



Pontificia Universidad Católica de Chile  
Escuela de Ingeniería  
Departamento de Ciencia de la Computación

# Clase 19

## SQA

IIC2143 - Ingeniería de Software  
Sección 1

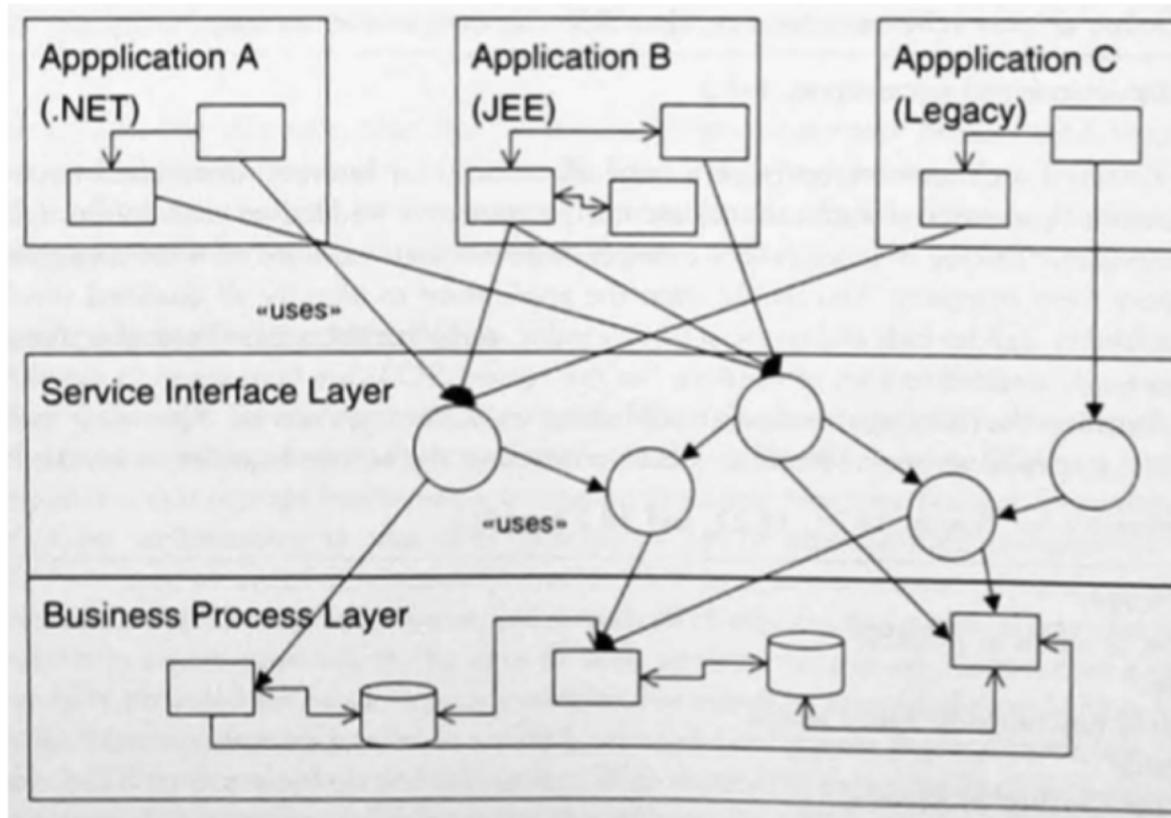
Rodrigo Saffie

[rasaffie@uc.cl](mailto:rasaffie@uc.cl)

