BeatIt!

Plan de Verificación y Validación

Versión 1.2

Historia de revisiones

Fecha	Versión	Descripción	Autor
23/8/2014	1.0	Completado todos los puntos	Emiliano Vázquez
24/8/2014	1.1	Formato y estilo	Emiliano Vázquez
31/8/2014	1.2	Revisión SQA	Pablo Olivera

Contenido

1. Introducción

1.1. Propósito

Este Plan de Verificación para el proyecto BeatIt! soporta los siguientes objetivos:

Hacer un seguimiento del proceso de desarrollo, garantizando que el comportamiento del software corresponde con lo indicado por el cliente.

Los resultados esperados serán obtenidos en un tiempo aceptable y serán suficientemente precisos.

La consistencia en los entregables, los criterios de programación y convenciones adoptadas durante los procesos, resultarán en un producto de calidad, por entenderse las metodologías aplicadas el camino indicado para contener la complejidad del sistema y realizar un producto mantenible.

Por tratarse de una aplicación volcada a la utilización de los diferentes componentes hardware del dispositivo móvil, recomendamos fuertemente definir controladores básicos de los mismos a modo de servicios, así pues utilizar la técnica de Service Mocking para hacer las pruebas de integración.

La misma representa un poderoso recurso para el testing ya que verificará la correcta interacción de componentes.

Por tratarse de un desarrollo iterativo y prototipado, una temprana implementación de un sistema de registro de errores permitirá hacer un seguimiento de defectos durante y después de cada iteración sea cual sea el ambiente de ejecución.

Por último centramos nuestra estrategia de verificación en la puesta en marcha de un Build Server de modo de lograr integración continua, con posibilidad de notificación inmediata ante la introducción de un defecto al software por parte del equipo de desarrollo.

Los recursos identificados como necesarios son:

- Una solución de integración continua vinculada al repositorio.
- La implementación de funcionalidad de software para proveer un fácil montaje de contextos transaccionales para pruebas unitarias y de integración.

  Esta implementación controlará el tiempo de cada prueba, verificando que no exceda los 3 segundos de retorno.
- Conjuntos de datos pertinentes para el montaje de las pruebas unitarias y de integración de los diferentes componentes.
- Herramientas de control de cubrimiento.
- Analizador sintáctico y de control de flujo en lo posible en tiempo real.
- Recurso humano suficiente para la codificación de tests y revisión de documentos entregables.

La estimación del esfuerzo realizado en cada punto es respectivamente:

- Investigar, configurar y verificar la correcta puesta en marcha de un Build Server conllevará 10 horas.
- El desarrollo de funcionalidad para hacer más cómodas las pruebas unitarias y de integración se estima en 15 horas.
- El tiempo dedicado a la creación de un ambiente de datos adecuado para las pruebas será de 1 hora por componente.
- El esfuerzo de investigación e integración de una herramienta de reporte de cubrimiento es calculado en 2 horas.
- Idem anterior pero relativo al analizador estático, se calcula en 2 horas también.

- La codificación de tests representará aproximadamente entre un 20-30% del tiempo de implementación de la funcionalidad a testear. Y la revisión supone 15 horas semanales.

Los entregables correspondientes al proyecto de verificación son:

- Plan de verificación.
- Cronograma de actividades de verificación por iteración.
- Especificaciones de casos de prueba para los distintos componentes.
- Código correspondiente al entorno de verificación y juego de datos necesarios.
- Conjunto de pruebas unitarias y de integración.
- Informe consolidado con todos los resultados de las pruebas en diferentes fases de desarrollo y resultados de verificación de documentación técnica y de trazabilidad entre requerimientos e implementación.
- Informe final de verificación que contendrá el relevamiento de actividades, conclusión respecto al esfuerzo estimado, completitud de lo planificado, descripción final del sistema (cobertura, gráficos de pruebas encontradas por unidad de tiempo).

1.2. Punto de partida

El proyecto BeatIt! quedará compuesto por tres principales componentes en su arquitectura.

En primera instancia un cliente grueso, en el cual se encargará de una gran porción del procesamiento total de cada caso de uso.

Un servidor web, el mismo expone funcionalidad suficiente de consulta y registro de información.

Y servidores de base de datos, tanto para un ambiente de producción como para desarrollo.

1.3. Alcance

Comenzando con pruebas unitarias que darán paso a la correcta implementación de los componentes del sistema, luego se pondrá en marcha pruebas vía Mock Services ayudados por contenedores de inyección de dependencias.

Así pues la interacción entre componentes estará verificada, dando lugar a un sistema de captura de interfaz de usuario a medida que se ejecutan los casos de uso.

Constatando que las capturas no cambian involuntariamente podremos verificar el sistema en general y elementos de diseño de interfaz.

