



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

Clase 20

Testing

IIC2143 - Ingeniería de Software
Sección 1

Rodrigo Saffie

rasaffie@uc.cl

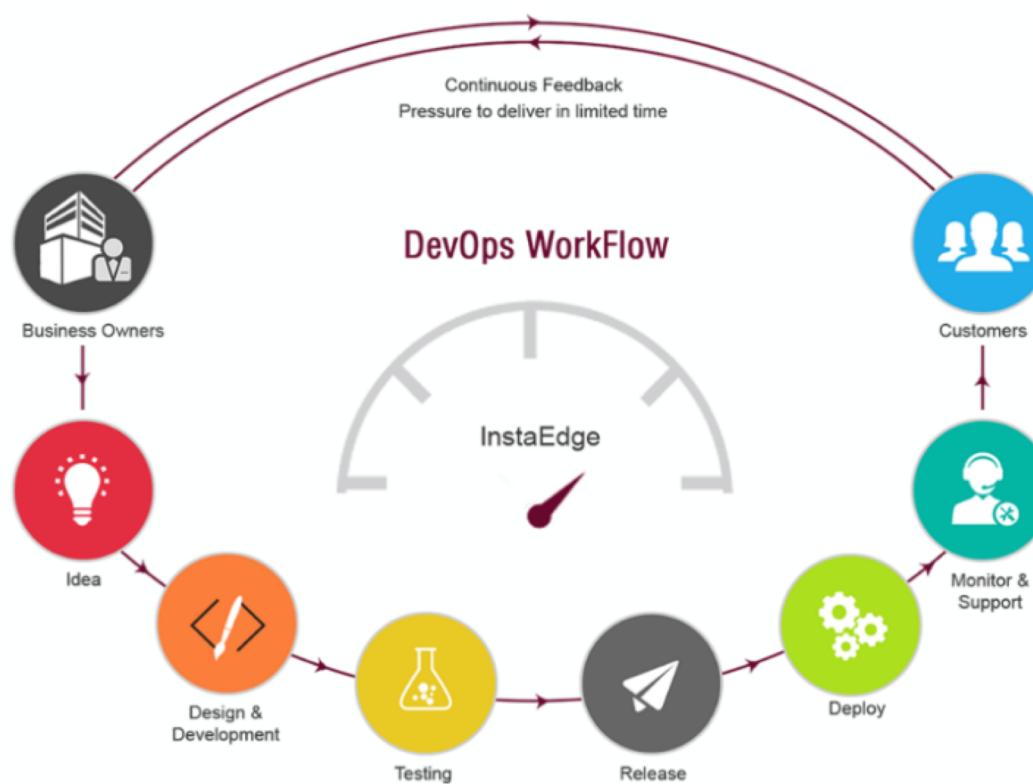
30 de mayo de 2018

Microservicios y DevOps

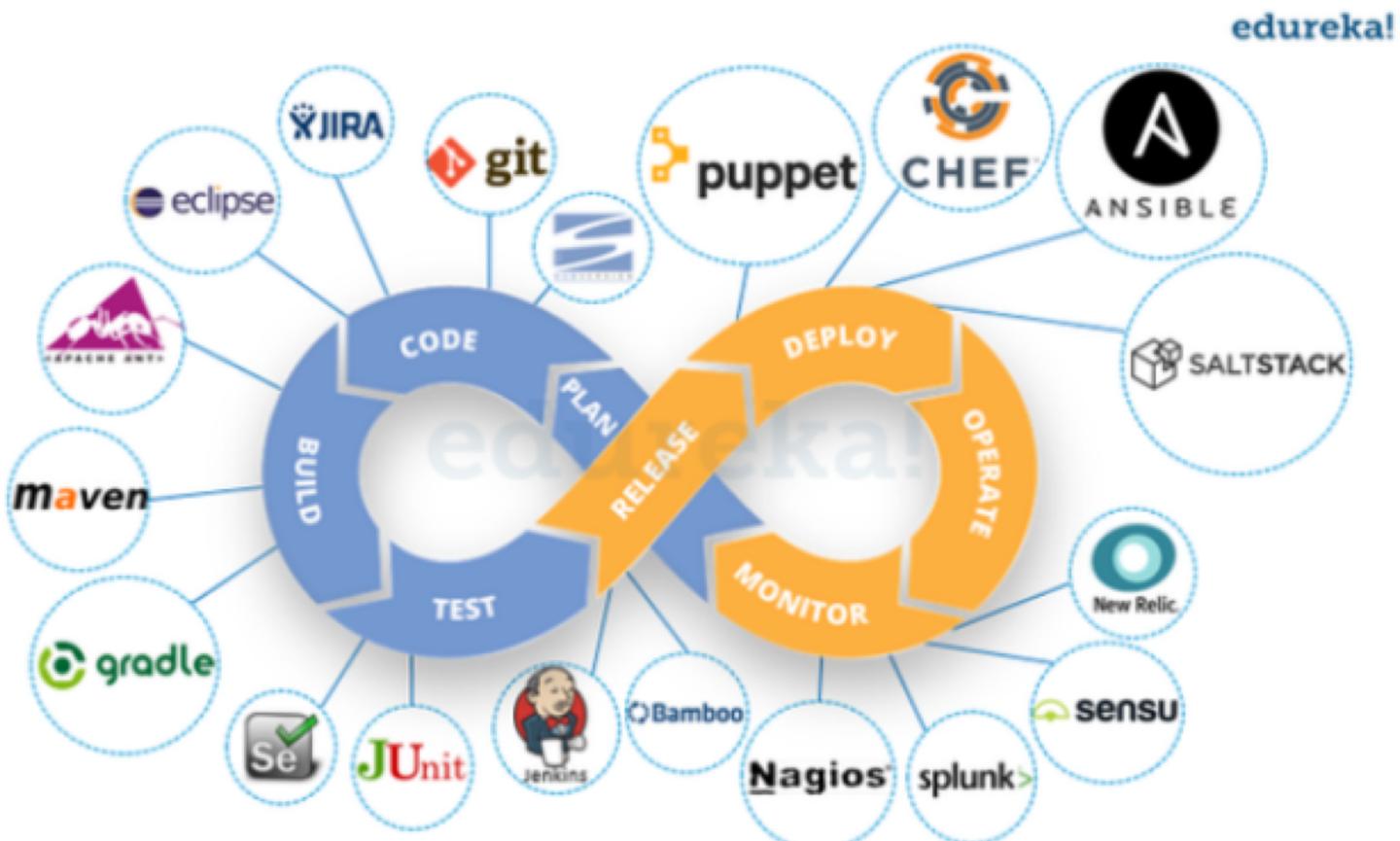
- DevOps – Unir desarrollo de operaciones con el objetivo de acelerar drásticamente la puesta en producción de nuevas funcionalidades.
