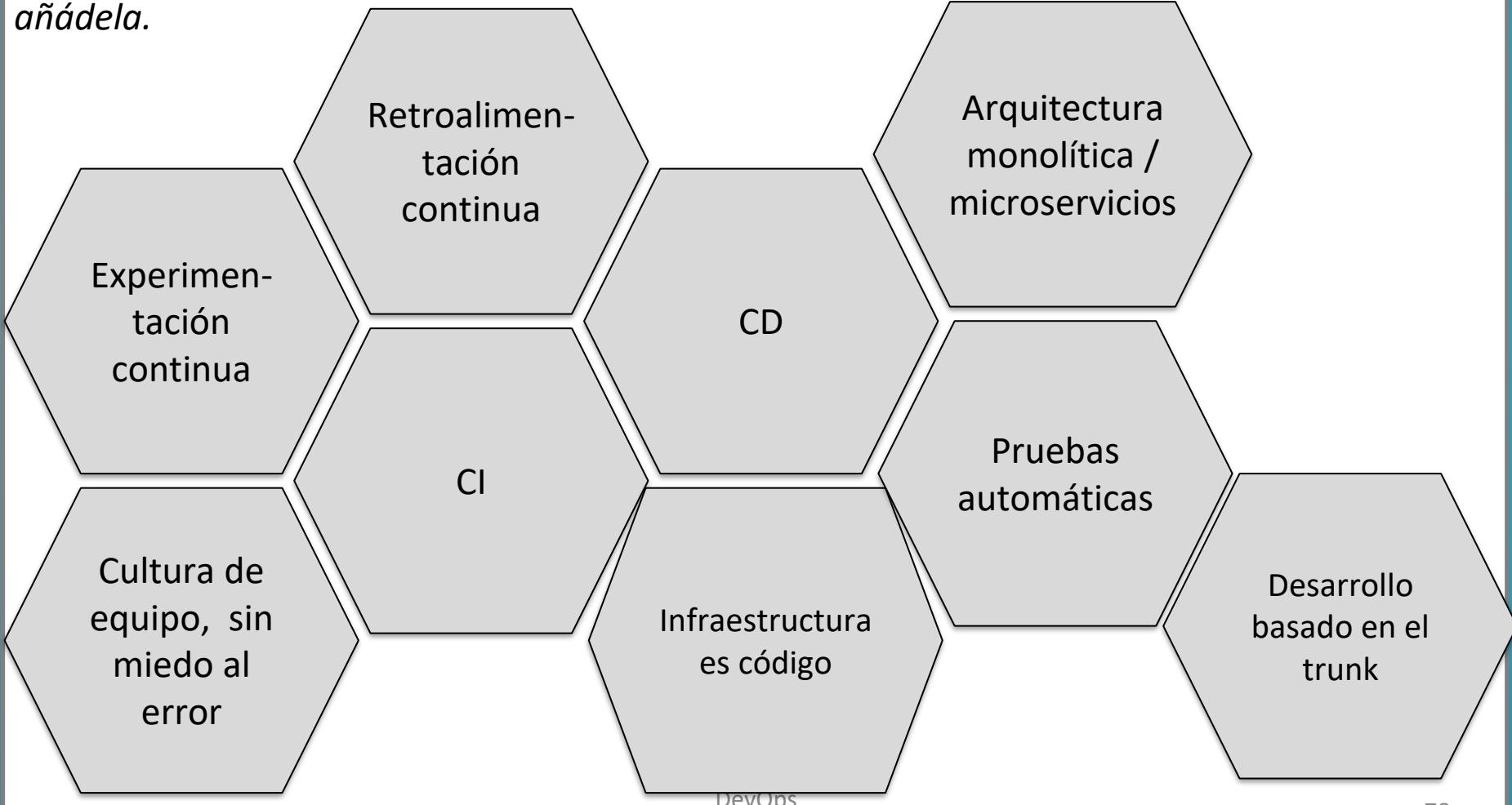
### **6.3.4 Arquitectura y Despliegues**

## **6.4 Estado Actual**

## **6.5 Casos de Estudio**

## 6.5 – Casos de Estudio

Lee los siguientes casos de estudio (traducidos y adaptados de *The DevOps Handbook*), y marca qué práctica(s) habituales en DevOps son utilizadas. Si crees que falta alguna añádela.



