

Clase 3

Programación extrema

(Parte 2)



1. Implementación

Conceptos involucrados con una implementación XP:

- **Cliente disponible**
- **Estándares de codificación**
- **Implementación dirigida por pruebas**
- **Programación en parejas**
- **Integración secuencial**
- **Propiedad colectiva del código**
- **Ritmo constante**

1. Implementación

1.1. Cliente disponible

Uno de los requisitos de XP es tener al cliente disponible, no sólo para ayudar al equipo de desarrollo, sino para **ser parte del mismo**.

Todas las fases de XP requieren comunicación con el cliente.



1. Implementación

1.1. Cliente disponible

Las historias de usuario son escritas por el cliente con la ayuda de los desarrolladores, además de establecer la prioridad de las mismas.

Su presencia asegura que los desarrollos cubran **toda la funcionalidad** descrita.

1. Implementación

1.1. Cliente disponible

Durante la reunión de planificación el cliente negocia las historias que se incluirán en la próxima entrega, así como su duración.

El cliente es **imprescindible** a la hora de realizar pruebas funcionales, en caso de error él será el encargado de decidir si el código **puede pasar a producción o no**.



1. Implementación

1.2. Estándares de codificación

El código debe seguir los estándares de codificación.

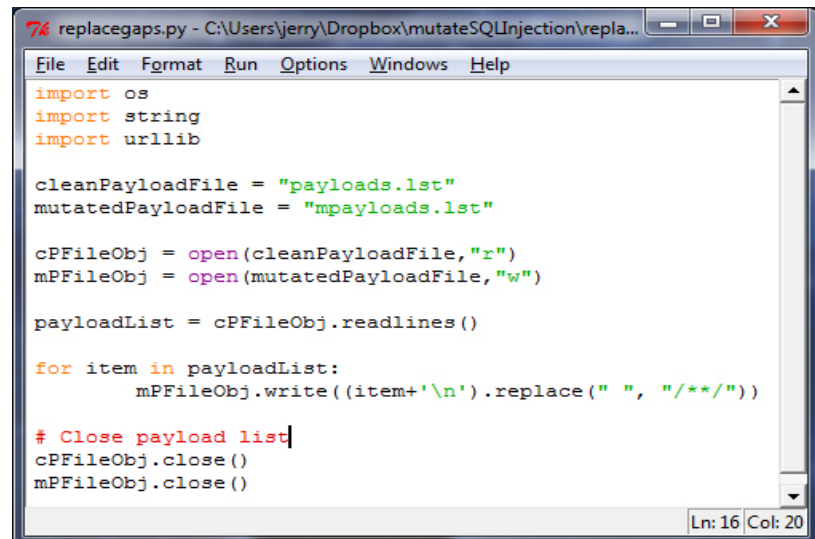


1. Implementación

1.2. Estándares de codificación

Todos los programadores deben **escribir** y **documentar** el código en la misma manera.

Las normas de codificación ayudan a mantener el código legible y fácil de mantener y refactorizar.



```
replacegaps.py - C:\Users\jerry\Dropbox\mutateSQLInjection\repla...
File Edit Format Run Options Windows Help
import os
import string
import urllib

cleanPayloadFile = "payloads.lst"
mutatedPayloadFile = "mpayloads.lst"

cPFileObj = open(cleanPayloadFile,"r")
mPFileObj = open(mutatedPayloadFile,"w")

payloadList = cPFileObj.readlines()

for item in payloadList:
    mPFileObj.write((item+'\n').replace(" ", " /**/"))

# Close payload list
cPFileObj.close()
mPFileObj.close()
Ln: 16 Col: 20
```

