



# it-Mentor

Pruebas de Software

<b>PRUEBAS DE SOFTWARE .....</b>	<b>3</b>
INTRODUCCIÓN .....	3
<i>Definiciones [1] .....</i>	<i>3</i>
<i>Filosofía y Economía .....</i>	<i>4</i>
<i>Justificación .....</i>	<i>4</i>
PRINCIPIOS [1] .....	7
NIVELES DE PRUEBAS .....	8
TIPOS DE PRUEBAS .....	8
CLAVES DEL CAMBIO EN LA FORMA DE TRABAJO .....	9
<i>Razones para automatizar las pruebas.....</i>	<i>9</i>
MÉTODOS DE PRUEBA .....	14
<i>Test incrementales.....</i>	<i>14</i>
<i>Top Down .....</i>	<i>14</i>
<i>Bottom Up .....</i>	<i>14</i>
<i>Caja Negra.....</i>	<i>15</i>
<i>Caja Blanca.....</i>	<i>16</i>
DISEÑO DE CASOS DE PRUEBAS.....	18
PRUEBAS FUNCIONALES Y DE ACEPTACIÓN .....	20
<i>Desde los casos de uso a los casos de pruebas.....</i>	<i>20</i>
<i>Automatización a partir del trabajo integrado – Fitnessse .....</i>	<i>21</i>
<i>Vinculo con el sistema bajo test.....</i>	<i>22</i>
<i>Fixture Clases.....</i>	<i>22</i>
<i>Diseño y edición de los test.....</i>	<i>23</i>
<i>Ejecución de los test .....</i>	<i>24</i>
<i>Historia de test.....</i>	<i>25</i>
<i>Integración con otros servers .....</i>	<i>25</i>
PRUEBAS DE CARGA Y STRESS.....	26
<i>Carga del servidor.....</i>	<i>26</i>
<i>Tiempo de respuesta de los queries y evolución de la dispersión después del arranque.....</i>	<i>27</i>
PLANIFICACIÓN .....	29
<i>Criterio de Completitud de las pruebas .....</i>	<i>29</i>
REVISIONES .....	31
<i>Objetivos .....</i>	<i>31</i>
<i>Beneficios .....</i>	<i>32</i>
<i>Formales vs Informales .....</i>	<i>33</i>
<i>Condiciones para comenzar.....</i>	<i>33</i>
<i>Checklists guías en revisiones.....</i>	<i>34</i>
REFERENCIAS .....	34

## Pruebas de Software

### INTRODUCCIÓN

#### Definiciones [1]

**Testing:** es el proceso orientado a demostrar que un programa no tiene errores. 1 - *Imposible*. 2 - *Tentación a diseñar tests que no detecten errores*.

**Testing:** es la tarea de demostrar que un programa realiza las funciones para las cuales fue construido.

**Testing:** es la tarea de probar que un programa realiza lo que se supone debe hacer. *Aún haciendo lo esperado, puede contener errores*.

**Testing:** es la ejecución de programas de software con el objetivo de detectar defectos y fallas. *Proceso destructivo, sádico*.

**Test Exitoso:** aquel que detecta errores

**Test No exitoso:** aquel que no los detecta

Problema psicológico, requiere un cambio de actitud ya que naturalmente somos constructivos.

**Error:** una equivocación de una persona al desarrollar alguna actividad de desarrollo de software.

**Defecto:** se produce cuando una persona comete un error.

**Falla:** es un desvío respecto del comportamiento esperado del sistema, puede producirse en cualquier etapa

Notas:

Defecto es una vista interna, lo ven los desarrolladores. Falla es una vista externa, la ven los usuarios.



## Filosofía y Economía

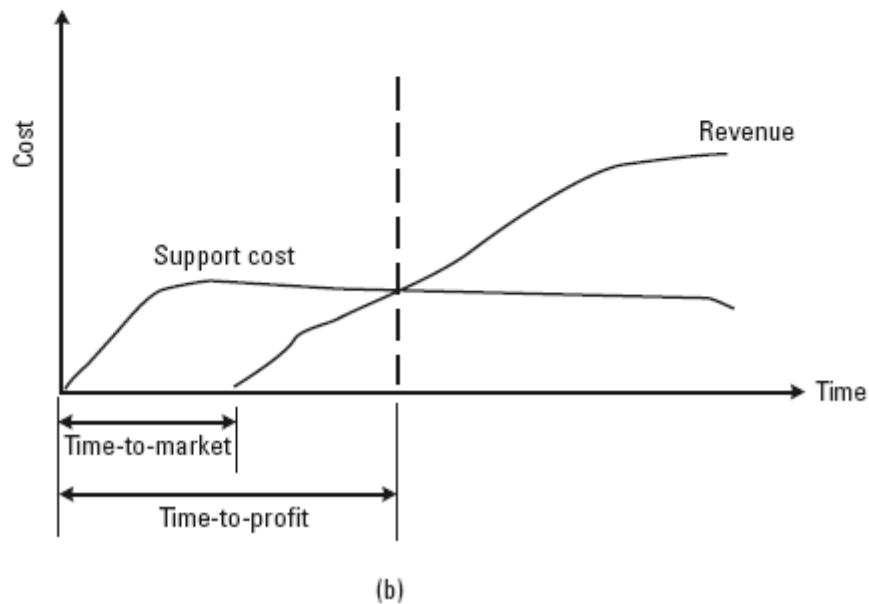
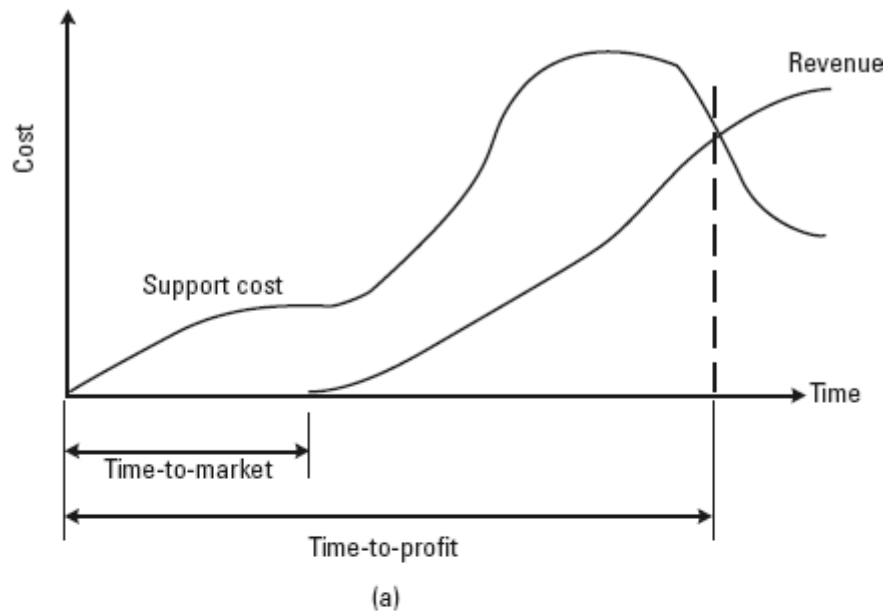


Gráfico tomado de Rakitin[2]

## Justificación

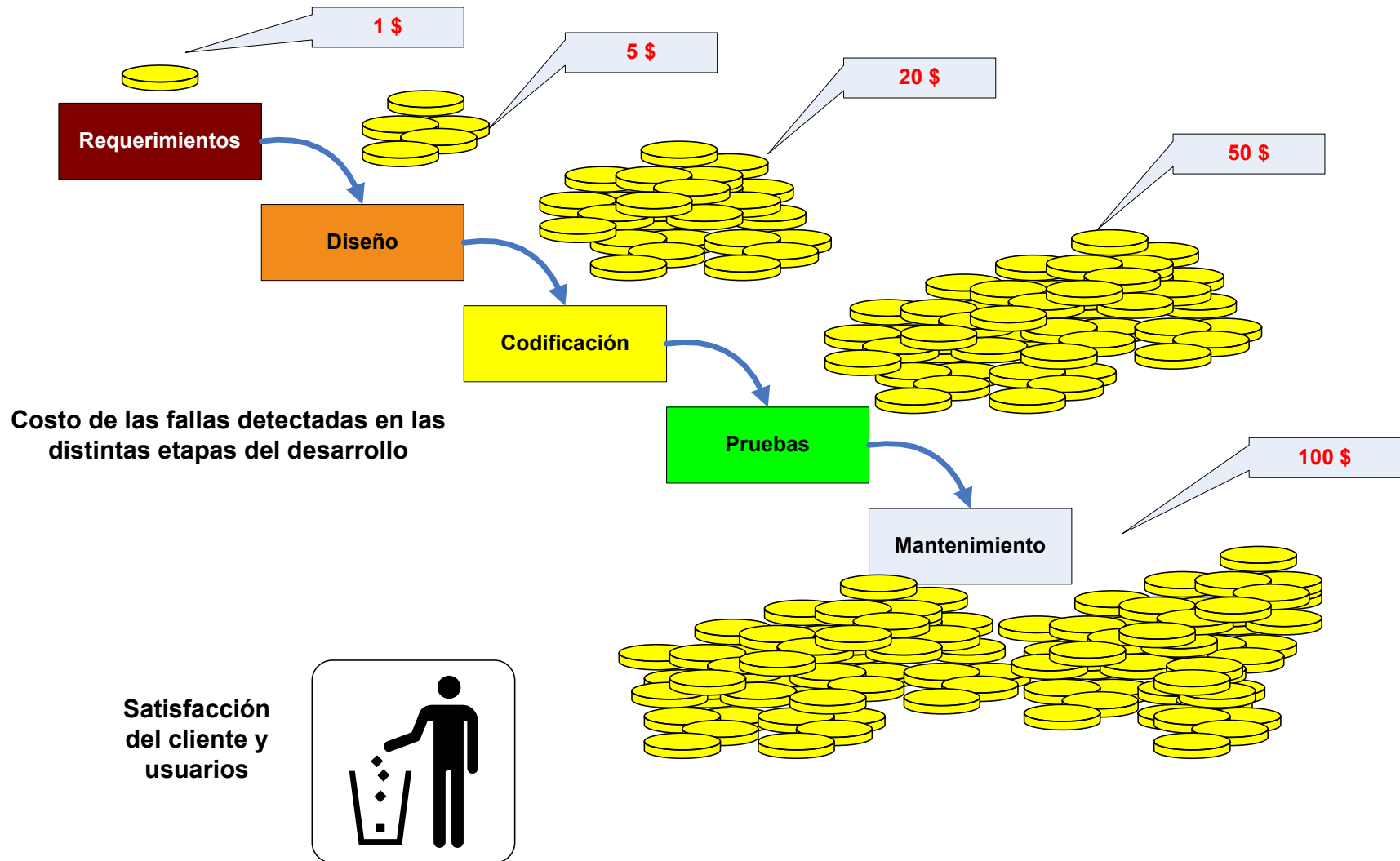
1. La realización de tareas de pruebas conlleva un costo asociado que puede inducir a tomar decisiones de no realizarlas.
2. No realizarlas también conlleva un costo asociado.

