Validación de Software

M.I. Capel

ETS Ingenierías Informática
y Telecommunicación
Universidad de Granada
Email: manuelcapel@ugr.es

Desarrollo de Software

Índice

Índice

Índice

Objetivos y principios fundamentales

Concepto

La actividad denominada *prueba de software* (*Testing*) consiste realmente en un conjunto de actividades sistemáticas que se planifican antes de comenzar el desarrollo de software (codificación) y son realizadas por *personal especializado* y, para proyectos muy grandes, por un equipo de expertos en pruebas (“*testers*")

Motivación

Corrección de defectos en la industria del software

Fase del desarrollo	Coste / defecto (USD)
Diseño y compilación	139
Compilación o encuadernación	455
Integración y pre–producción	977
En el mercado	7,136

Costes ocasionados a la industria por software defectuoso (Basili–Boehm)

El coste promedio a la industria por corregir cada defecto de un software que haya salido de la responsabilidad del equipo de desarrollo y que sea detectado por el cliente que recibe el producto es de 14,102 USD.

Motivación

Corrección de defectos en la industria del software

Fase del desarrollo	Coste / defecto (USD)
Diseño y compilación	139
Compilación o encuadernación	455
Integración y pre–producción	977
En el mercado	7,136

Costes ocasionados a la industria por software defectuoso (Basili–Boehm)

El coste promedio a la industria por corregir cada defecto de un software que haya salido de la responsabilidad del equipo de desarrollo y que sea detectado por el cliente que recibe el producto es de 14,102 USD.

Origen de los defectos del software

Porcentaje de defectos respecto de la fase del desarrollo

%	Fase (introducidos en)
85	Diseño y codificación
< 2	Compilación y encuadernación
< 2	Integración
> 5	En el mercado

¿Qué es y qué no es la actividad denominada “prueba del software”?

Ejemplo: cómo probar un coche antes de comprarlo

¿Qué exámenes o comprobaciones debemos hacer antes de comprarnos un coche nuevo?

- Desde el punto de vista de usuarios–conductores del vehículo, no podemos repetir todos los diferentes tipos de pruebas que ha realizado el fabricante antes de sacarlo al mercado.
- Hemos de limitar nuestras pruebas a comprobaciones factibles a través de Internet o en el concesionario de automóviles.

¿Qué es y qué no es la actividad denominada “prueba del software”? – II

Ejemplo de las pruebas a realizar (punto de vista del conductor)

- Validar la financiabilidad de la compra
- Que nos guste el coche
- Comprobar el grado de confort
- Capacidad, versatilidad, mantenibilidad
- Prestaciones:
  - Gasto combustible / 100 Km
  - Tipo de combustible
  - Tiempo mínimo para conseguir velocidad
  - Estabilidad en curvas
  - Estabilidad en alta velocidad ($>$ 180 KM/h)
  - Tiempo promedio entre mantenimientos

Pruebas del software

Idea fundamental

- Las pruebas sólo pueden verificar el sistema y su operación respecto de criterios predeterminados o *requerimientos* previos del software
- La calidad del software ha de encontrarse dentro de éste, no es algo que se decida o se pueda incluir en la fase de pruebas

Tipos de pruebas aplicadas en el ejemplo de la compra de vehículo

Correspondencia con la terminología de pruebas

Tipo prueba	Nombre técnico
Examinar el coche sin conducirlo	Pruebas estáticas
Comprobar caracte-rísticas sin conducirlo	Pruebas funcionales y estructurales
Probar a conducirlo	Prueba de rendimiento

Tipos de pruebas del software

Verificación

¿Estamos construyendo el producto correctamente?

Verificación se refiere al conjunto de tareas que aseguran que el software producido implementa correctamente una función especifica.

Validación

¿Estamos construyendo el producto correcto?

Validación se refiere a un conjunto diferente de tareas para poder seguir(“traceable”) al software hasta los requisitos.

Tipos de pruebas del software

Verificación

¿Estamos construyendo el producto correctamente?

Verificación se refiere al conjunto de tareas que aseguran que el software producido implementa correctamente una función especifica.

Validación

¿Estamos construyendo el producto correcto?

Validación se refiere a un conjunto diferente de tareas para poder seguir(“traceable”) al software hasta los requisitos.

Tipos de pruebas del software

Verificación

¿Estamos construyendo el producto correctamente?

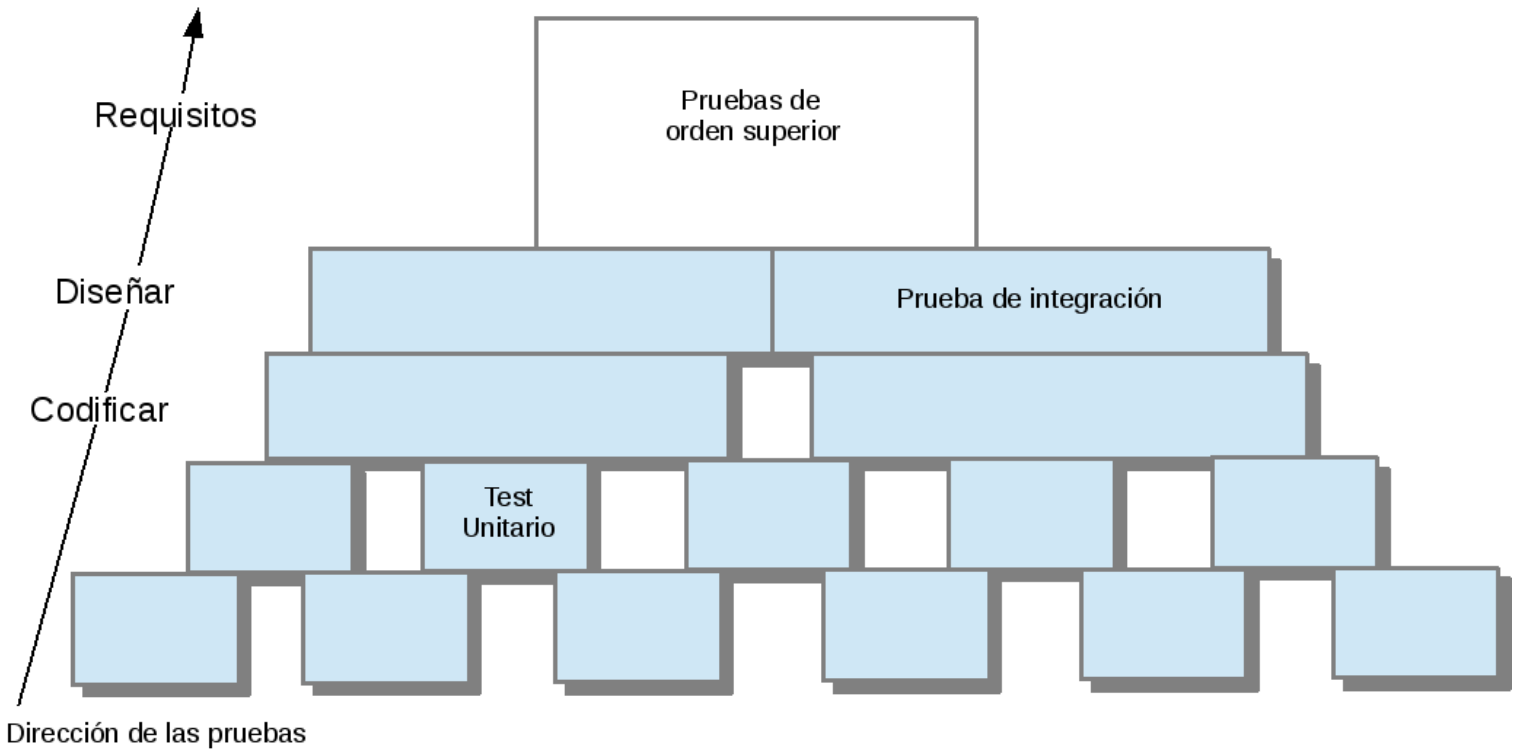
Verificación se refiere al conjunto de tareas que aseguran que el software producido implementa correctamente una función especifica.

Validación

¿Estamos construyendo el producto correcto?

Validación se refiere a un conjunto diferente de tareas para poder seguir(“traceable”) al software hasta los requisitos.

Proceso de prueba del software



Proceso de prueba del software – II

Pasos del proceso

1. *Pruebas estáticas*: se refieren a las pruebas que se realizan cuando la aplicación no se está ejecutando
2. *Pruebas unitarias*: comprueban caminos específicos en la estructura de control de un componente
3. *Prueba de integración*: los componentes se ensamblan para formar un paquete software completo
4. *Las pruebas de validación*: el software ha de cumplir todos los requisitos: *funcionales*, *comportamiento*, *rendimiento*...
5. *Pruebas del sistema*: comprueba que todos los elementos (software+hardware) se combinan bien y que se consigue el rendimiento/funcionamiento global esperado

Criterios para determinar el fin del proceso de prueba del software

Técnicas para determinar el fin

- *Cleanroom Software Engineering*: utilización de técnicas estadísticas de utilización
- Utilización de modelos estadísticos y teoría de confiabilidad del software para predecir la completitud de las pruebas
- Recogiendo muestras (y métricas) durante la realización de pruebas del software
- Utilizando modelos de confiabilidad del software

Pruebas Estáticas

Idea fundamental

- Para realizar este tipo de pruebas no es necesario ejecutar la aplicación que queremos probar
- Reducción de defectos del software debido a la mejora de la documentación, a partir de la cual se desarrollan
- Ayudan a conseguir la operación correcta por el usuario final, cuando el software está ya desarrollado y entregado
- Es una de las formas más costo–efectivas de identificar los probables defectos en el software de las aplicaciones
- Son recomendables para cualquier fase del desarrollo, excepto en la de instalación del software

Documentación para revisar

1. Software Development Manager/ Gestor de Desarrollo de Software
2. Software Developers/ Desarrolladores del Software
3. Testers / Equipo de Pruebas
4. Administrator / Administrador del Sistema:
   - Guías de Instalación
   - Guías de Operación/Administración
5. Documentación de Usuario:
   - Guías de usuarios
   - Pantallas de ayuda
   - Manuales de entrenamiento

Técnicas de Pruebas Estáticas

Revisión de Contenidos (basadas en)

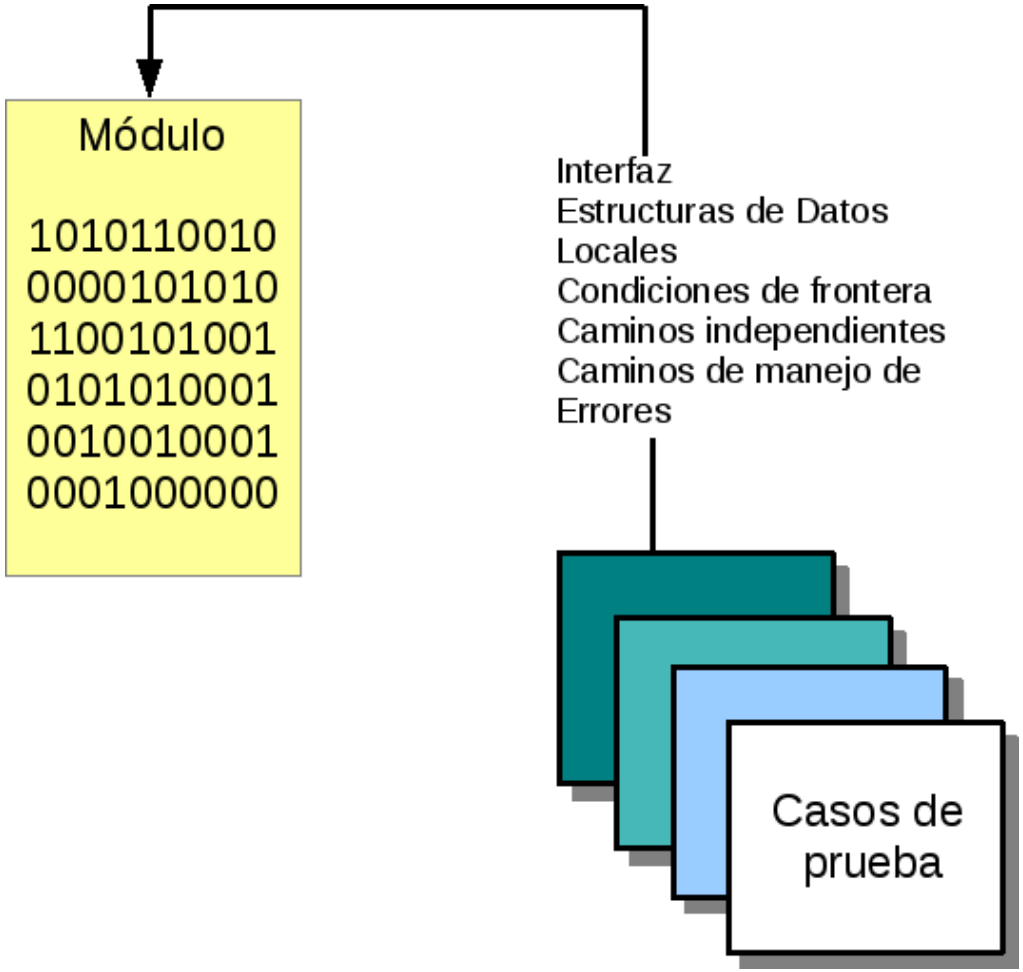
- Pruebas de Escritorio (“Desk Checking")
- Inspecciones
- Pruebas Estructuradas de Recorrido (“Walkthroughs")

Pruebas Unitarias

Antecedentes

- Concentra el esfuerzo de comprobación en la unidad más pequeña de diseño de software
- Trata de encontrar errores en la lógica interna de proceso o en las estructuras de datos de un componente
- Se prueban caminos de control importantes para descubrir errores dentro del ámbito de cada prueba
- Los errores que se pueden producir y la complejidad de las comprobaciones están, por tanto, limitados