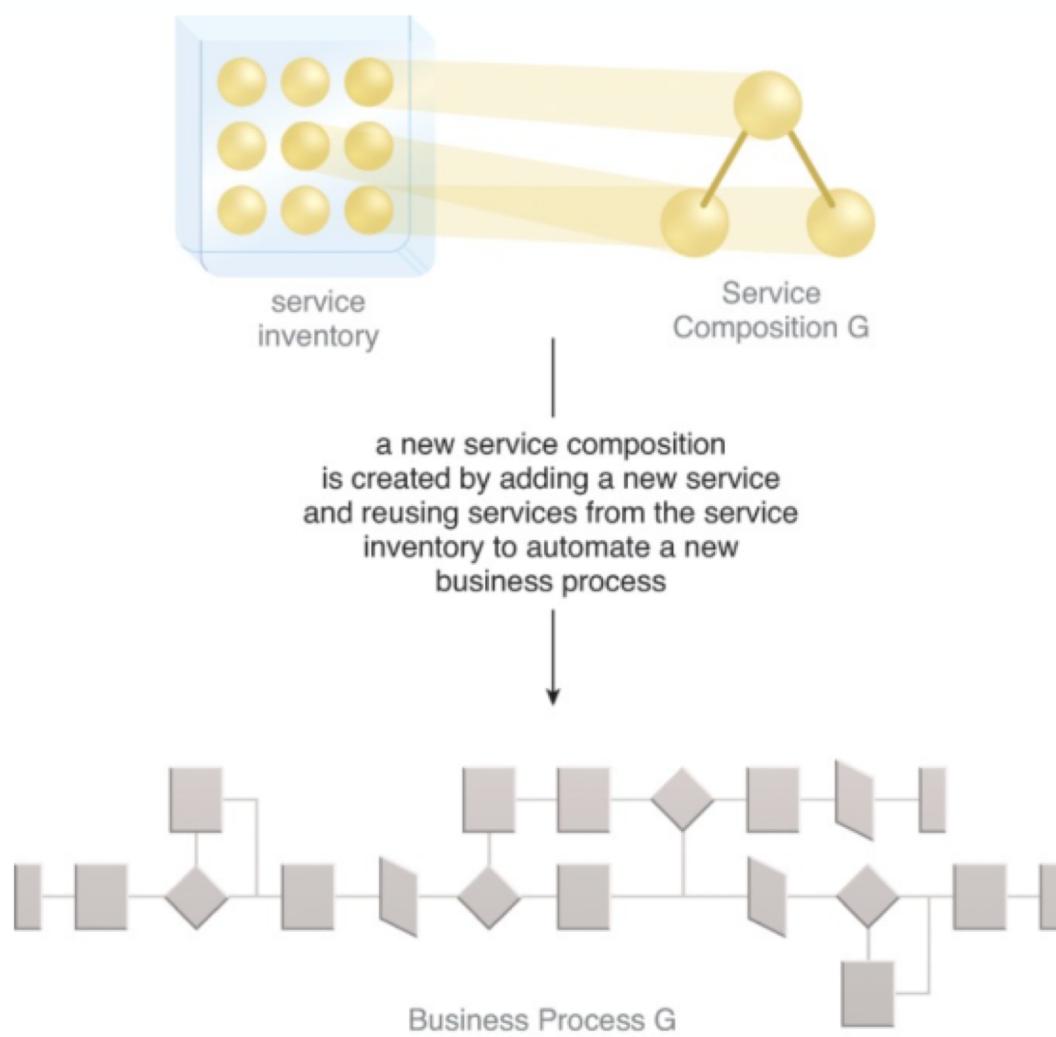
28 de mayo de 2018

# Patrones arquitectónicos

*Service Oriented Architecture (SOA)*



# Con arquitectura de servicios



# Aparición de los microservicios

- Primera generación de SOA comenzó a hacerse demasiado compleja
  - SOAP -> middleware
  - *Monolithic REST API*
- Microservicios: representan una maduración de las ideas de *SOA* a la luz de la experiencia práctica
- No requieren capa de *middleware*
  - cada servicio expone una API (contrato)

# Características de arquitectura de microservicios

- Componentes son servicios
- Altamente cohesivos y desacoplados
- Usan principios y protocolos de la Web (HTTP, REST)
- Organización en base a lógica de negocio y no de especialización tecnológica (UI, *middleware*)
- Gobierno descentralizado (un servicio es totalmente independiente)
- Manejo de datos descentralizado

# Microservicios vs SOA

2000's

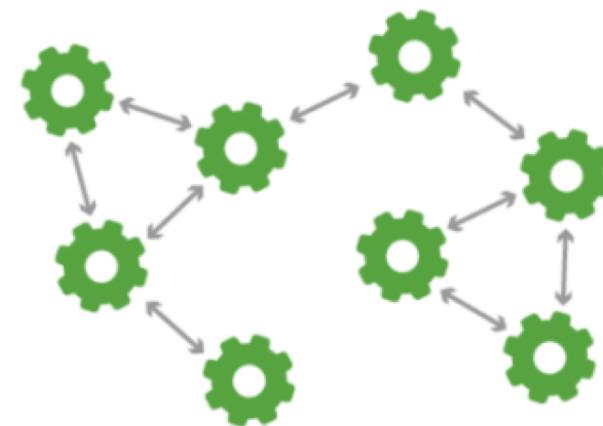
## SERVICE ORIENTED ARCHITECTURE



**SOA** based applications are comprised of more loosely coupled components that use an Enterprise Services Bus messaging protocol to communicate between themselves.

2010's

## MICROSERVICES ARCHITECTURE

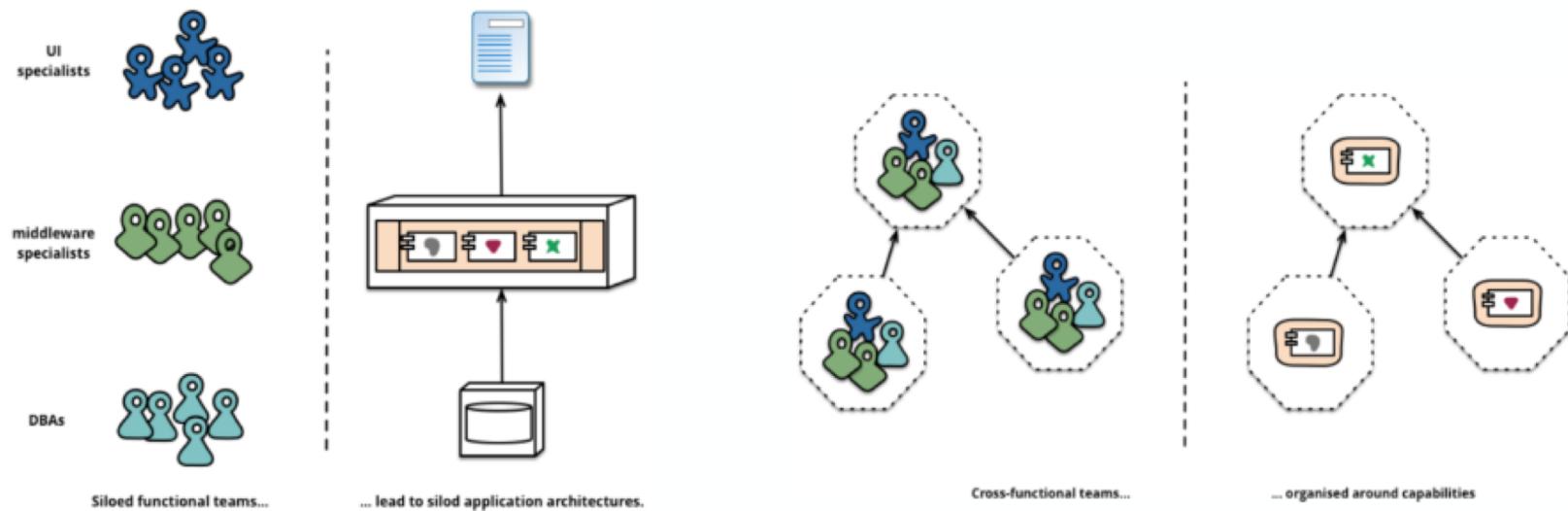


**Microservices** are a number of independent application services delivering one single functionality in a loosely connected and self-contained fashion, communicating through light-weight messaging protocols such as HTTP, REST or Thrift API.

# Tipo de Microservicios

- **Servicios funcionales:** se exponen hacia fuera de la organización
- **Servicios de infraestructura:** Para uso interno de los otros servicios
  - Autenticación y autorización
  - Monitoreo
  - Logging y auditoría