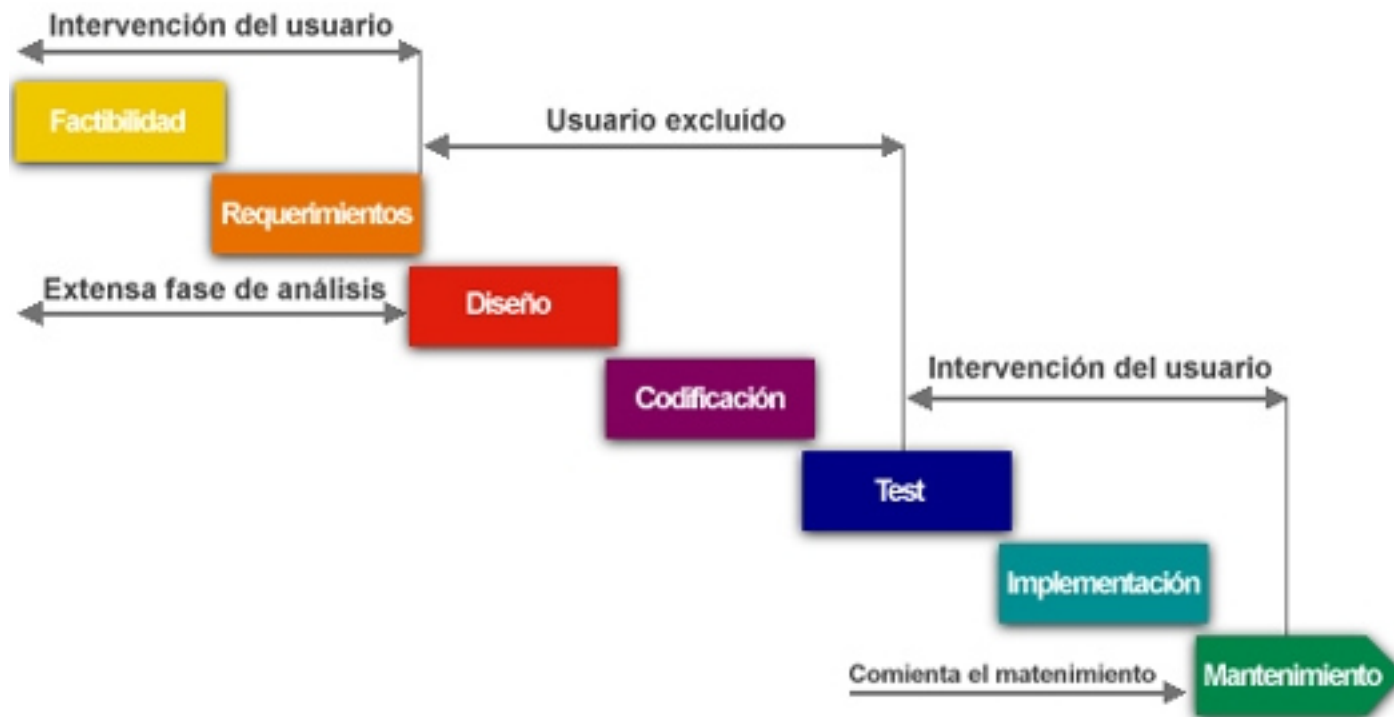
1. Implementación

1.3. Implementación dirigida por pruebas

En las metodologías tradicionales, la fase de pruebas, incluyendo la definición de los *tests*, se realiza **al final** del proyecto, o al final del desarrollo de cada módulo.

1. Implementación

1.3. Implementación dirigida por pruebas



1. Implementación

1.3. Implementación dirigida por pruebas

XP propone un modelo inverso, en el cual primero se escriben los *test* que el sistema debe pasar.

El desarrollo debe ser el **mínimo necesario** para pasar las pruebas previamente definidas. La definición de estos *test* al comienzo, **condiciona y dirige el desarrollo**.

1. Implementación

1.4. Programación en parejas

XP promueve que todo el código sea escrito en parejas trabajando en la misma computadora.

La programación en parejas **incrementa la calidad del código** sin impactar en la fecha de entrega.



1. Implementación

1.4. Programación en parejas

En contra de lo que parece, dos personas que trabajan en un mismo equipo añadirán la misma funcionalidad que dos personas trabajando por separado, excepto que el código será de mucha mayor **calidad**.