Pruebas Unitarias – II

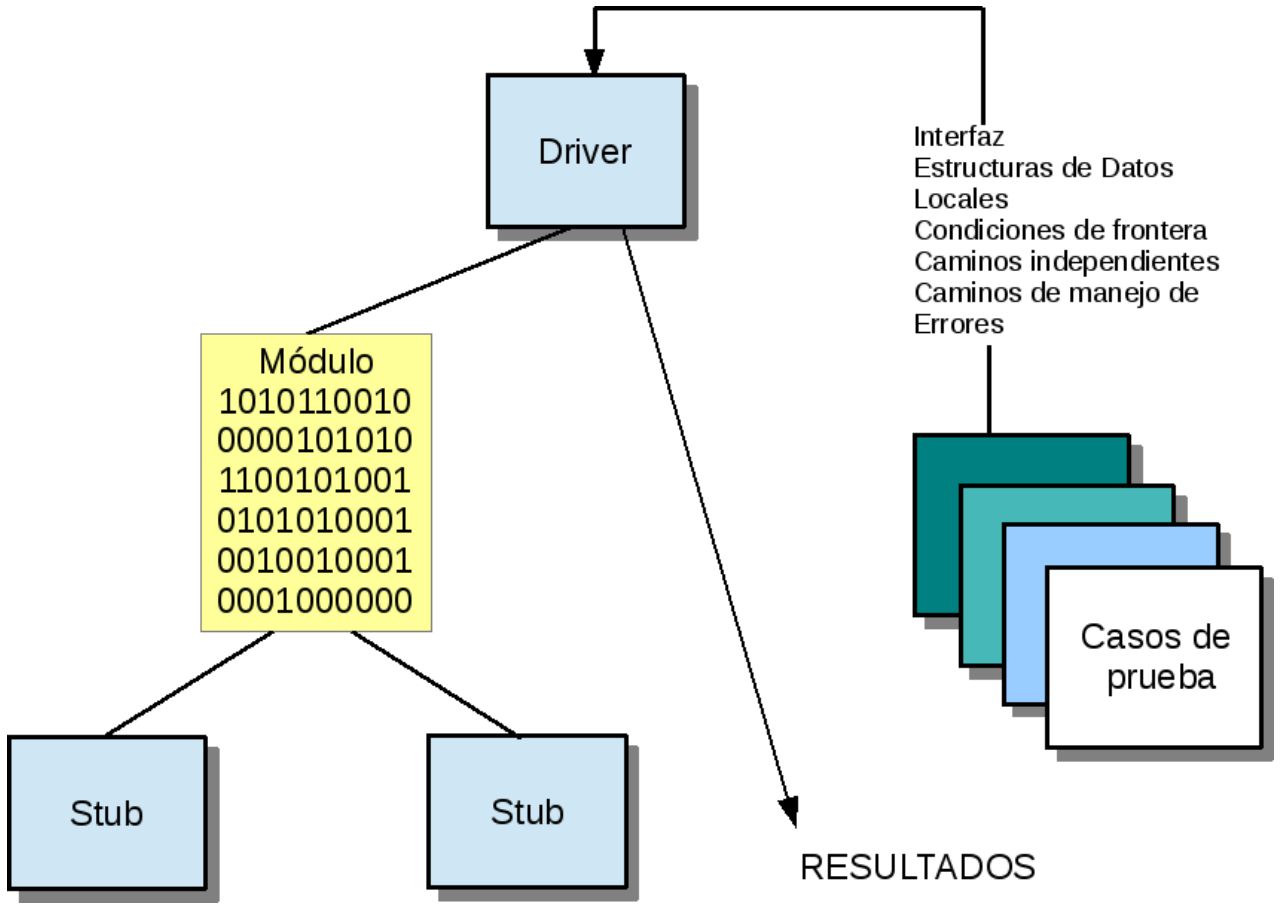


Pruebas Unitarias – III

Prueba Unitaria

- Lo primero es comprobar el flujo de datos a través de la interfaz del componente
- Las Estructuras de Datos han de ser *ejercitadas*
- Determinar el impacto que producen en los datos globales de la aplicación
- Seleccionar los caminos de ejecución a comprobar
- Diseñar casos para descubrir errores de computación, comparaciones incorrectas o flujos de control impropios
- Un buen diseño de software ha de anticipar errores y establecer su tratamiento cuando estos se produzcan

Entorno para Pruebas Unitarias



Pruebas unitarias de componentes

- Las pruebas unitarias se consideran normalmente como adjuntas a la actividad de codificación del software
- El diseño de cada prueba unitaria puede ocurrir antes de que comience la codificación de un software, o después

Para probar componentes hay que desarrollar:

- Concepto de *Driver*–software
- Concepto de *Stubs*
- Un *stub* utiliza la interfaz de un módulo subordinado (pero sin pasarle la llamada) y devuelve el control al componente
- Drivers y stubs representan la sobrecarga del software debida a su prueba

Pruebas de integración

Cuestión Fundamental

Si todos los componentes software, tras efectuar las pruebas unitarias, funcionan individualmente ¿Por qué dudar de que funcionen también cuando se combinan todos en una aplicación?

Falta de composicionalidad de las pruebas:

- Problemas de interfaz entre componentes
- Cuando se combinan sub–funciones pueden no producir la función principal deseada
- Problemas de composición entre estructuras de datos globales repartidas entre componentes
- Los errores individuales de los componentes se amplifica

Concepto de prueba de integración

Pressman, 2010

"Se trata de una técnica sistemática para construir la arquitectura del software mientras que, al mismo tiempo, se realizan pruebas para descubrir errores asociados con la inter-actuación de los componentes (o problema de las interfaces) de un software"

¿Dónde (fase el ciclo) se deben llevar a cabo?

Se combinan bien con la construcción de la arquitectura de software del sistema completo

Técnicas de Pruebas de Integración

Integración incremental vs. "Big–Bang"

Todas las pruebas de los componentes combinadas de una sola vez, y el programa se prueba entero: ¡¡¡¡El Caos!!!!

La *integración incremental* es la antítesis de hacer *"Big–Bang"*

El caso de las interfaces

La prueba de las interfaces no se presta a la utilización de una técnica "Big Bang"

Tipos de pruebas de integración de software

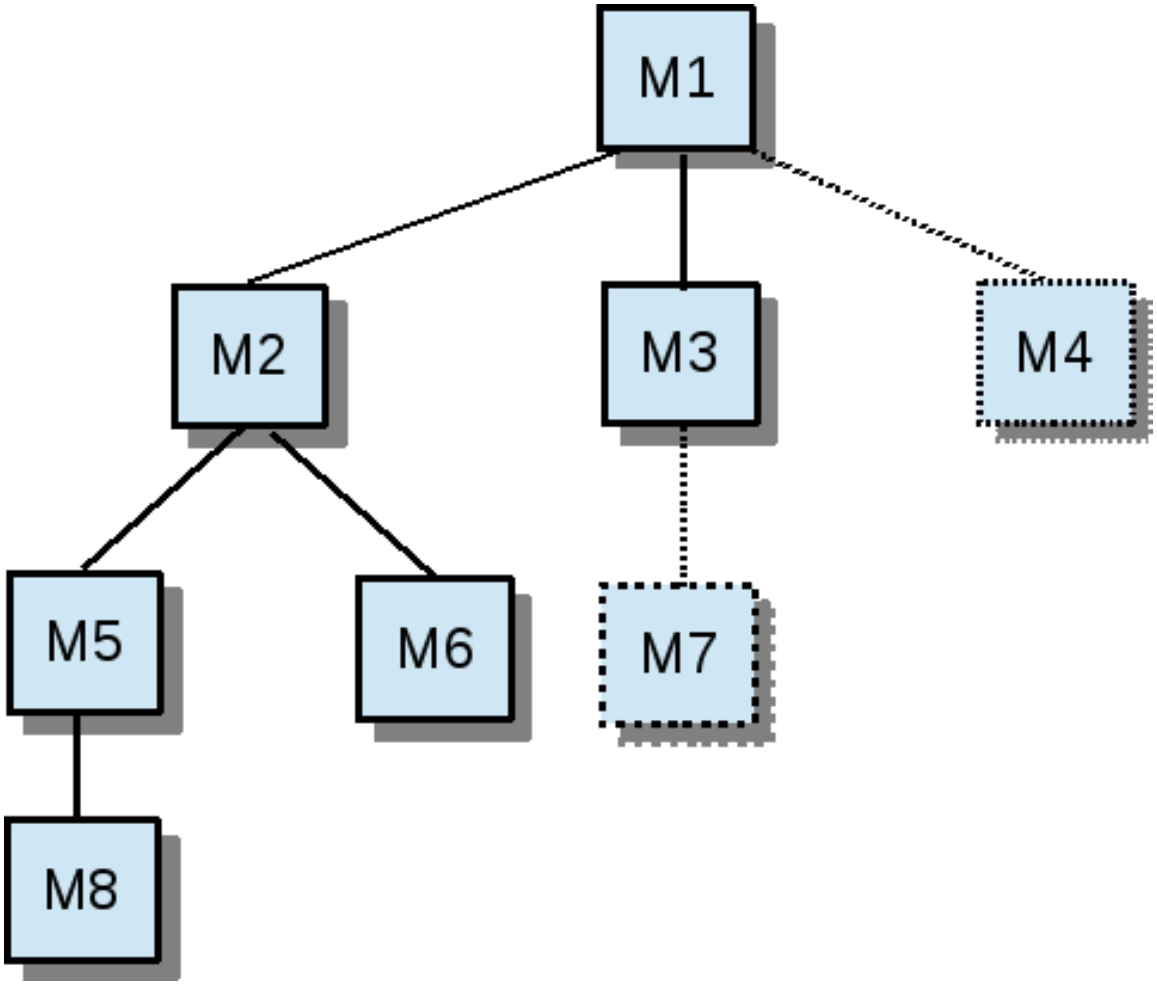
- Top-Down
- Bottom-Up

Integración Top-Down

Idea fundamental

- Se trata de una prueba de integración incremental
- Seguir la jerarquía de control (se supone en forma de árbol), hacia las *hojas*
- Comenzar con el *módulo de control principal*(raíz del árbol)
- Se incorporan a la estructura jerárquica, recorriéndola
- Seleccionar el orden de integración de las pruebas (recorriendo el árbol en profundidad o en anchura) dependerá de las características de la aplicación

Integración Top-Down II



Integración Top-Down III

Pasos del Proceso de Integración Top–Down

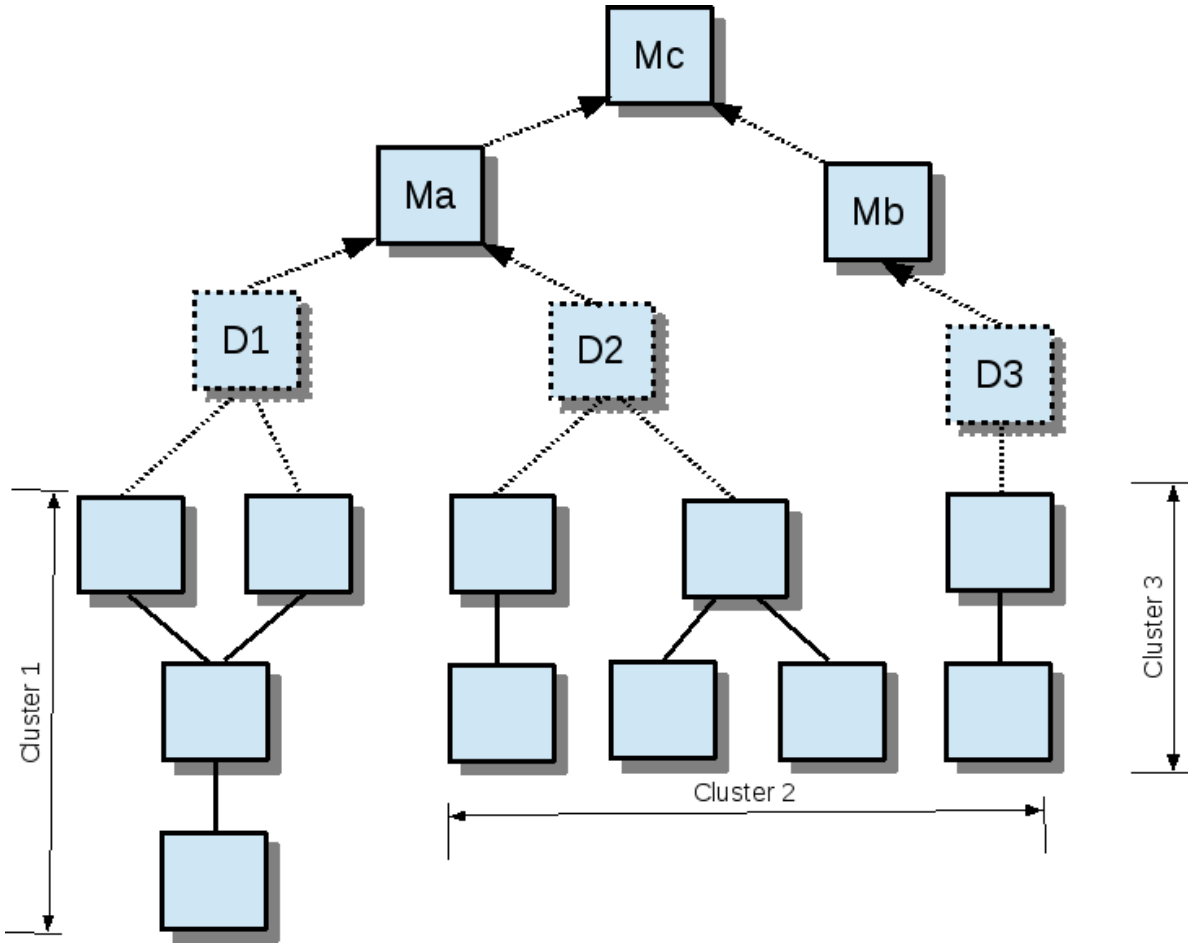
1. *Driver* de la prueba: módulo de control principal
2. *Stubs*: módulos subordinados al principal
3. *Estrategia*: reemplazo de cada *stub* por el componente real al que representa, después volver a ejecutar la prueba
4. Se realizará una *prueba de regresión* para asegurar que no se han introducido nuevos errores

Integración Bottom–Up

Idea fundamental

Se comienza construyendo y probando primero *módulos atómicos*, es decir, componentes en los niveles más bajos de la estructura del programa, eliminando, de esta forma, la necesidad de crear *stubs* en la estructura de la prueba de integración

Integración Bottom–Up II



Integración Bottom–Up III

Pasos del Proceso de Integración Bottom–Up

1. Los módulos atómicos se combinan en “builds"
2. Se programa un *driver* para cada *build*
3. Se prueba el *build*
4. Se van eliminando *drivers* y los *builds* se combinan en *clusters*
5. Desplazamiento hacia arriba en la estructura jerárquica de la prueba de integración

Prueba de Regresión

Concepto

- El software cambia, durante una prueba de integración, cada vez que se añade un módulo. Tales cambios pueden producir problemas en el comportamiento de funciones que antes funcionaban perfectamente.
- Se trata de la re-ejecución de algunos subconjuntos de pruebas que ya han sido realizadas para asegurar que los cambios no se han propagado produciendo efectos colaterales indeseados en el resto del software ya comprobado

Prueba de Regresión II

Contenidos de la prueba:

1. Un ejemplar representativo de todas las posibles comprobaciones, que ejerciten todas las funciones del software a probar
2. Comprobaciones que se centran en la prueba de componentes software que han sido cambiados
3. Comprobaciones adicionales que se enfocan a la prueba de funciones de software que podrían verse afectadas por los cambios

Pruebas funcionales

Concepto

- Validar el comportamiento del software con respecto a lo que tiene que hacer, tal como aparece documentado
- Ejercitar lo más completamente que sea posible el software que soporta la actividades diarias de un negocio, mediante la ejecución de *casos de prueba*:
  - Se obtienen a partir de los *casos de uso* previos
  - Inicialmente para componentes individuales: menús,ítems de datos, páginas web, bases de datos, etc.
  - Finalmente, se prueban los componentes software de forma conjunta y en orden inverso al del código producido

Tipología de casos de prueba utilizados normalmente

- Prueba de navegación del usuario
- Pruebas de ventanas de transacciones
- Prueba del flujo de las transacciones
- Prueba de ventanas de información al usuario
- Prueba de flujo de ventanas de información
- Pruebas de eliminación/ actualización/ recuperación/ creación de registros en una base de datos

Pruebas de Rendimiento

Concepto

- Se trata de validar la *velocidad de ejecución* del software respecto de la *necesidad de velocidad* del negocio, tal como se expresó en el documento de requerimientos
- La denominada *velocidad* del software se consigue como una buena combinación del tiempo de respuesta y la carga de trabajo cuando se producen *cargas–pico* debidas a los usuarios activos
- Pruebas de rendimiento: una serie de tests que introducen de forma incremental más carga de trabajo por una combinación de transacciones de un negocio para varias ventanas temporales de ejecución de la aplicación

Metas Importantes de las Pruebas de Rendimiento

Desafíos a Afrontar

1. Identificar correctamente a qué transacciones y actividades concretas del negocio se les necesita *medir el rendimiento*
2. Por cada grupo de transacciones relevante del negocio, determinar la *utilización–pico* de recursos computacionales y cuándo ocurren (ventanas de tiempo) los picos
3. Determinar cuántos picos de carga de trabajo hay que comprobar para conseguir una buena prueba de rendimiento

Técnicas de Planificación de las Pruebas

Pasos para pruebas de carga de trabajo

- Llevar a cabo una intensificación de la carga de trabajo del software hasta llegar a una situación de *carga–pico*
- Ejecutar medidas de rendimiento en el entorno de dicho pico de carga de trabajo
- Llevar a cabo una disminución de la carga de trabajo del software desde la última situación de *carga–pico*

Técnicas de Planificación de las Pruebas-II

Documentación de Requisitos de Carga de Trabajo

- El análisis de carga de trabajo de una aplicación software debe identificar los grupos de transacciones y sus requisitos de rendimiento

Grupo Transacciones	Tiempo respuesta
Navegar menús	3s. (máx)
Des/Conexión	3s. (máx)
Información productos	4s. (máx)
Operación compra	7s. (máx)
Búsqueda catálogo	10s. (máx)
Pago con Tarjeta	30s. (máx)
Envío	24h. (máx)

Tabla: Ejemplo de aplicación de compras por Internet

Técnicas de Planificación de las Pruebas-II

Documentación de Requisitos de Carga de Trabajo

- El análisis de carga de trabajo de una aplicación software debe identificar los grupos de transacciones y sus requisitos de rendimiento

Grupo Transacciones	Tiempo respuesta
Navegar menús	3s. (máx)
Des/Conexión	3s. (máx)
Información productos	4s. (máx)
Operación compra	7s. (máx)
Búsqueda catálogo	10s. (máx)
Pago con Tarjeta	30s. (máx)
Envío	24h. (máx)

Tabla: Ejemplo de aplicación de compras por Internet

Técnicas de Planificación de las Pruebas II

Documentación de Picos de Carga de Trabajo

- Para caracterizar un pico de carga lo importante es determinar los usuarios activos en ese momento, no los usuarios conectados
- La competencia por los recursos entre usuarios de una aplicación sólo se produce dentro de una misma ventana temporal
- Instrumentación de las pruebas de carga-pico: determinar *cómo de bien* compiten las transacciones respecto de diferentes cargas de trabajo
- Hay que añadir la previsión de uso máximo de cada grupo de transacciones a la planificación de la carga de trabajo

Técnicas de Planificación de las Pruebas III

Documentación de Picos de Carga de Trabajo (2)

Grupo Transacciones	Tiempo respuesta	Activos	Fecha de la actividad
Navegar menús	3s. (máx)	2000	L-V: 12-13h
Des/Conexión	3s. (máx)	2000	L-V: 12-13h
Información productos	4s. (máx)	2000	L-V: 12-13h
Operación compra	7s. (máx)	500	S: 9-11h
Búsqueda catálogo	10s. (máx)	2000	L-V: 12-13h
Pago con Tarjeta	30s. (máx)	500	S: 9-11h

Tabla: Planificación de la carga de trabajo para obtener rendimiento

Técnicas de Planificación de las Pruebas IV

Documentación de Picos de Carga de Trabajo (3)

- Para obtener una estimación del rendimiento de la aplicación en todo momento, hay que averiguar cuántos *picos de carga de trabajo distintos* hay que comprobar

Grupo Transacciones	Tiempo respuesta	Activos	Fecha de la actividad
Navegar menús	3s. (máx)	500	S: 9-11h
Des/Conexión	3s. (máx)	500	S: 9-11h
Información productos	4s. (máx)	500	S: 9-11h
Operación compra	7s. (máx)	500	S: 9-11h
Búsqueda catálogo	10s. (máx)	500	S: 9-11h
Pago con Tarjeta	30s. (máx)	500	S: 9-11h

Tabla: Planificación de la carga (sábados) para obtener rendimiento

Técnicas de Planificación de las Pruebas IV

Documentación de Picos de Carga de Trabajo (3)

- Para obtener una estimación del rendimiento de la aplicación en todo momento, hay que averiguar cuántos *picos de carga de trabajo distintos* hay que comprobar

Grupo Transacciones	Tiempo respuesta	Activos	Fecha de la actividad
Navegar menús	3s. (máx)	500	S: 9-11h
Des/Conexión	3s. (máx)	500	S: 9-11h
Información productos	4s. (máx)	500	S: 9-11h
Operación compra	7s. (máx)	500	S: 9-11h
Búsqueda catálogo	10s. (máx)	500	S: 9-11h
Pago con Tarjeta	30s. (máx)	500	S: 9-11h

Tabla: Planificación de la carga (sábados) para obtener rendimiento

Técnicas de Ejecución de las Pruebas

Idea general

- Para poder predecir el rendimiento de la aplicación hay que crearse los picos de carga de trabajo en un entorno de pruebas
- Esto se realiza en 3 pasos:
  1. Intensificación de la carga de trabajo hasta alcanzar el pico
  2. Medida de rendimiento en el pico
  3. Disminución de la carga de trabajo desde el pico

Prueba de Rendimiento de un Componente

Idea general

- Tener una estimación temprana acerca de si la inclusión de un nuevo componente software nos puede llevar cerca del valor máximo del tiempo de respuesta previsto en la planificación de carga de trabajo

Prueba de Rendimiento de un Componente II

Rendimiento de "*viaje completo*" (roundtrip)

- Se trata de la medida del tiempo de respuesta de la aplicación desde que el usuario envía su petición hasta que los resultados se han mostrado completamente
- Esta medida de rendimiento incluye:
  - Procesamiento implicado y realizado en la parte del usuario
  - Comunicación necesaria a través de la red
  - Procesamiento secundario en otros computadores de soporte para la transacción

Prueba de Rendimiento de un Componente III

Ejemplo de Medida de Rendimiento de "*Viaje Completo*"

0.5s	venta de compra en el cliente
0.2s	transmisión a la red
2.4s	registro en servidor
1.3s	generar orden al almacén (servidor)
0.7s	generar confiración compra (servidor)
0.1s	tramsión confirmación (red)
0.6s	mostrar registro confirmación (cliente)

Total tiempo de respuesta $= 5.8s <$ tiempo máximo en pico (previsto en la planificación de carga de trabajo)

Pruebas de Administración

Idea fundamental

- Se trata de una ampliación de las pruebas funcionales de las actividades de un negocio
- Prueba de las actividades de *soporte* a un negocio
- Los componentes administrativos de una aplicación podrían haberse desarrollado antes que los funcionales, en ese caso los resultados de la prueba serán el punto de comienzo de la prueba de estos ultimos
- Si ocurre los contrario, los archivos de configuración manual utilizados para probar la funcionalidad de la aplicación se pueden utilizar como los resultados esperados de las pruebas de componentes administrativos

Pruebas del Sistema

Idea fundamental

- Configurar una aplicación instalada consiste en seleccionar entre una lista de opciones cómo ha de operar el software para cumplir con las reglas específicas
- Fijar parámetros de arranque (“start up"): ubicación archivos, número de sesiones, ID/password, zonas...
- Reglas de procesamiento: clases de seguridad, planificación de arranque y backups del sistema,...

Pruebas de Humo

- Este tipo de prueba se utiliza para verificar de forma no exhaustiva que una aplicación software instalada puede ser posteriormente bien configurada
- La dificultad consiste en identificar las combinaciones de configuración más probables de la aplicación para probarlas
- Cada nueva configuración se prueba diferencialmente con respecto a una que anteriormente funcionó

Pruebas Estructurales

Concepto

- Se trata de validar el comportamiento del software que utilizan (dan soporte) las aplicaciones de usuario
- Reduce el riesgo de fallo del denominado *software de plataforma*
- Son pruebas de tipo *no–funcional*
- Las pruebas de *caja blanca* no se pueden aplicar
- La mayoría de las pruebas de *caja negra* tampoco

Técnicas de Pruebas Estructurales

Pruebas de Interfaz

- Probar el paso de datos entre la aplicación y los distintos componentes de la plataforma se realiza correctamente
- Han de ser probados: archivos de datos, APIs, peticiones a bases de datos, mensajes por la red
- Se puede realizar en 4 pasos:
  1. Producir datos pero mantener la trasferencia inhibida
  2. Eliminar los inhibidores de transferencia de datos
  3. Escribir tests para que la aplicación pida datos desde otros componentes de la plataforma
  4. Permitir que los componentes de la plataforma–software alimenten con datos reales de entrada a la aplicación

Técnicas de Pruebas Estructurales – II

Pruebas de Seguridad

- Utilizar clases de equivalencia para probar el comportamiento relativo a la seguridad
- Puede implicar la encriptación de las palabras de paso, lo cual complica la prueba basada en clases de equivalencia
- El equipo de pruebas ha de verificar la degradación del rendimiento de la aplicación por motivo de esta prueba
- Desde el punto de vista de las pruebas de regresión, conviene realizar pruebas de seguridad lo antes posible en la aplicación

Técnicas de Pruebas Estructurales – III

Pruebas de Instalación

- Se trata de comprobar si la nueva aplicación se ha situado correctamente en entorno de producción
- Es necesario probar el proceso de instalación para asegurase que los clientes podrán hacer funcionar la aplicación
- Normalmente se ha de contar con una plataforma software+hardware análoga al entorno de utilización para hacer la prueba de la aplicación
- Hay que proporcionar al cliente información acerca de si el proceso de instalación se desarrolló correctamente

Técnicas de Pruebas Estructurales – IV

Pruebas de Backup y Recuperación

- Los archivos de *backup* se utilizan para restaurar el software a un estado próximo al que tenía antes de fallar
- Si no se prevén situaciones de fallo en las aplicaciones de negocio durante su desarrollo, entonces probablemente éstas no se recuperarán en la eventualidad de un fallo
- Realización de backups de archivos críticos del negocio: *master files*, *transacciones* y *antes* y *después* de instalación de imágenes del sistema
- Hay que realizar varios backups, interrumpir la aplicación anormalmente y restaurar la aplicación utilizando sólo los backups guardados

Caso de Estudio: "*Compras del Sábado*"

Comentarios a la figura

- En un entorno de pruebas, se lanza la ejecución de la aplicación, inicialmente sin usuarios activos
- Se lanzan cada vez más copias de la misma transacción
- Cada una de ellas representa a un usuario activo, y observar la forma de la gráfica bajo unas condiciones de carga de trabajo creciente
- El tiempo (eje de ordenadas) representa el *peor tiempo de respuesta* de todas las transacciones actualmente activas
- El tiempo de respuesta promedio entre todas las transacciones no se puede utilizar para determinar si se cumple o no el requisito de rendimiento

Caso de Estudio: "*Compras del Sábado*"

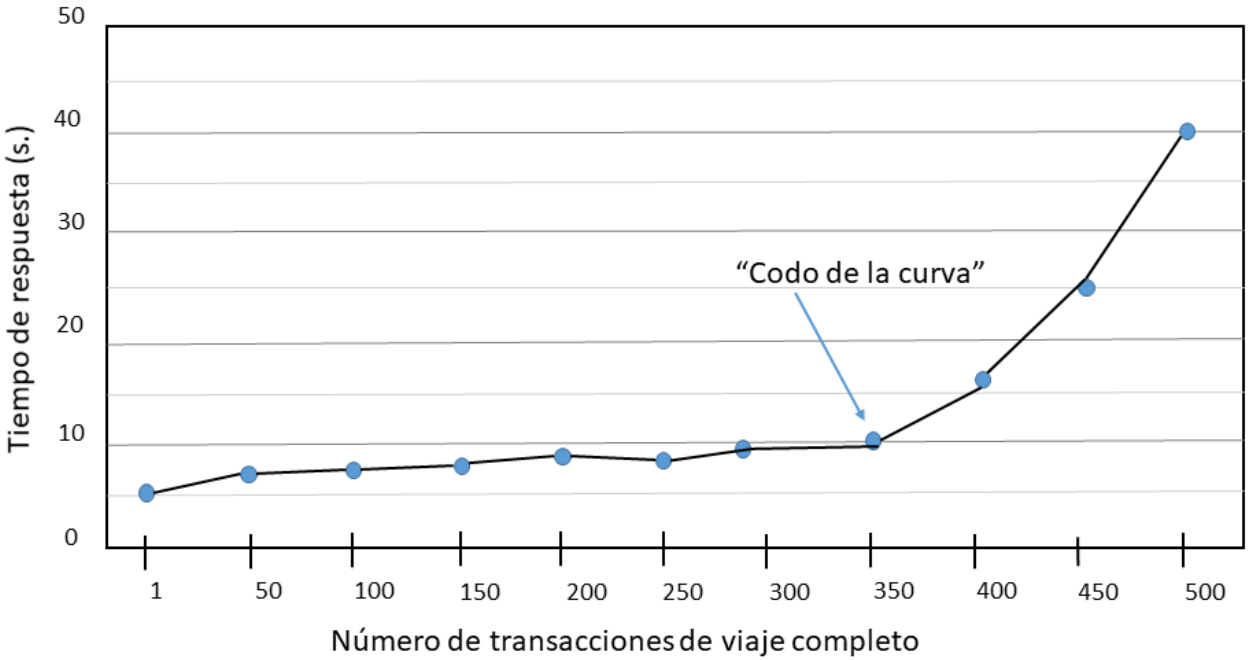


Figura: Rendimiento de viaje completo

Caso de Estudio: "*Compras del Sábado*"

