Clase 20: Testing

Rodrigo Arturo Saffie Kattan

Pontificia Universidad Católica de Chile

rasaffie@ing.puc.cl

18 de octubre de 2016

Contenidos

¿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 defectuoso

Validación y Verificación

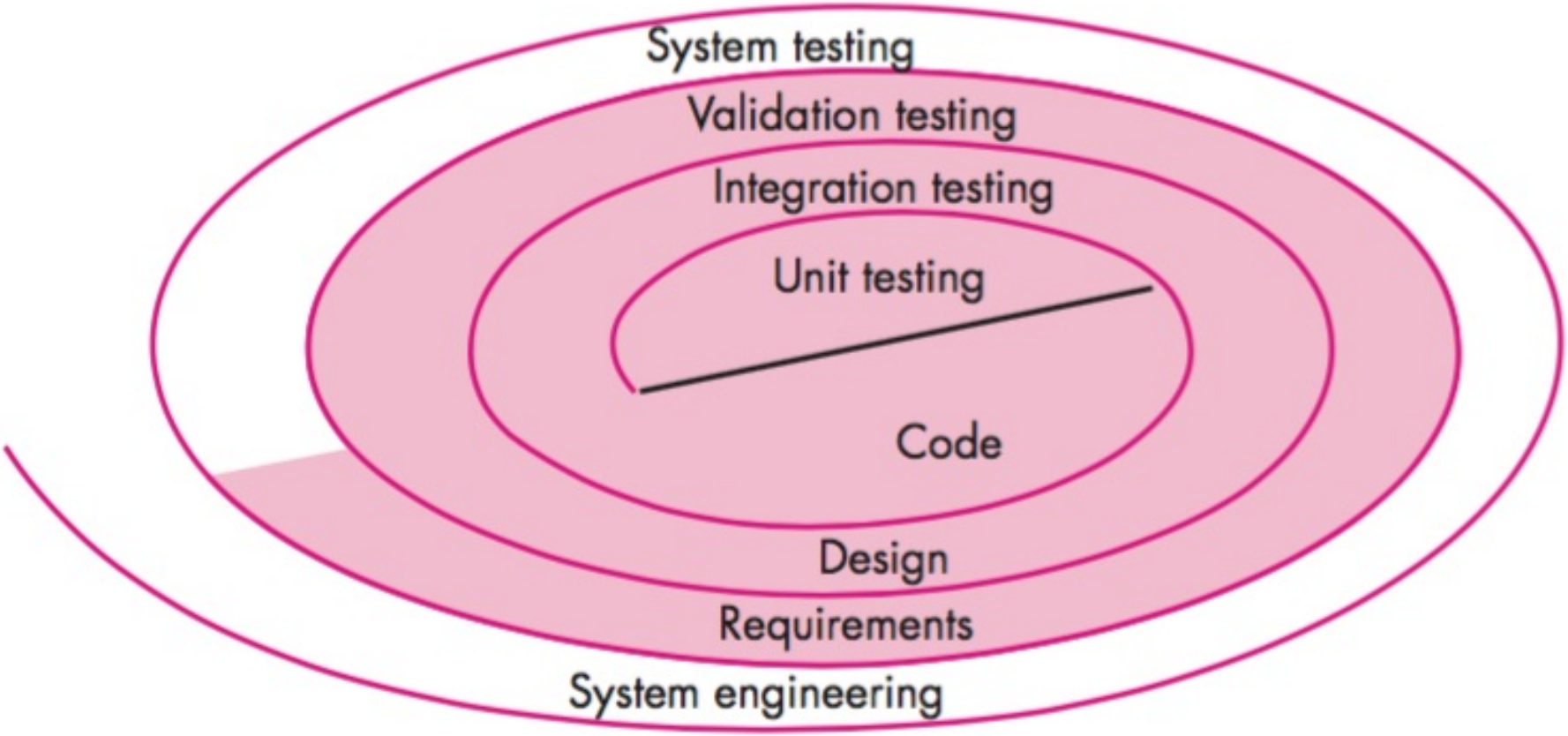
Validación:

¿Estamos construyendo el producto **correcto**?