El problema es determinar cuál de estos costos es mayor.

**Presuponemos los siguientes objetivos:**

Menores costos, menores tiempos de desarrollo y mayor satisfacción del cliente.





## PRINCIPIOS [1]

1. Una parte necesaria de un test es la definición de los resultados esperados
2. Un programador debe evitar probar su propio desarrollo
3. Una organización no debe probar sus propios desarrollos
4. Revise los resultados de los test en profundidad
5. Los test deben incluir entradas inválidas e inesperadas así como las válidas y esperadas
6. Revisar un programa para verificar que hace lo que se espera que haga es sólo la mitad de la prueba; la otra mitad consiste comprobar que no haga lo que no se espera
7. No tirar los test a la basura a menos que el programa sea basura
8. No planear esfuerzos de pruebas asumiendo que no se encontrarán errores
9. La probabilidad de encontrar errores en una sección de un programa es proporcional al número de errores ya encontrados en esa sección
10. El “testing” constituye una tarea creativa e intelectualmente desafiante

## NIVELES DE PRUEBAS

- Test Unitarios
- Test de Componentes / Test de Integración
- Test de Funcionalidad
- Test de Sistema
- Test de Aceptación
- Test de Instalación

Niveles de pruebas				
Test	Objetivo	Participantes	Ambiente	Método
<b>Unitario</b>	Detectar errores en los datos, lógica, algoritmos	Programadores	Desarrollo	Caja Blanca
<b>Integración</b>	Detectar errores de interfaces y relaciones entre componentes	Programadores	Desarrollo	Caja Blanca, Top Down, Bottom Up
<b>Funcional</b>	Detectar errores en la implementación de requerimientos	Testers, Analistas	Desarrollo	Funcional
<b>Sistema</b>	Detectar fallas en el cubrimiento de los requerimientos	Testers, Analistas	Desarrollo	Funcional
<b>Aceptación</b>	Detectar fallas en la implementación del sistema	Testers, Analistas, Cliente	Productivo	Funcional

## TIPOS DE PRUEBAS

- Test de Facilidad
- Test de Volumen
- Test de Stress
- Test de Usabilidad
- Test de Seguridad
- Test de Performance
- Test de Configuración
- Test de Instalabilidad
- Test de Fiabilidad



- Test de Recuperación
- Test de Documentación
- Test de Mantenibilidad

## CLAVES DEL CAMBIO EN LA FORMA DE TRABAJO

- ⇒ **Automatización**
- ⇒ **Prueba como criterio de diseño**

### **Razones para automatizar las pruebas**

- Ciclo de prueba manual es muy largo
- Proceso de prueba manual es propenso a errores
- Liberar a la gente para realizar tareas creativas
- Generar un ambiente de confianza soportado por los test
- Obtener realimentación de forma temprana y con alta frecuencia
- Generar conocimiento del sistema en desarrollo a partir de los test
- Generar documentación del código consistente
- Generar una mejor utilización de los recursos a partir de menores costos

### **Obstáculos para automatizar las pruebas**

- Actitud de los programadores
- La joroba de dolor
- Inversión inicial
- Código que siempre cambia
- Sistemas legacy
- Temor
- Viejos hábitos

### **Qué debería automatizarse**

- Pruebas unitarias y de componentes
- Pruebas de funcionalidad sin interfaces de usuario
- Pruebas de sistema con interfaces de usuario

En la figura que sigue se muestra la llamada pirámide de las pruebas dónde se indican los aspectos a automatizar y no.



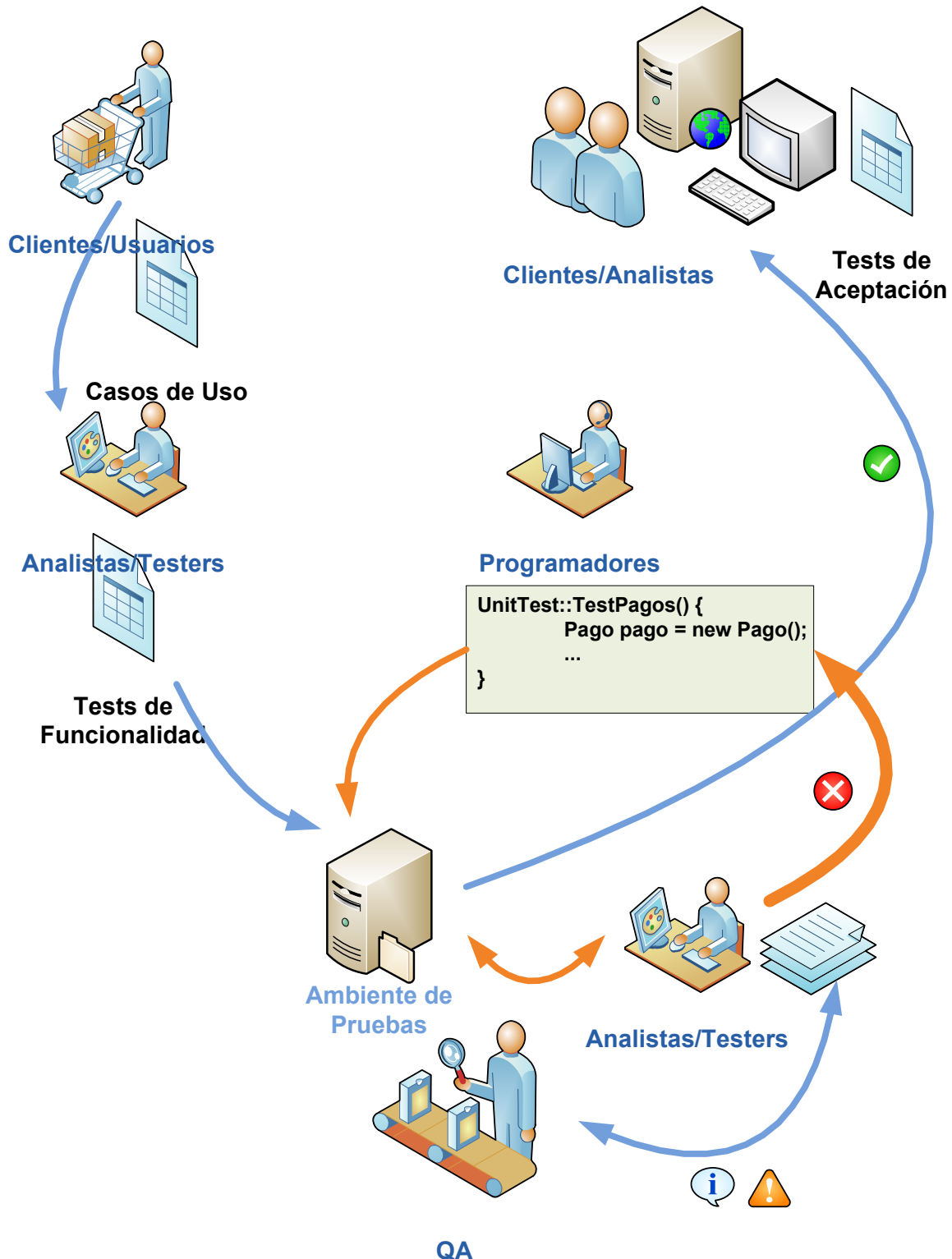
#### **Qué no debería automatizarse**

- Pruebas de usabilidad
- Pruebas exploratorias
- Pruebas que no fallarán
- Tareas únicas de fácil ejecución manual y difícil automatización

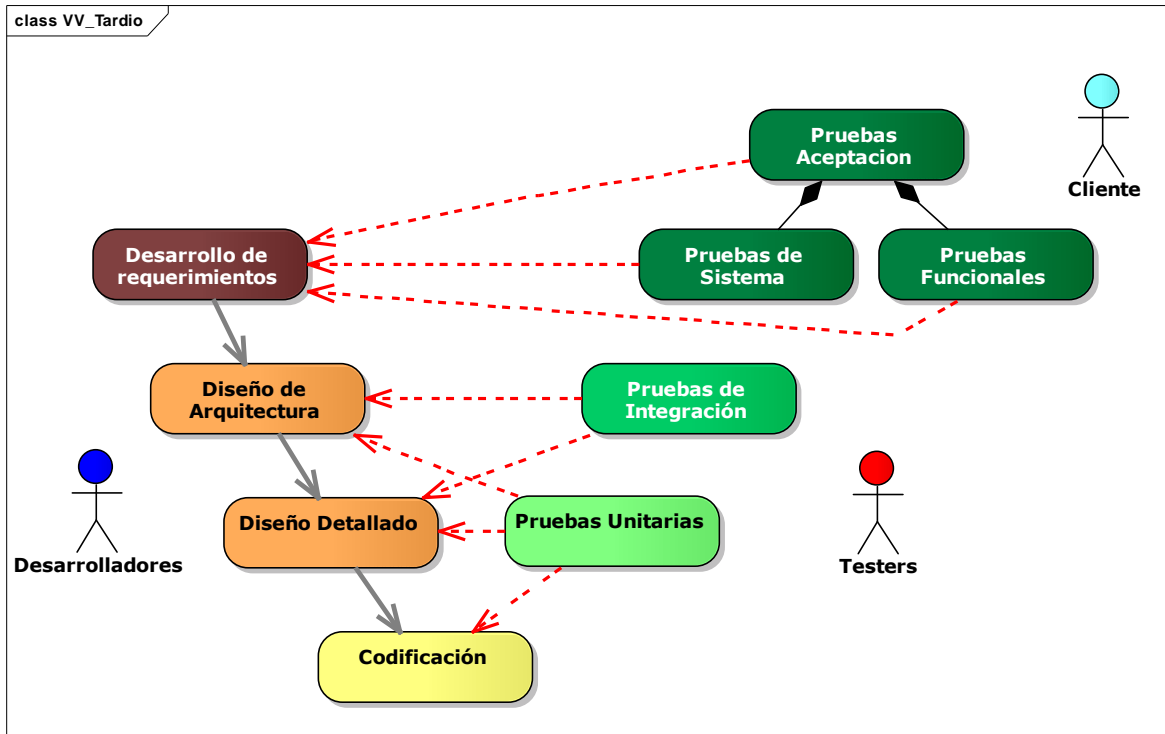
#### **Estrategia para comenzar la automatización**