- Enfoque de microservicios es particularmente apropiado para esto por:
 - Desarrollo descentralizado
 - Independencia del resto de la aplicación
 - Puede ser probado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

La revolución de DevOps

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



Automatización de DevOps



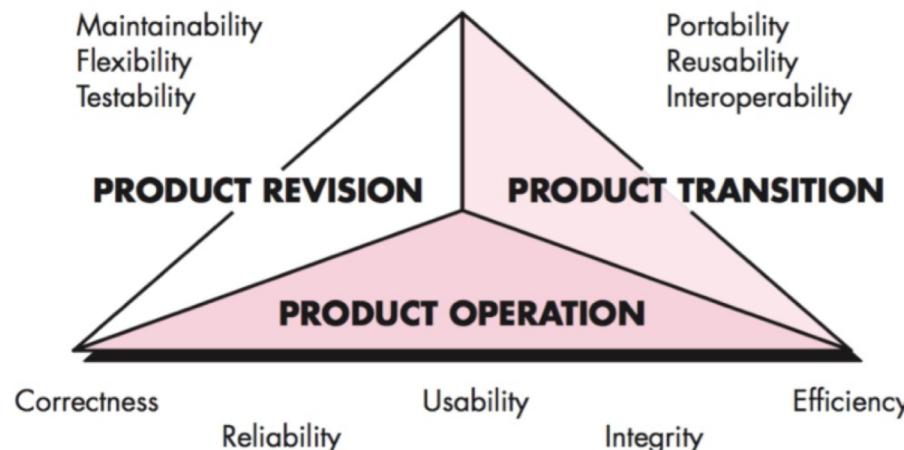
¿Por qué es difícil con arquitecturas monolíticas?