1. Implementación

1.4. Programación en parejas

Ventajas:

- a) La mayoría de los errores se descubren en tiempo de codificación, dado que el código es permanentemente revisado por dos personas.
- b) La cantidad de defectos encontrados en las pruebas es estadísticamente menor.

1. Implementación

1.4. Programación en parejas

Ventajas:

c) Los diseños son mejores y el código más corto.



1. Implementación

1.4. Programación en parejas

Ventajas:

d) El equipo resuelve problemas en forma más rápida.

e) Las personas aprenden significativamente más, acerca del sistema y acerca de desarrollo de software.

1. Implementación

1.4. Programación en parejas

Ventajas:

f) El proyecto termina con más personas que conocen los detalles de cada parte del código.



1. Implementación

1.4. Programación en parejas

Ventajas:

g) Las personas aprenden a trabajar juntas, generando mejor dinámica de grupo y haciendo que la información fluya rápidamente.

h) Las personas disfrutan más de su trabajo.

1. Implementación

1.5. Integración secuencial

Todos los desarrolladores necesitan trabajar siempre con la **última versión**.

Realizar cambios o mejoras sobre versiones antiguas causan graves problemas, y retrasan al proyecto.

1. Implementación

1.6. Propiedad colectiva del código

La propiedad colectiva anima a todos los miembros del equipo a aportar nuevas ideas.



1. Implementación

1.6. Propiedad colectiva del código

Cualquier desarrollador puede cambiar líneas de código para añadir funcionalidad, solucionar errores, mejorar el diseño o refactorizar el código.

1. Implementación

1.6. Propiedad colectiva del código

Cuando una persona realiza un desarrollo tiene que subir al repositorio el código más sus pruebas unitarias funcionando al 100%.

Así cualquier otra persona que se lo descargue puede confiar en que tiene un código que funciona y desarrollar a partir del mismo.

1. Implementación

1.7. Ritmo constante

La metodología XP indica que debe llevarse un **ritmo sostenido de trabajo**.



1. Implementación

1.7. Ritmo constante

Tradicionalmente esto se denominaba “semana de 40 horas”. Sin embargo, lo importante para XP no es si se trabajan 30, 40 ó 50 horas por semana.



1. Implementación

1.7. Ritmo constante

El concepto que se desea establecer con esta práctica es el de planificar el trabajo de manera de mantener un ritmo constante y razonable, **sin sobrecargar al equipo**.

1. Implementación

1.7. Ritmo constante

Cuando un proyecto se retrasa, trabajar tiempo extra puede ser más perjudicial que beneficioso.

El trabajo extra desmotiva inmediatamente al grupo e impacta en la calidad del producto.



1. Implementación

1.7. Ritmo constante

En la medida de lo posible, se debería renegociar el plan de entregas realizando una nueva reunión de planificación con el cliente, los desarrolladores y los gerentes.

Además, agregar más desarrolladores en proyectos ya avanzados no siempre resuelve el problema o acelera el progreso.

2. Pruebas

Las pruebas unitarias son una de las **claves** de XP.

Todos los módulos deben pasar las pruebas unitarias antes de ser liberados o puestos en producción.



2. Pruebas

La modalidad es “*Test-driven programming*”, es decir, las pruebas deben ser definidas antes de realizar el código.

El sistema y el conjunto de pruebas debe ser guardado junto con el código, para que pueda ser utilizado por otros desarrolladores en caso de tener que corregir, modificar o recodificar parte del mismo.

3. Valores en XP

Los estándares de XP destacan estos 4 valores:

- **Simplicidad**
- **Comunicación**
- **Retroalimentación**
- **Coraje**

3. Valores en XP

3.1. Simplicidad

Hacer exactamente
lo que se ha
pedido, **no más**.



3. Valores en XP

3.2. Comunicación

La comunicación entre los componentes del equipo XP es fundamental.

Dado que la documentación es escasa, el **diálogo cara a cara** entre desarrolladores, gerentes y usuarios es el medio básico de comunicación.