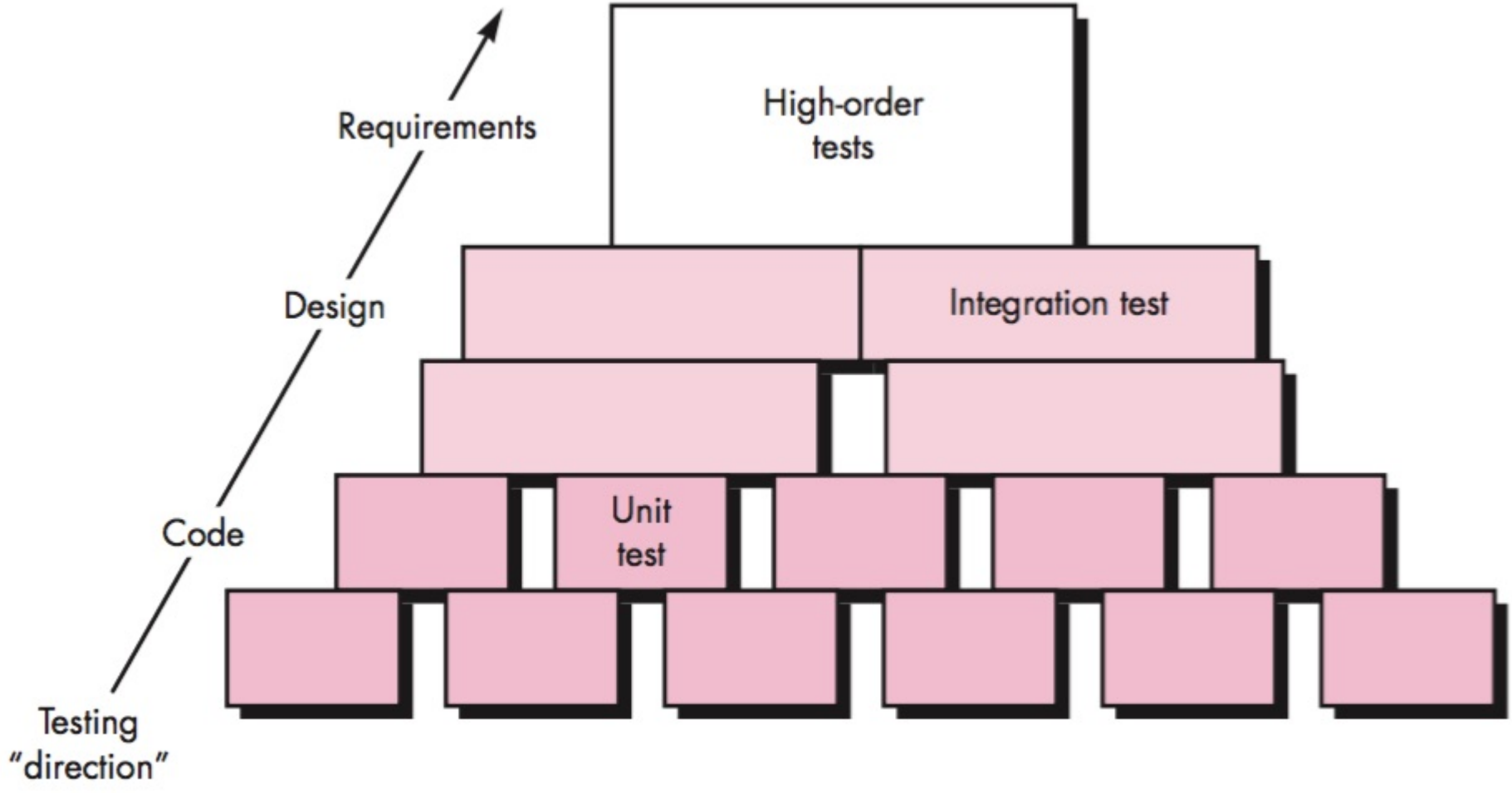
Verificación:

¿Estamos construyendo el producto **correctamente**?