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

Hacia ausencia de defectos

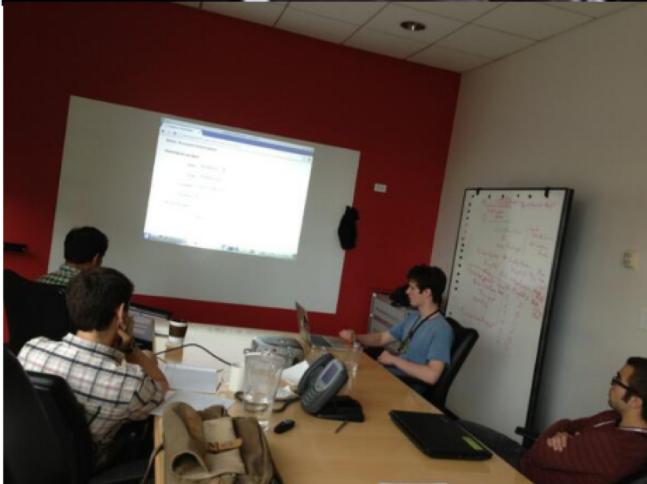
- No incorporarlos al construir el *software* (muy difícil)
- Análisis estático
 - Se examina el código con herramientas de *software* en busca de problemas o cosas sospechosas (código duplicado, peligros de seguridad, manejo de errores, entre otros)
- Inspección formal de código
 - Código es examinado 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

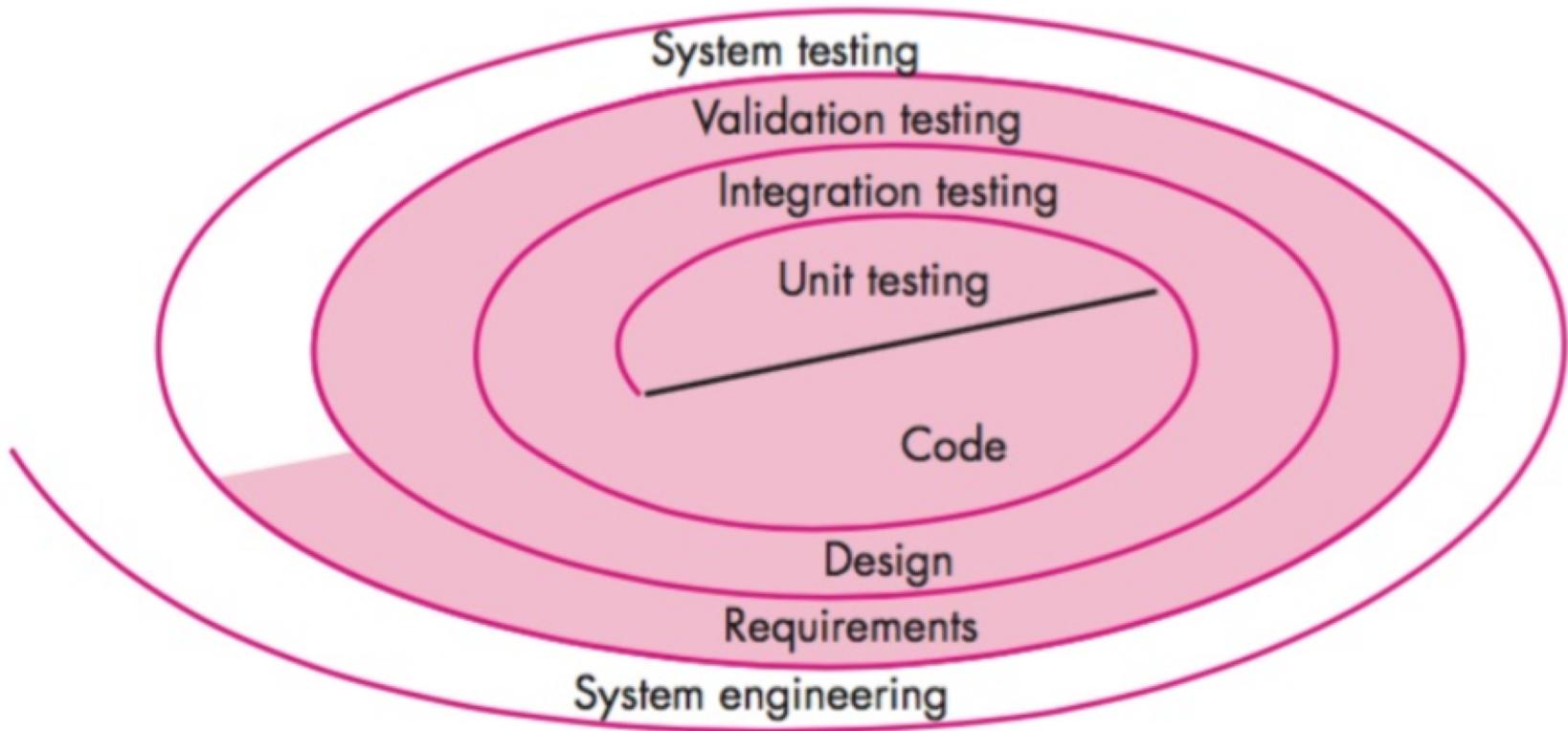
- Forma efectiva para producir código de calidad
- Código es revisado por equipo de pares con el objetivo de detectar la mayor cantidad de defectos
- Equipo revisor se prepara antes de sesión de revisión
- Pueden usarse *checklists* para búsquedas más dirigidas
- Ejemplos: *Pull Requests* o sesiones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