## 6.5.1 Casos de Estudio – Amazon (2001-2015)

Una de las transformaciones arquitectónicas más estudiadas ocurrió en Amazon. Werner Vogels (CTO en Amazon) explica en una entrevista que Amazon.com comenzó en 1996 como una “aplicación monolítica, ejecutada sobre un servidor web y comunicada con una base de datos en el back end. Esta aplicación, apodada Obidos, evolucionó para encargarse de toda la lógica de negocio, de display, y toda la funcionalidad por la que Amazon acabaría siendo famoso: parecidos, recomendaciones, listas, revisiones, etc...”

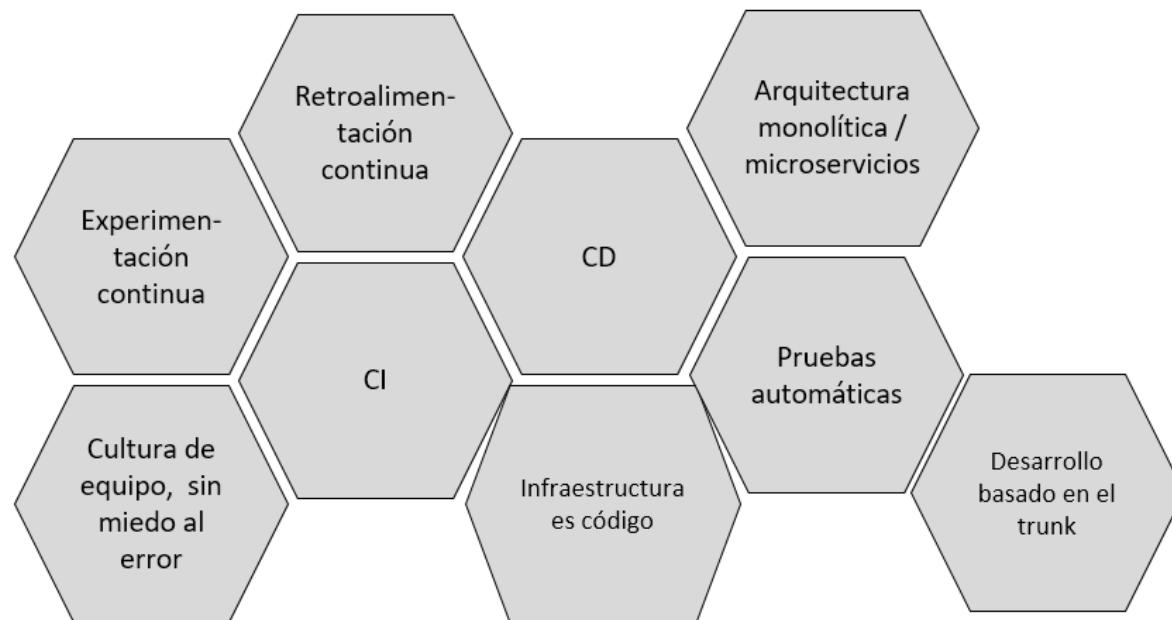
A medida que pasaba el tiempo, Obidos evolucionó creando una maraña, con relaciones complejas de manera que partes individuales no podían escalarse como se necesitaba. Vogels cuenta que esto significaba que “muchas cosas que te gustaría ver en un buen entorno software, ya no podían realizarse; había muchos trozos de software complejos combinados dentro de un solo sistema. Ya no podía evolucionar más”.

Describiendo el proceso por el que fue ideando su arquitectura deseada, cuenta: “Pasamos por un período de seria introspección y concluimos que una arquitectura orientada a servicios nos daría el nivel de aislamiento que permitiera construir muchos componentes software rápidamente”.

Vogels añade, “El gran cambio arquitectónico por el que pasó Amazon durante 5 años [2001-2005] fue cambiar de una plataforma monolítica de 2 niveles a una totalmente distribuida, descentralizada y plataformas que daban servicio a muchas aplicaciones diferentes. Hizo falta mucha innovación para que esto sucediera, ya que fuimos de los primeros en tomar esta medida”.

En 2011, Amazon realizaba 15000 despliegues por día. En 2015, 136000.

## 6.5.1 Casos de Estudio – Amazon (2001-2015)



## 6.5.2 Casos de Estudio – Etsy (2014)

Al contrario que Facebook, donde los despliegues son gestionados por ingenieros encargados de las releases, en Etsy los despliegues son realizados por cualquiera que quiera hacerlo, sean de Desarrollo, Operaciones o InfoSec. El proceso de despliegue en Etsy se ha convertido en algo tan seguro y rutinario que hasta un ingeniero nuevo en la empresa puede llegar a realizar un despliegue a producción en su primer día de trabajo.