3. Valores en XP

3.2. Comunicación

Una buena comunicación tiene que estar presente durante todo el proyecto.



3. Valores en XP

3.3. Retroalimentación

Siempre hay que tener en cuenta la valoración o *feedback* del cliente una vez que se hace una entrega, e intentar mejorar haciendo cambios en el proceso si es necesario.



3. Valores en XP

3.4. Coraje

Se trata de que el equipo asuma la responsabilidad de su trabajo, tanto si es un éxito como un fracaso, además de ser **emprendedor** a la hora de implementar cambios en la aplicación, por ejemplo con refactorizaciones.

4. Roles

La programación extrema implica la existencia de roles, que normalmente son:

- **Cliente**
- **Programador**
- **Encargado de pruebas (*tester*)**
- **Encargado de seguimiento (*tracker*)**
- **Entrenador (*coach*)**
- **Gestor (*big boss*)**

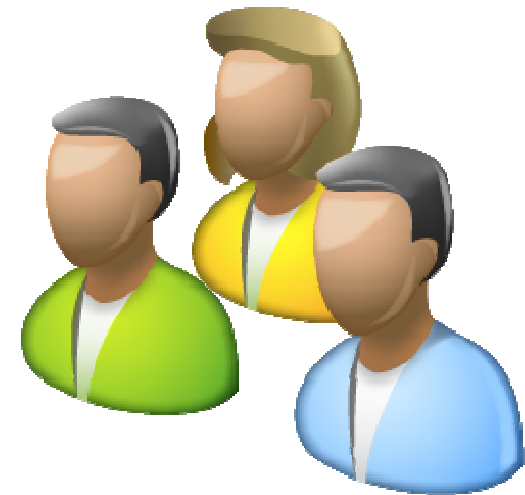
4. Roles

4.1. Cliente

El cliente o usuario es el responsable de **conducir el proyecto**.

Define el proyecto y sus objetivos.

Cuanto más preciso es su trabajo y cuanto más se involucre, mayores serán las oportunidades de éxito.



4. Roles

4.1. Cliente



4. Roles

4.1. Cliente

Toma **decisiones de negocio** y debe entender los cambios del mismo a largo del tiempo, identificando:

- Si una historia tiene el mismo valor ahora que cuando se identificó;
- Si se puede retrasar o simplificar.

4. Roles

4.1. Cliente

Debe responder a preguntas como:

a) ¿Qué debería hacer esta nueva característica?

Colaborando con los desarrolladores, escribiendo las historias de usuario o requisitos, aclarando los detalles además de planificarlas.

4. Roles

4.1. Cliente

Debe responder a preguntas como:



b) ¿Cómo sabremos cuando está lista?

Creando baterías de tests de aceptación del producto, verificando que las nuevas características están completas.

4. Roles

4.1. Cliente

Debe responder a preguntas como:

c) ¿Cuánto podemos gastar? ¿Cuándo comenzamos a trabajar?

Participando en la reunión de planificación de las historias de usuario para la siguiente iteración.

4. Roles

4.1. Cliente

El cliente representa al usuario final y a los intereses económicos de la empresa.



4. Roles

4.1. Cliente

Derechos

- a) Maximizar la inversión, escogiendo las historias de usuario para cada iteración.
- b) Cambiar el alcance del proyecto para hacer frente a los cambios en la planificación, añadiendo o eliminando tareas de la iteración si las estimaciones demuestran ser incorrectas.

4. Roles

4.1. Cliente

Derechos

- c) Determinar que funcionalidades serán las siguientes en implementarse.
- d) Medir el progreso del proyecto en cualquier momento, lanzando las pruebas de aceptación.

4. Roles

4.1. Cliente

Derechos

e) Detener el proyecto en cualquier momento sin perder la inversión realizada, manteniendo en producción un producto estable y continuando con la planificación de otras funcionalidades más importantes.



4. Roles

4.1. Cliente

Responsabilidades

- a) Confiar en las decisiones técnicas de los desarrolladores.
- b) Analizar los riesgos correctamente.
- c) Seleccionar las historias que aportan más valor para cada iteración.