Testing

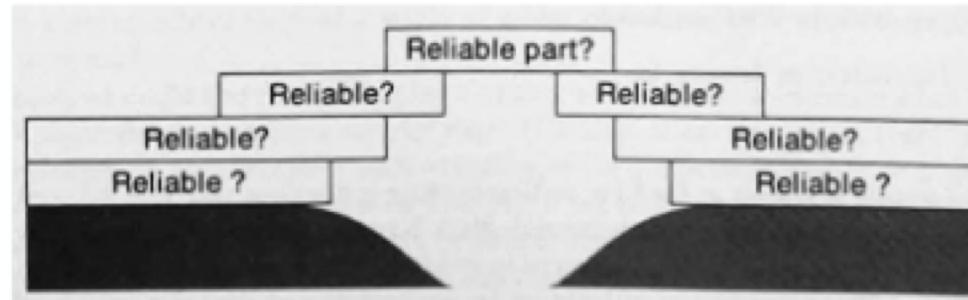
- Proceso de ejecutar un programa con el objetivo de encontrar un error
- Un buen caso de prueba es uno con una alta probabilidad de encontrar un error oculto
- Un *test* exitoso es aquel que descubre un error que no se conocía

Niveles de *tests*



Tests unitarios

- Se centran en verificar las unidades más pequeñas del *software* (componentes y módulos)
- Se pueden realizar antes, durante o después de la codificación
- Se debe tener un control de los resultados esperados (*inputs* y *outputs*)



Tests de Blackbox y Whitebox

- *Blackbox*: No se conocen detalles del *software*, solo se considera *input* y *output*
- *Whitebox*: Se conocen los detalles del *software* y se puede utilizar en diseño del *test*

Coverage

- Una métrica que mide la proporción de las líneas únicas de código fuente que son ejecutadas por una batería de *tests*.
- Distintos criterios de medición:
 - ***Function coverage***: proporción de métodos que son invocados
 - ***Statement coverage***: proporción de instrucciones ejecutadas
 - ***Branch coverage***: proporción de caminos independientes recorridos
 - ***Condition coverage***: proporción de predicados probados