Comentarios a la figura (2)

- El codo de la gráfica se debe a algún tipo de embotellamiento en el que se interfieren las transacciones, que se produce en algún momento de sus caminos de procesamiento
- La gráfica no informa de las causas del embotellamiento, sólo de las circunstancias en las que se produce
- La única forma de descubrir dónde se produce el *codo* es ejecutar las pruebas incrementando la carga de trabajo hasta que aparezca en la gráfica

Prueba de Rendimiento de un Componente IV

Previo a la Prueba de Carga de Trabajo

1. En un *sistema vacío*, para todos los tipos y todas las transacciones de cada tipo, el rendimiento de viaje completo inicial ha de ser menor que el tiempo máximo en pico previsto en los requerimientos del software
2. El rendimiento de viaje completo de las transacciones empeorará conforme aumente la competencia por recursos en un sistema cada vez más ocupado
3. Todas las transacciones (color verde) que cumplan la condición (1) anterior están preparadas para ser utilizadas en una prueba de carga de trabajo

Rendimiento de viaje completo de las transacciones

Grupo Transacciones	Tiempo respuesta	Activos	Fecha	Rendimiento
Navegar menús	3s. (máx)	500	S:9-11h	4.5s
Des/Conexión	3s. (máx)	500	S:9-11h	2.0s
Información productos	4s. (máx)	500	S:9-11h	15.3s
Operación compra	7s. (máx)	500	S: 9-11h	3.0s
Búsqueda catálogo	10s. (máx)	500	S: 9-11h	1.6s
Pago con Tarjeta	30s. (máx)	500	S: 9-11h	103s

Tabla: Planificación de la carga (sábados) para obtener rendimiento

Caso de Estudio: "*Compras del Sábado*"

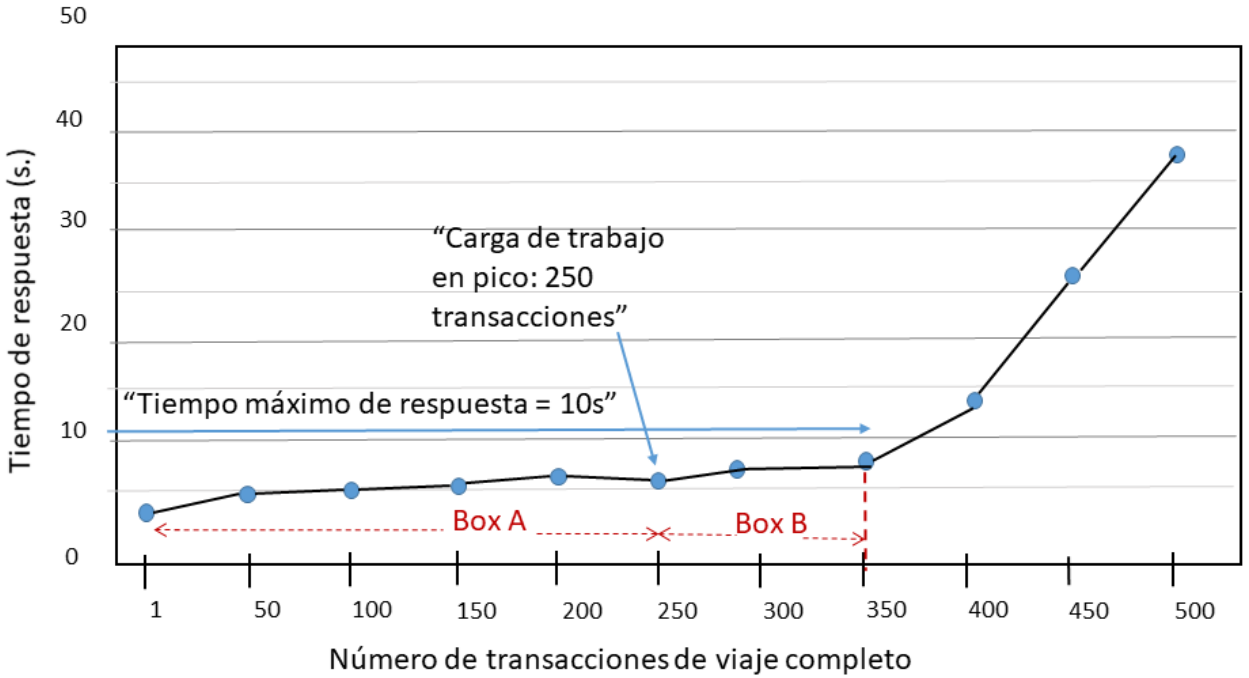


Figura: Rendimiento - Transacciones de búsqueda en catálogo

Caso de Estudio: "*Compras del Sábado*"

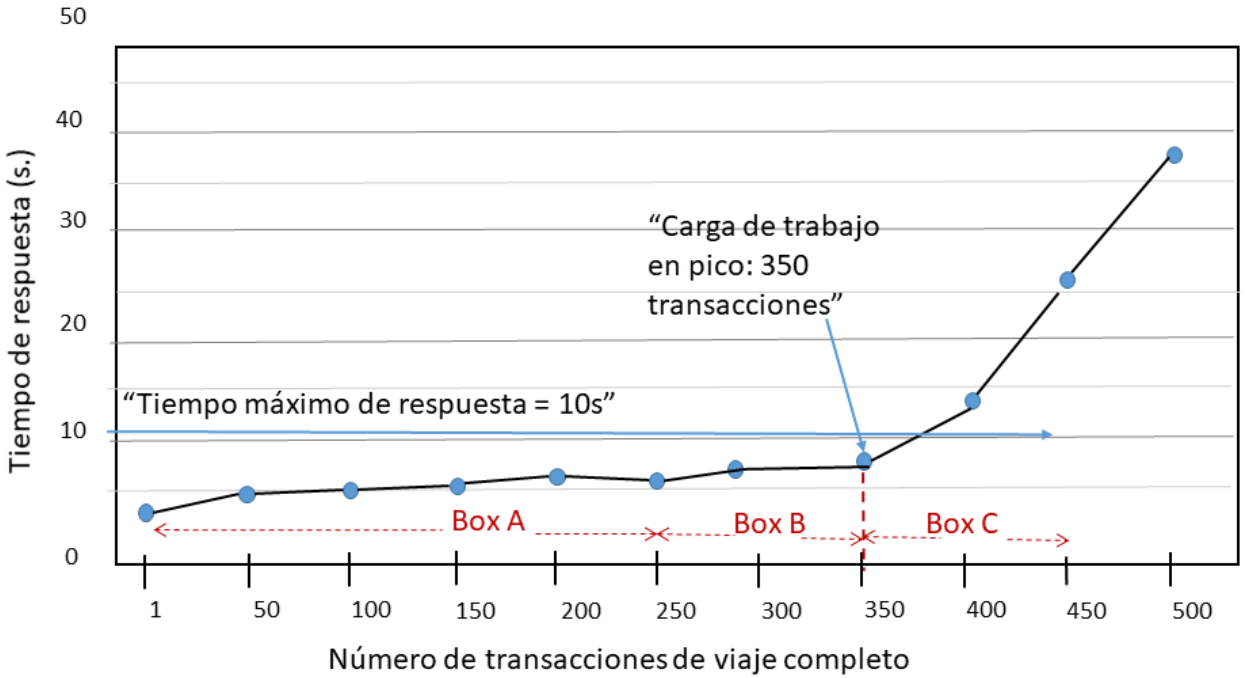


Figura: Rendimiento - Transacciones de búsqueda en catálogo

Caso de Estudio: "*Compras del Sábado*"

Gráficas del t.respuesta para la búsqueda en catálogo

- "*Box A*" incluye el tiempo de respuesta de peor caso para el *pico de carga de trabajo* que se observa con 250 transacciones activas (5.6 s)
- "*Box B*" extiende el test hasta las 350 transacciones activas, obteniéndose una gráfica lineal y valores debajo del máximo requerido (6.8s en el pico)
- "*Box C*" extiende el test hasta las 450 transacciones , lo cual hace aparecer el codo de la gráfica que eleva el tiempo de respuesta en el pico de carga a 26.4 s
- Al llegar a las 350 transacciones activas al mismo tiempo, la ejecución de la aplicación comenzará a ser muy lenta

Caso de Estudio: "*Compras del Sábado*"

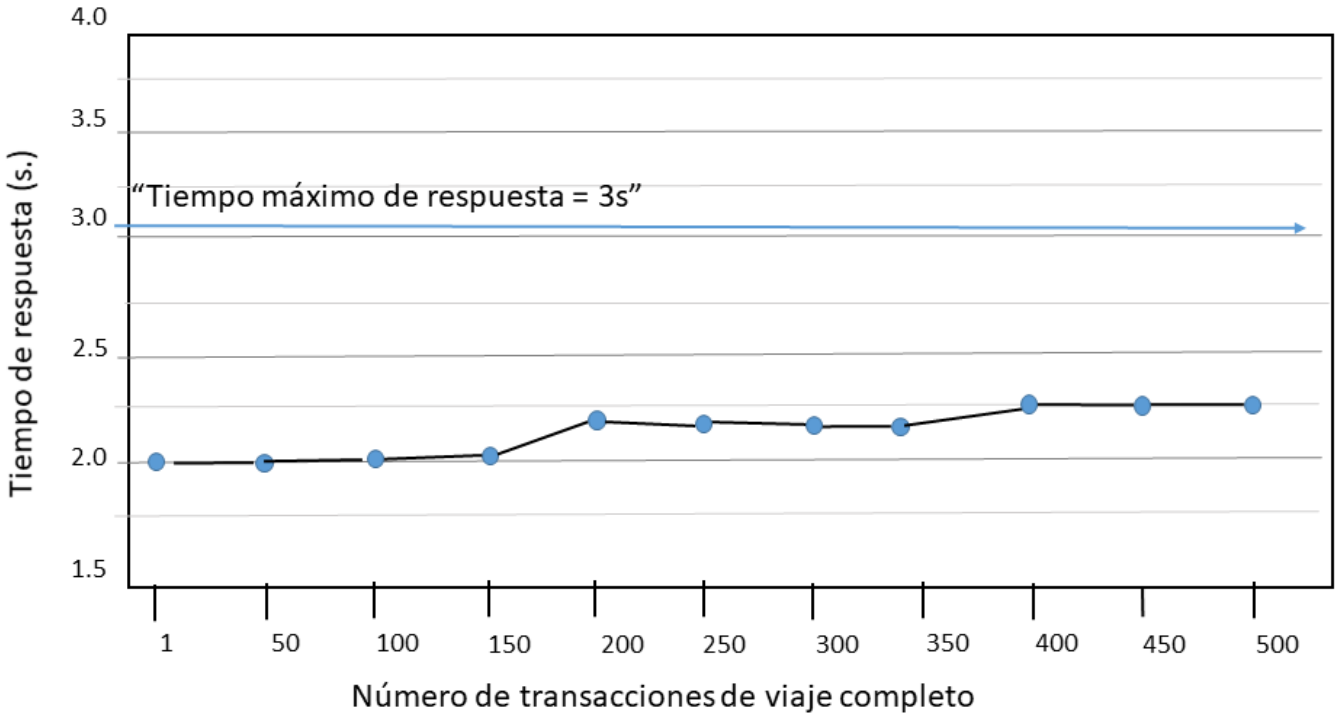


Figura: Pico de Trabajo - Transacciones de conexión (logon)

Caso de Estudio: "*Compras del Sábado*"

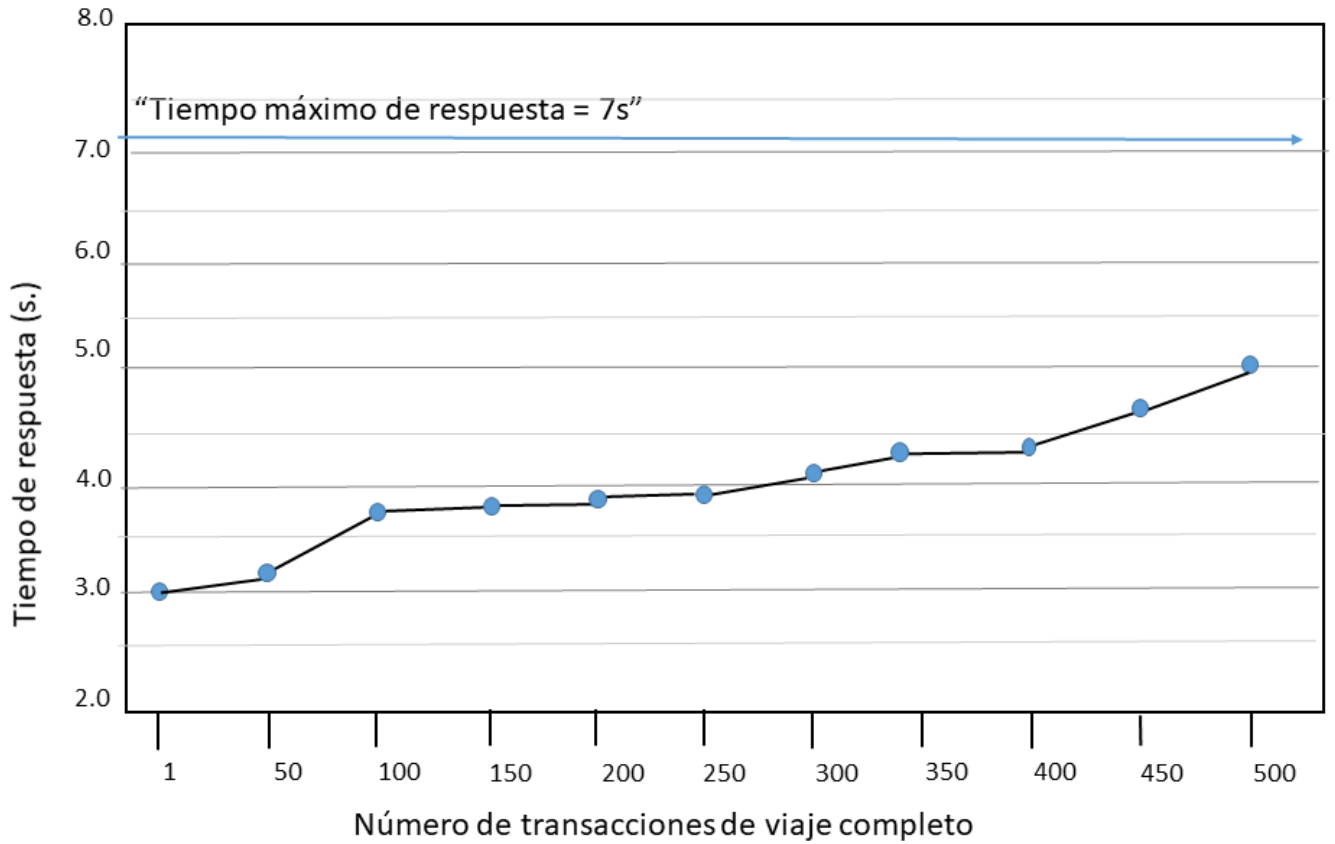


Figura: Pico de Trabajo - Transacciones de compra artículo

Caso de Estudio: "*Compras del Sábado*"