Noah Sussman, arquitecta de pruebas en Etsy, escribió: “Sobre las 8 a.m. de un día laborable normal, más de 15 personas comienzan a hacer cola [en la pipeline], todos esperando a desplegar en total hasta 25 conjuntos de cambios antes de acabar el día.”

Aquellos ingenieros que quieren desplegar su código deben entrar [log-in] en un chat, donde cada uno se añade asimismo a la cola de despliegue, ven la actividad de despliegue en progreso, quién más está en la cola, anuncian sus actividades y puede pedir ayuda al resto de ingenieros si lo necesitan. Cuando es su turno para desplegar se les notifican en el chat.

El objetivo en Etsy ha sido hacer fácil y seguro desplegar a producción con el menor número posible de pasos ceremoniales. Antes de que el desarrollador haga un checkin de su código, ejecutará en su puesto de trabajo 4500 tests unitarios en menos de un minuto (todas las llamadas a sistemas externos, como bases de datos, son simuladas o realizadas de manera parcial).

Tras hacer cambios en el trunk principal del control de versiones, cerca de 7000 tests automáticos son ejecutados en el trunk en servidores dedicados a integración automática. Sussman escribe: “Median ensayo y error, hemos establecido 11 minutos como el tiempo más largo que los tests automáticos pueden tardar al realizar un push. Así, eso deja tiempo para relanzar los tests durante el despliegue (si alguien ha roto la pipeline y hay que arreglarla), sin sobrepasar un límite de 20 minutos.

## 6.5.2 Casos de Estudio – Etsy (2014)

Si todos los tests se ejecutaran secuencialmente, Sussman indica que “los 7000 tests en el trunk tardarían sobre media hora en ejecutarse. Así que dividimos los tests en subconjuntos, y los distribuimos en las 10 máquinas de nuestro clúster de Jenkins [CI]… Dividir nuestra test suite y ejecutarlos en paralelo nos da el tiempo deseado de 11 minutos”.

Los siguientes tests en ejecutarse son los smoke tests, que son tests a nivel de sistema que lanzan llamadas para ejecutar pruebas unitarias PHPUnit. Después, se ejecutan tests funcionales, los cuales ejecutan pruebas dirigidas por GUI en un servidor (con el entorno de QA o el entorno de stage llamado “Princess”) cuya configuración nunca se cambia para asegurarnos que es exactamente como el entorno de producción.

Cuando llega el turno de despliegue de un ingeniero, “vas a el Deployinator [herramienta interna de Etsy], y pulsas el botón para ir a QA. Desde ahí se va a Princess… Entonces, cuando el código está listo, pulsas el botón “Prod” y tu código ya ha salido a producción, y todo el mundo en el canal de chat IRC sabe quién integró ese código y tiene disponible un link para ver las diferencias con la versión trunk anterior. A los que no estaban en el IRC se manda un correo con la misma información.

En 2009, el proceso de despliegue en Etsy era causa de estrés y miedo. En 2011, se había convertido en una operación rutinaria que sucedía de 25 a 50 veces por día, ayudando a que los ingenieros subieran rápidamente su código a producción, llevando valor a sus clientes.

# ক্রেডিট কোর্স মার্কেট

logged in as: ekastner [root] QA: 25145-trunk-20100518-220720-UTC Princess: 25145-trunk-20100518-220720-UTC  
Production: 25145-trunk-20100518-220720-UTC

**Deploy to QA (Trunk)**

deploying revision: 25150

Message:

**Push to QA →**

**Princess is in the other castle**

deploying version: 25145-trunk-20100518-220720-UTC

**Save the Princess**

**Deploy to Production**

deploying version: 25145-trunk-20100518-220720-UTC

**PROD!!! →**

Important links:

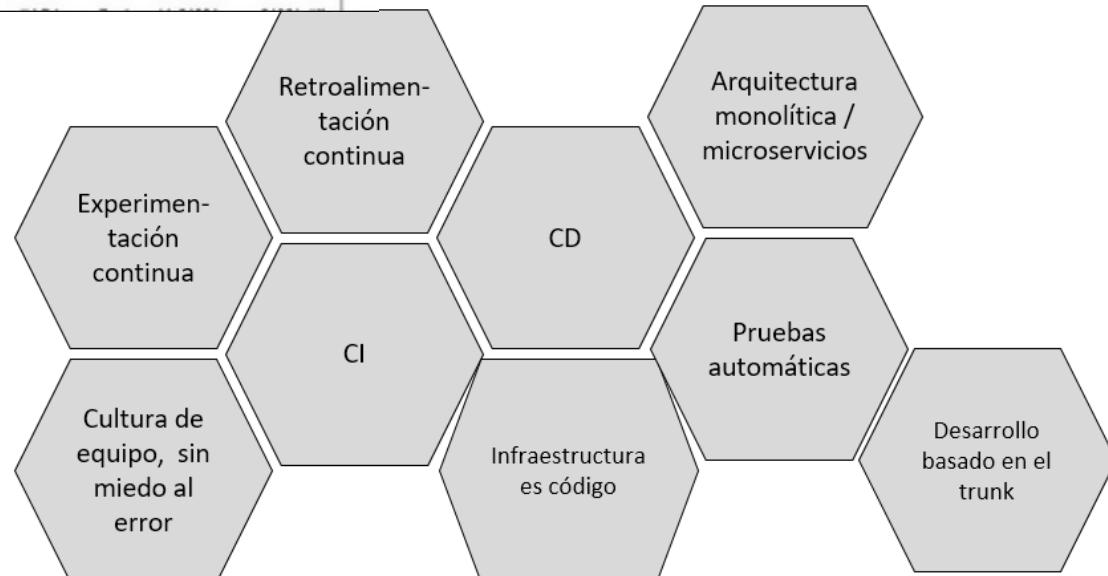
**Log**

Commits since last prod deploy!!  
 Auto scroll command output?

Add an arbitrary log message:  
#sozeyr--- to link to jira and add a comment

- [ web ] | 2010-05-18 22:20:50 | PRODUCTION | sandrews | Production Deploy: old 25134, new: 25145 [diff](#)
- [ web ] | 2010-05-18 22:18:00 | PRINCESS | sandrews | Princess Deploy: old: 25144, new: 25145 [diff](#)
- [ web ] | 2010-05-18 22:17:22 | QA | sandrews | kyles bug fix old: , new: 25145 [diff](#)
- [ web ] | 2010-05-18 22:12:03 | QA | sandrews | pushing again -- banned user cache busting old: , new: 25144 [diff](#)
- [ web ] | 2010-05-18 22:06:39 | PRINCESS | sandrews | Princess Deploy: old: 25134, new:
- [ web ] | 2010-05-18 22:02:35 | QA | sandrews | ~~old: 25134, new: 25144~~ old: 25134, new: 25144 [diff](#)
- [ web ] | 2010-05-18 20:56:50 | PRODUCTION | cmunnns | Production Deploy: old 25134, new: 25134 [diff](#)
- [ web ] | 2010-05-18 20:49:02 | PRODUCTION | cmunnns | Production Deploy: old 25134, new: 25134 [diff](#)
- [ web ] | 2010-05-18 20:44:43 | PRODUCTION | ahashim | Production Deploy: old 25030, new: 25134 [diff](#)
- [ web ] | 2010-05-18 20:41:17 | PRINCESS | ahashim | Princess Deploy: old: 25030, new: 25134 [diff](#)
- [ web ] | 2010-05-18 20:40:38 | QA | ahashim | peanut butter jelly time!!!! old: 25030, new: 25134 [diff](#)
- [ web ] | 2010-05-17 16:23:26 | PRODUCTION | sandrews | Production Deploy: old 24951, new: 25030 [diff](#)
- [ web ] | 2010-05-17 16:12:12 | PRINCESS | sandrews | Princess Deploy: old: 24951, new: 25030 [diff](#)
- [ web ] | 2010-05-17 16:08:05 | QA | sandrews | Kissmetrics amongst others old: 24951, new: 25030 [diff](#)
- [ web ] | 2010-05-14 20:16:47 | PRODUCTION | zgarrett | Production Deploy: old 24884, new: 24951 [diff](#)

## 6.5.2 Casos de Estudio – Etsy (2014)



## 6.5.3 Casos de Estudio – Bazaarvoice (2012)

Ernest Mueller ayudó a transformar el proceso de desarrollo y producción en Bazaarvoice en 2012. Bazaarvoice suministra contenido generado por los clientes (revisiones, puntuaciones,...) a miles de minoristas, como BestBuy, Nike y Walmart.