- Capacitación a analistas, testers y programadores
- Seleccionar una forma de trabajo
- Seleccionar herramientas
- Desarrollar proyectos pilotos
- Institucionalizar

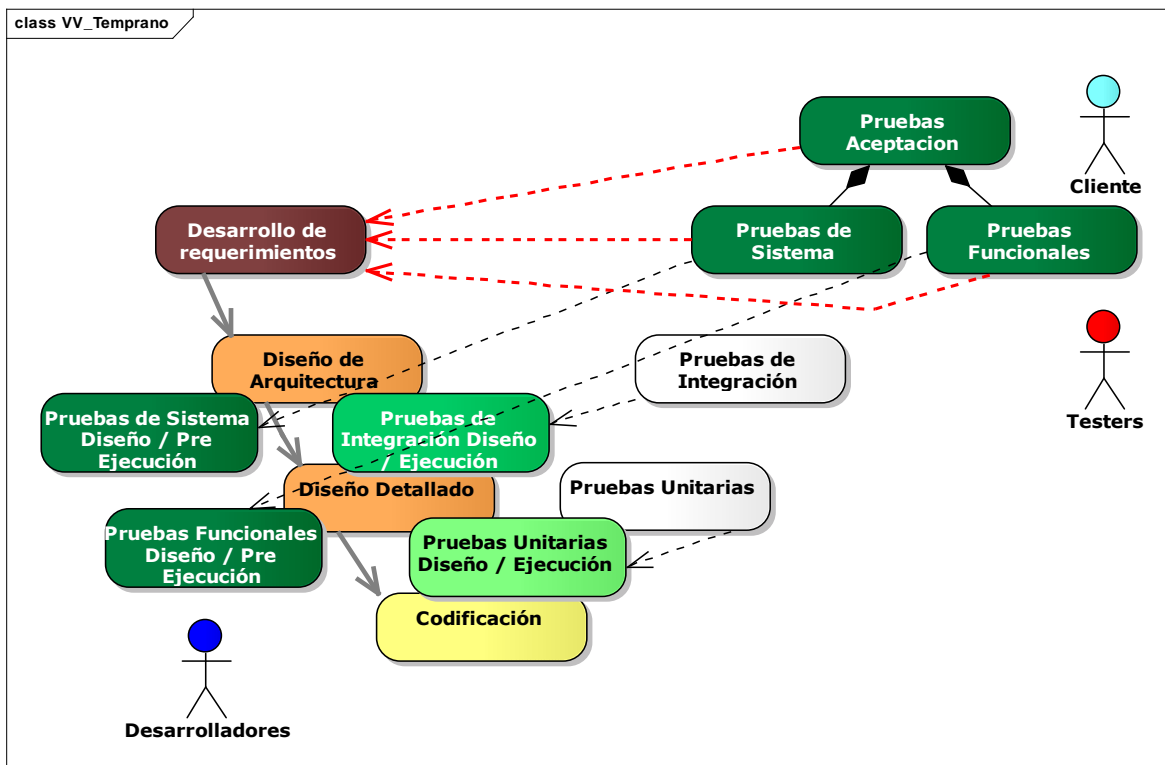
## Trabajo con tests manuales



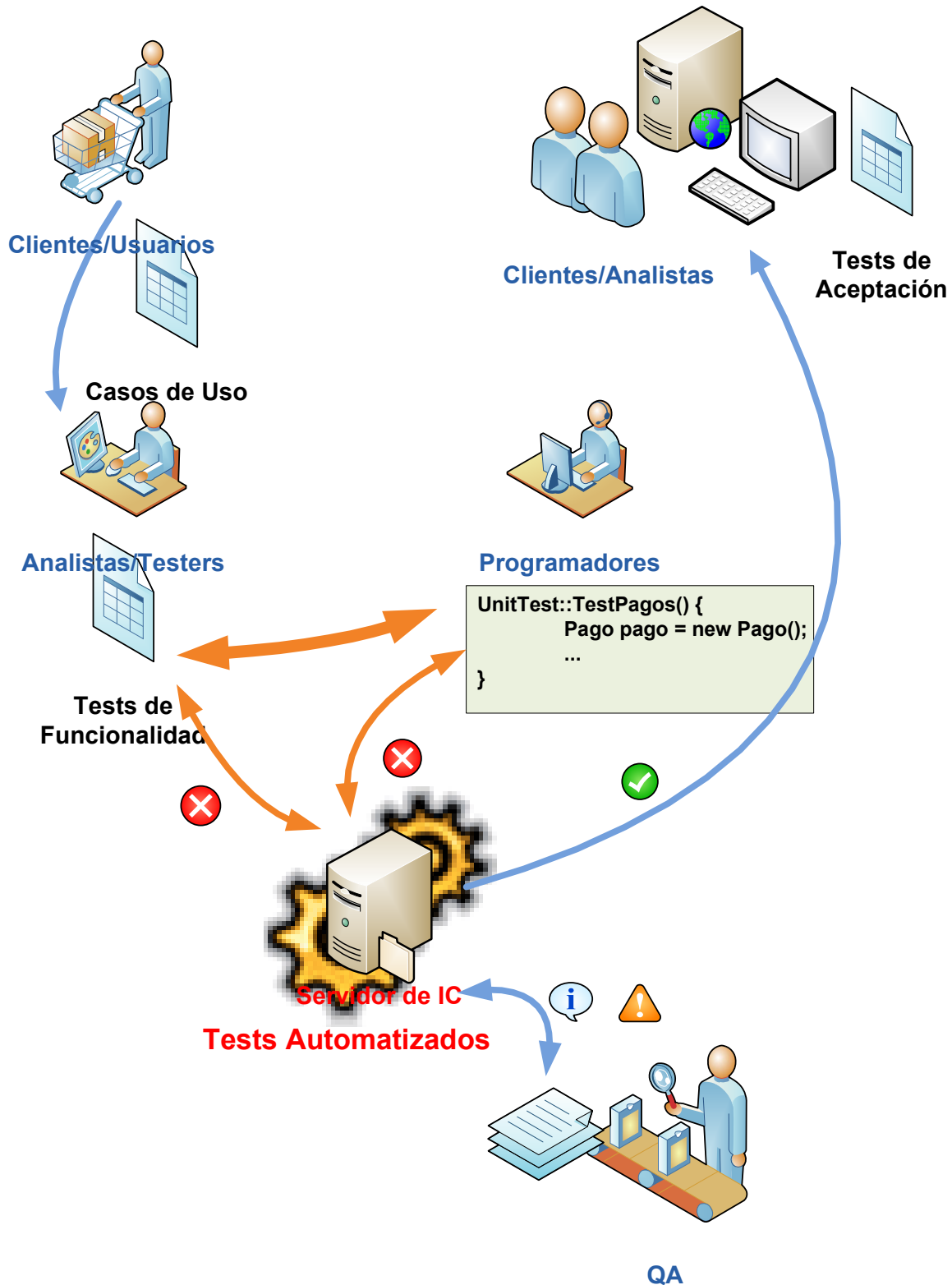
## Modelo tradicional



## Modelo actualizado



## Trabajo con tests automatizados



## MÉTODOS DE PRUEBA

### Test incrementales

Testeo continuo, distribuye las pruebas de integración en la integración diaria del código compartido.

#### Top Down

- ⇒ Se requieren Stubs para suplantar los módulos inferiores aún no implementados
- ⇒ Los Stubs se quitan a medida que se desarrollan los diferentes módulos
- ⇒ Un test por módulo que se suma
- ⇒ Realizar test de regresión sobre los módulos

#### Desventajas

- ⇒ Se retrasa la prueba del procesamiento real realizado generalmente en módulos de más bajo nivel
- ⇒ Desarrollar Stubs que emulen a los módulos es mucho trabajo

#### Bottom Up

- ⇒ Las pruebas comienzan en el más bajo nivel con la integración de algoritmos que realizan procesamiento
- ⇒ Se escriben test que dan el contexto de ejecución a los módulos
- ⇒ Se prueban los módulos
- ⇒ Se desarrolla e integran funcionalidades del módulo superior y se repite

#### Desventajas

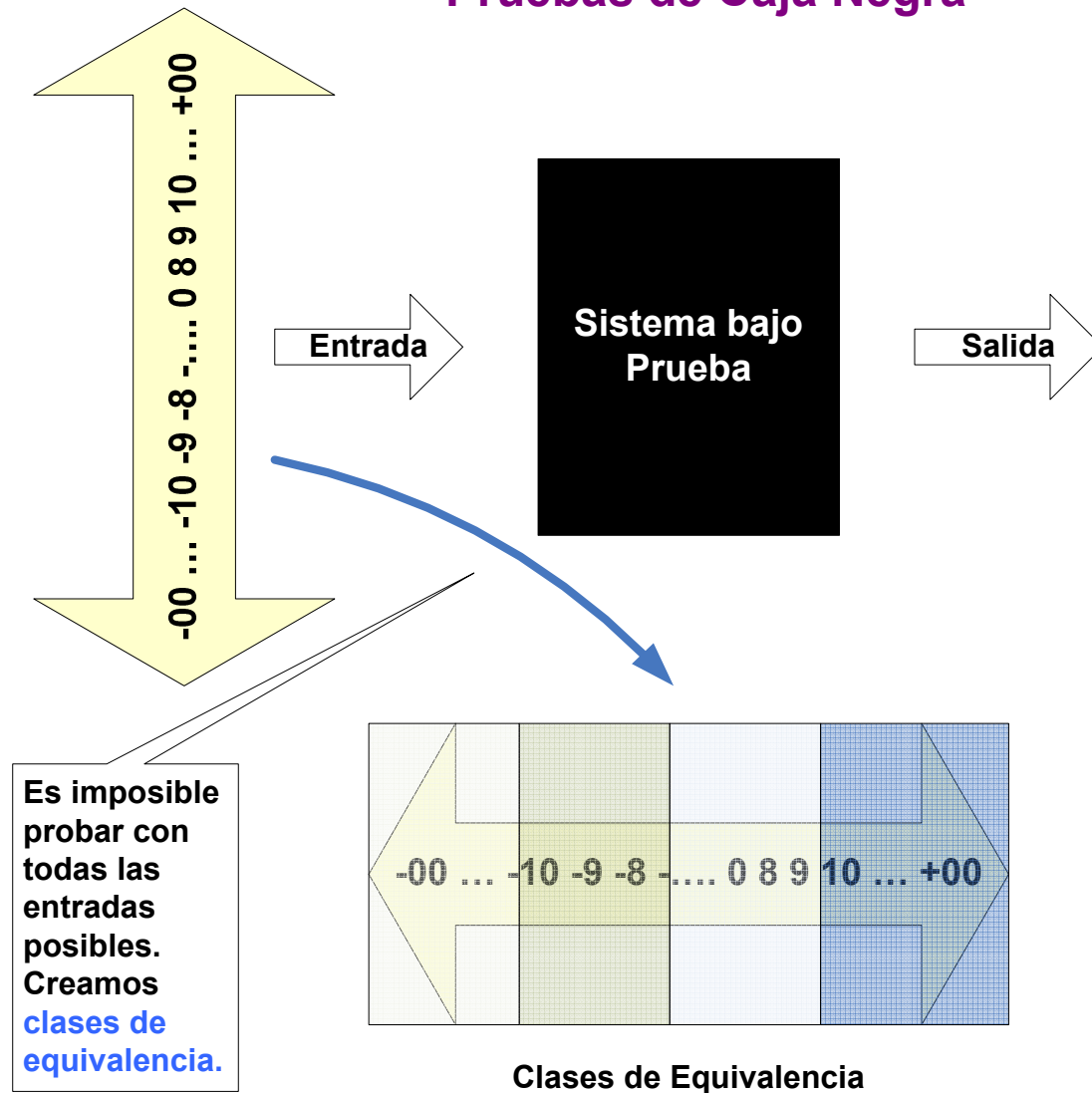
- ⇒ Hasta que se logra un nivel determinado, la aplicación no es visible
- ⇒ Problemas asociados a volumen, recursos y tiempo se prueban en etapas tardías