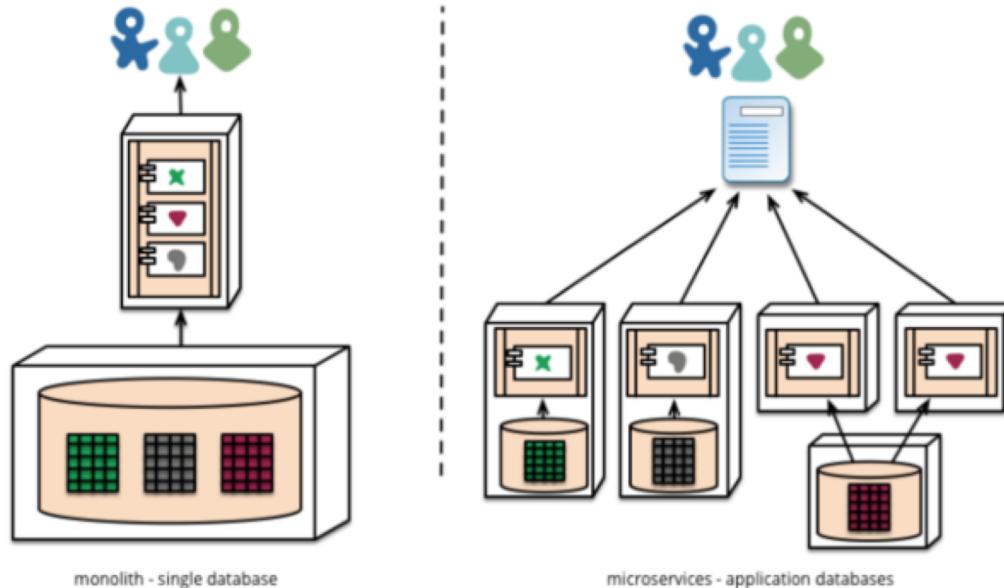
# Equipos en microservicios



**Clásico: front-end, back-end, DB**

**Microservices: equipos multifuncionales**

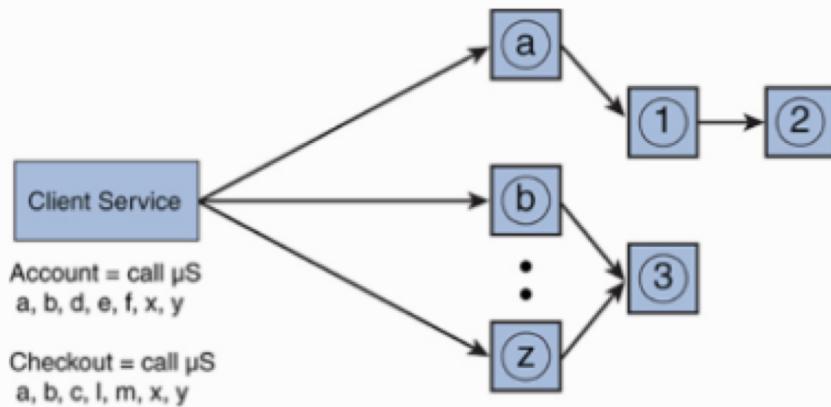
# Manejo de datos descentralizados



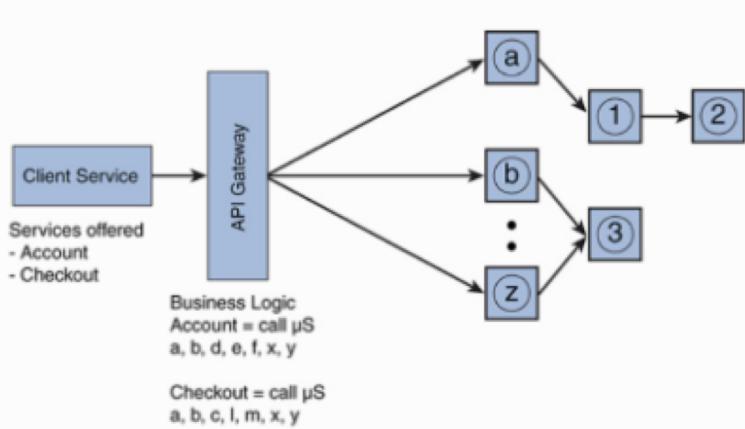
- Cada servicio maneja su propia DB
- No se usan transacciones para coordinar, consistencia eventual
- Respuesta rápida y escalabilidad se paga con posibles grados de inconsistencia temporal