Validación y Verificación



Validación y Verificación

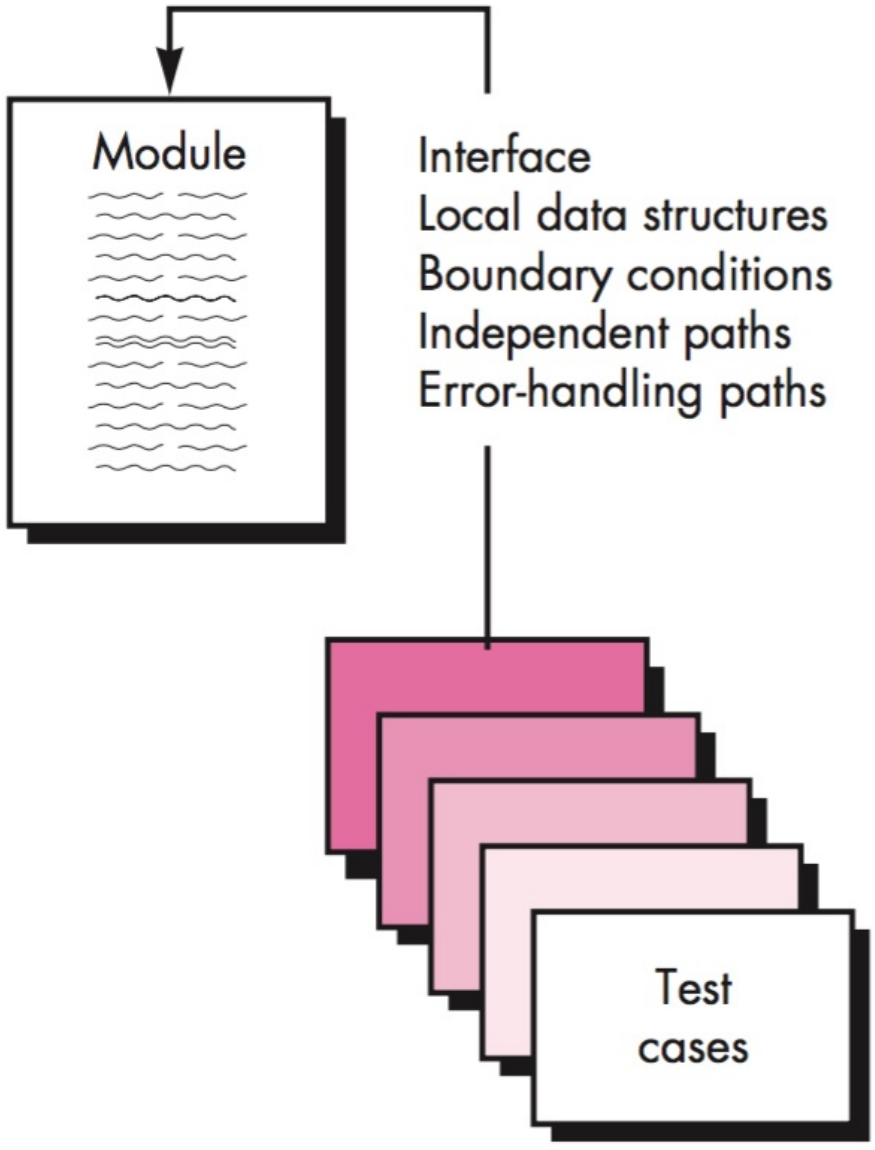


Unit Testing

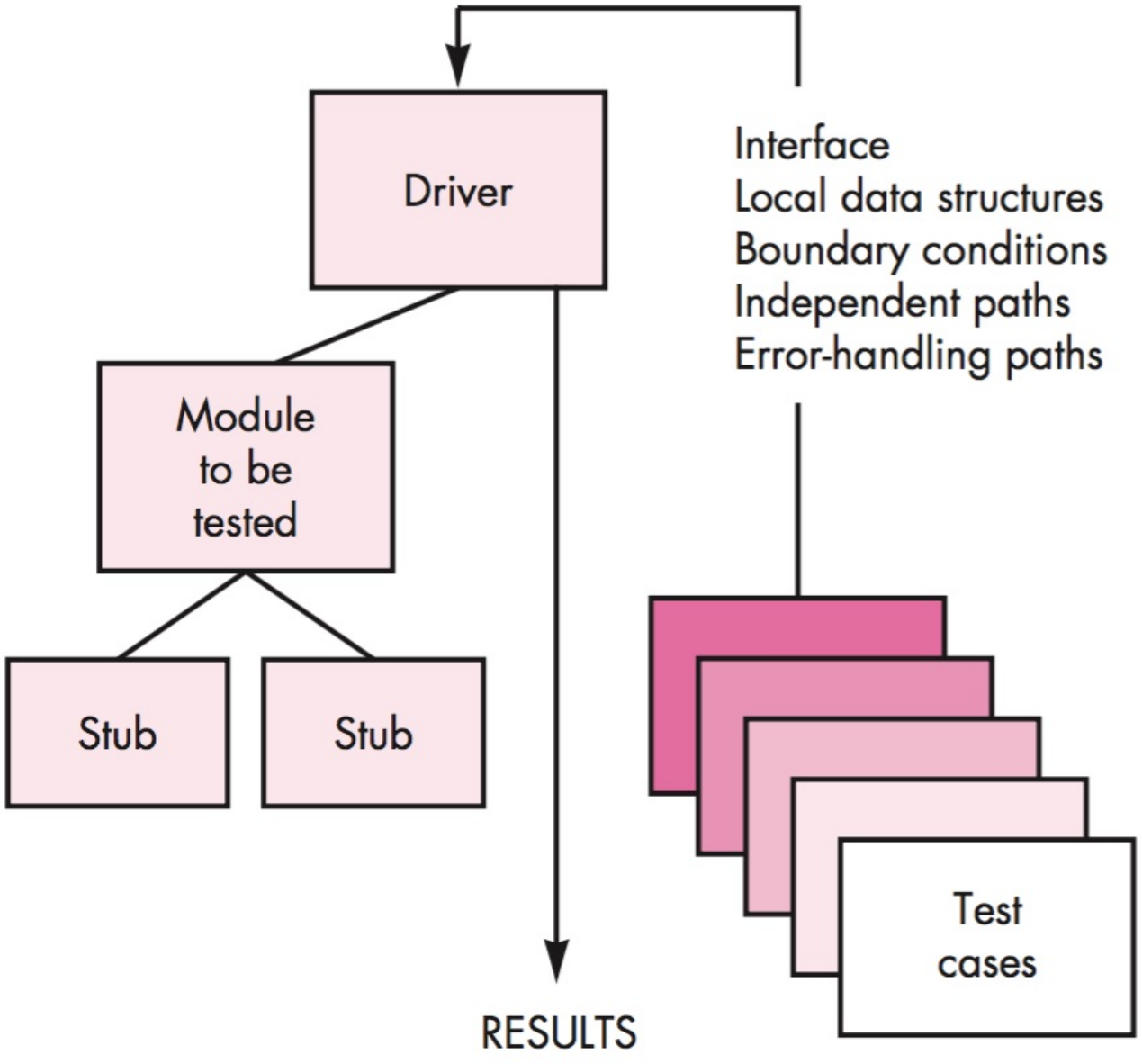
Se centran en verificar las unidades más pequeñas del software (componentes y módulos).

- Se puede realizar antes, durante o después de la codificación
- Se debe tener un control de 'resultados esperados'
- Realizarlo en componentes con clara cohesión es más simple

Unit Testing



Unit Testing



Unit Testing

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.