## Caja Negra

Pruebas funcionales sin acceso al código fuente de las aplicaciones, se trabaja con entradas y salidas

## Pruebas de Caja Negra

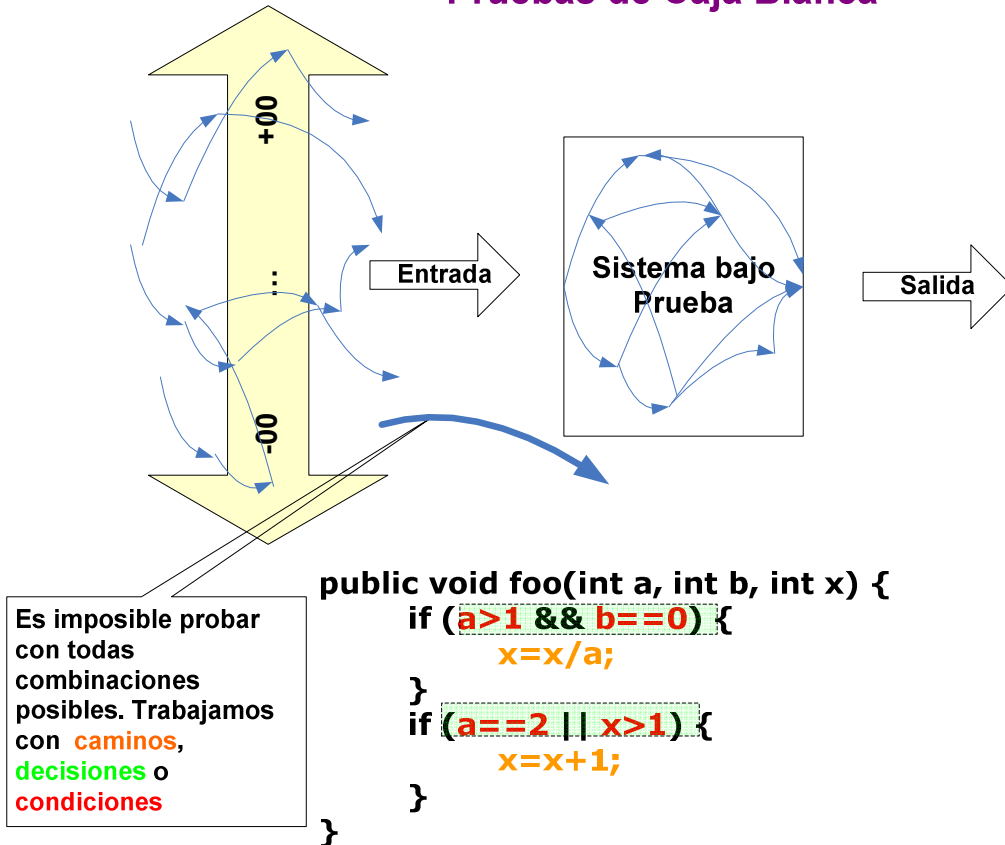


Condición de entrada	Clase de equivalencia válida	Clase de equivalencia inválida
<b>Rango de valores (1, 99)</b>	Valores > 1 y Valores < 99	Valores < 1 Valores > 99
...	...	...
...	...	...

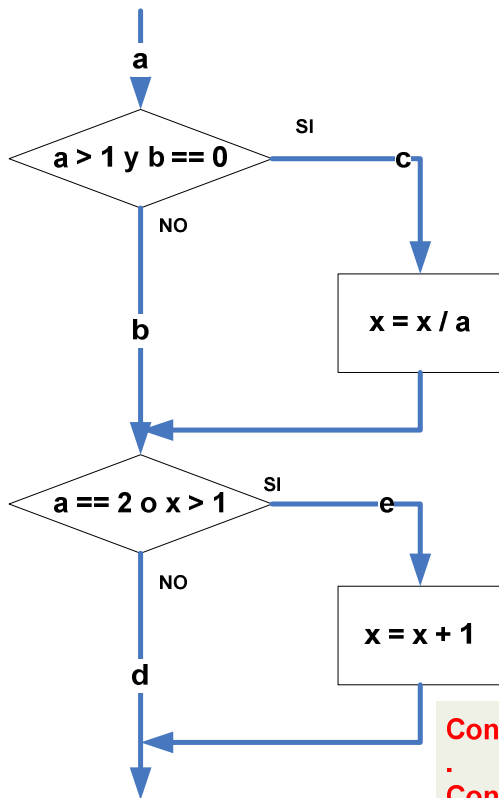
## Caja Blanca

Pruebas con acceso al código fuente (datos y lógica). Se trabaja con entradas, salidas y el conocimiento interno.

## Pruebas de Caja Blanca







**Caminos:** a = 2, b = 0, x = 2

**Falencias:**

- si 'y' debería ser 'o' no se detecta.
- si x>0 en lugar de '1' no se detecta
- si x sin modificar (a,b,d) es un error no se detecta

**Decisiones:** para cada decisión un test por true y uno por false. O sea (ace, abd) o (acd, abe)

Test 1 - a = 3, b = 0, x = 3

Test 2 - a = 2, b = 1, x = 1

**Falencias:** 50% de posibilidad al elegir

- Si elijo el test 2, si en lugar de x>1 era x<1, no sería detectado

**Condiciones:** un test por cada condición (true, false)

**Condiciones:** (4) a>1, b=0, a=2, x>1

**Tests:**

a > 1, a <= 1, b = 0, b <> 0, a = 2, a <> 2, x > 1, x <= 1

**Tests resultantes:**

a = 2, b = 0, x=4 (ace)

a = 1, b = 1, x=1 (adb)

**Falencias:** no se ensayan todas las decisiones por solapamiento de condiciones

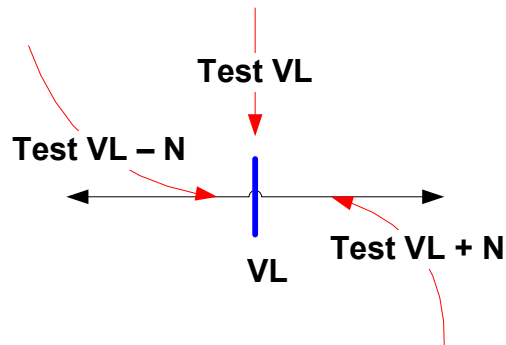
a = 1, b = 0, x = 1

a = 2, b = 1, x = 3

No se prueba nunca (ac)

Valores límite





## DISEÑO DE CASOS DE PRUEBAS

- ⇒ Clases de equivalencia
- ⇒ Decisiones/condiciones
- ⇒ Valores límites
- ⇒ Tester Visitante

Ejemplo:

```
public class EvaluaEstrategia {

    public EstrategiaPago getEstrategiaPago(Proyecto pr){

        EstrategiaPago estrategia = null;

        if(pr.getCosto() >= 100000.00){
            estrategia = new EstrategiaPagoAlta();
        }
        else if(pr.getCosto() < 100000.00 && (pr.getCosto() >= 50000.00)){
            estrategia = new EstrategiaPagoMedia();
        }
        else{
            estrategia = new EstrategiaPagoBaja();
        }
        return estrategia;
    }

}
```