# Variaciones

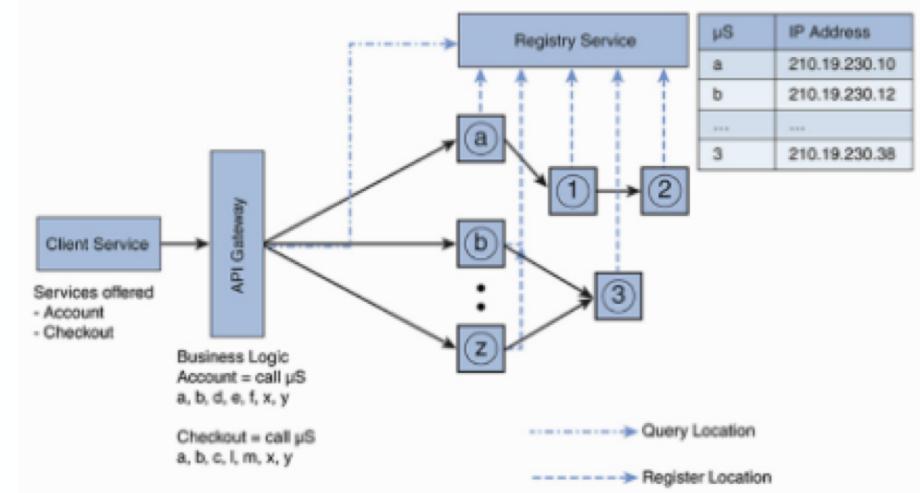
## Sin API Gateway



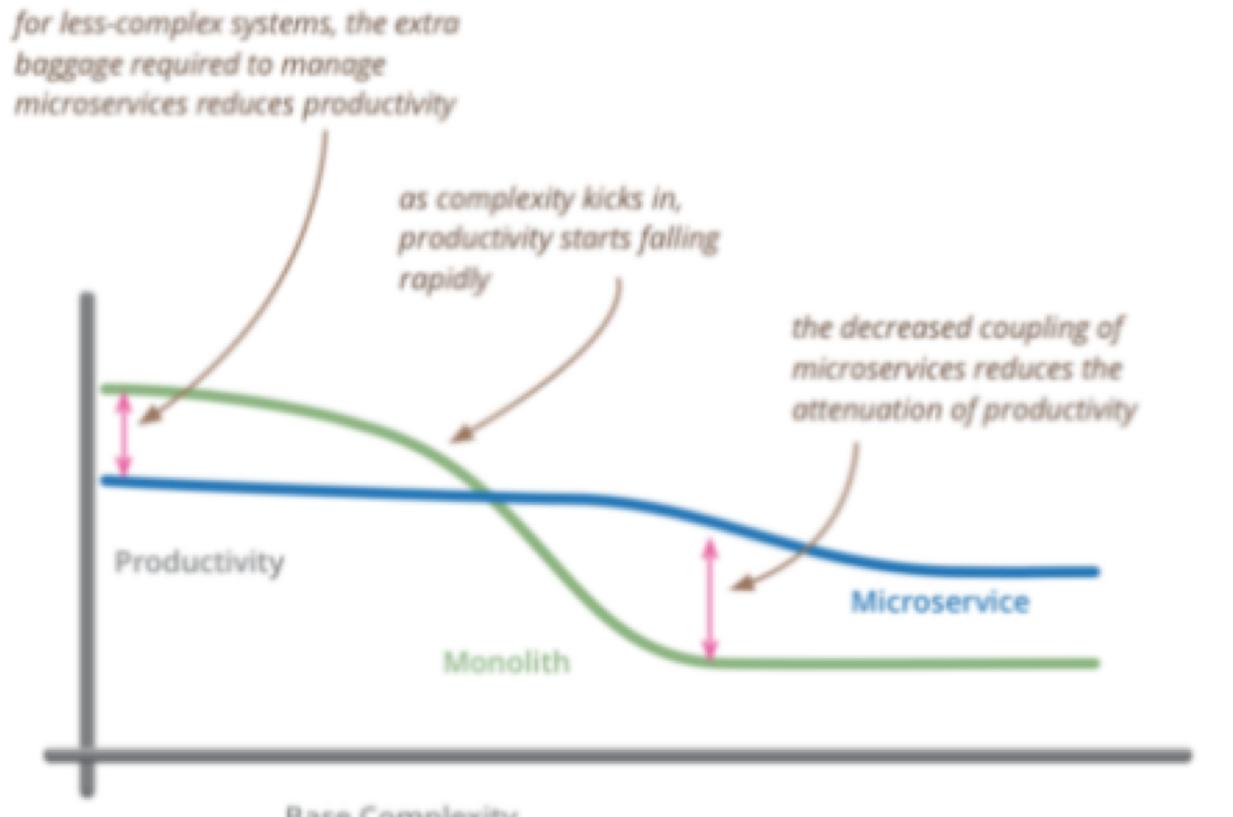
## Con API Gateway



## Con API Gateway y Registry



# No es la panacea



*but remember the skill of the team will outweigh any monolith/microservice choice*

# Desafíos

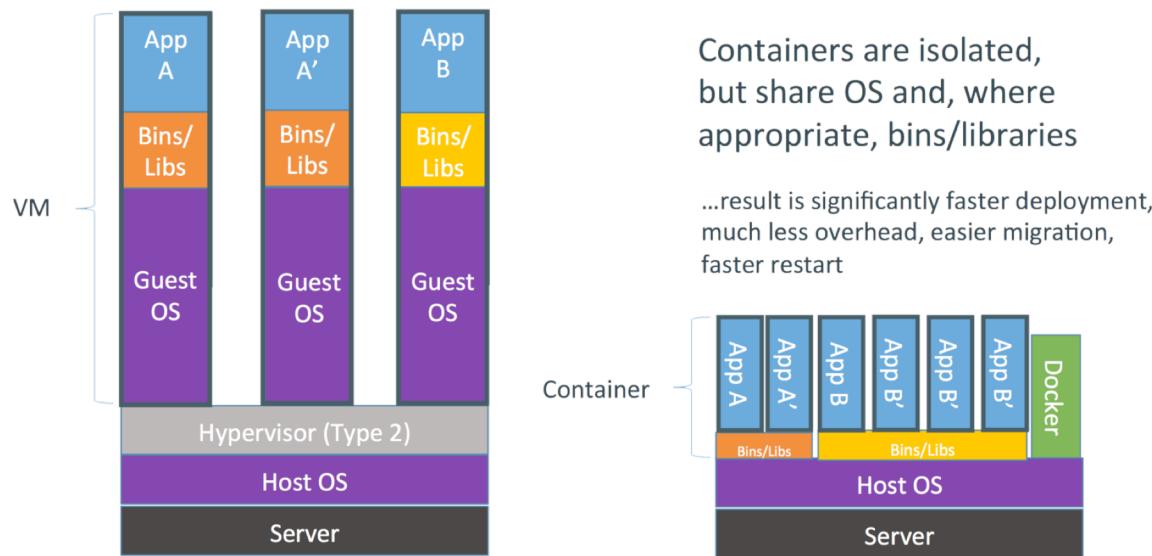
- Complejidad puede ser mayor que el de una aplicación monolítica equivalente
- Congestión de la red y latencia
- Integridad de datos
- Pensar de manera distribuida y asíncrona

# Mejores prácticas

- Modelar servicios de acuerdo a dominio de negocio
- Alta cohesión y bajo acoplamiento
- Descentralizar todo (equipos, código)
- Datos privados para cada servicio
- Comunicación a través de APIs bien definidas
- API Gateway no debe saber nada del dominio

# Microservicios y contenedores

- Un microservicio puede correr en una máquina virtual propia
- Sin embargo, hoy en día se ha hecho popular el uso de contenedores