4. Roles

4.1. Cliente

Responsabilidades

- d) Definir las historias de usuario de forma precisa, permitiendo a los desarrolladores realizar estimaciones precisas.
- e) Participar en el equipo, dando directrices y feedback tan pronto y preciso como sea posible.

4. Roles

4.2. Programador

Una vez que se han comprendido las historias de usuario, XP propone adjudicar a los programadores la responsabilidad de tomar decisiones técnicas.



4. Roles

4.2. Programador

Los desarrolladores estiman el **tiempo** que les va a llevar cada historia de usuario y las transforman en código.



4. Roles

4.2. Programador

Deben responder a preguntas como:

- ¿Cómo implementamos esta funcionalidad?
- ¿Cuánto tiempo nos va a llevar?
- ¿Cuáles son los riesgos?

4. Roles

4.2. Programador

Derechos

- a) Estimación de su propio trabajo, teniendo autoridad para tomar decisiones técnicas.
- b) Delimitar el trabajo responsabilizándose de aquellas tareas que van a ser capaces de llevar a cabo.

4. Roles

4.2. Programador

Derechos

- c) Implementar sólo la funcionalidad que cubra las necesidades del cliente.
- d) No tomar decisiones de negocio.



4. Roles

4.2. Programador

Responsabilidades

- a) Seguir las directrices del equipo.
- b) Desarrollar las funcionalidades realmente necesarias.
- c) Comunicación fluida con el cliente.

4. Roles

4.3. Encargado de pruebas

El *tester* ayuda al cliente a definir y escribir las pruebas de aceptación de las historias de usuario.

Es responsable de realizar periódicamente las pruebas e informar de los resultados al equipo.



4. Roles

4.3. Encargado de pruebas

A medida que el volumen de pruebas aumenta, el encargado de pruebas necesitará una herramienta para crear y mantener la batería de pruebas, ejecutarlas y obtener los resultados más rápidamente.

4. Roles

4.4. Encargado de seguimiento

El *tracker* hace el seguimiento de acuerdo a la planificación.



4. Roles

4.4. Encargado de seguimiento

Una métrica clave para XP es la **velocidad del equipo**, que se define como el tiempo ideal estimado para las tareas frente al tiempo real dedicado.

Esta métrica ayuda a determinar si el proyecto está dentro del tiempo de la iteración.

4. Roles

4.4. Encargado de seguimiento

Para medir la velocidad del equipo, el encargado de seguimiento pregunta uno por uno a los desarrolladores cuántas tareas ha completado.



4. Roles

4.4. Encargado de seguimiento

El mejor procedimiento es preguntárselo en persona, en un ambiente informal y distendido. La sinceridad es fundamental por parte de los desarrolladores, y el encargado de seguimiento no debe entrar en valoraciones.

4. Roles

4.4. Encargado de seguimiento

Esta métrica ayuda a controlar el flujo del proyecto en posteriores iteraciones.



4. Roles

4.5. Entrenador

El *coach* no es un rol cubierto en todos los equipos de XP.

Su papel es guiar y orientar al equipo, especialmente cuando se comienza a trabajar por primera vez con XP.



4. Roles

4.5. Entrenador

Esto se debe a que no es fácil aplicar XP de forma consistente.

Aunque son prácticas de sentido común, se necesita un tiempo para interiorizarlas.

También hay situaciones especiales en las que se requiere la sabiduría de un especialista en XP para aplicar sus normas frente a un obstáculo en el proyecto.

4. Roles

4.5. Entrenador

El objetivo de un entrenador es que el equipo comprenda las directrices de XP.



No se trata de que sean solamente lecciones teóricas, si no de **dar el ejemplo** y proponer ideas para mejorar.

4. Roles

4.6. Gestor

El *big boss* es el gerente del proyecto. Debe tener una idea general del proyecto y estar familiarizado con su estado. El cliente también puede asumir este papel.