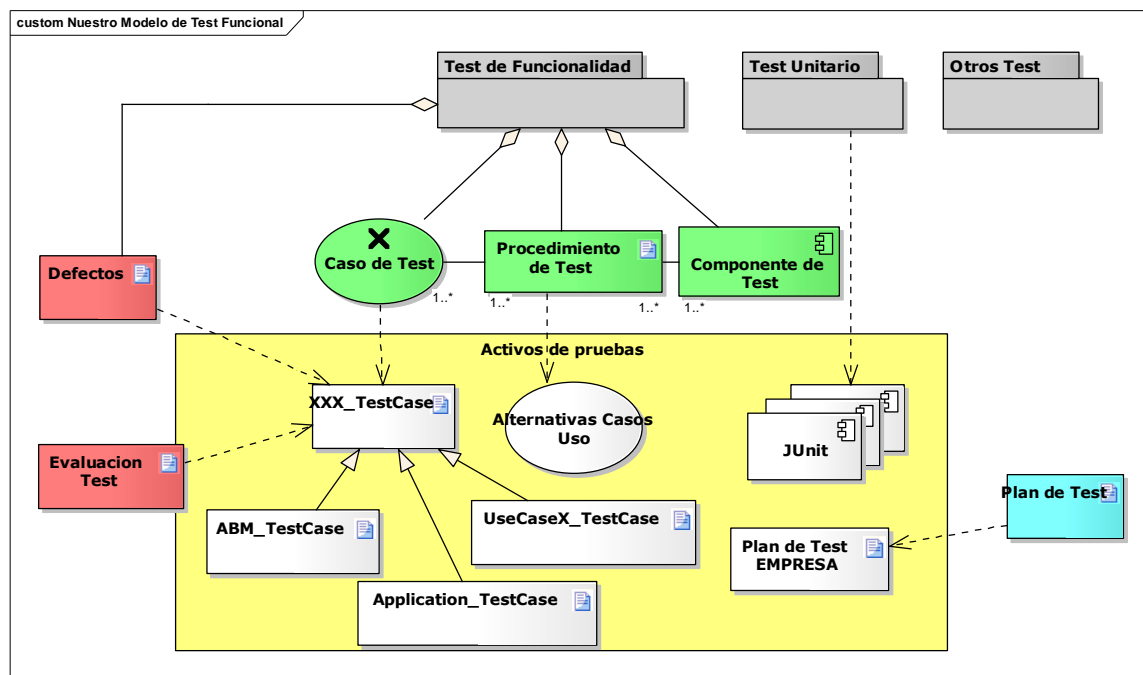


Clases de Equivalencia				
Clases	Condición Entrada	Clase Equivalencia	Test	Id Test
1	Costo del proyecto	Valor positivo > 0.00	Prueba con entrada costo = 150000.00	1
2		Valor cero (0)	Prueba con entrada costo = 0.00	2
3		Valor < 0	Prueba con entrada costo = - 1000.00	3
Decisiones / Condiciones				
Condición	Lógica	Condición a probar	Test	
1	Costo >= 100000.00	true	Prueba con entrada costo = 150000.00	1
		false	Prueba con entrada costo = 60000.00	4
2	Costo < 100000.00	true	Prueba con entrada costo = 60000.00	4
		false	Prueba con entrada costo = 150000.00	1
3	Costo >= 50000.00	true	Prueba con entrada costo = 150000.00	1
		false	Prueba con entrada costo = 10000.00	5
Valores Límites				
Límite	Valor	Condición a probar	Test	
1	100000.00	=	Prueba con entrada costo = 100000.00	6
		>	Prueba con entrada costo = 100001.00	1
		<	Prueba con entrada costo = 99999.00	4
2	50000.00	=	Prueba con entrada costo = 50000.00	7
		>	Prueba con entrada costo = 50001.00	1
		<	Prueba con entrada costo = 49999.00	5

## PRUEBAS FUNCIONALES Y DE ACEPTACIÓN

### Desde los casos de uso a los casos de pruebas.

- Pruebas de funcionalidad
  - Aspectos claves
    - Buena especificación
    - El diseño conceptual de interfaces
    - Modelo de dominio
- Pruebas de aplicación
  - Aspectos claves
    - Definición precisa de interfaces



## Automatización a partir del trabajo integrado – Fitnessse

### Herramienta

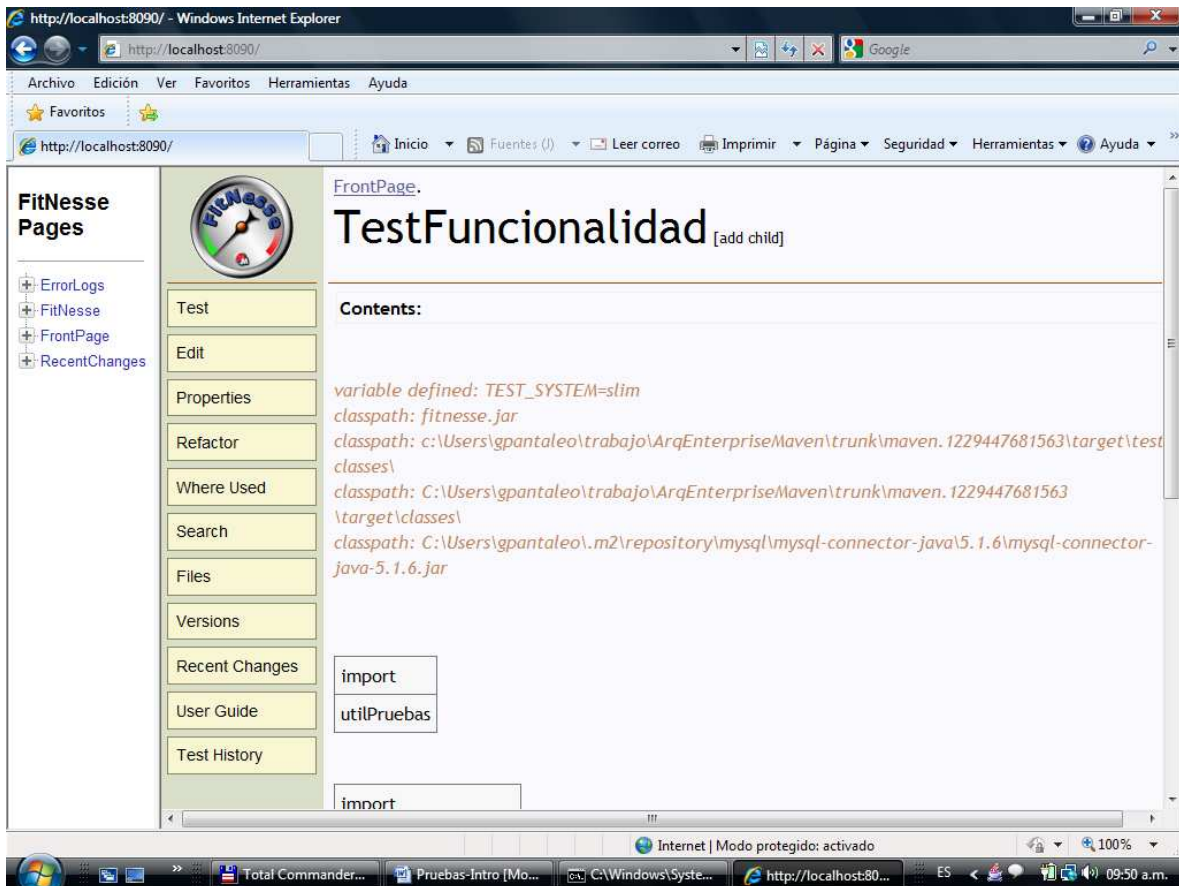
- FIT – Fitnessse (Framework for Integrated Tests)

### Roles

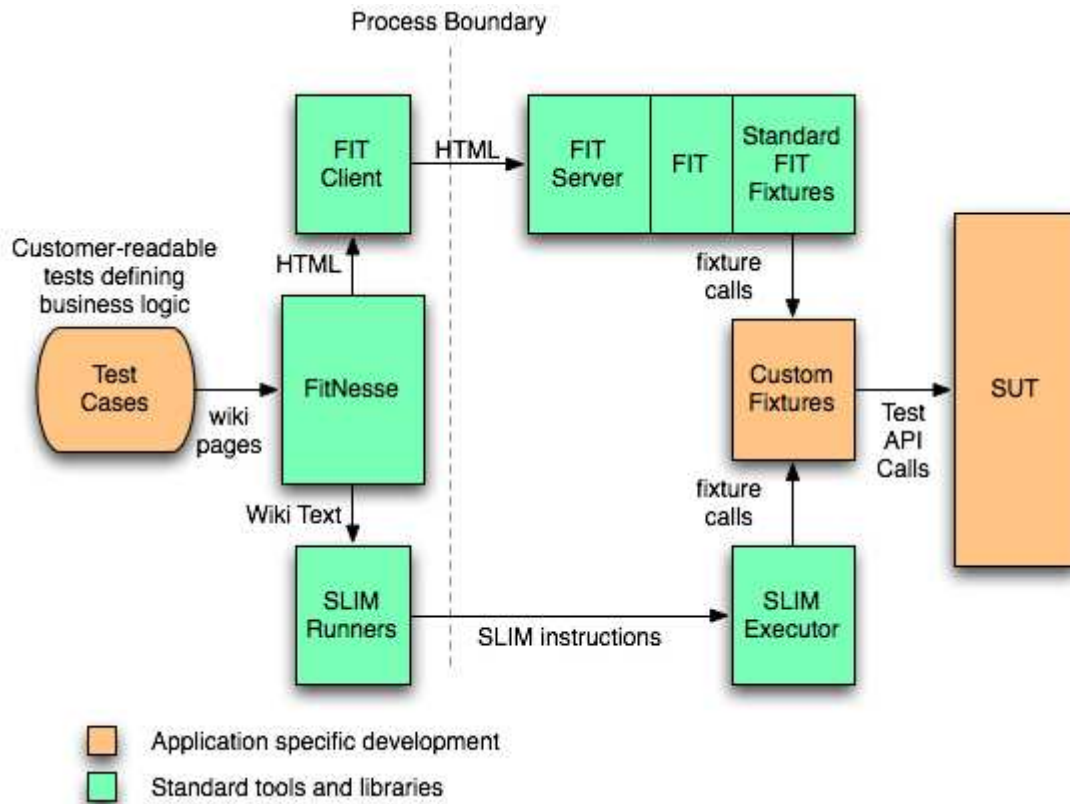
- Analista
- Tester
- Programador

### Características

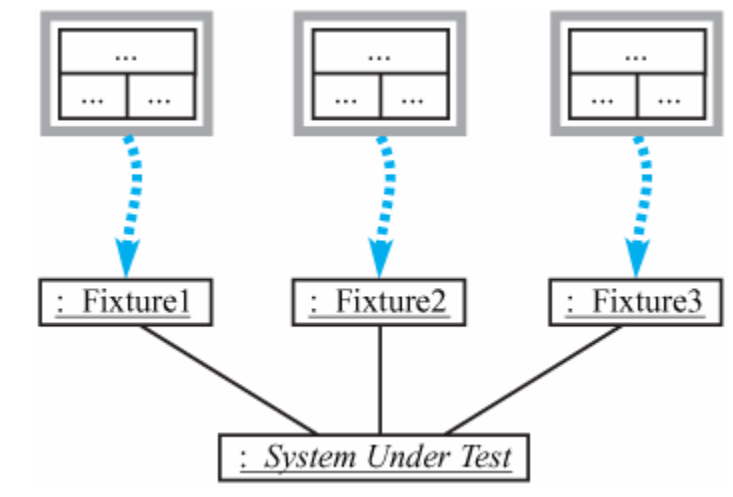
- Administración de tablas (orientada a NO programadores)
- Perspectiva del negocio (validación de requerimientos, reglas de negocio y flujo de trabajo)



