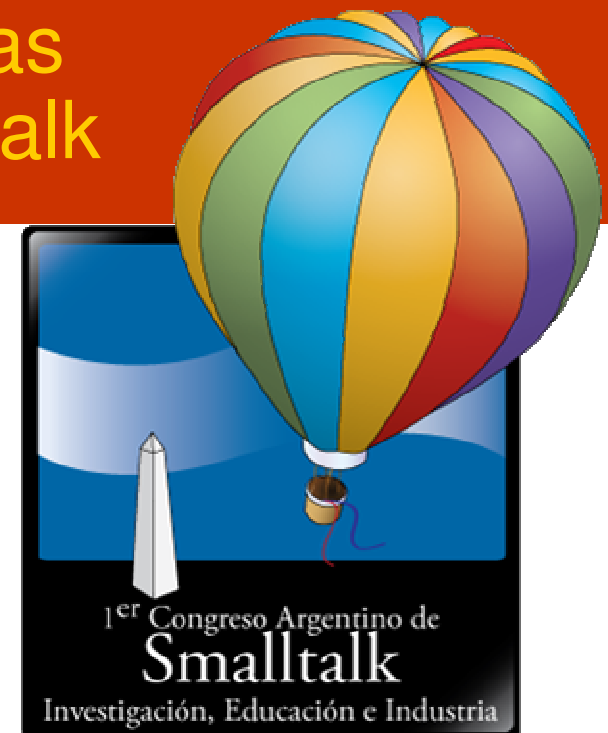




Experience Report

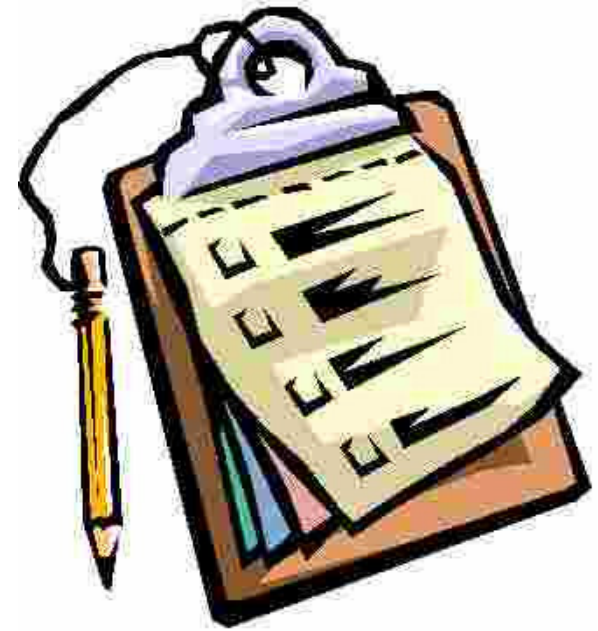
Herramientas para escalar
desarrollando sistemas
complejos con Smalltalk

Lic. Maximiliano Contieri
md.contieri@mercapsoftware.com
Mercap S.R.L.



Agenda

- Escenario y Objetivos
- Integración Continua
- Estándares de Calidad
- Migración de objetos
- Conclusiones



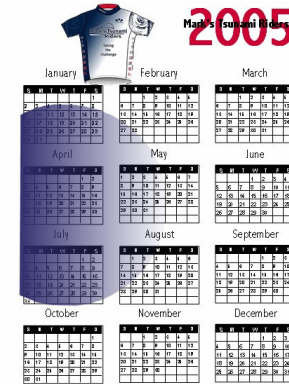
Agenda

- Escenario y Objetivos
- Integración Continua
- Estándares de Calidad
- Migración de objetos
- Conclusiones





Escenario y Objetivos



- Año 2005
- Desarrollar un sistema grande para el mercado financiero desde cero



- Utilizar conocimiento financiero adquirido previamente con nuevas estrategias de desarrollo

Estrategia de desarrollo

- Utilizar prácticas tomadas de Metodologías Ágiles

- XP
- Scrum



- Programar usando Test Driven Development



- Ambiente de Desarrollo: Visual Age Smalltalk



- Ambiente de Persistencia: GemStone/S





Objetivos - Desarrollo

- Realizar integraciones continuas
- No restringir la evolución del diseño debido a la historia del sistema
- Cualquier programador puede modificar cualquier parte del sistema
- Entregar versiones periódicamente a los clientes
- Correr el sistema en dos ambientes de manera transparente
 - Desarrollo (VisualAge) – Sencillo Debuguear – Poco Volumen de Datos
 - Producción (GemStone/S) – Gran Volumen de datos