Stubs:

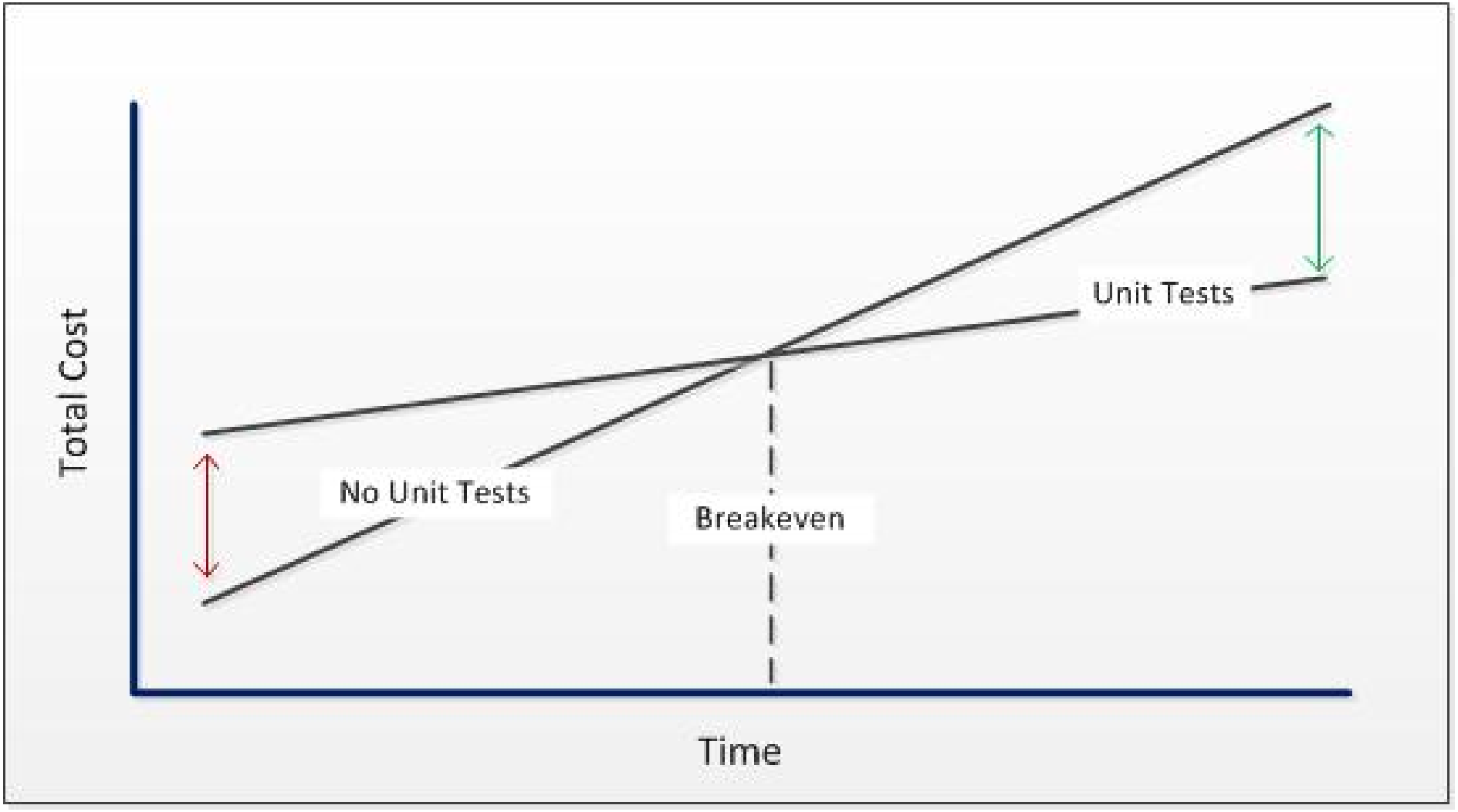
Son imitaciones de objetos, que proveen resultados predefinidos para ciertas invocaciones sobre ellos. Son útiles para probar de forma aislada los componentes.

