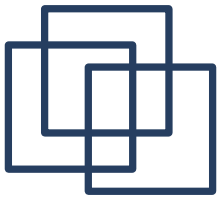
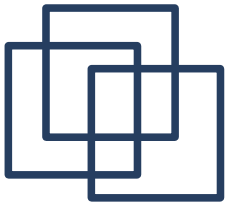
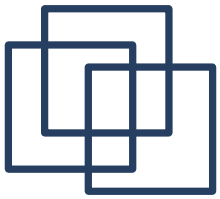


Testing



If debugging is the process of removing bugs, then programming must be the process of putting them in.

Edsger W. Dijkstra



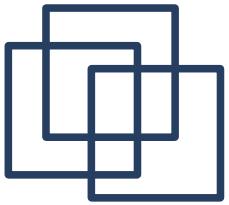
Probar / testear / depurar

- ¿Qué prácticas utilizaban hasta ahora para probar, testear, depurar su código?

- ¿Imprimir en consola?
  - console.log(”lleguo hasta acá 1”);
- ¿Breakpoints?
- ¿Inspectores / watches ?

- ¿Pros / contras? ¿Cuándo y para que sirven?


Testing de Unidad con SUnit



Generación de Bugs

- Comunicación imperfecta
- Optimismo mágico
  - Cambios menores, grandes errores

Testing sistemático

- Determinar que hace un artefacto
- Determinar que no hace un artefacto

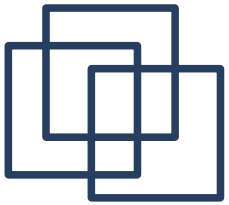
- Mantener funcionalidad minima

- Tener una medida de avance
- Tener una medida de éxito
- Estimaciones más realistas

El artefacto debe:

Hacer lo que se espera

No hacer lo que no se espera



Paradoja del Testing

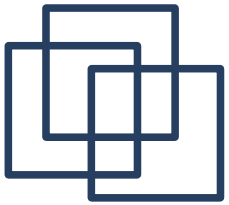
- Escribir casos de testing es deseable
- Escribir casos de testing costoso y aburrido(?)

- Testear todos los métodos no es práctico
  - Ej:

```
name
  ^name
```

```
name: aString
  name := aString
```

Objetivo: min. los casos y max. 'covertura'

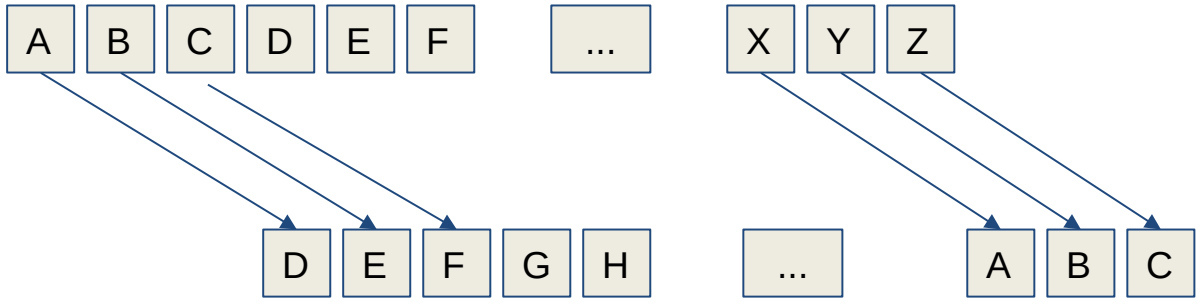


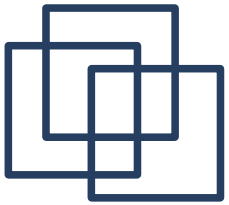
Ejemplo Motivador: Privacidad de Mensajes (CesarCipher)

Cifrado de Texto

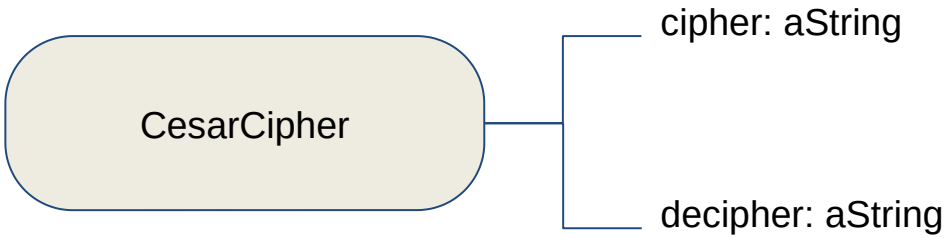
Un texto cifrado es el resultado de aplicar un algoritmo que ha ofuscado un texto original (plain text)

El cifrado de César es un cifrado de sustitución en el que una letra en el texto original es reemplazada por otra letra que se encuentra un número fijo de posiciones más adelante en el alfabeto.





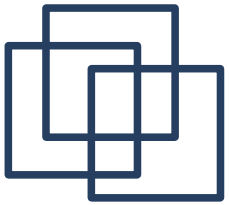
Cifrado Texto: Objetos



Considerando que existe un objeto que implementa el cifrado de César (CesarCipher) y que el objeto entiende dos mensajes:

- cipher:
- decipher:

Cómo deberiamos testear que la implementación del cifrado de César es correcta?