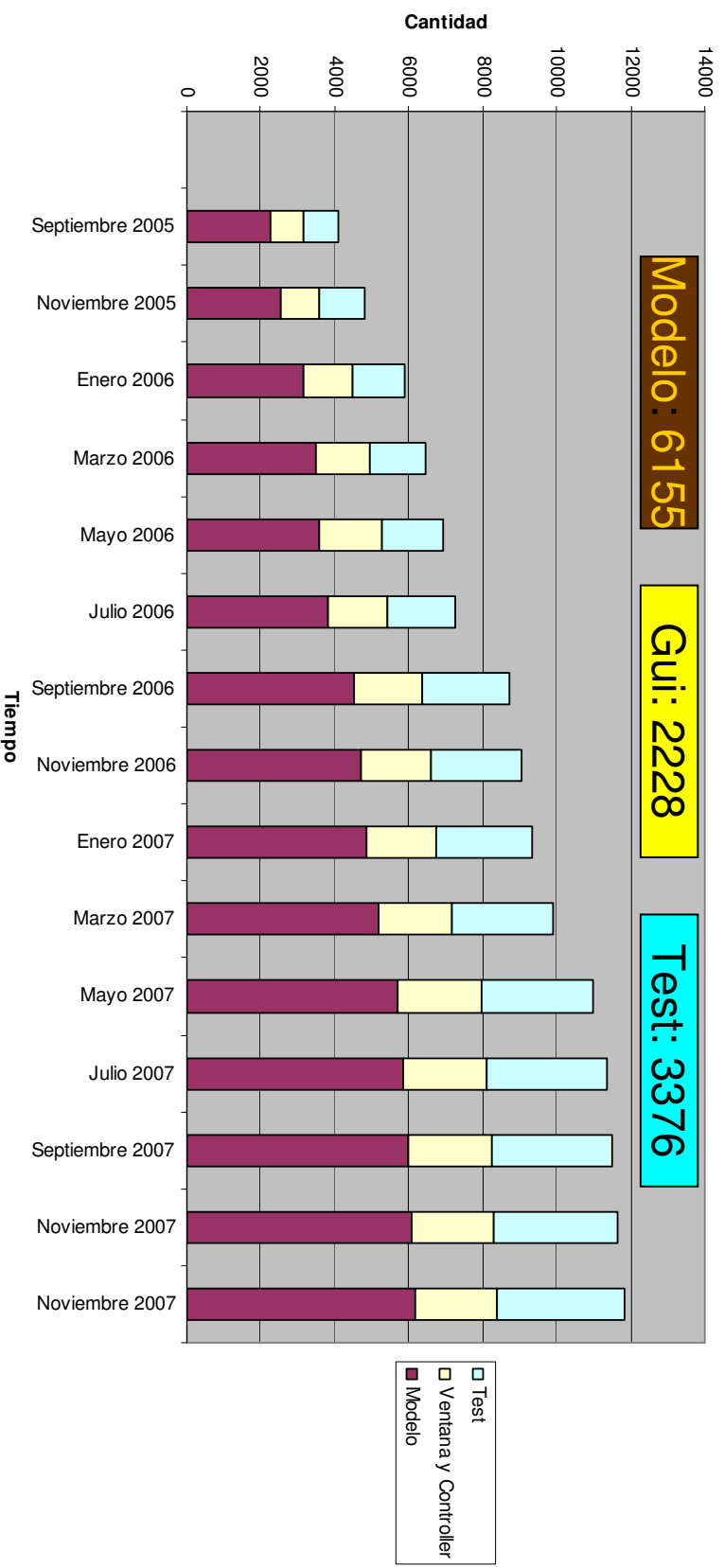
Objetivos - Calidad

- Correr tests unitarios todo el tiempo
- Correr tests idénticos en ambiente de desarrollo y ambiente de persistencia
- Correr tests de regresión en cada versión
- Tomar métricas de manera periódica