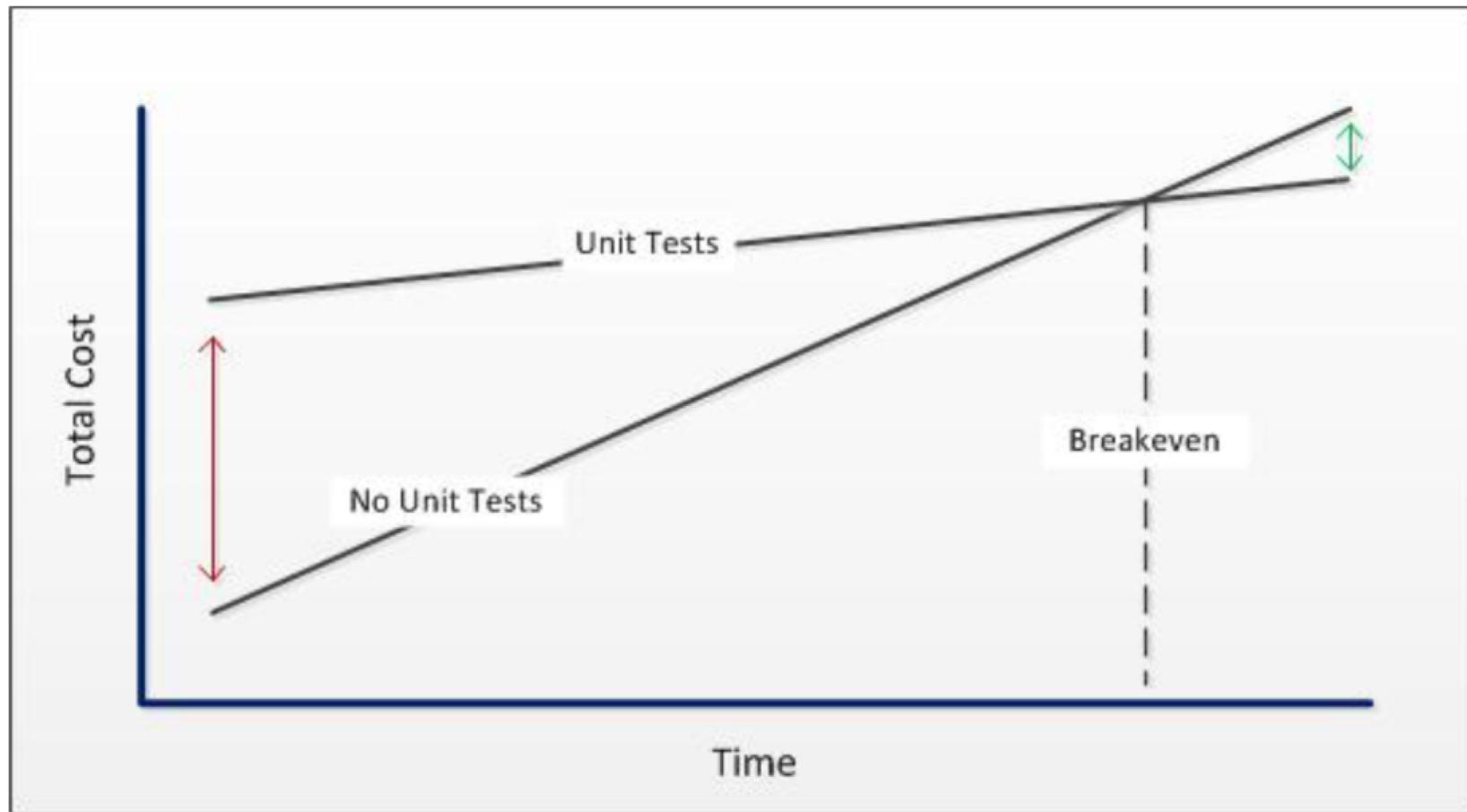
Tests unitarios: Beneficios

- Permiten hacer cambios al código de manera segura
- Ayudan a entender el diseño y funcionalidades a programar
- Sirven como apoyo a la documentación (como ejemplos)