## Vinculo con el sistema bajo test

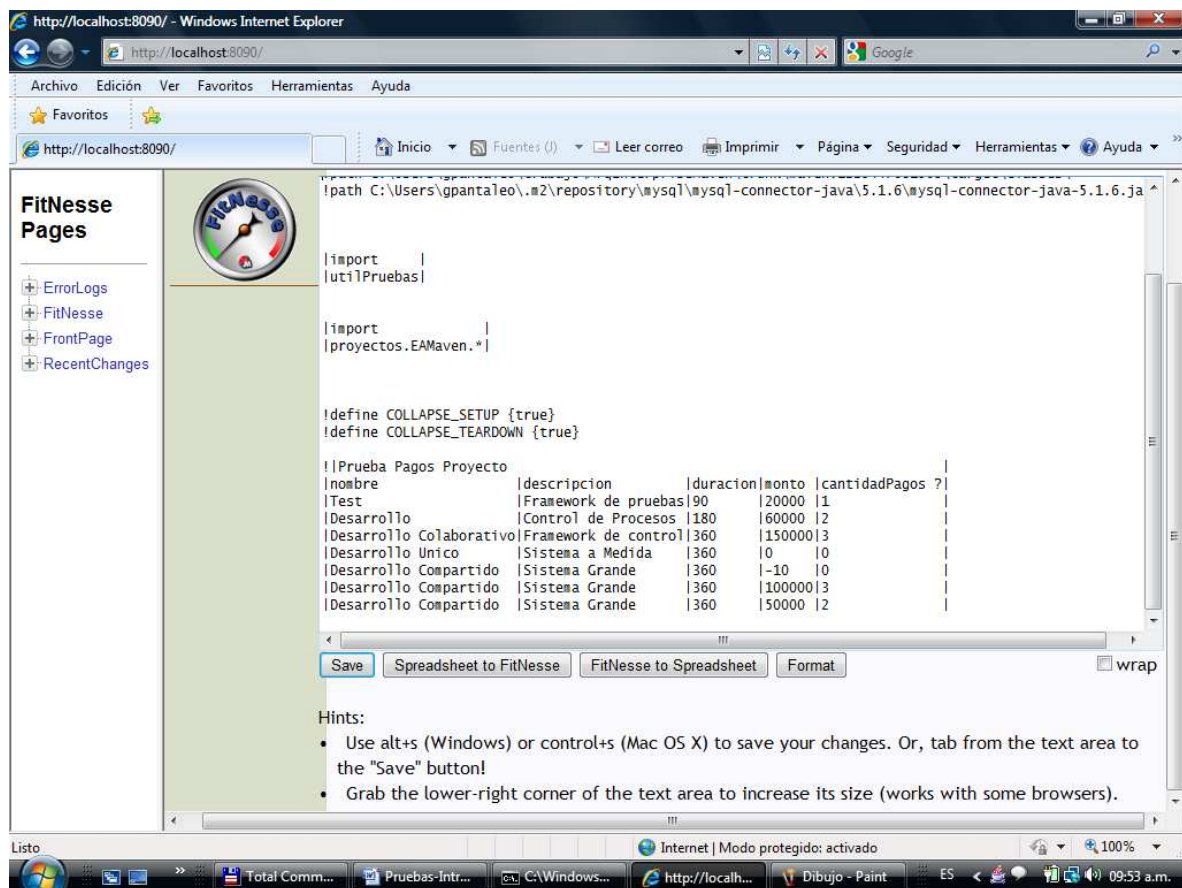


## Fixture Clases

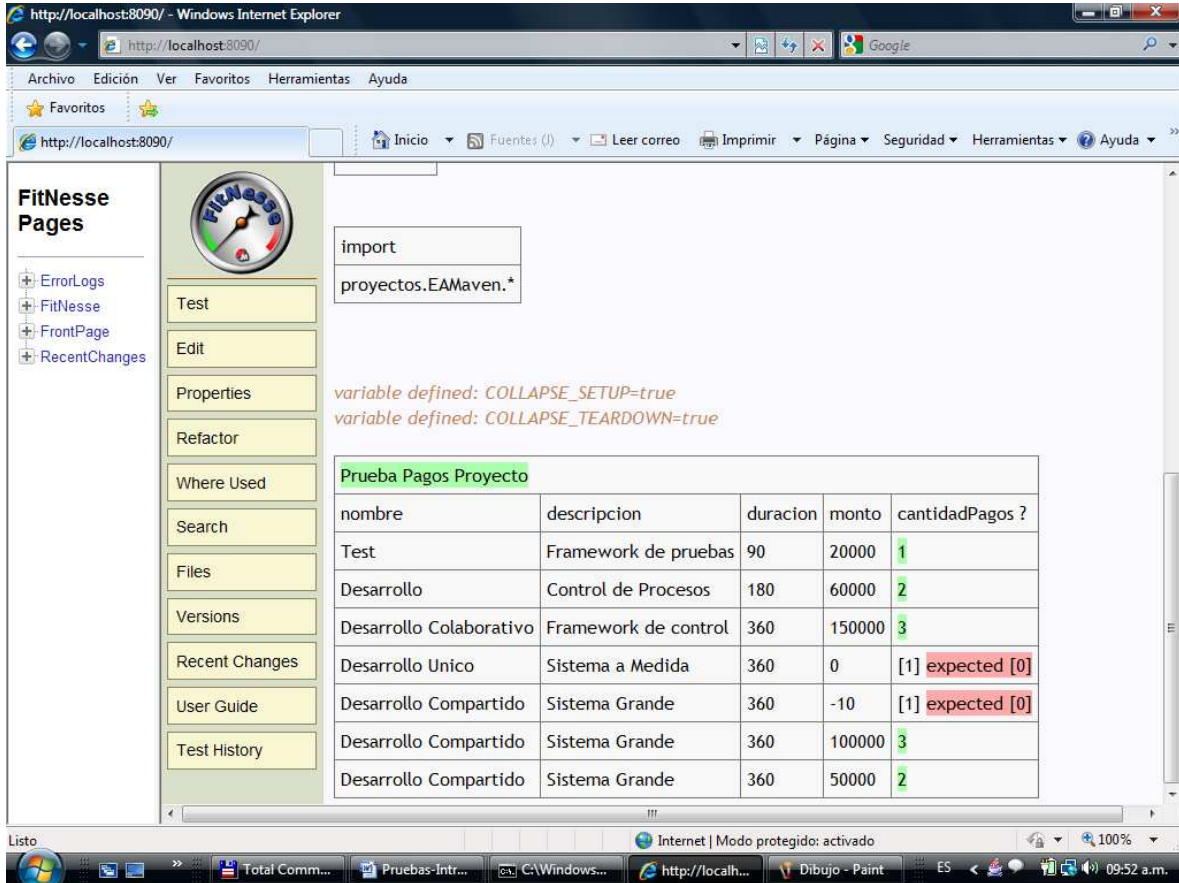


## Diseño y edición de los test

Prueba Pagos Proyecto					
nombre	descripcion	duracion	monto	cantidadPagos ?	
Test	Framework de pruebas	90	20000	1	
Desarrollo	Control de Procesos	180	60000	2	
Desarrollo Colaborativo	Framework de control	360	150000	3	
Desarrollo Unico	Sistema a Medida	360	0	0	
Desarrollo Compartido	Sistema Grande	360	-10	0	
Desarrollo Compartido	Sistema Grande	360	100000	3	
Desarrollo Compartido	Sistema Grande	360	50000	2	



## Ejecución de los test



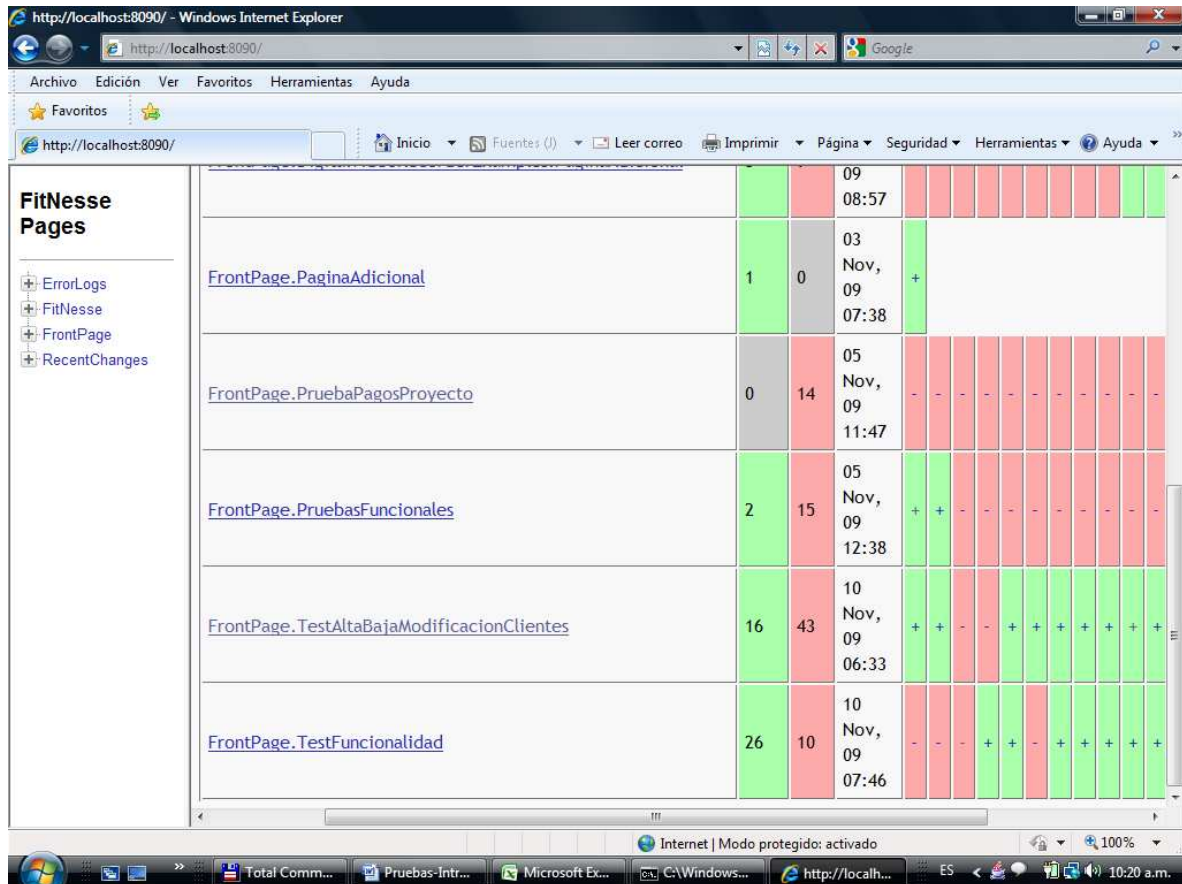
import  
proyectos.EAMaven.\*

variable defined: COLLAPSE\_SETUP=true  
variable defined: COLLAPSE\_TEARDOWN=true

nombre	descripcion	duracion	monto	cantidadPagos ?
Test	Framework de pruebas	90	20000	1
Desarrollo	Control de Procesos	180	60000	2
Desarrollo Colaborativo	Framework de control	360	150000	3
Desarrollo Unico	Sistema a Medida	360	0	[1] expected [0]
Desarrollo Compartido	Sistema Grande	360	-10	[1] expected [0]
Desarrollo Compartido	Sistema Grande	360	100000	3
Desarrollo Compartido	Sistema Grande	360	50000	2