Evolución del Sistema – Clases

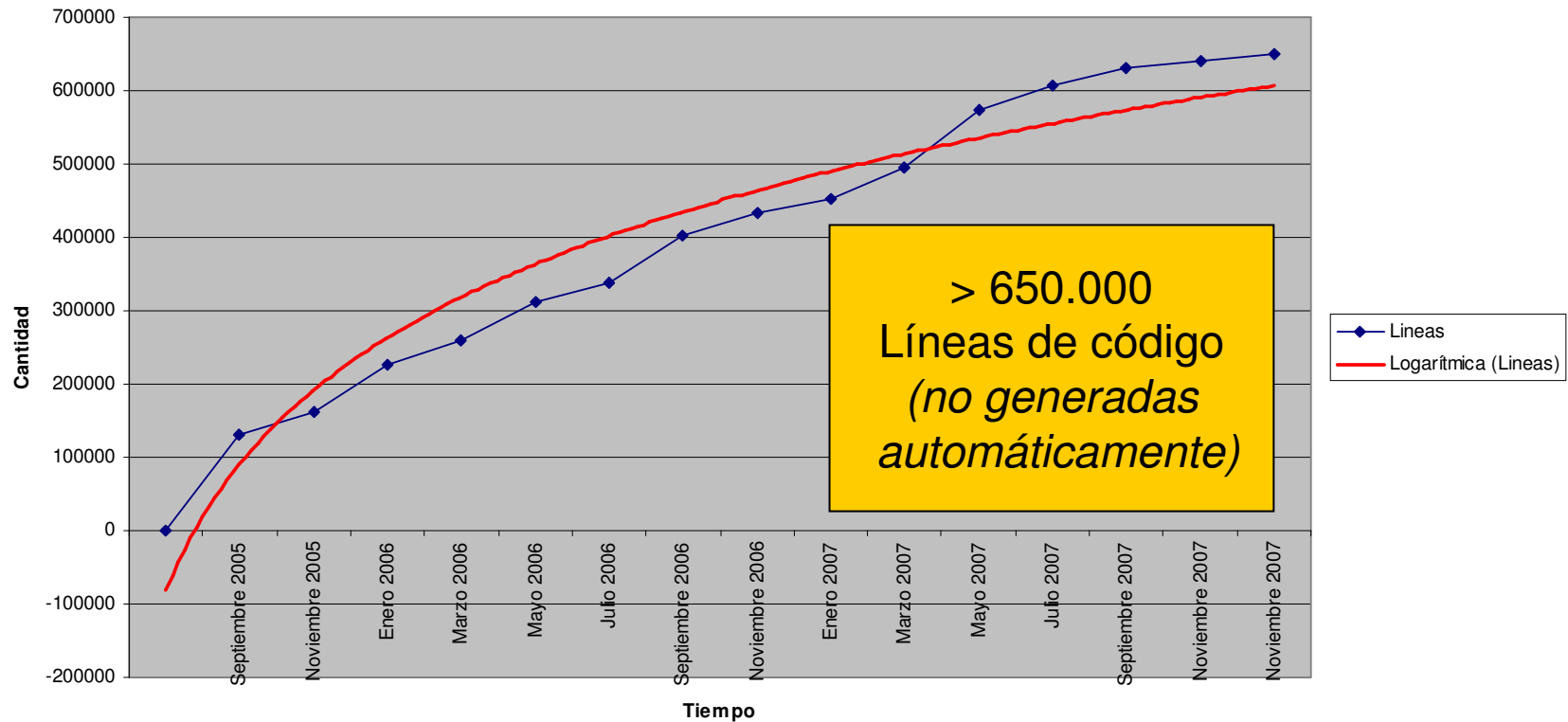
Evolución de Clases





Evolución del Sistema - SLocs

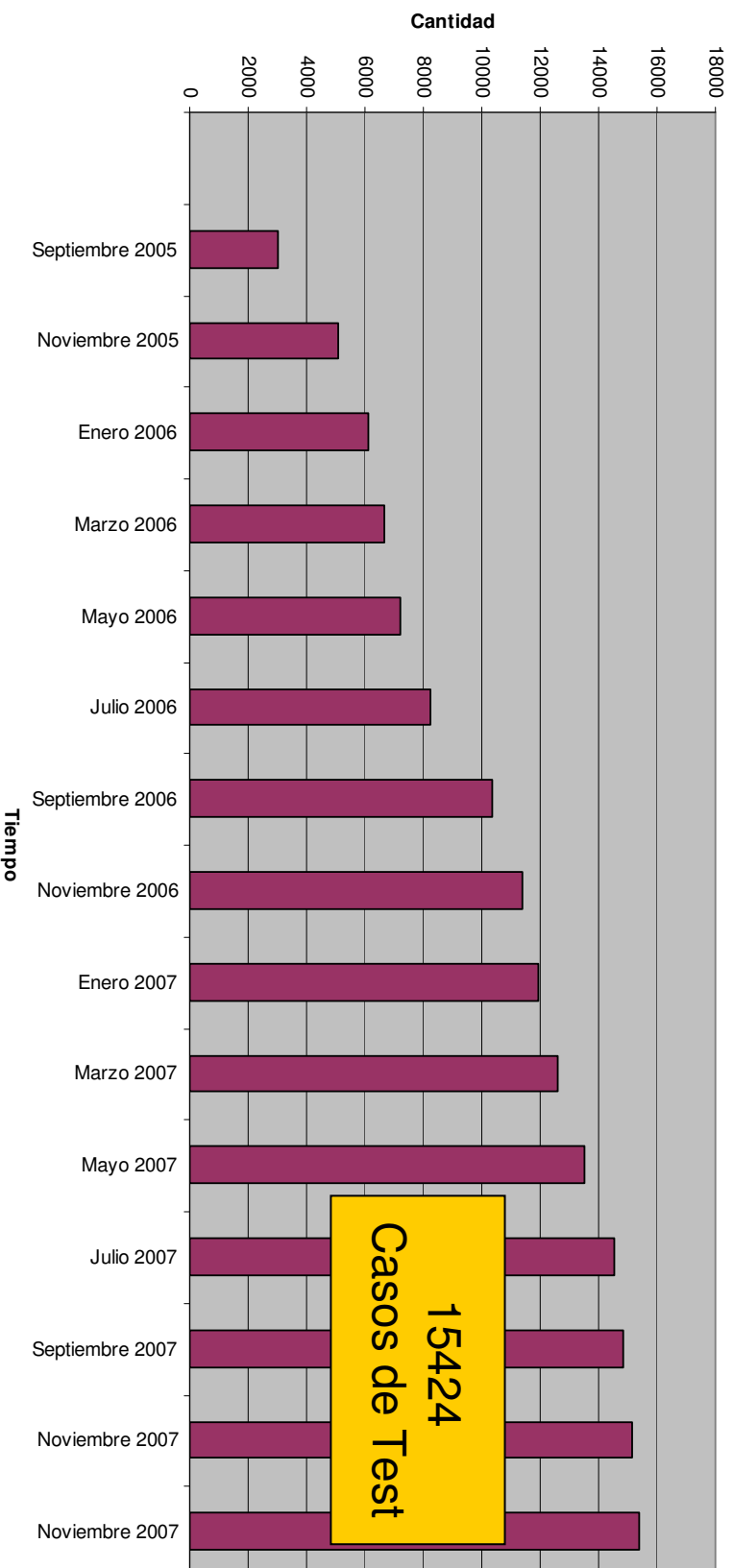
Líneas de Código





Evolución del Sistema – Tests Unitarios

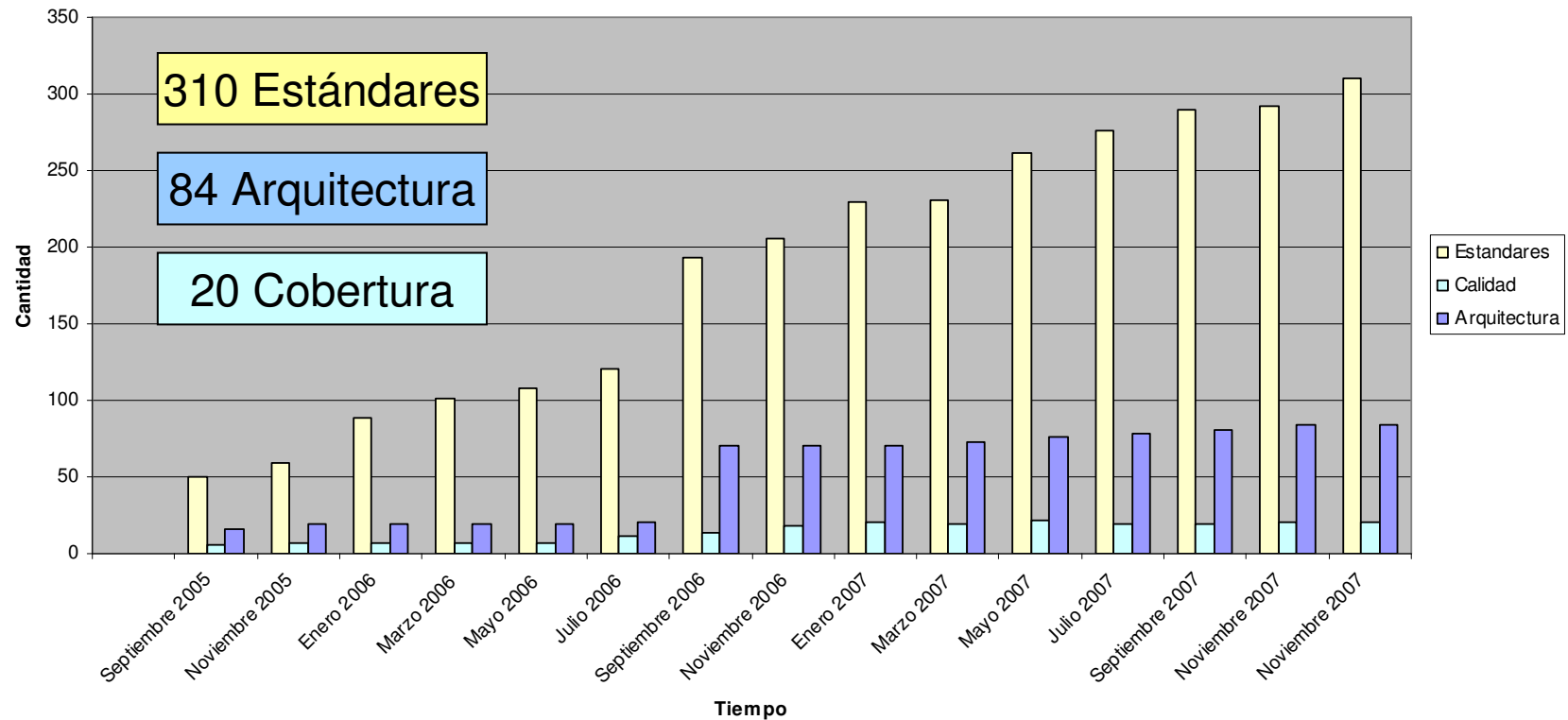
Evolución de Tests Unitarios





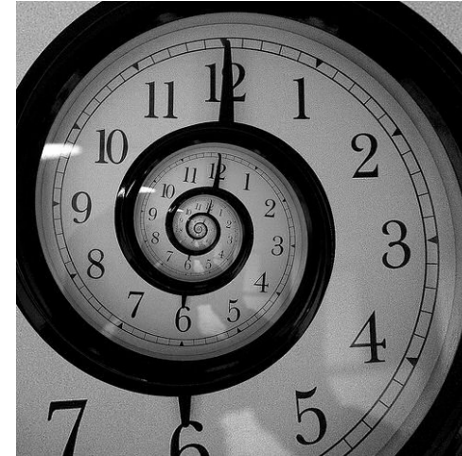
Evolución del Sistema – Otros Tests

Evolución de Tests



Agenda

- Escenario y Objetivos
- Integración Continua
- Estándares de Calidad
- Migración de objetos
- Conclusiones