Unit Testing

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)

Unit Testing

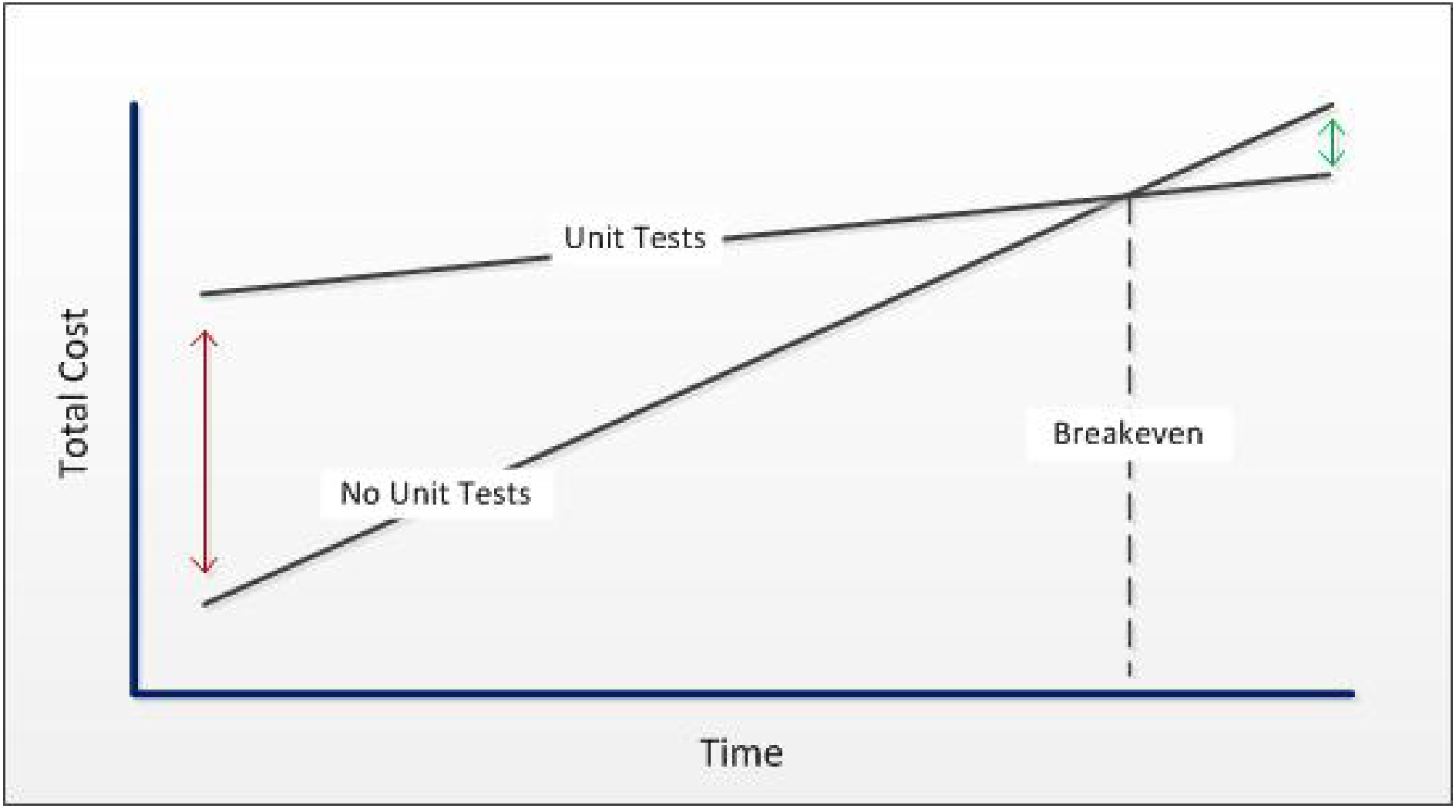


Unit Testing

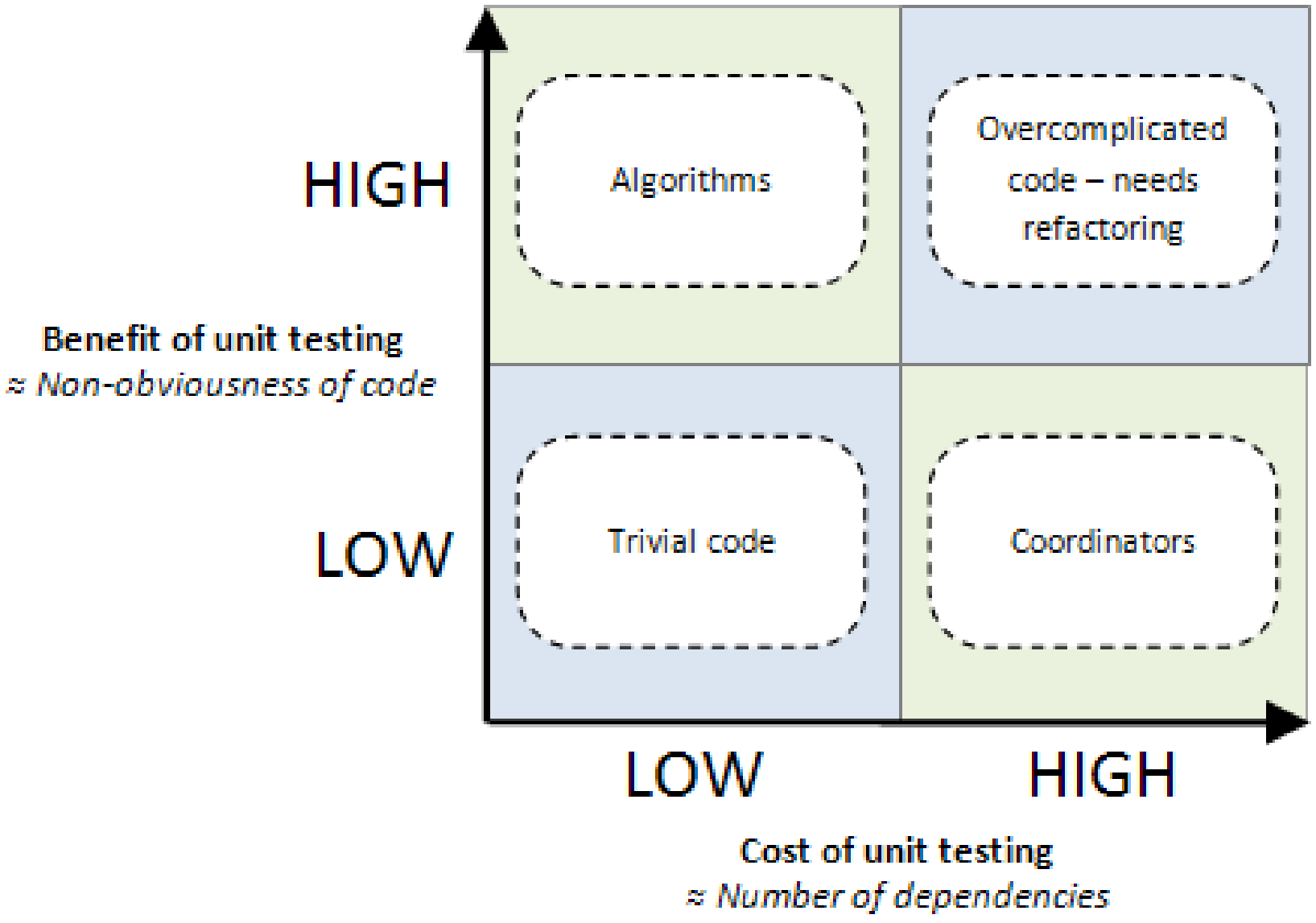
Costos:

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

Unit Testing



Unit Testing



Unit Testing

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** (or predicate coverage): proporción de predicados probados