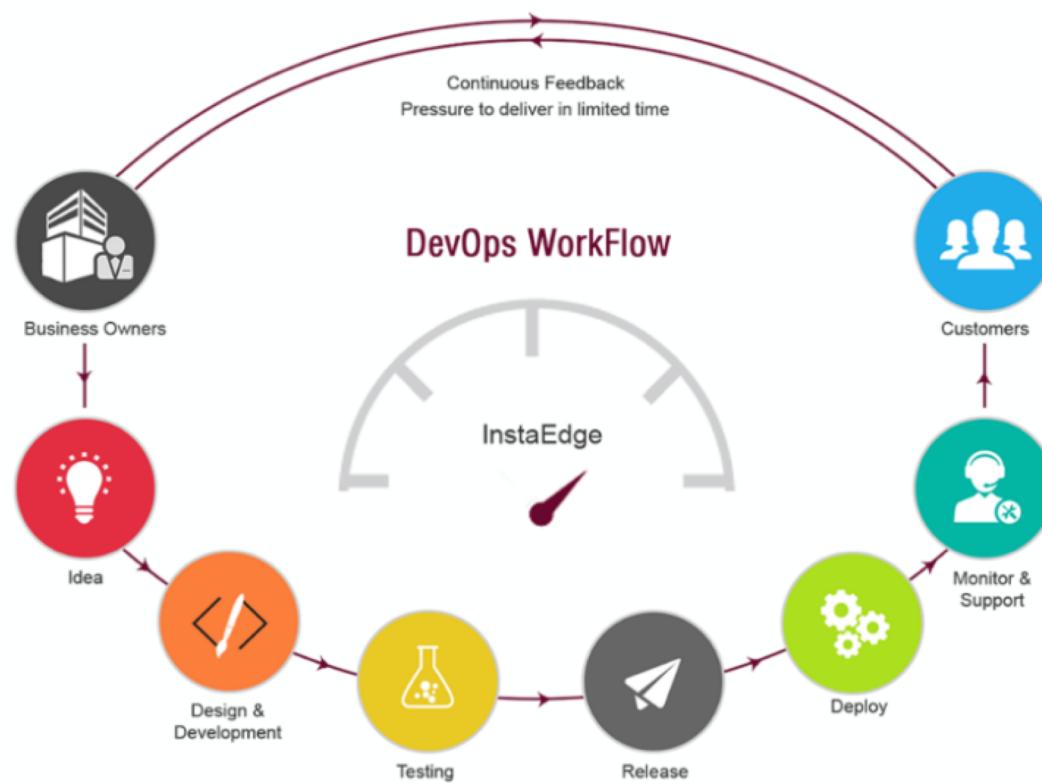


# Microservicios y DevOps

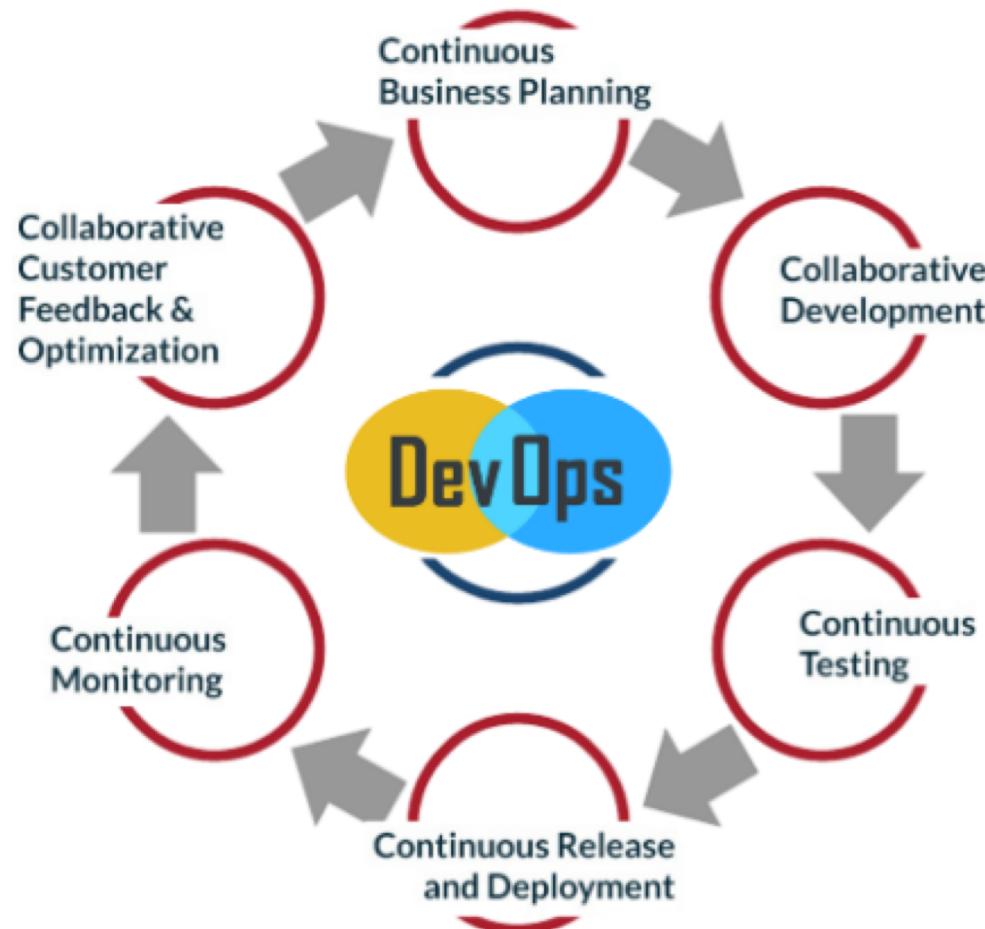
- DevOps – Unir desarrollo de operaciones con el objetivo de acelerar drásticamente la puesta en producción de nuevas funcionalidades.
- Enfoque de microservicios es particularmente apropiado para esto por:
  - Desarrollo descentralizado
  - Independencia del resto de la aplicación
  - Puede ser probado en forma separada

# La revolución de DevOps

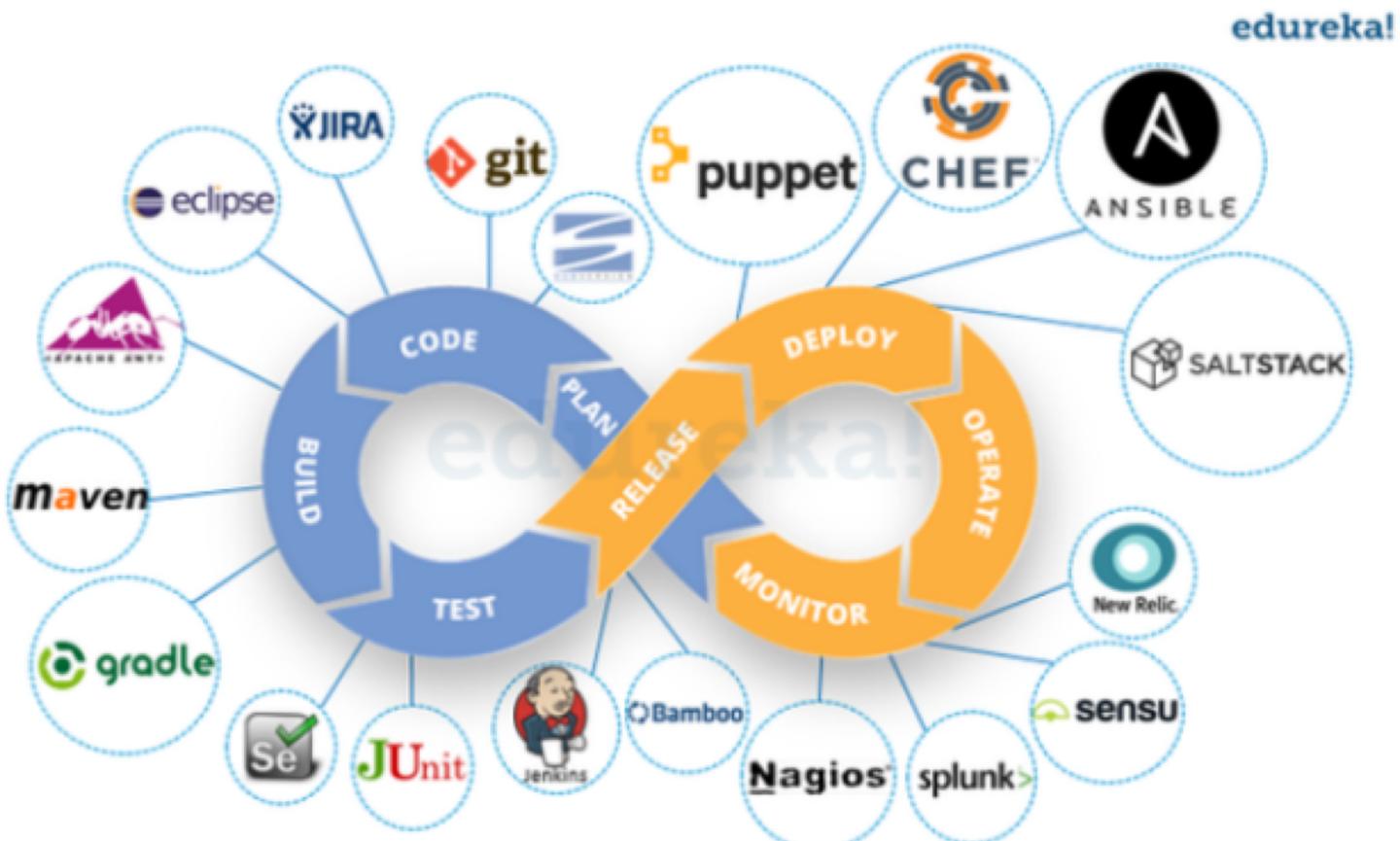
- Una empresa competitiva requiere poner en producción nuevas funcionalidades con rapidez