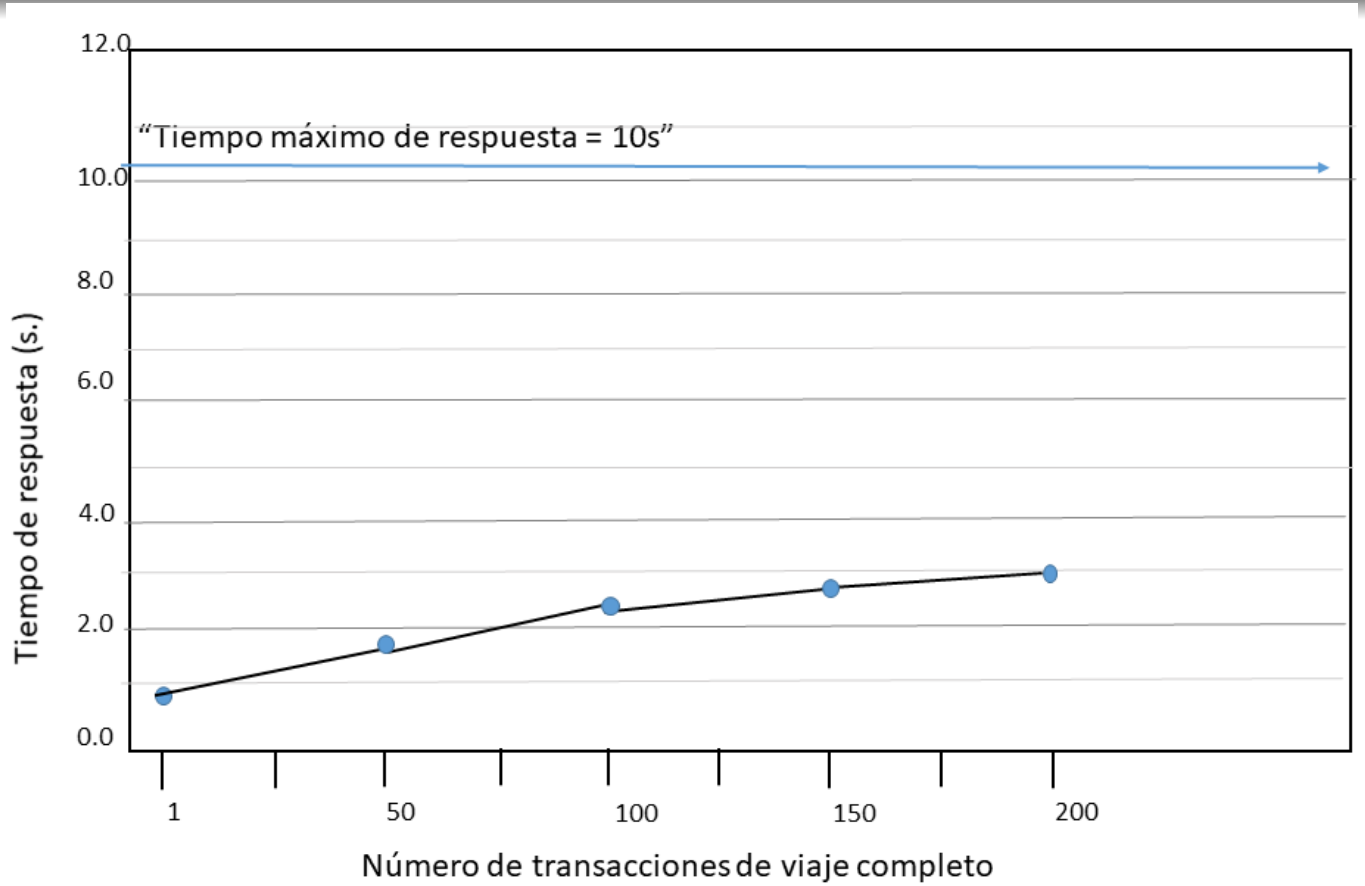


Figura: Pico de Trabajo - Transacciones de búsqueda en catálogo

Caso de Estudio: "*Compras del Sábado*"

Gráficas del t.respuesta para cada transacción individual

- Por simplicidad, se estudian sólo 3 grupos de transacciones (*conexión*, *búsqueda en catálogo*, *compra*)
- Se elije la planificación de carga de trabajo de las *compras de los sábados* porque necesitan simular menos transacciones (500) hasta el pico de carga que la de la planificación de los días laborables (2000 transacciones)
- Se encontró que los tiempos de respuesta de las 3 transacciones consideradas individualmente están bastante por debajo de máximo requerido para el pico de carga de los sábados de cada transacción

Caso de Estudio: "*Compras del Sábado*"

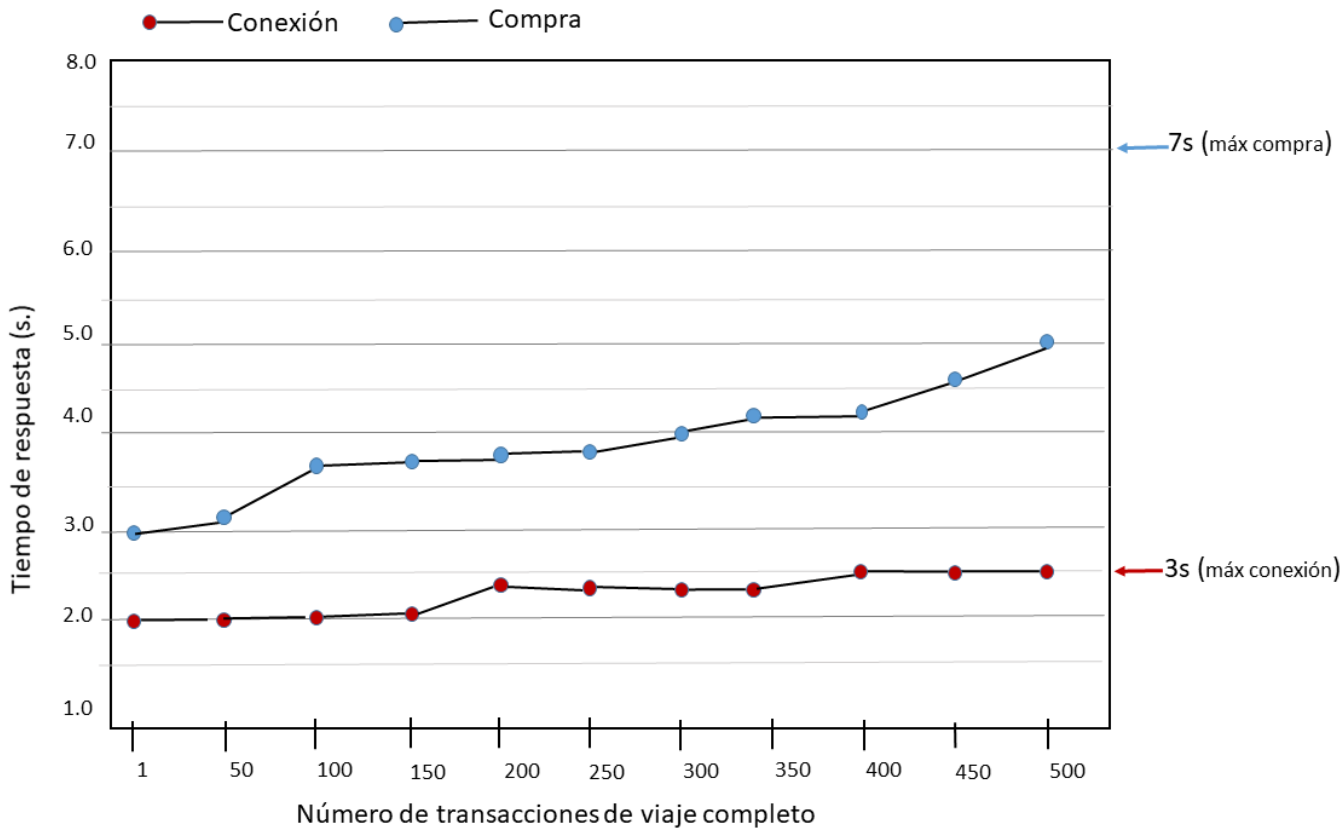


Figura: Pico de Trabajo - Transacciones de conexión+compra

Caso de Estudio: "*Compras del Sábado*"

Gráficas del t.respuesta para mezclas de transacciones

- Las mezclas de las transacciones han de hacerse gradualmente (*conexión+ compra*, *conexión+ compra+ búsqueda*) para no introducir indeterminación respecto de la que causa la pérdida de rendimiento
- El tiempo de respuesta de la transacción de conexión se convierte en un elemento marginal para el cálculo del rendimiento si se superan las 400 transacciones activas
- Por consiguiente, existe un conflicto en el acceso a recursos entre la transacción de compra y la de búsqueda

Caso de Estudio: "*Compras del Sábado*"

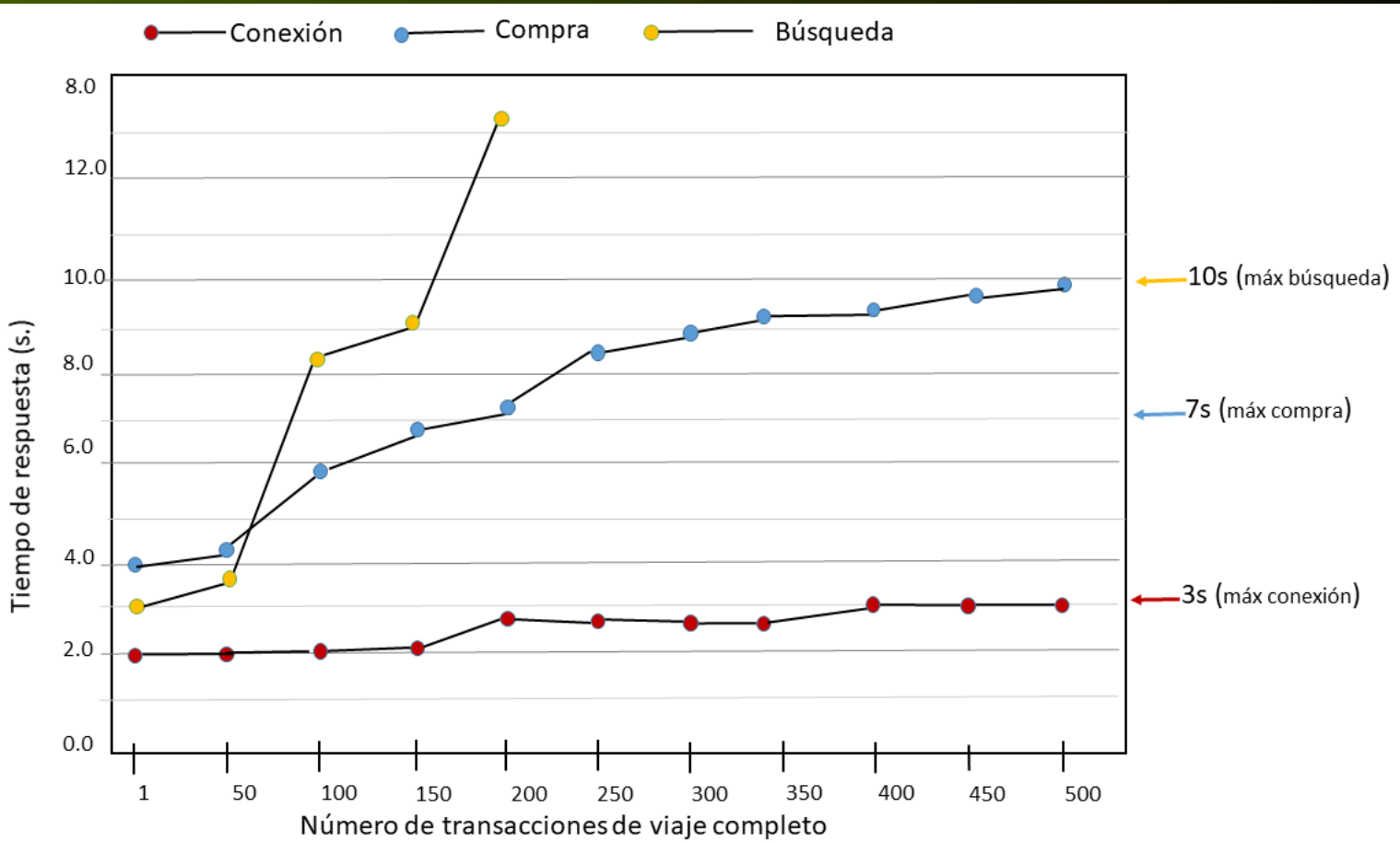


Figura: Transacciones de conexión+compra+búsqueda

Caso de Estudio: "*Compras del Sábado*"

Gráficas del t.respuesta para mezclas de transacciones (2)

- Para la búsqueda en catálogo por encima de las 150 transacciones, la gráfica que contempla la mezcla de las 3 transacciones (conexión+compra+búsqueda) muestra un tiempo de respuesta inaceptablemente alto
- Se deben detener las pruebas de rendimiento (situación denominada "*showstopper*") y devolver el software a equipo de desarrollo

Caso de Estudio: "*Compras del Sábado*"

Gráficas para cada transacción individual *corregida*

- Se recibió el software de la aplicación corregido y con las pruebas funcionales de regresión realizadas:
  - El módulo de *conexión* de los usuarios interfería con el módulo de compra por motivo de pérdida de memoria asignada al primero, que le retiraba el segundo módulo a partir de 400 transacciones activas
  - Degradación del rendimiento de una librería dinámica compartida entre el módulo de compra y el de búsqueda en catálogo cuando se cargan en memoria rutinas de utilidades para su ejecución

Caso de Estudio: "*Compras del Sábado*"