Unit Testing

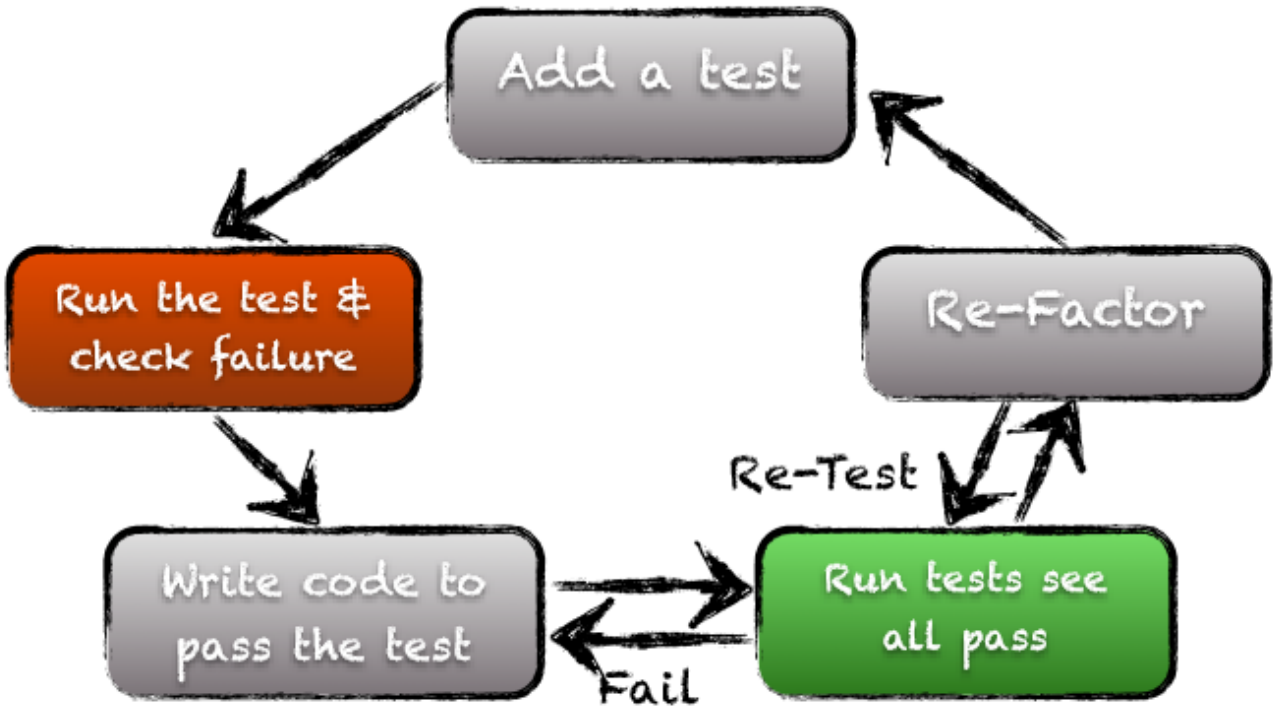
Mitos:

- Es más costoso y lento, porque se escribe más código
- El programa es de 'alta calidad' si tiene gran *coverage* con tests que pasan
- Es más difícil hacer cambios, porque además hay que cambiar tests
- Si el código es simple no necesita tests
- Si se cuenta con tests de integración y aceptación, no se necesitan tests unitarios

Test Driven Development

Test Driven Development: desarrollar código en base a ciclos pequeños e iterativos, que incluyen:

- Diseñar tests para un requerimiento
- Desarrollar código hasta que los tests pasen
- Mejorar la implementación



Referencias

Pressman, R. S. (2009)

Software Engineering: A Practitioner's Approach

7th ed., *McGraw-Hill Education*

Martin Fowler (2007)

http://martinfowler.com/articles/mocksArentStubs.html

Tom Fischer (2012)

https://www.simple-talk.com/dotnet/net-framework/unit-testing-myths-and-practices/

Steve Sanderson (2009)

http://blog.stevensanderson.com/2009/11/04/selective-unit-testing-costs-and-benefits/

Daniel Lynn (2016)

http://www.agile42.com/en/blog/2016/02/18/making-time-refactoring/

Fin