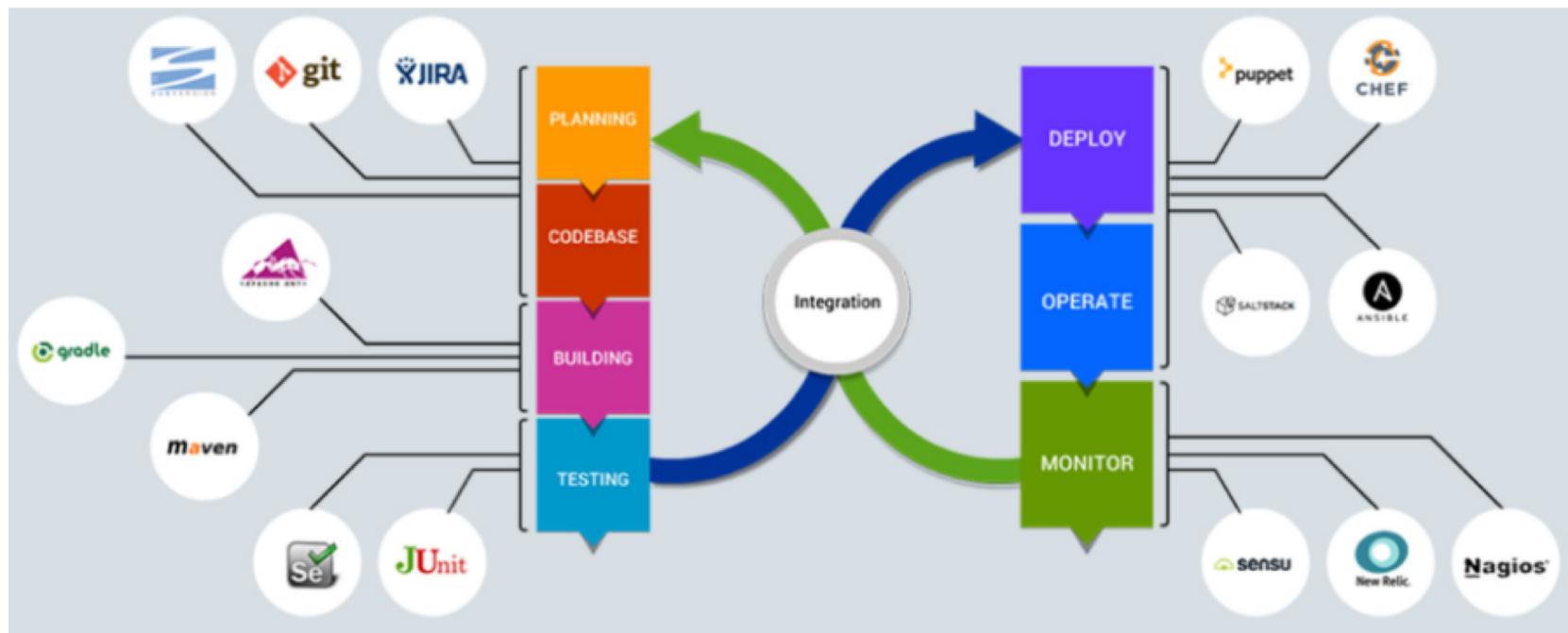
# El ciclo de DevOps



# Automatización de DevOps



# Automatización de DevOps



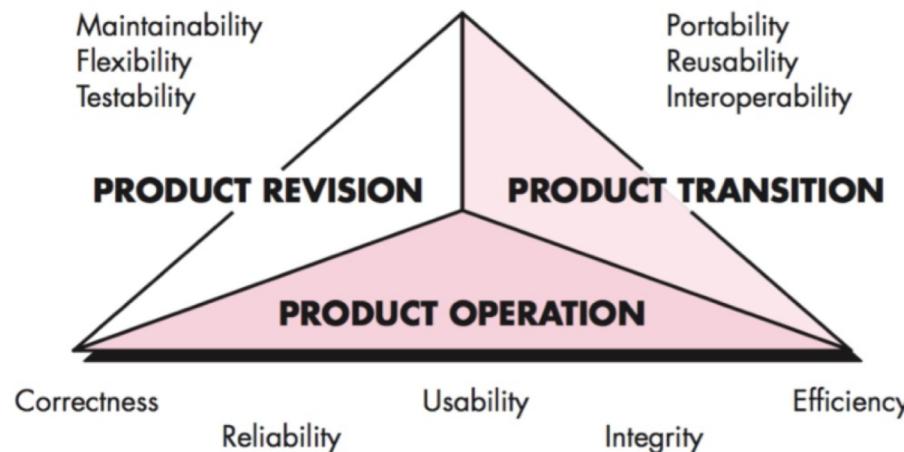
# ¿Por qué es difícil con arquitecturas monolíticas?