Por entonces, Bazaarvoice tenía unos ingresos de 120 millones de dólares y se estaba preparando para un IPO (primera venta de acciones al público). El negocio era dirigido principalmente por la aplicación Bazaarvoice Conversations, una aplicación Java monolítica de casi 5M LOC escritas en 2006 y distribuidas en 15000 archivos. Se ejecutan en 1200 servidores de 4 centros de datos y múltiples proveedores de servicios en la nube.

En parte como resultado por haber adoptado un proceso de desarrollo ágil con intervalos de desarrollo de 2 semanas, había un gran deseo de aumentar la frecuencia de despliegue, entonces programado de manera fija cada 10 semanas. También habían comenzado a separar partes de su aplicación monolítica, desglosándolas en microservicios.

Su primer intento de despliegue programado en 2 semanas fue en Enero de 2012. Mueller dice: “No fue bien. Se formó un caos masivo, con 44 incidentes en producción indicados por nuestros clientes. La reacción principal de los directores fue básicamente: ‘No hagamos esto nunca más’ “.

Mueller se encargó del proceso de despliegue poco después, con el objetivo de conseguir hacerlo cada 2 semanas sin interrumpir el servicio a los clientes. Los objetivos de negocio para los despliegues frecuentes incluían realizar tests A/B más rápidos e incrementar el flujo de características en producción. Mueller identificó 3 problemas principales:

1. La falta de tests automáticos hacía que los tests de cualquier nivel fueran inadecuados para prevenir fallos a gran escala.
2. Su estrategia de ramificar en control de versiones permitía a los desarrolladores hacer check-in de nuevo código directamente en la versión de producción.
3. Los equipos encargados de microservicios estaban realizando también despliegues independientes.

## 6.5.3 Casos de Estudio – Bazaarvoice (2012)

Mueller llegó a la conclusión de que debía estabilizarse el proceso de despliegue de la aplicación monolítica Conversations, lo cual requeriría integración continua. En las 6 semanas siguientes, los desarrolladores dejaron de implementar características para enfocarse en escribir suites de tests automáticos, los cuales incluían tests unitarios en Junit, tests de regresión en Selenium, y establecer una pipeline de despliegue sobre TeamCity. “Al ejecutar estos tests constantemente, sentíamos que podíamos hacer cambios con cierto nivel de seguridad. Y lo más importante, podíamos detectar inmediatamente cuándo alguien estropeaba algo, en vez de describirlo solo una vez que ya está en producción”.

También se movieron a un modelo de despliegue trunk/rama, en el cual cada 2 semanas creaban una rama dedicada al próximo despliegue, sin permitir nuevos *commits* a esa rama al menos que fuera una emergencia – todos los cambios se hacían offline, por turno o por equipo-. Esa rama iría a través de un proceso de AQ, y entonces pasada a producción.

Las mejoras en la predictibilidad y calidad de las *releases* fueron asombrosas:

- Enero 2012: 44 incidencias de clientes (empieza el esfuerzo en integración continua).
- 6 Marzo 2012: 5 días tarde, 5 incidencias de clientes.
- 22 Marzo 2012: a tiempo, 1 incidencia de cliente.
- 5 Abril 2012: a tiempo, 0 incidencias de cliente.

“Tuvimos tal éxito desplegando cada 2 semanas, que nos pasamos a 1 semanal, lo cual no supuso apenas cambios dentro de los equipos. Al convertirse los despliegues en algo tan rutinario, no supuso más que doblar el número de despliegues en el calendario y hacerlos cuando la agenda lo decía. En serio, ya no suponía un evento especial. La mayoría de cambios pedidos pertenecían a nuestros equipos de servicio al cliente y publicidad, los cuales tuvieron que cambiar sus procesos, como por ejemplo cambiar sus correos semanales a los clientes para avisarles de las nuevas características desplegadas. Después de esto, comenzamos a trabajar en nuestros nuevos objetivos, los cuales llevaron a acelerar nuestros tiempos de pruebas de >3h a <1h, y reduciendo el número de entornos de 4 a 3 (Dev, Test, Producción, eliminando Staging), y así cambiamos a un modelo de entrega continua en la que realizamos despliegues rápidos de 1-clic.”

## 6.5.3 Casos de Estudio – Bazaarvoice (2012)