Objetivos – Integración de Código

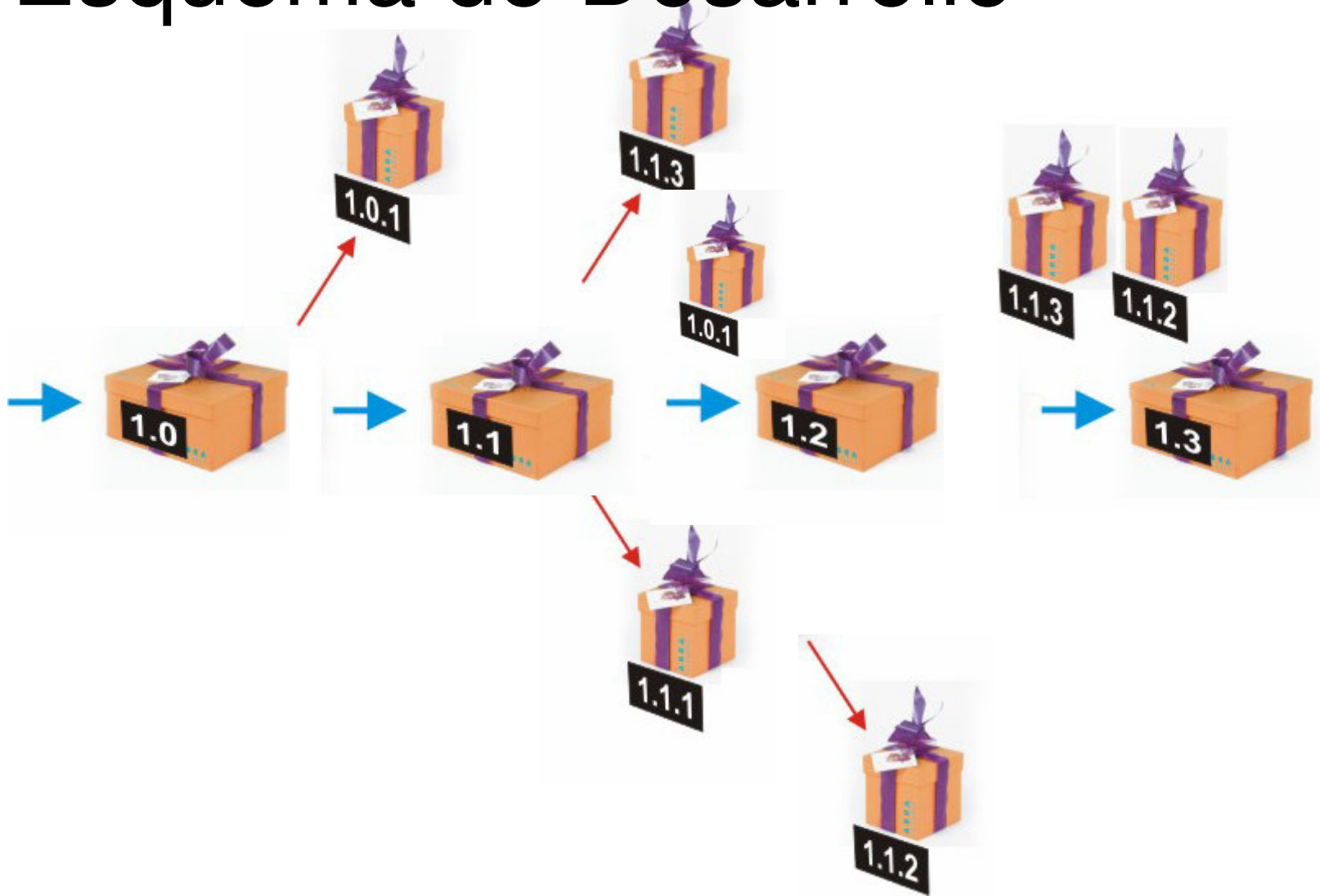
- Mantener una única línea base de producto (Común a todos los clientes)
- Alentar los desarrollos cortos
- Minimizar los conflictos de commit al repositorio de código
- Encontrar errores de manera temprana



Integración - Mecánica

- Cada programador mantiene su propia rama de desarrollo (Imagen Smalltalk)
- El 100% de los tests deben funcionar antes de enviar a integrar su rama
- Si el desarrollo es largo el programador debe ir actualizándose **desde** la línea base