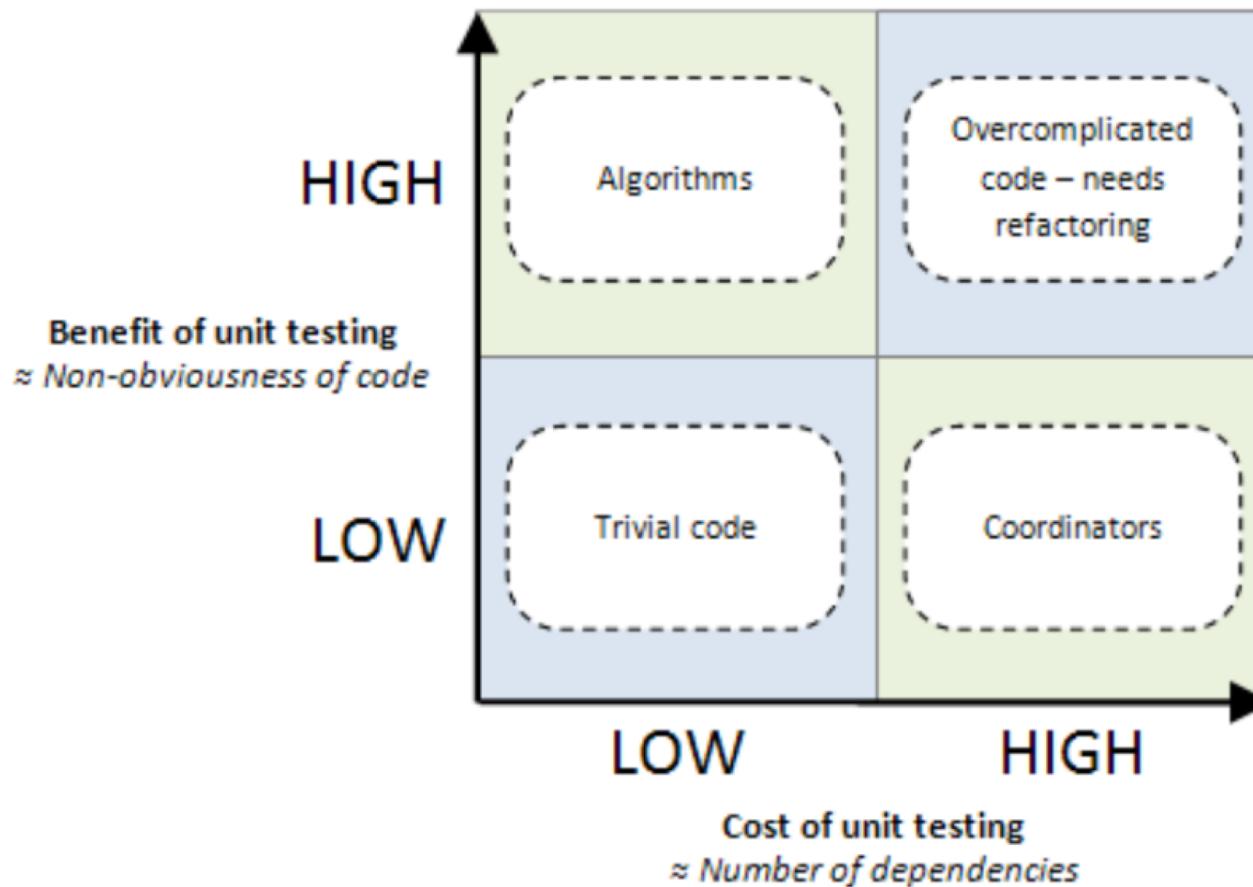
Tests unitarios: Costos

- Consumen más tiempo en el corto plazo
- No todos los *tests* agregan el mismo valor
- No representan ni garantizan la calidad del código