## Historia de test



Page Name	Status	Count	Time	Results
FrontPage.PaginaAdicional	Pass	1	03 Nov, 09 08:57	+
FrontPage.PruebaPagosProyecto	Fail	0	05 Nov, 09 11:47	- - - - -
FrontPage.PruebasFuncionales	Pass	2	05 Nov, 09 12:38	+ + - - - - -
FrontPage.TestAltaBajaModificacionClientes	Pass	16	10 Nov, 09 06:33	+ + - + + + + + +
FrontPage.TestFuncionalidad	Pass	26	10 Nov, 09 07:46	- - - + + - + + + + +

## Integración con otros servers

### Organización de proyectos

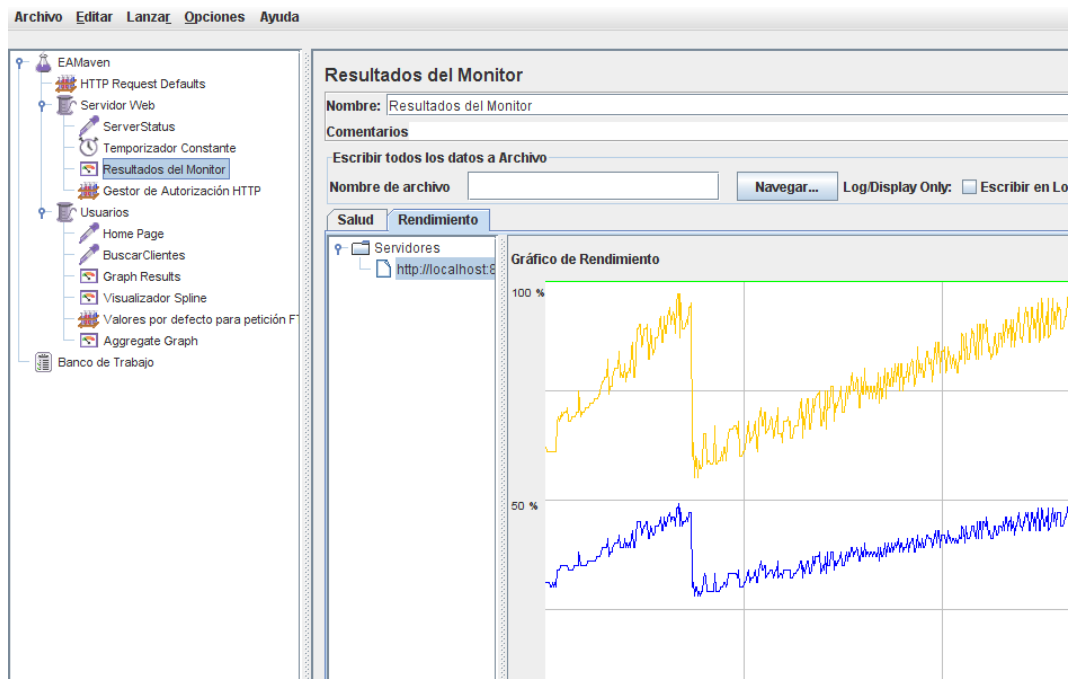
⇒ Maven2: mvn fitness:remotecall

### Integración Continua

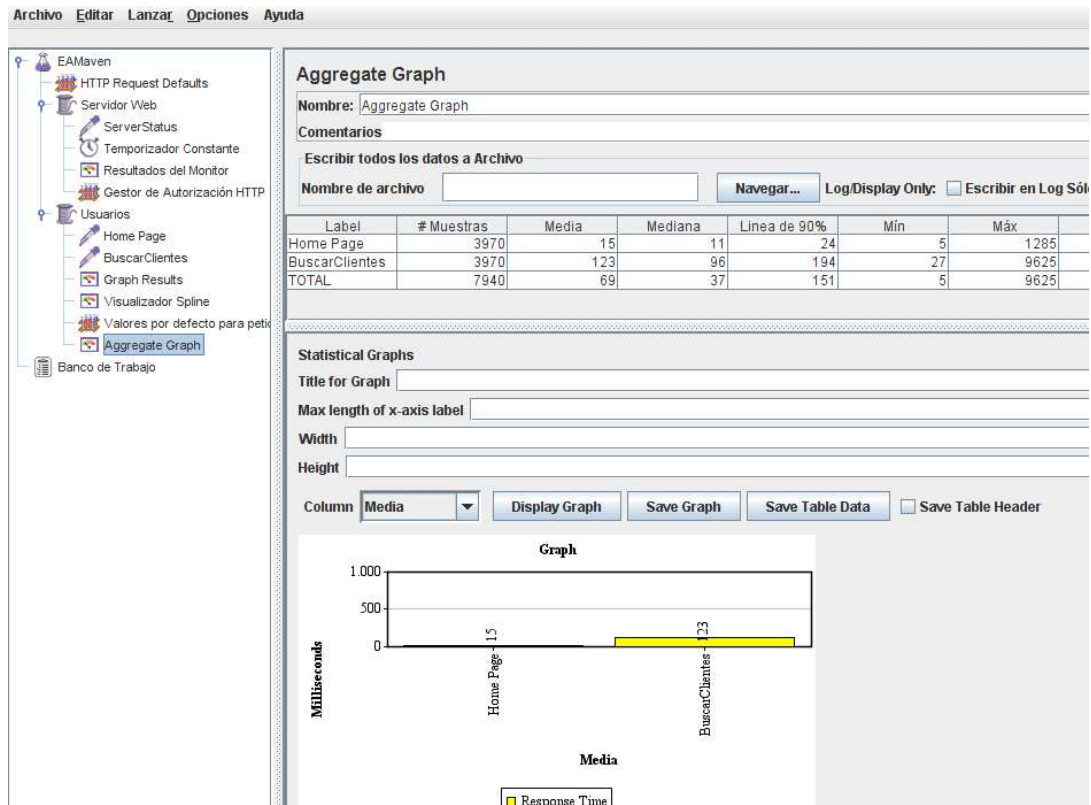
⇒ Continuum: vía Maven

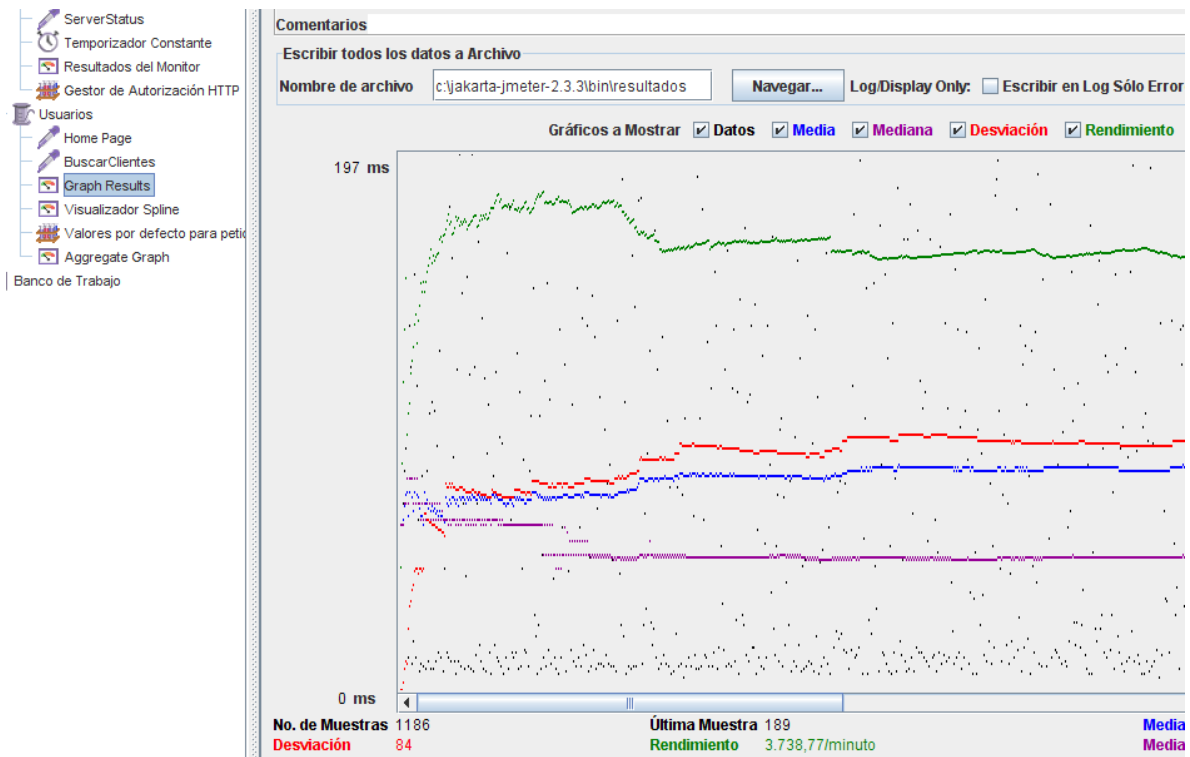
## PRUEBAS DE CARGA Y STRESS

### Carga del servidor



## Tiempo de respuesta de los queries y evolución de la dispersión después del arranque





## PLANIFICACIÓN

- ⇒ Planificación General
  - Objetivos
  - Criterios de Completitud
  - Cronograma
  - Responsabilidades
- ⇒ Planificación Técnica
  - Estándares de Casos de Pruebas
  - Herramientas
  - Infraestructura
  - Procedimientos

### **Criterio de Completitud de las pruebas**

1. Parar cuando se agotó el tiempo asignado
2. Parar cuando los test dan todos resultados esperados

#### **Desventajas:**

- ✓ No garantiza la realización de las pruebas (1), si el tiempo asignado a los test fue usado en desarrollo
- ✓ No garantiza buenos test (2), condiciona a veces a escribir test exitosos (no detectan errores)

#### **Otros criterios más concretos y eficientes**

1. Cuando todos los test den resultados esperados, los cuales fueron diseñados tal que satisfagan criterios de Condiciones y un análisis de Valores Límites
2. Cuando hayan sido detectados y reparados N errores
3. Cuando haya pasado M días sin detectar errores

### **Ejemplo:**

Después de las revisiones: 5 errores cada 100 líneas (métricas)

Objetivos: 98% de codificación, 95% de diseño

Programa: 10.000 líneas

Errores estimados:  $10.000 / 100 * 5 = 500$  errores.