- Automatización de *tests* e instalación es demasiado compleja
- Base de datos central grande y con rol preponderante
- Difícil asegurar que todo anda bien antes de cambiar a nuevo *release*

# Aseguramiento de Calidad del Software (SQA)

¿Qué entendemos por calidad del *software*?

- Satisfacción total a requerimientos
  - Errores en requerimientos
  - Requerimientos poco claros
- Atributos de calidad (poco percibidos por usuarios)

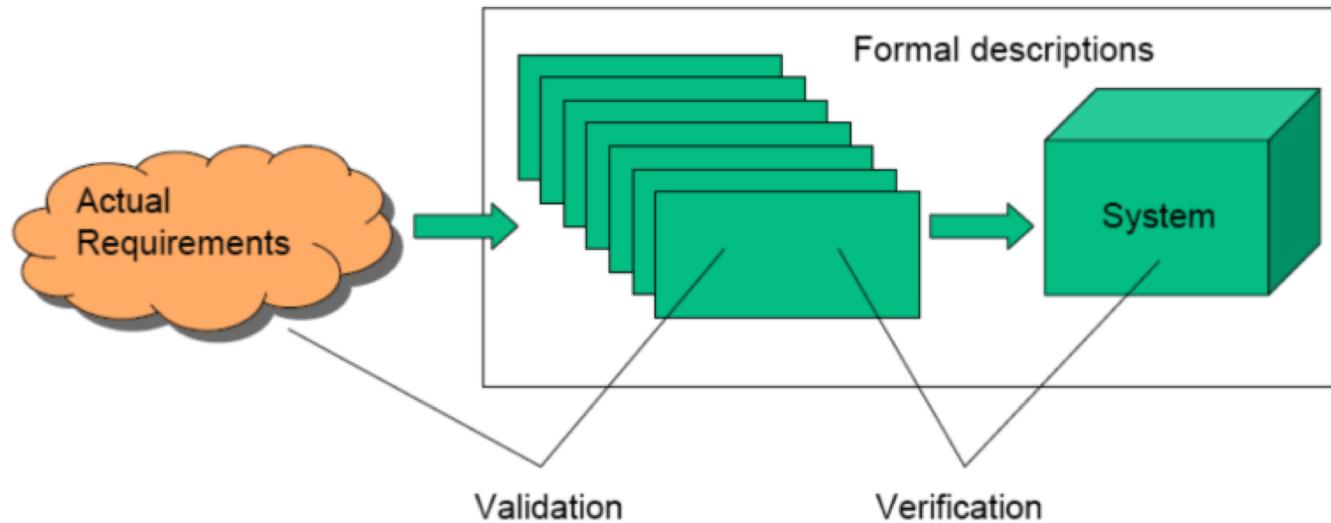


# Aseguramiento de Calidad del Software (SQA)

¿Por qué falla el *software*?

- En los requisitos:
  - Faltan requisitos
  - Requisitos mal definidos
  - Requisitos no realizables
  - Diseño de *software* defectuoso
- En la implementación:
  - Algoritmos incorrectos
  - Implementación defectuosa

# Verificación vs Validación



- Validación – el sistema hace lo que realmente se necesitaba que hiciera
- Verificación – el sistema hace correctamente lo que se especificó

# Verificación vs Validación

- Validación:
  - ¿Estamos construyendo el producto **correcto**?
- Verificación:
  - ¿Estamos construyendo el producto **correctamente**?

# Hacia ausencia de defectos

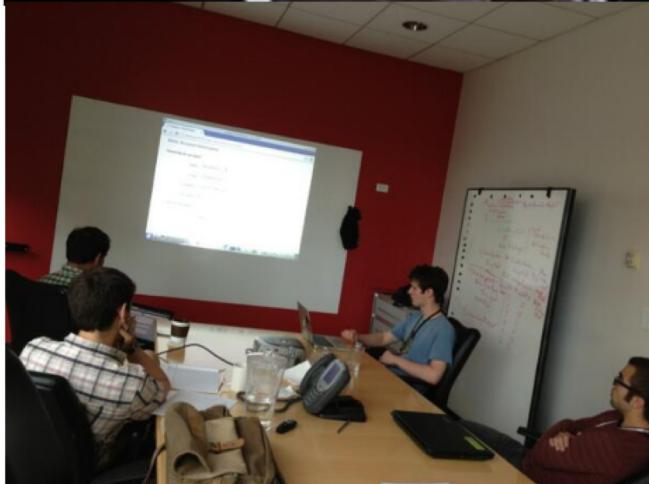
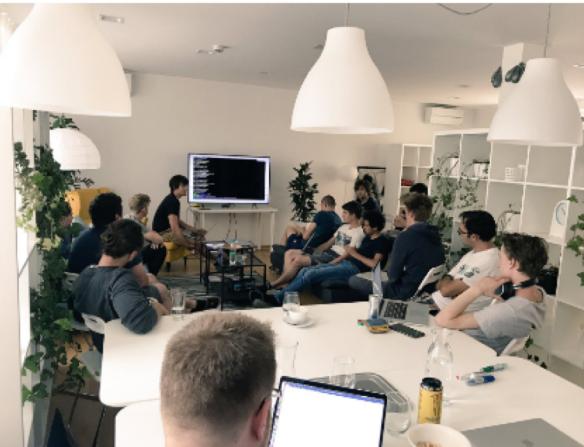
- No incorporarlos al construir el *software* (muy difícil)
- Análisis estático
  - Se examina el código con herramientas de *software* en busca de problemas o cosas sospechosas (código duplicado, peligros de seguridad, manejo de errores, entre otros)
- Inspección formal de código
  - Código es examinado por una o más personas que no son quienes lo escribieron
- *Testing*
  - Se diseñan casos de prueba y se somete el *software* a ellas

# Análisis estático

- Análisis del código sin ejecutarlo, mediante herramientas que permiten detectar elementos sospechosos
  - Ejemplos: [Rubocop](#), [Brakeman](#), [Reek](#), [Flay](#), [bundler-audit](#)
- Muchos falsos positivos
- Muchos defectos importantes no son detectados

# Inspección formal de código

- Forma efectiva para producir código de calidad
- Código es revisado por equipo de pares con el objetivo de detectar la mayor cantidad de defectos
- Equipo revisor se prepara antes de sesión de revisión
- Pueden usarse *checklists* para búsquedas más dirigidas
- Ejemplos: *Pull Requests* o sesiones de inspección