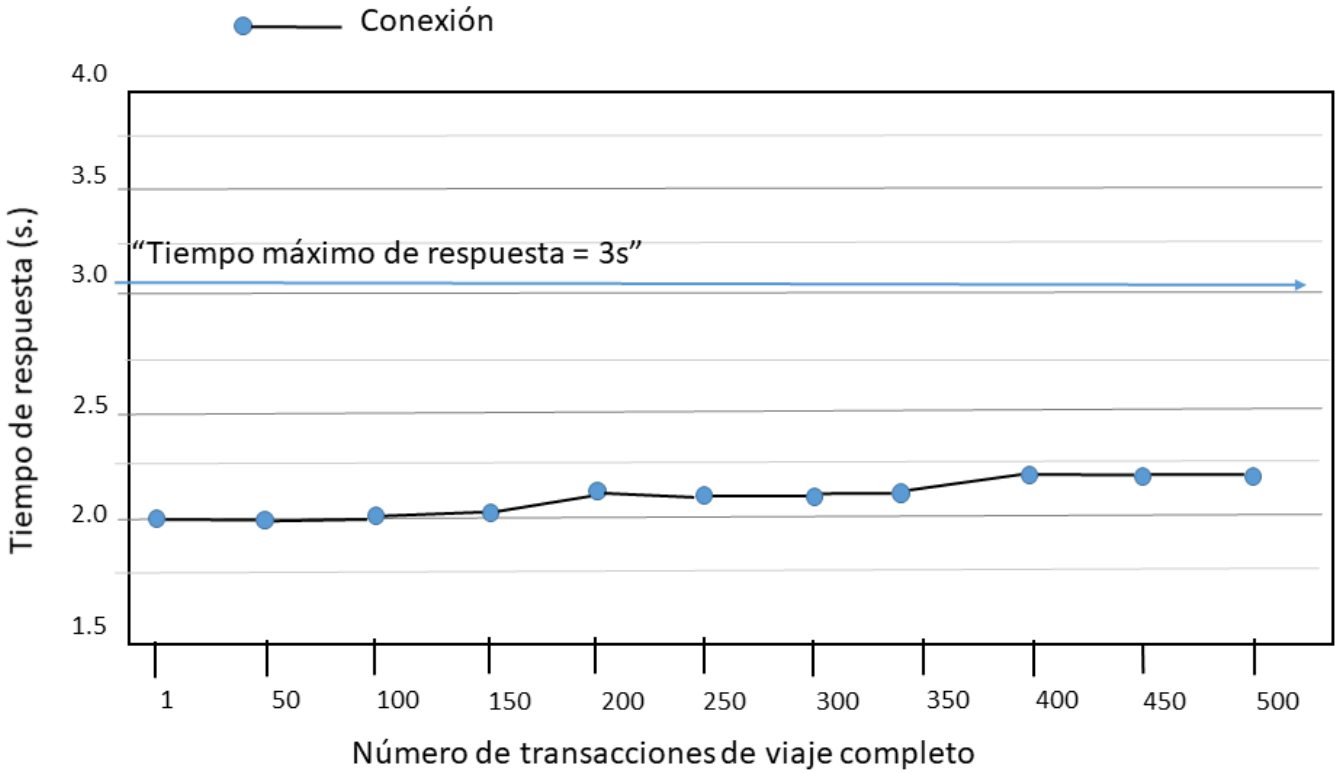


Figura: Transacciones de conexión después de correcciones

Caso de Estudio: "*Compras del Sábado*"

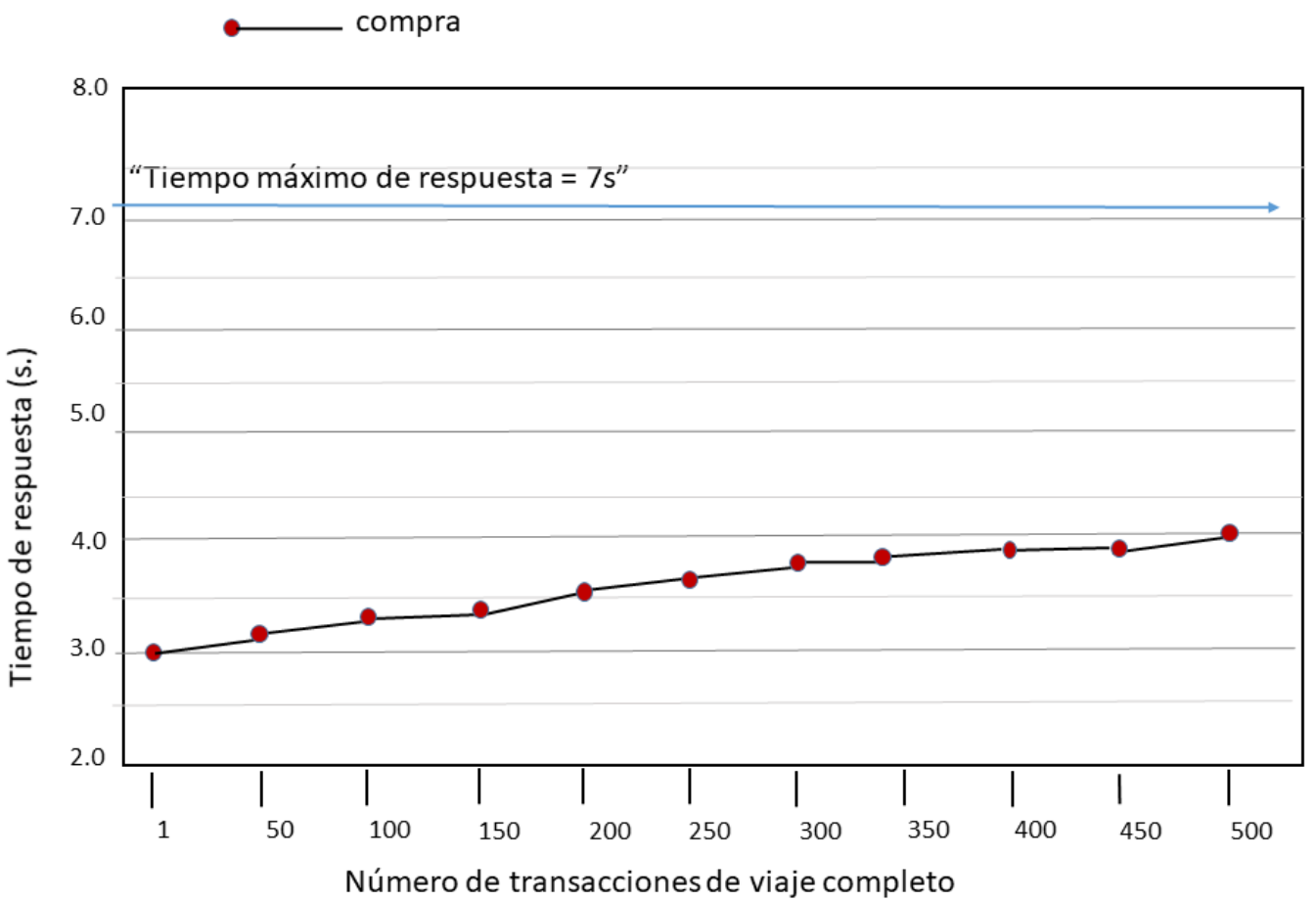


Figura: Transacciones de compra después de correcciones

Caso de Estudio: "*Compras del Sábado*"

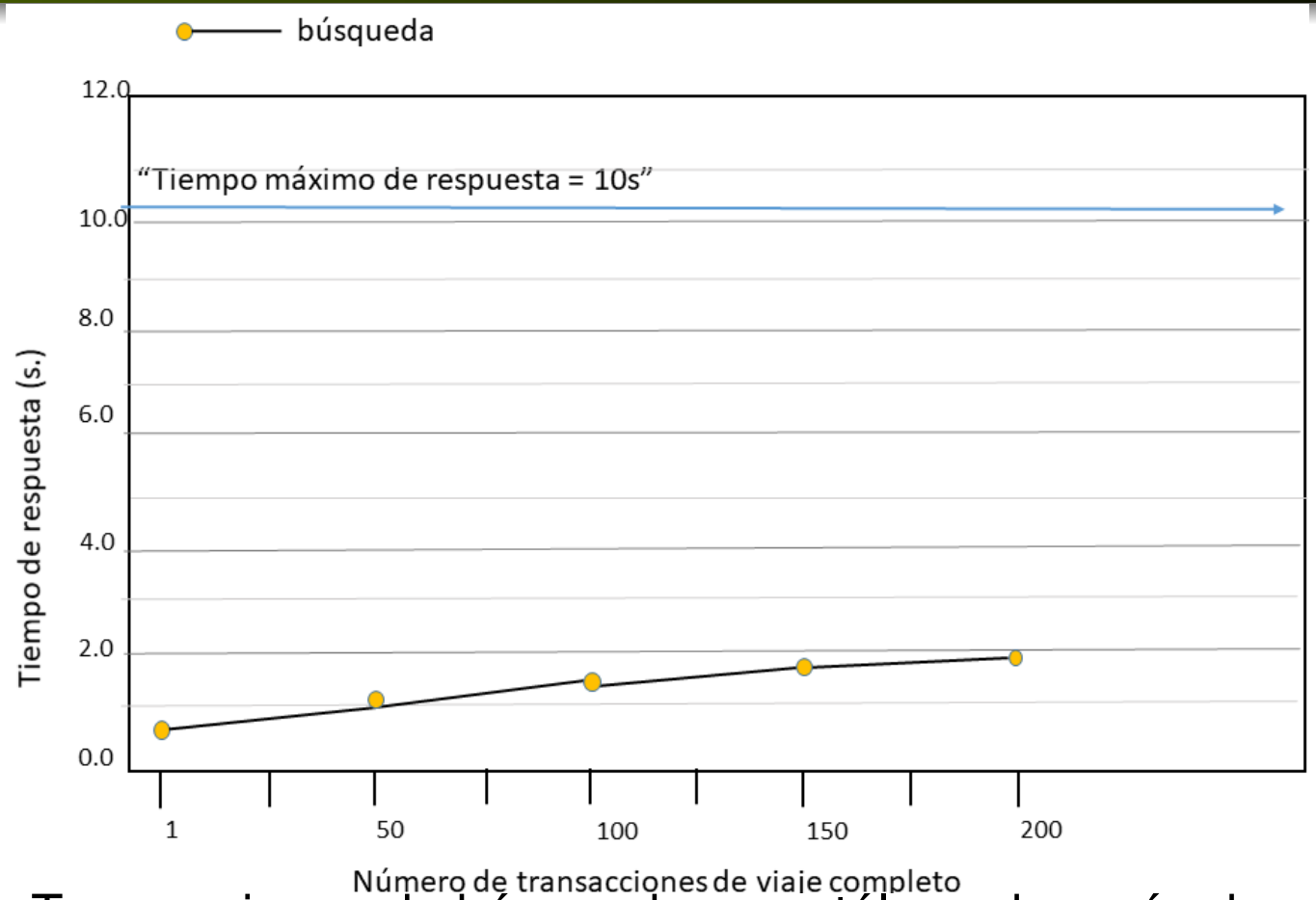


Figura: Transacciones de búsqueda en catálogo después de correcciones

Caso de Estudio: "*Compras del Sábado*"

Gráficas del para cada transacción individual *corregida*

- El rendimiento del código del módulo de conexión no se ha visto afectado por los cambios
- Los módulos de compra y búsqueda en catálogo muestran un ligero aumento de su rendimiento para cada transacción individual
- Aunque los cambios del código se centraron en corregir la degradación del rendimiento debido a una mala implementación de librerías compartidas, también mejoró el comportamiento individual de algunos módulos

Caso de Estudio: "*Compras del Sábado*"

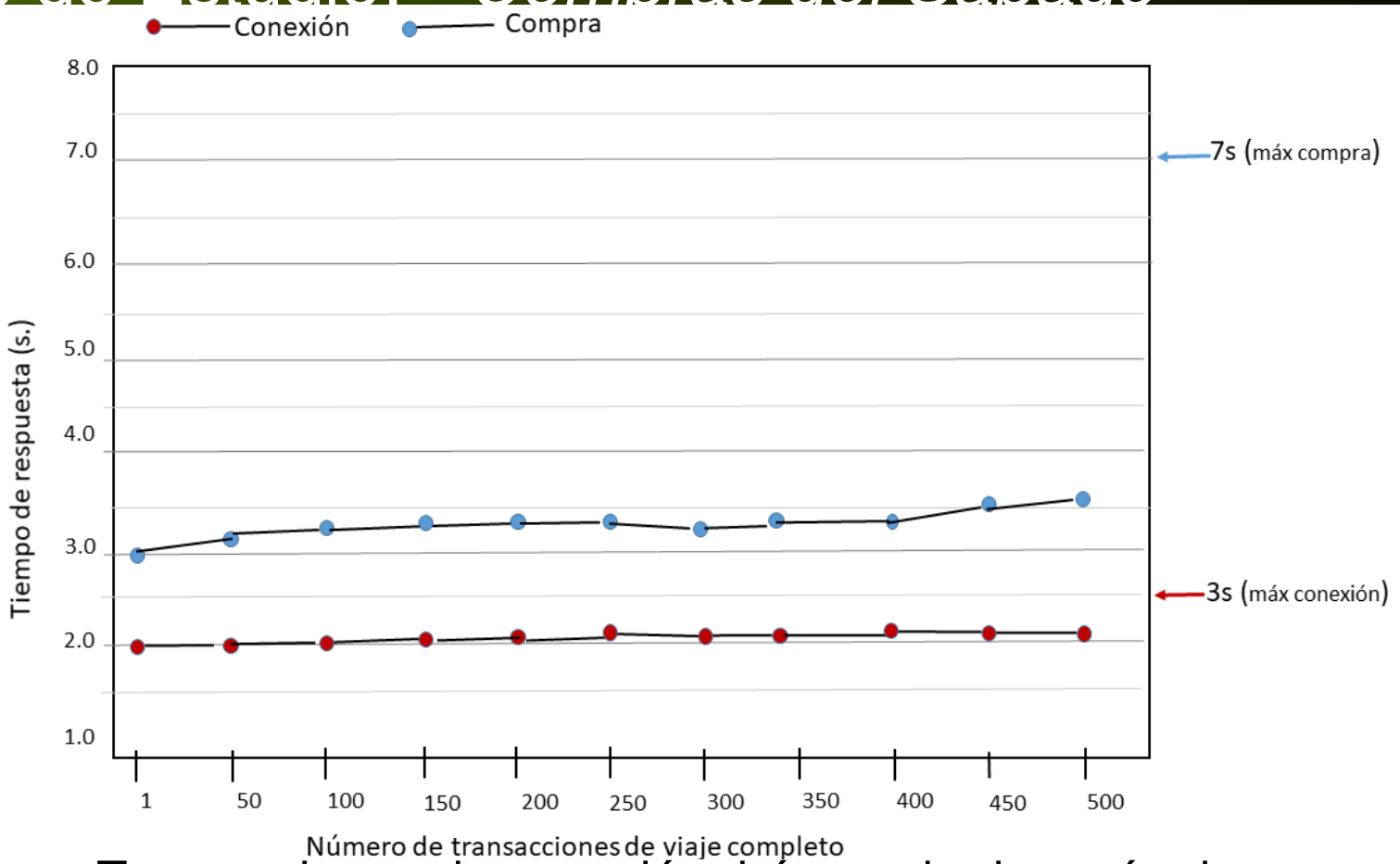


Figura: Transacciones de conexión+búsqueda después de correcciones

Caso de Estudio: "*Compras del Sábado*"