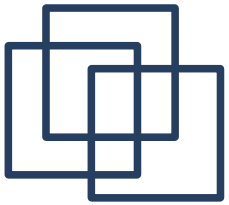
Particiones Equivalentes

- Definir conjuntos de datos
  - si un elemento pasa el test, otros del mismo conjunto pasarán

  {CesarCipher}
  {CesarCipher de caracteres que no producen “loopback”}
  {CesarCipher de caracteres que producen “loopback”}

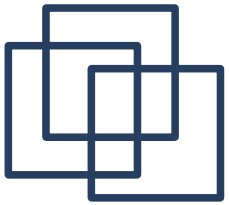
- Asegurarse que el artefacto no hace lo que no debe

  CesarCipher no agrega o elimina caracteres

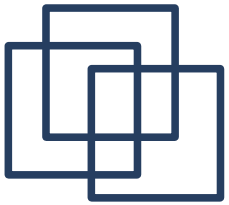


Valores de Borde

- La mayoría de los errores ocurren en los bordes o límites entre conjuntos
  - `Cifrado de caracteres que producen loopback`
  - `Cifrado con alfabeto nulo`
  - `Cifrado con cadena de texto vacia`
- Cuál es el comportamiento esperado si el caracter no es parte del alfabeto?

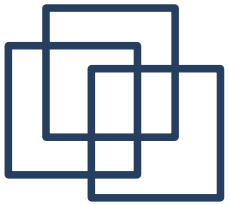


Sunit
-Framework for Unit Testing-



SUnit

- Framework de testing para desarrollar 'test cases' en Smalltalk
- Originalmente desarrollado por Kent Beck.
- Existen implementaciones para muchos lenguajes.
  - JUnit, CUnit, PyUnit, etc
- Existen extensiones testear ventanas y tambien web applications.



Sunit's Customization

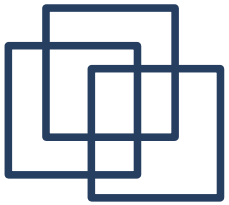
Crear subclase de `TestCase`

Realizar toda inicialización necesaria en `setUp`

Realizar toda limpieza necesaria en `tearDown`

Para la funcionalidad Y en la clase X, tener un método #testY en la clase TestX.
Usar:

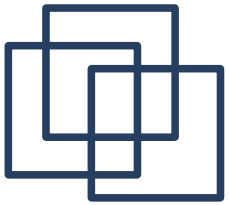
a)self `assert:` (expresión booleans).
b)self `deny:` (expresión booleans).



Crear una clase

```
TestCase subclass: #CesarCipherTest
  instanceVariableNames: 'cipher'
  classVariableNames: ''
  category: 'CipherTest'
```

Mars Global Surveyor



setUp y tearDown

setUp

```
cipher := self cesarCipher
```

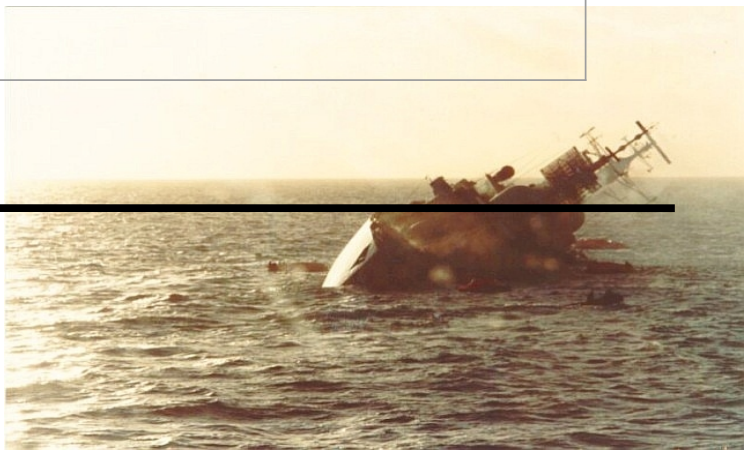
tearDown

```
cipher := nil
```

K-19

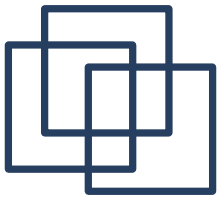
Casos de testing

```
testCipherCharacters
    self assert: (cipher cipher: 'X') equals: 'A'.
    self assert: (cipher cipher: 'A') equals: 'D'.
    self assert: (cipher cipher: 'W') equals: 'Z'
```



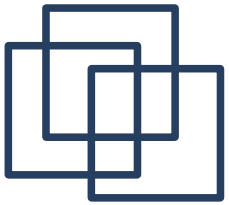
```
testCipherDecipher
    self assert: (cipher decipher: (cipher cipher: 'WIKIPEDIA'))

        equals: 'WIKIPEDIA'
```



¿Verde quiere decir terminado?

- Los tests de unidad confirman que:
  - Nuestro programa (o parte) hace lo que dijimos que hace
  - No hace algo que no queremos que haga
- Pero puede ser que nuestro código todavía tenga problemas que lo hacen dificil de entender y mantener
  - Antes de quedarnos tranquilos, debemos chequear nuestro programa cumple con los criterios y heuristicas de diseño vistos
  - No podemos automatizar esta parte (o casi no podemos porque Pharo nos da algunas pistas con su Critic Browser)



Resumen

- Test Unidad implementa un mecanismo para probar la implementacion de métodos de un objetos
- Cada test case debería implementar 1 aspecto de la funcionalidad
- Todos los test cases conforman una “libreria” de pruebas con valor para:
  - dar por terminada una tarea => da una medida tangible de avance
  - para hacer test de regresión
  - ayudar en la integración de versiones (branchs) en el repositorio de codigo
- **Particiones Equivalentes es complemetario de “Valores de Borde”**