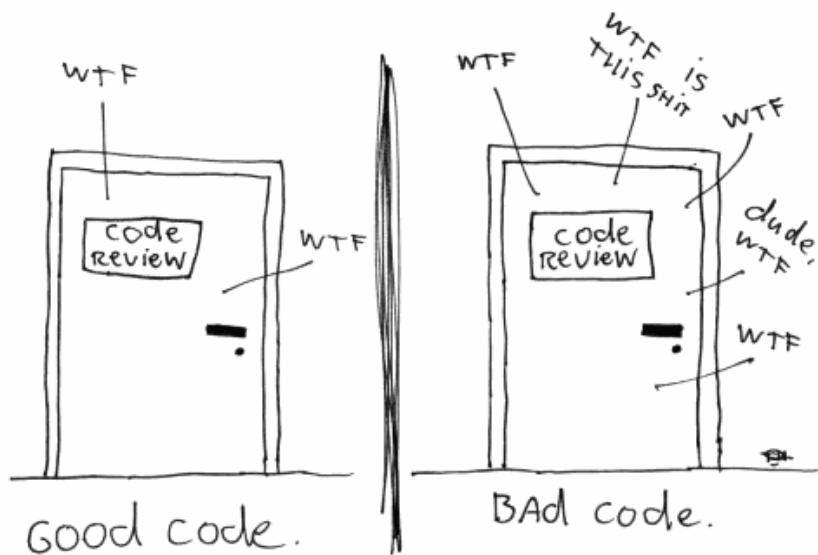
## The Peer Code Review



# Sesión de inspección

- Un facilitador dirige al grupo sobre el código
- Los revisores plantean sus dudas para ver si se trata de un problema
- Si es un problema se registra (no se corrige de inmediato)

The ONLY VALID MEASUREMENT  
OF CODE QUALITY: WTFs/MINUTE



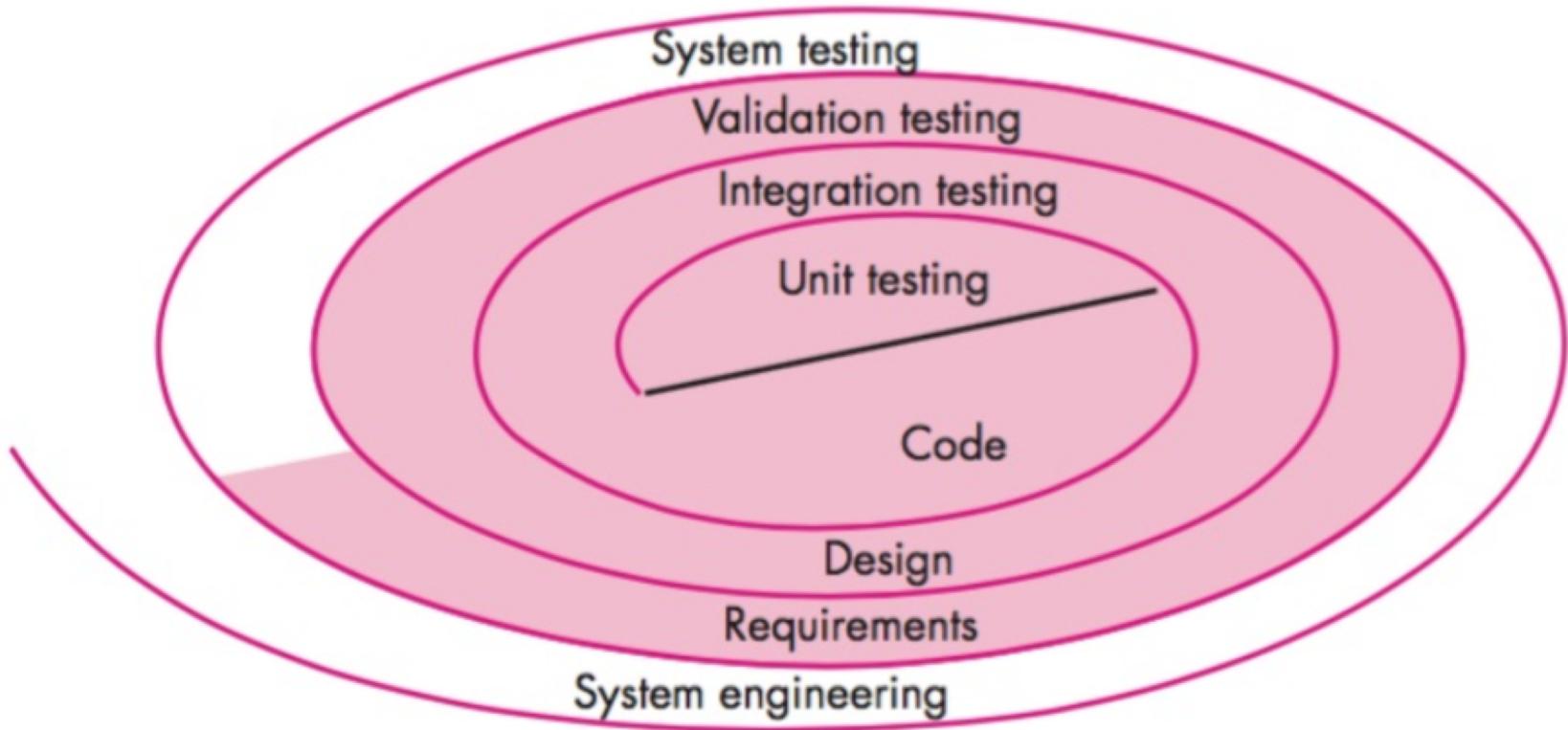
# Recomendaciones

- Métricas
  - Densidad de defectos: número promedio de errores por línea de código
  - Tasa de defectos: errores por hora de revisión
  - Tasa de inspección: líneas revisadas en una hora
- Revisar máximo 400 líneas de código por sesión
- No más de 1 hora por sesión
- Cultivar cultura positiva de revisión

# *Testing*

- Proceso de ejecutar un programa con el objetivo de encontrar un error
- Un buen caso de prueba es uno con una alta probabilidad de encontrar un error oculto
- Un *test* exitoso es aquel que descubre un error que no se conocía

# Niveles de *tests*





Pontificia Universidad Católica de Chile  
Escuela de Ingeniería  
Departamento de Ciencia de la Computación

# Clase 19

## SQA

IIC2143 - Ingeniería de Software  
Sección 1

Rodrigo Saffie

[rasaffie@uc.cl](mailto:rasaffie@uc.cl)

28 de mayo de 2018