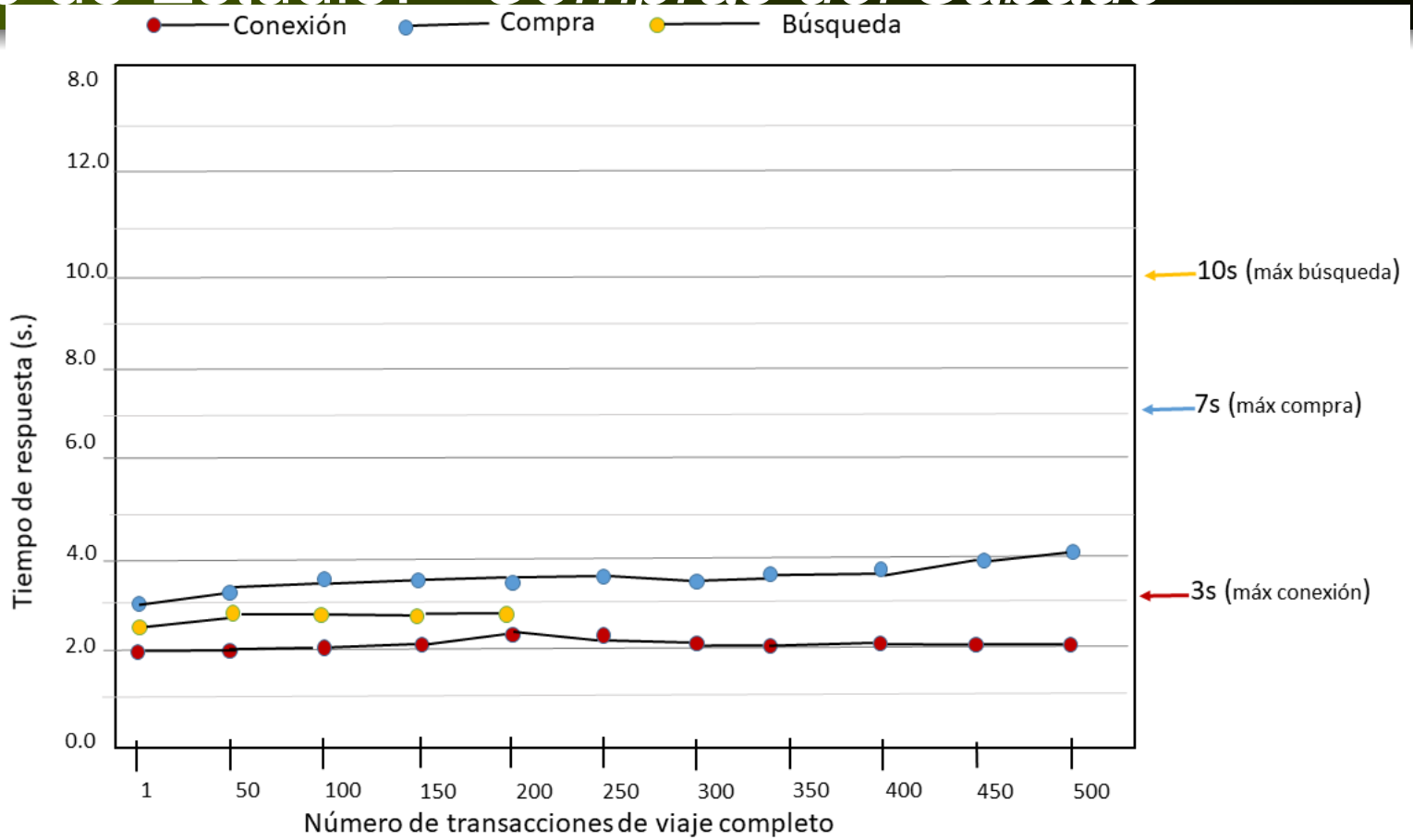


Figura: Transacciones de conexión+búsqueda+compra después de correcciones

Caso de Estudio: "*Compras del Sábado*"

Gráficas del t.respuesta para mezclas corregidas

- Se resolvió el problema de la interferencia con la compra (cuando existen $>$ 400 transacciones-compra activas)
- Ambos grupos de transacciones (*conexión* y *compra*) producen un peor tiempo de respuesta muy por debajo del máximo tiempo en pico (en los requerimientos)
- Añadir *búsqueda* en catálogo a las anteriores no afecta ahora al rendimiento de ninguna transacción cuando compiten por recursos
- Los tests de rendimiento han de formar parte de las pruebas de regresión para demostrar que la inclusión de nueva funcionalidad no degrada el rendimiento global

Software Orientado a Objetos

Cómo probarlo

- ¡La naturaleza del software OO modifica la *estrategia* y la *táctica* de la prueba del software!
- Cambia el concepto de *unidad* que hemos estudiado en la pruebas unitarias de los componentes software
  - Foco de las pruebas unitarias: las *clases*
  - Las unidades "probables" más pequeñas ahora: *métodos*
  - Necesidad de que una suite de pruebas unitarias abarque a varias clases
- La prueba de clases en el software OO es el equivalente de la prueba unitaria en el software convencional
- La prueba de clases en software OO es conducida por sus operaciones encapsuladas y el comportamiento del estado (simulado) de los objetos durante dicha prueba

Diseño de tests para pruebas de software OO

Características de los métodos de prueba utilizados en POO

- Las pruebas unitarias se aplican clase por clase
- Se han de ejercitar todas las operaciones (*métodos*) que declara cada clase
- Se diseñan *suites* de tests para asegurar que las operaciones importantes son ejercitadas durante la prueba unitaria
- Se ha de examinar el estado de cada clase (valores de sus atributos simulados en una *fixture*) para determinar si existe algún error todavía no identificado en un conjunto de valores anterior

Diseño de tests para pruebas de software OO (II)

Desafíos

- La encapsulación propia de las clases puede hacer difícil extraer el estado concreto de cualquier objeto durante un momento de la ejecución de la prueba unitaria
- El mecanismo de herencia puede complicar la realización de las pruebas de una clase
- La herencia múltiple complicaría aún más las pruebas, ya que incrementa el número de contextos de utilización

Diseño de tests para pruebas de software OO (III)

Categorias de métodos de prueba

1. Pruebas basadas en fallos
2. Pruebas aleatorias
3. Pruebas de partición

1. Pruebas basadas en fallos

Características

- Partiendo del modelo de análisis OO, el comprobador trata de identificar posibles fallos
- Se diseñan casos de prueba para ejercitar el diseño del sistema o su codificación
- Se buscan posibles fallos en las llamadas a los métodos y las comunicaciones a través de mensajes:
  - llamadas a operaciones: hay que examinar su comportamiento
  - tipos de errores: *resultado inesperado*, *operación/mensaje incorrecto*, *invocación incorrecta*
  - Se trata de determinar si existen errores en el código que llama, no en el llamado

2. Pruebas aleatorias

Características

- Este tipo de prueba se utiliza para ejercitar una clase
- Existen restricciones en el orden de invocación de las operaciones de una clase, debido al problema
- Incluso con dichas restricciones, existen muchas posibles permutaciones de la secuencia de llamadas a operaciones
- Se generan aleatoriamente diferentes secuencias para ejercitar distintas *historias* de la *vida* de las instancias

3. Pruebas de Partición

Características

- Respecto de la prueba aleatoria, reduce el número de tests necesarios para realizar la prueba de una clase
- Se categorizan las entradas y las salidas de las operaciones y se diseñan tests para ejercitarlas
- *Particionamiento por estados*: clasifica las operaciones según su habilidad para cambiar el estado de la clase
- *Particionamiento basado en los atributos*: clasifica las operaciones según los atributos de la clase que utilizan
- *Particionamiento basado en categorías*: clasifica las operaciones de la clase según las funciones genéricas

Pruebas de Integración

Estrategias

- El integrar operaciones de una en una, en una sola clase, es, a menudo, imposible
- (1) Pruebas *basadas en hebras*: cada hebra de la aplicación es probada individualmente
- Hay que aplicar la prueba de regresión (muy importante)
- (2) Pruebas *basadas en la utilización*
- Después de probar las clases *independientes*, se prueba la siguiente capa de clases
- La secuencia de prueba de capas de clases dependientes continúa hasta se construye el sistema completo

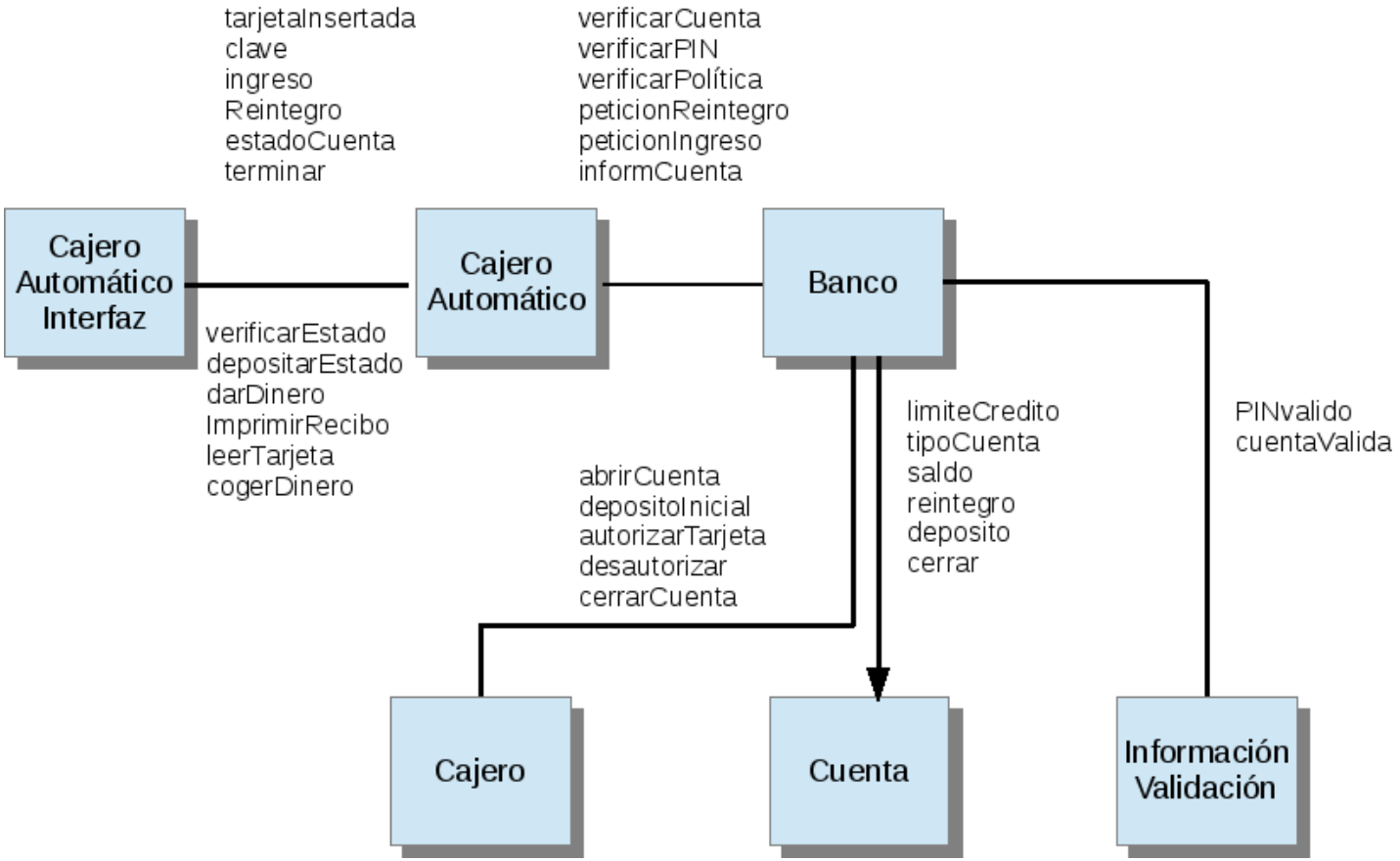
Pruebas de Integración II

Pruebas de múltiples clases

1. Para cada clase cliente: usar la lista de operaciones para generar secuencias de prueba aleatorias (las operaciones enviarán mensajes a otras clases)
2. Para cada mensaje: determinar la clase colaboradora y la operación correspondiente en el objeto servidor
3. Para cada operación en el objeto servidor: determinar los mensajes que transmite
4. Para cada uno de estos mensajes: determinar el siguiente nivel de operaciones que son invocadas e incluirlas en la secuencia inicial de prueba