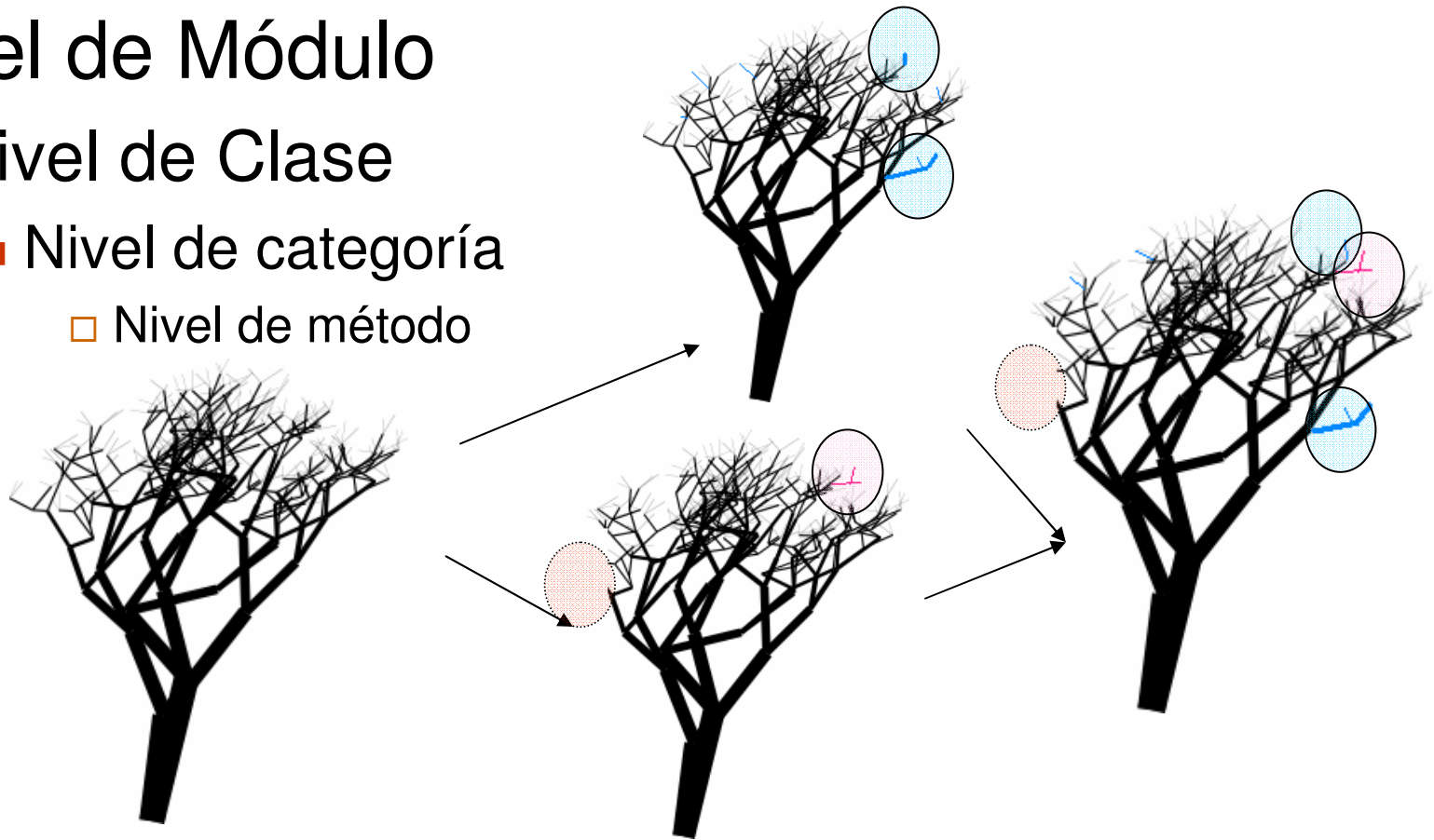
Esquema de Desarrollo





Integración – Granularidad

- Nivel de Módulo
 - Nivel de Clase
 - Nivel de categoría
 - Nivel de método





Integración – Ejemplo en Vivo



Integración Automática Conflictos

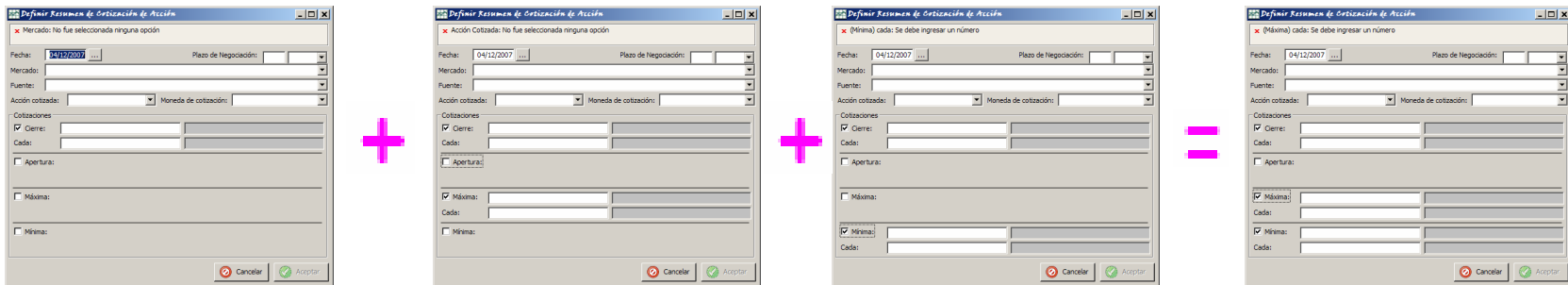


- Aplicación/Clase/Método Borrado y modificado a la vez
 - Resolución Manual
 - Ej. *Rename Class* (Ver presentación de Diego Tubello)
- Método modificado por dos personas
 - Resolución Manual

- Resultados: el conflicto a nivel método se da en el 0.47 % de los casos

Integración – Casos Especiales

- Integración de definición visual
 - Se trata como un método Smalltalk cualquiera



■ Integración de Base de Objetos

- La estructura esta dada por la definición de las clases
- Se trata como una clase Smalltalk cualquiera

Agenda

- Escenario y Objetivos
- Integración Continua
- **Estándares de Calidad**
- Migración de objetos
- Conclusiones





Estándares de Calidad

- ✓ Homogenizan el código
- ✓ Detectan fallas
- ✓ Favorecen el polimorfismo
- ✓ Unifican los nombres de las clases, métodos, variables, etc
- ✓ Sugieren refactorizaciones
- ✓ Detectan código duplicado
- ✓ Evitan uso de mensajes incorrectos
- ✓ Evitan referenciar a clases base no comunes a ambos ambientes
- ✓ Detectan abusos de jerarquías
- ✓ Permiten Inspecciones automatizadas
- ✓ Adherencia a implementación de patrones



Estándares de Calidad - Ejemplos

- Métodos muy largos
- Referencias a clases abstractas
- Falta redefinición de SubclassResponsability
- Métodos especiales que no deberían ser redefinidos
- Mensajes que no deberían ser utilizados
- Clasificación de métodos (categorías)
- Problemas con el #= y #hash
- Errores en definición de variables
- Métodos que no referencian a self (utilities)
- Referencias a clases no presentes en uno de los ambientes
- Cobertura de Tests Unitarios Pobre
- Incorrecta utilización de patrones



Estandares de Calidad - Implementacion

- Controles estáticos
 - Se utiliza *Smalllint* incluido en el *Refactoring Browser*
 - Se modelan las fallas conocidas como casos particulares
 - Test unitarios que utilizan el metamodelo (ventaja Smalltalk !!)
 - 87 Inspecciones
- Cobertura de métodos
 - Nivel de entrada a método
 - Se mide con un *methodWrapper* ejecutando los tests unitarios

Estándares de Calidad – Ejemplo en Vivo



Agenda

- Escenario y Objetivos
- Integración Continua
- Estándares de Calidad
- Migración de objetos
- Conclusiones