Tests unitarios: Costos



Tests unitarios: Costos



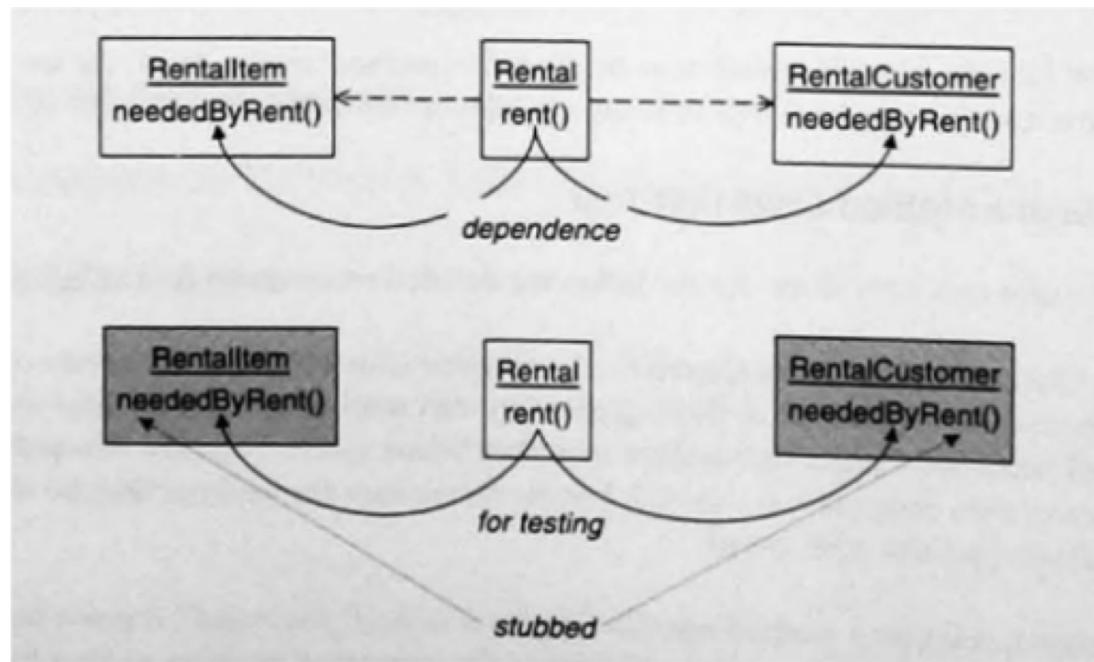
Tests de integración

- Cada uno de los componentes puede tener pruebas unitarias, pero aún así el sistema necesita pruebas



Uso de *stubs*

- Puede ser necesario escribir clases/métodos que no existen para probar el que nos interesa
- En ese caso se escribe lo mínimo necesario



Stubs y Mocks

- ***Stubs***: son imitaciones de objetos, que proveen resultados predefinidos para ciertas invocaciones sobre ellos. Son útiles para probar de forma aislada los componentes.
- ***Mocks***: son imitaciones de objetos, de las cuales se espera que ciertos métodos sean invocados durante un *test*. De no ser así, el *test* falla.

Tests de Aceptación

- Validación efectuada por el cliente con el objeto de aceptar o rechazar el producto desde una mirada funcional
- Se pueden utilizar:
 - *tests* de usabilidad
 - *A/B testing*
 - *Alpha tests*
 - *Beta tests*

Tests de usabilidad



A/B testing

- Ampliamente utilizado con sitios Web
- Se ponen a prueba dos versiones de la página y se ve cual funciona mejor



Alpha/Beta testing

- Pruebas de la aplicación simulando ambiente real de utilización
- *Alpha testing:*
 - Usuarios o desarrolladores prueban el producto en el sitio de desarrollo bajo la dirección de un encargado de las pruebas
- *Beta testing:*
 - Número de usuarios controlado prueba libremente el producto en su propio ambiente de uso

Tests de sistema

- Pruebas de la aplicación en su ambiente de ejecución
- Validan los requisitos no funcionales del sistema:
 - Performance: que cumpla con los requerimientos de desempeño
 - Carga: cómo se comporta el sistema bajo carga de usuarios
 - Stress: cómo responde el sistema bajo una carga mayor para la cual fue diseñado
 - Seguridad: ataques simulados para encontrar fortalezas y debilidades del sistema

Tests de sistema

- Ejemplos:
 - [loader.io](#)
 - [blazemeter.com](#)
 - [jmeter.apache.org](#)
 - [bugcrowd.com](#)



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

Clase 20

Testing

IIC2143 - Ingeniería de Software
Sección 1

Rodrigo Saffie

rasaffie@uc.cl

30 de mayo de 2018