Ejemplo de prueba que implica a múltiples clases

Diagrama de Colaboración de Clases



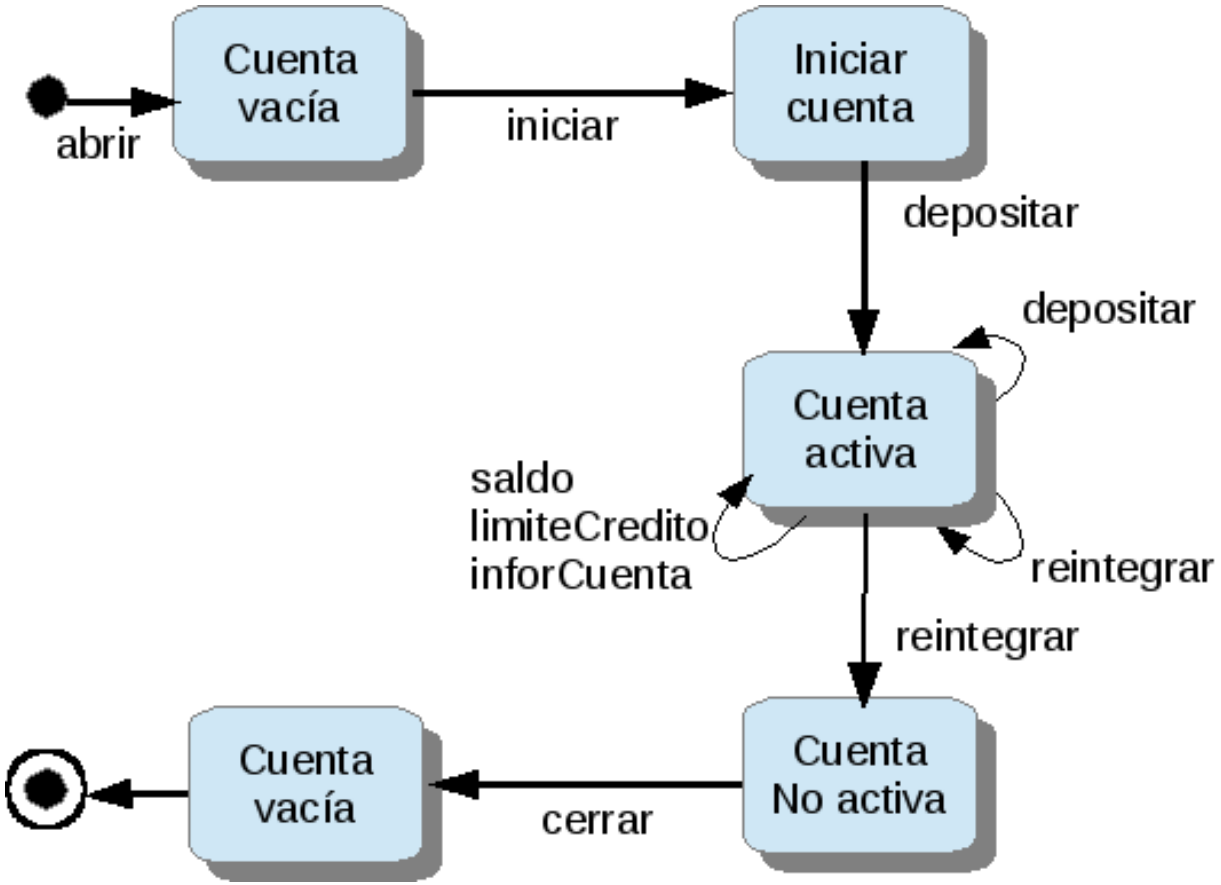
Pruebas para múltiples clases

Ejemplo: *Acceso a cuenta de cliente desde un cajero basada en el diagrama de colaboración de clases*

- La prueba de partición para múltiples clases es similar a la de 1 clase
- Ahora, la secuencia es ampliada para que incluya las operaciones que se activan en las clases colaboradoras, después de recibir los mensajes:
  - secuencia para*Banco* : *verificarCuenta*() · *verificarPIN*() · [[*verificarPolitica*() · *peticionReintegro*()]|*peticionIngreso*()|*inforCuenta*()]
  - Una secuencia aleatoria:Test 3: *verificarCuenta*() · *verificarPIN*() · *peticionIngreso*()
  - Secuencia con colaboraciones:Test 4: *verificarCuenta*() · [*Banco* : *InformacionValidacion*.*inforCuenta*()] ·*verificarPIN*()[*Banco* : *InformacionValidacion*.*validarPIN*()] ·*peticionIngreso*()[*Banco* : *Cuenta*.*deposito*()]

Ejemplo de prueba que implica a múltiples clases - II

Diagrama de estados UML



Pruebas derivadas de los modelos–UML de comportamiento

Diagramas Estados–UML

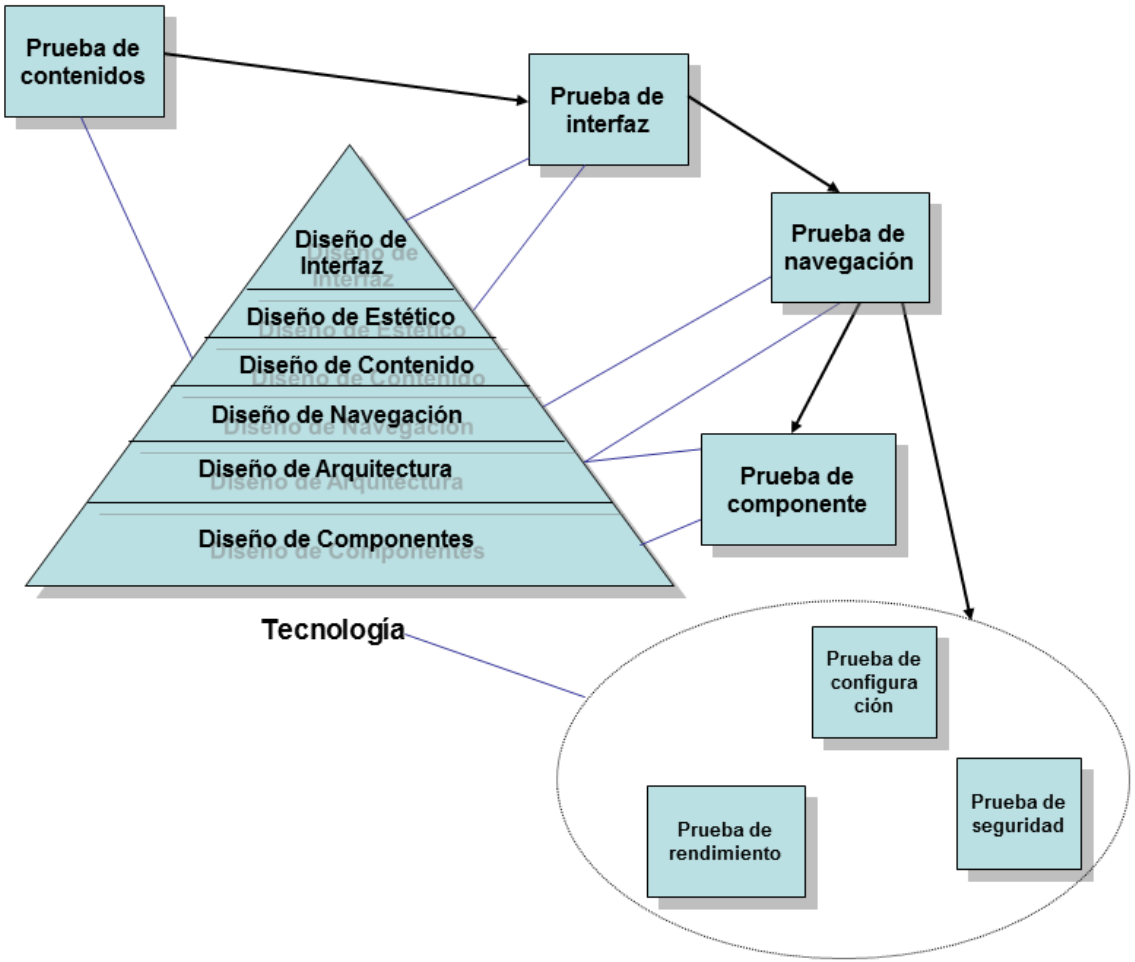
- Se utilizan para representar el comportamiento dinámico de una clase
- Ahora, lo usamos para derivar una secuencia de pruebas que ejercite el comportamiento dinámico de la clase y sus colaboradoras
- Los tests diseñados han de incluir todos los estados del diagrama–UML
- El modelo de estados puede ser atravesado de forma *breadth–first*

Pruebas para Aplicaciones Web

Antecedentes

- Se siguen los principios básicos y los pasos de la pruebas de software en general: unitarias, integración, regresión, etc.
- Se aplican las tácticas de la prueba de software OO
- Lo que se quiere llegar a probar:
  - Exclusión de errores de navegación
  - Preocupación por la *usabilidad*
  - Compatibilidad con diferentes plataformas y configuraciones
  - Confiabilidad (= seguridad+robustez+fiabilidad+disponibilidad)
  - Rendimiento

Pruebas para Aplicaciones Web (II)



Pruebas para Aplicaciones Web III

Pasos del Proceso de Prueba

1. Revisar el modelo de contenidos
2. Comprobar que todos los casos de uso son tratados en el modelo de interfaz
3. Descubrir cualquier error de navegación en el modelo de diseño
4. Exclusión de errores de la mecánica de navegación o de la presentación en la interfaz de usuario
5. Prueba unitaria de cada componente fucnional
6. Navegación a través de toda la arquitectura software de la aplicación
7. Tests de compatibilidad con plataformas y configuraciones
8. Pruebas de seguridad, exclusión de vulnerabilidades de la aplicación o de su entorno
9. Tests de rendimiento

Pruebas de Contenidos

Caracteríticas

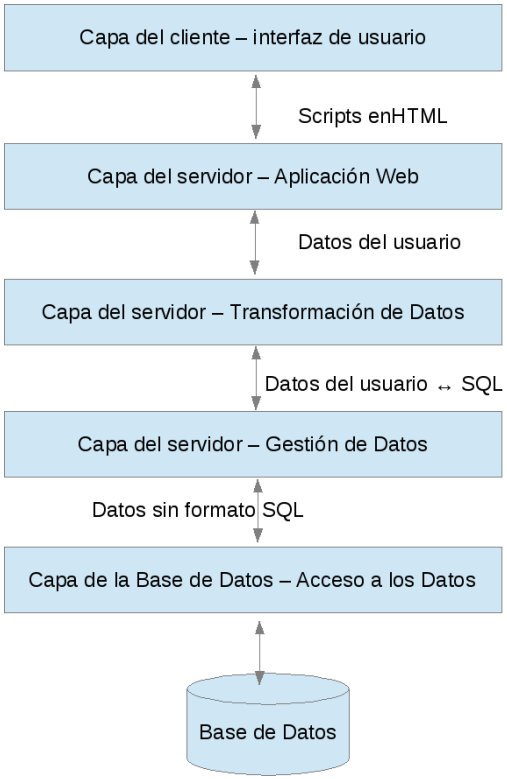
- Combina las revisiones con la generación de pruebas ejecutables
- Objetivos:
  - Descubrir errores sintácticos
  - Errores semánticos
  - Errores en la organización de la página o en la estructura de los contenidos que se presentan al usuario

Interfaz entre Aplicación Web–SGBD

Características

- Descubrir errores producidos en la traducción de las peticiones del usuario al lenguaje de consulta del SGBD
- Identificar errores de comunicación entre la aplicación–Web y la BD remota
- Demostrar la validez de los datos sin formato recibidos por el servidor de la aplicación Web
- Los contenidos dinámicos han de ser transmitidos al cliente de una forma que puedan ser mostrados (comprobar compatibilidad con diferentes plataformas y configuraciones del usuario)

Capas de Interacción entre Aplicación y Base de Datos



Prueba de la Interfaz de Usuario

Ubicación en el ciclo de vida

- Durante el Análisis de Requerimientos
- Durante el Diseño
- Durante la Prueba del Software
- Se trataría de descubrir errores relacionados con mecanismos específicos de la interfaz
- y los errores relacionados con la semántica de la navegación, funcionalidad de la aplicación Web o contenidos mostrados

Prueba de la Interfaz de Usuario (II)

Actividades

- Probar mecanismos de la Interfaz
- Comprobación de *cookies* en el lado del servidor y del cliente
- Prueba de la Semántica de la Interfaz
- Pruebas de Usabilidad
- Pruebas de Compatibilidad

Pruebas de Navegabilidad

Antecedentes

- Se trata de asegurar que los mecanismos que permiten al usuario navegar a través de la aplicación Web son todos funcionales
- Validar que cada *unidad semántica de navegación* (NSU) puede ser alcanzada por la categoría adecuada de usuarios
- Se hacen pruebas de *Sintaxis de Navegación* y de su Semántica

Pruebas de Sintaxis

- Enlaces de navegación
- Redirecciones
- Marcas de libro (*bookmarks*)
- Frames y framesets
- Tabla de contenidos de sitio
- Motores de búsquedas internos

Pruebas de Semántica

Definición de NSU

- Un conjunto de información y estructuras de navegación relacionadas que colaboran para que se puedan satisfacer un subconjunto relacionado de requerimientos del usuario

Pruebas de Semántica II

- La prueba de navegabilidad ejercita cada NSU para asegurarse que los requerimientos se alcanzan:
  - ¿Todos los nodos de navegación se alcanzan desde el contexto definido para un NSU?
  - ¿Todos los caminos importantes hacia/desde el NSU se han probado?
  - ¿Funcionan adecuadamente los mecanismos de navegación dentro de los nodos de navegación grandes?
  - ¿Son los NSU tolerantes a fallos y errores?

Pruebas de Seguridad

Tipos de amenazas

- Desde el cliente: errores en navegadores, correo electrónico, software de comunicaciones, acceso a cookies, *spoofing*
- Desde el servidor: ataques de denegación de servicio, scripts maliciosos, robo de datos

Elementos de Seguridad

- Firewalls
- Autenticación de todos los clientes y servidores
- Encriptación (y certificados digitales)
- Autorización
- Las pruebas de seguridad ha de diseñarse para probar que todas las tecnologías anteriores funcionan; se intenta descubrir agujeros de seguridad

Prueba de Carga

Concepto

- Examina la carga que puede experimentar el sistema en el mundo real, a varios niveles y en una variedad de combinaciones
- Prueba de *Stress*: fuerza el incremento de la carga hasta el punto de *ruptura* del sistema para determinar la capacidad máxima que puede aguantar

Prueba de Carga II

Conjunto de condiciones del test:

- N: número de usuarios concurrentes de la aplicación–Web
- T: número dede transacciones en línea por unidad de tiempo
- D: datos procesados por el servidor por transacción

Prueba instantánea

Determinar si una combinación de las tres medidas (N, T, D) puede asociarse a un decrecimiento importante del rendimiento en un momento:

$$Throughput = N \times T \times D \quad (1)$$

Prueba de Stress

Condiciones

Las variables N, T, D de la *prueba de carga* son obligadas a llegar a los límites operacionales del sistema y, después, a excederlos

Prueba de Stress (II)

Cosas a Determinar

- ¿Se degrada el sistema suavemente?
- ¿El software del sistema genera mensajes “server not available"?
- ¿Almacena peticiones de los usuarios y deja vacía la cola una vez que se supera el ímite?
- ¿Se pierden transacciones?
- ¿Qué valores de N, T, D ocasionan que el servidor falle?
- ¿La integridad de los datos se ve afectada?
- ¿Cuando tardará el sistema en recuperarse?

Para ampliar, referencias bibliográficas: I


Black (2007).
Pragmatic Software Testing.
Wiley.


Everett and Raymond, . (2007).
Software Testing.
Wiley.


Lewis.


Perry (2005).
Effective Methods for Software Testing.
Wiley.


Spiller (2007).
Softwre Testing Process: Test Management.
Rocky Nook.