Migración de Objetos

■ Paso 1

- Se actualiza el repositorio GemStone con las nuevas clases y métodos

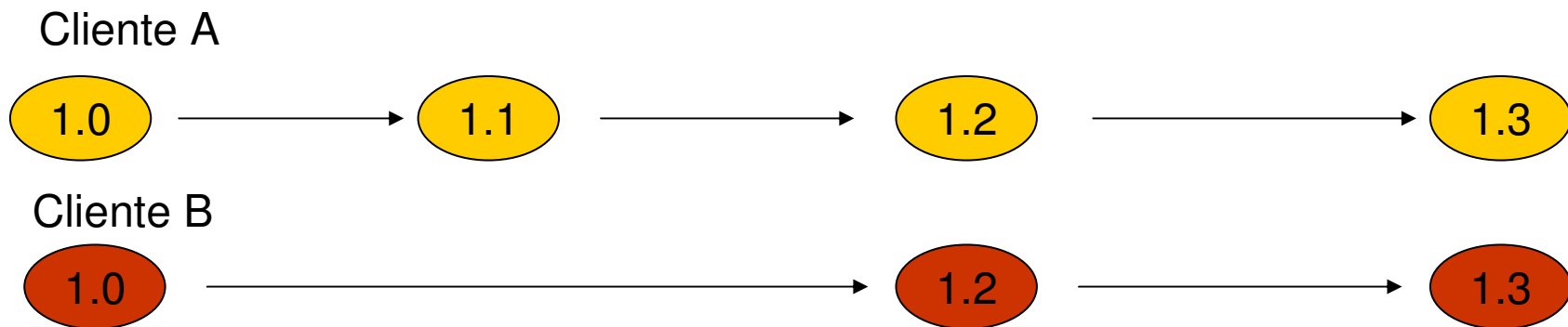
■ Paso 2

- Se realizan transformaciones y se eliminan rastros de la historia de las clases

Migración de Objetos

■ Invariantes

- Se mantiene una única versión en la historia de las clases por lo que deben migrarse los objetos a la versión actual
- Se debe poder migrar acumulando scripts





Migración – Casos complicados

- Clases con nuevas variables de instancia
 - Se deben inicializar
- Inicialización de variables de clase
- Movimiento de clases de symbolList (Aplicación)
- Movimiento de clases entre segmentos con diferentes permisos
- Scripts de transformación cuando cambia la representación
- Instalación de sistemas en ‘caliente’
- Las referencias a clases no se pueden migrar
 - Regla de diseño: No usar las clases mas que para crear instancias



Migración de Objetos

Casos Especiales

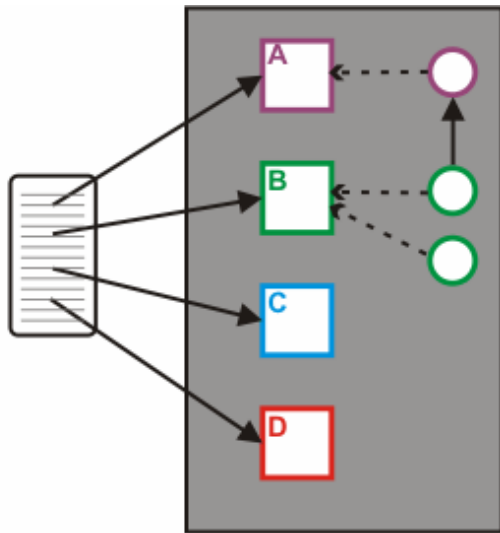
- ❑ Clases borradas con instancias
- ❑ Clases renombradas
- ❑ Clases movidas de segmento
- ❑ Clases con transformaciones (por ej: se declaran singleton)
- ❑ Scripts de transformación



Migración de Objetos - Integridad

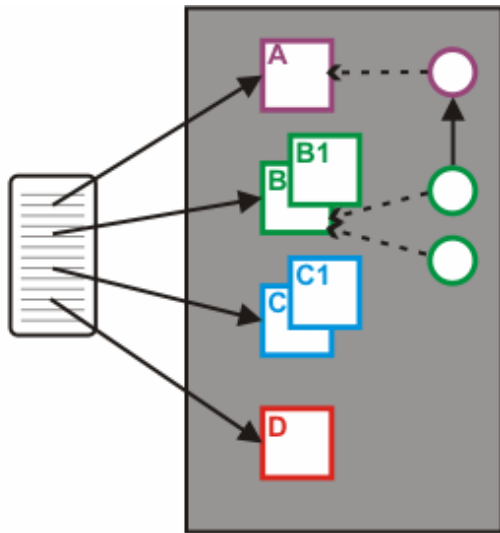
- Tests de Sincronización
 - Controlan que los métodos y clases en el ambiente VisualAge es correspondan unívocamente con los del ambiente GemStone
- Tests de Consistencia
 - Verifican que no haya clases con historia, clases no publicadas, instancias de meta clases con clases no publicadas
- Tests de Integridad
 - Tests particulares verificando los invariantes de representación y reglas de negocio

Migración – Ejemplo



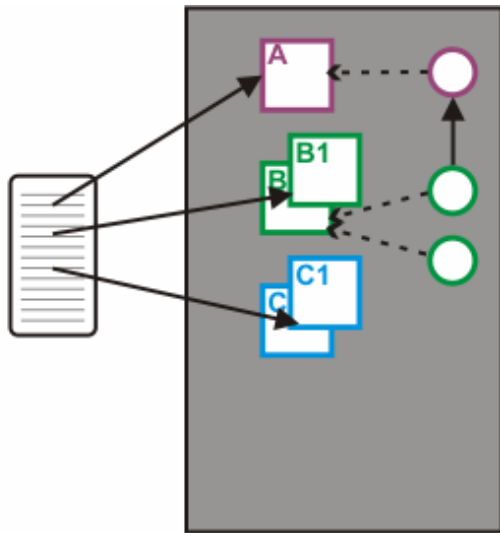
- Antes de la migración
- Las versiones de clases se publican en un diccionario global