Distribución de errores por tareas	
Requerimientos Funcionales	8.12 %
Diseño de Arquitectura	26.92 %
Diseño Detallado	22.44 %
Codificación	26.05 %
Integración	8.98 %
Pruebas	2.76 %

<b>Inespecificados</b>	<b>4.73 %</b>
------------------------	---------------

Codificación (180), Diseño (320).

Objetivo de las pruebas:

- ⇒ Detectar  $180 * 98 / 100 = 176$  errores de codificación
- ⇒ Detectar  $320 * 95 / 100 = 304$  errores de diseño

**Si los errores no se detectan después de N tiempo, y los casos son OK, terminamos.**

La evolución del número de errores es una ayuda interesante para la toma de decisiones como se ve en la figura:



## Estimating completion by plotting errors detected by unit time.

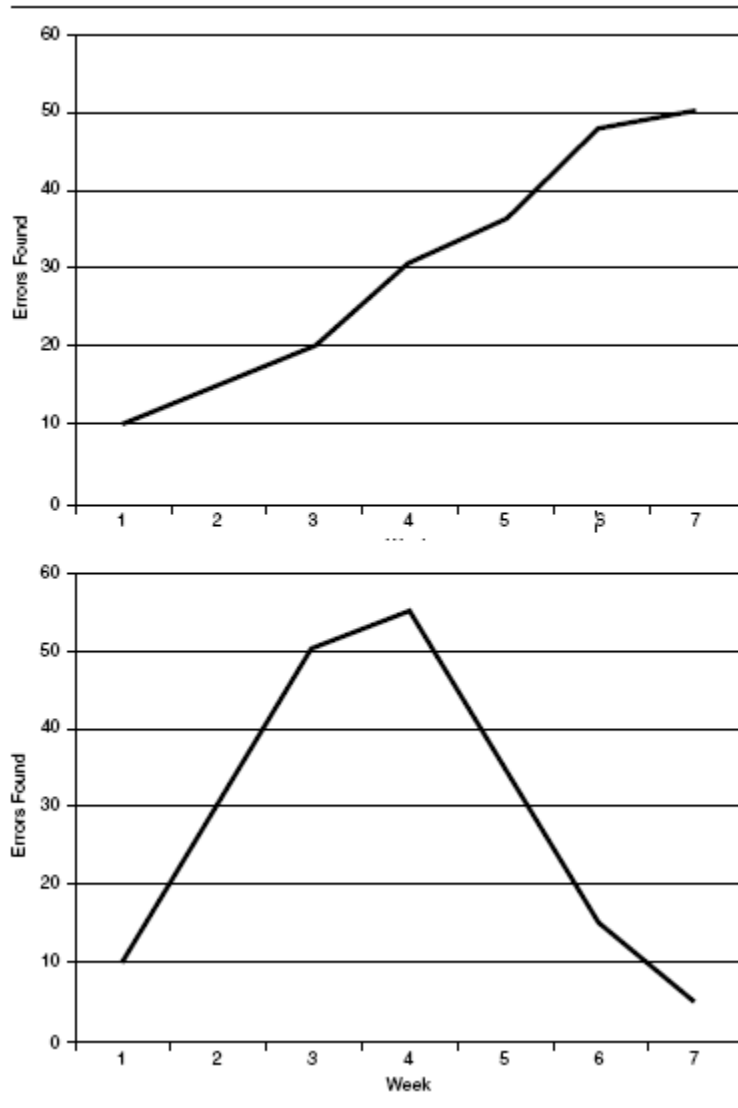


Gráfico tomada de Myers [1]

## REVISIONES

Revisión rigurosa y en profundidad de un artefacto de software realizado con el fin de detectar errores.

### Objetivos

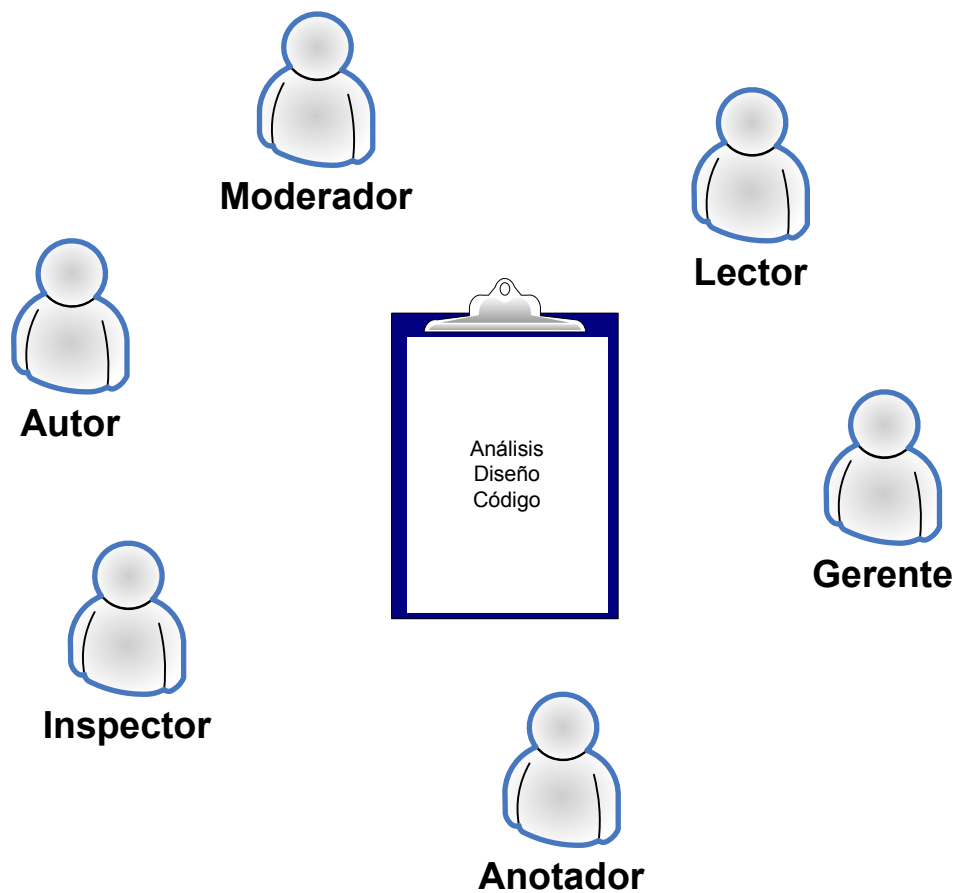
1. Detectar problemas de análisis, diseño y código en forma temprana
2. Definir y acordar criterios de retrabajo para su resolución
3. Verificar que se resolvió de acuerdo al criterio acordado

## Beneficios

1. Genera datos acerca del producto y el proceso de desarrollo
2. Genera conocimiento entre miembros del grupo de desarrollo
3. Aumenta la efectividad de la validación y verificación
4. Contribuye a la instalación del concepto de calidad

**Formales:** Con roles y responsabilidades y un procedimiento definido

**Informales:** Con roles desdibujados y sin procedimiento





### Formales vs Informales

Atributo	Formal	Informal
<b>Objetivos</b>	Detectar errores	Detectar errores
	Verificar re trabajo	Discutir alternativas de solución
	Focalizada sobre si o no los productos cubren los requerimientos	Focalizada en demostrar cómo los productos cubren los requerimientos
<b>Decisiones</b>	Decisiones concensuadas	Decisiones del autor
<b>Responsable</b>	Moderador entrenado	Autor
<b>Asistentes</b>	Pares con asistencia registrada	Pares y responsables técnicos, sin registrar
<b>Material</b>	Presentador por el Lector	Presentado por el autor
<b>Métricas</b>	Requeridas	Opcionales
<b>Procedimiento</b>	Formalmente registrado	Informal
<b>Entrenamiento</b>	Requerido para todos los roles	No requerido

### Condiciones para comenzar

Tipo Inspección	Activos a Inspeccionar	¿Listo para realizar revisión?	Material requerido para el grupo
<b>Requerimientos</b>	ERS	Entrenamiento realizado Documento de visión acordado	EERS Cchecklists
<b>Diseño</b>	EDA, EDD	Entrenamiento realizado ERS revisada y todos los problemas detectados resueltos	ERS EDA EDD Checklists
<b>Código</b>	Fuentes	Entrenamiento realizado EDA y EDD revisadas y todos los problemas detectados resueltos Módulos seleccionados según criterio definido Código compilado sin errores	Fuentes Estándares definidos Checklists
<b>Validación</b>	Pruebas	Entrenamiento realizado	ERS Procedimientos de

Tipo Inspección	Activos a Inspeccionar	¿Listo para realizar revisión?	Material requerido para el grupo
<b>Pruebas</b>	Procedimientos	Entrenamiento realizado ERS revisada y todos los problemas detectados resueltos	validación  Test Checklists

## Checklists guías en revisiones

- ⇒ Requerimientos
- ⇒ Diseño
- ⇒ C++
- ⇒ Java

## REFERENCIAS

1. *The Art of Software Testing, Second Edition*, Glenford J. Myers, John Wiley & Sons, Inc., 2004.
2. *Software Verification and Validation for Practitioners and Managers, Second Edition*, Steven R. Rakitin, Artech House, 2001.
3. *Code Complete, Second Edition*, Steve McConnell, Redmond, Wa.: Microsoft Press, 2004.
4. *Fit for Developing Software: Framework for Integrated Tests*, Rick Mugridge, Ward Cunningham, Prentice Hall PTR, 2005.
5. FitFitsse, Pruebas de funcionalidad y aceptación. Basado en una Wiki para Java, <http://fitnesse.org/>
6. JMeter Apache Jakarta Project Pruebas de sistema, <http://jakarta.apache.org/jmeter/>