Migración – Evolución de Objetos



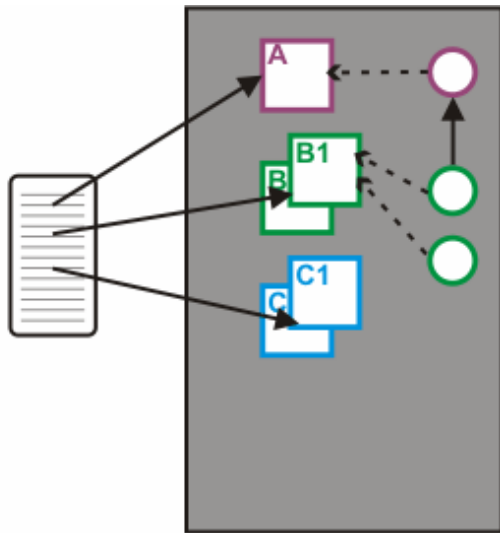
- Luego de la sincronización se crean nuevas versiones de las clases de estructura incompatible

Migración – Evolución de Objetos



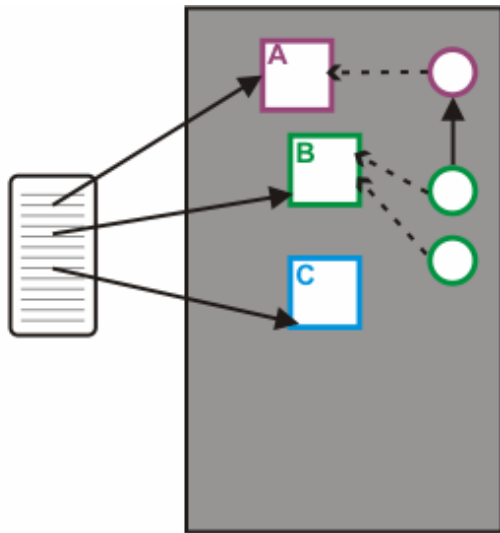
- Las versiones nuevas son publicadas
- Las clases eliminadas pierden las referencias
 - Serán recolectadas por el GC

Migración – Evolución de Objetos



- Las instancias se migran a las versiones publicadas
- Se realizan todas las transformaciones necesarias para adecuar la nueva estructura de clase

Migración – Evolución de Objetos



- Las versiones viejas pierden toda referencia
 - Serán recolectadas
- La Base de objetos esta en un nuevo estado seguro

Migración de Objetos – Ejemplo en Vivo



Agenda

- Escenario y Objetivos
- Integración Continua
- Estándares de Calidad
- Migración de objetos
- Conclusiones





Conclusiones

Integración Automática

- Redujo los tiempos de integración
 - Tasa de 5 desarrollos/día a mas de 30 d/día
- Redujo la cantidad de conflictos
 - %0.47 de los métodos integrados
- Permitted a los desarrolladores ponerse al día con la línea base
 - Gaps de Integración mas cortos
- Test de Herramienta: 1385 Tests Unitarios



Conclusiones - Calidad

- Se evitan errores ante nuevas situaciones
- La cobertura permite escribir código defensivo
- Las reglas están escritas y son enforzadas automáticamente
 - Nuevos programadores cumplen desde el primer día
 - Viejos programadores no las olvidan
- Cobertura de Tests de todo el código: 45%
 - Enorme para un sistema de mas de 600.000 Slocs
- Test de Herramientas: 721 Tests Unitarios



Conclusiones - Migración

- No se restringe la evolución del desarrollo
 - El desarrollo es independiente y se evitan mantener objetos con diseño 'legacy'
- Transparente para usuarios en producción
- Tests de integridad validan el repositorio de objetos
- Test de Herramienta: 510 Tests Unitarios



Conclusiones Generales

- Smalltalk permite hacer evolucionar un ambiente donde los objetos de negocio y herramientas conviven de manera **uniforme**
- Se pudo desarrollar un sistema de mas de *650.000 Slocs* con grupos reducidos de entre 4 y 8 personas
- Gracias a las herramientas desarrolladas los programadores se dedican **únicamente** a resolver problemas del negocio sin perder el tiempo en realizar integraciones, preocuparse por el estado del repositorio de objetos o por revisar el código de los demás
- Muchos de los errores se cometen una **única vez** y luego se agregan reglas de calidad automáticas

Trabajo Futuro

- Integración
 - Integrar refactorings
- Estándares de calidad
 - Medir código repetido
- Migración
 - Empaquetar comandos para entregar al cliente
 - Migración sin necesidad de estar conectado a un repositorio
 - Deducir transformaciones en base a la evolución de clases



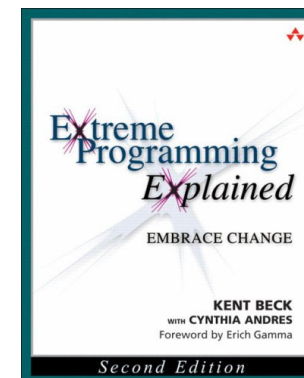
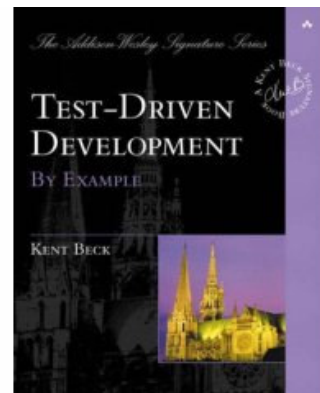
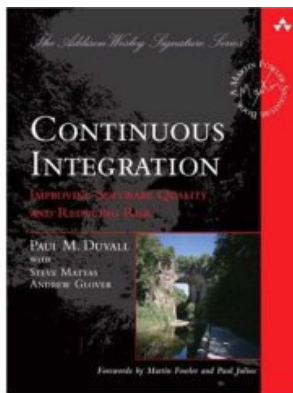
Recursos

Integración

- <http://www.integratebutton.com/>
- <http://martinfowler.com/articles/continuousIntegration.html>

Calidad

- <http://st-www.cs.uiuc.edu/users/brant/Refactory/RefactoringBrowser.html>
- Imaginación y Coraje





Preguntas