Las características principales en que centramos la verificación serán la capacidad de rápida respuesta al usuario y la correctitud de la información.

Entre lo que reconocemos no prioritario serán las pruebas de carga, volumen y esfuerzo, ya que el sistema se desplegará en Microsoft Azure, facilitando aspectos de escalabilidad.

Reconocemos por último que existen riegos propios en incursión de la verificación propuesta, ya sean:

El equipo no se adapta a la metodología, mitigando con horas capacitación o como plan de contingencia encontrar una verificación alternativa.

Las herramientas ha utilizar son complejas y resultan en un desacierto en productividad, la mitigación (de ser posible) será haciendo uso parcial de las mismas sin descuidar aspectos fundamentales y la contingencia sería única y representaría encontrar otra herramienta equivalente más simple.

Las tecnologías no se integran o surgen problemas de incompatibilidad, mismo proceder que el riesgo anterior.

1.4. Identificación del proyecto

Los documentos usados para elaborar el Plan de Verificación son los siguientes:

Descripción de la Arquitectura

Especificación de Requerimientos

Modelos de casos de uso

1.5. Estrategia de evolución del Plan

Dando comienzo al proyecto, en las primeras semanas se montará el sistema de integración continua de la mano del Responsable de SCM.

Las pruebas unitarias serán contruidas por cada Implementador sin necesidad de ser TDD (ya que consideramos que la técnica requiere mucho tiempo de adaptación).

Las pruebas de integración serán dadas por el Responsable de Verificación y por último las pruebas de sistema dada la tecnología utilizada recaerá en los Diseñadores de Interfaz de usuario.

Al cabo de cada iteración será estudiada la opinión de cada implicado en cuanto a rendimiento y calidad del plan, de detectarse alguna carencia se modificará a fin de mejorar, llegando a una solución entre quien detecte la carencia y el Responsable de Verificación.

Cada modificación será comunicada por las vías especificadas a todos los implicados y reformulado el documento de Plan de Verificación y Validación.

2. Requerimientos para verificar

Además del alcance ya definido entre los requerimientos funcionales encontramos los siguentes casos de uso:

- Iniciar sesión
- Cerrar sesión
- Ver perfil
- Iniciar desafío
- Cancelar desafío
- Completar desafío
- Ver desafío finalizado
- Listar Ranking

3. Estrategia de Verificación

La verificación se enfocará principalmente sobre los siguientes tipo de errores:

- De algoritmos
- De sintaxis
- De precisión y cálculo
- De documentación
- De borde
- De recuperación

- Relativos al hardware

En base a la relevación de requerimientos no funcionales con el cliente se deja de lado la verificación de los tipos de errores de estrés, sincronización, desempeño y recuperación ante fallos.

Puesto que el cliente no espera una solución a problemas de conexión y supone Microsoft Azure como solución de escalabilidad, no representandonos una preocupación al momento de verificar.

Tanto lo que refiere a los errores de algoritmos y borde, sera verificado con técnica de caja negra, dividiendo el dominio en clases de equivalencia con el fin de alcanzar cubrimiento de deisión, finalemente buscaremos completar el cubrimiento con técnica de caja blanca.

De la sintaxis será verificado, branches inalcanzables, variables no usadas, referencias perdidas, inconsitencia en estilos de nombramiento y formato en general, discrepancia de tipos de variable o retorno, entre otras, con la utilización de un analizador sintactico de tiempo real.

En cuanto a la documentación, por medio de la inspección visual del Responsable de SQA y sus asistentes se verificará el estilo y contenido previo a su respectiva entrega.

Idem, la verificación del comportamiento del hardware será principalmente basado en inspección visual por parte del Responsable de Verificación y sus asistentes.

Como último a destacar de la estrategía, fijaremos el criterio de aceptación cuando alcancemos una cobertura de código de un 70% y un estimado de defectos remanentes de un 10% en base a la proyección del gráfico proporsionado por la herramienta de integración continua de cantidad de defectos totales por unidad de tiempo.

Adicionalmente ninguna prueba propuesta en la verificación podrá fallar al momento de la liberación.

3.1. Tipos de pruebas

Pruebas Unitarias

Las pruebas unitarias tienen como objetivo descubrir discrepancias entre la especificación de la unidad y su comportamiento una vez que ha sido codificado.

Pruebas de Integración

El objetivo de las pruebas de integración es verificar el correcto funcionamiento entre los distintos componentes con el fin de comprobar que interactúan correctamente a través de sus interfaces, cubren la funcionalidad establecida y se ajustan a los requisitos no funcionales especificados.

Pruebas de Regresión

Su objetivo es verificar la no regresión de la calidad luego de un cambio. Asegurar que los cambios no introducen un comportamiento no deseado u errores adicionales. Implican la reejecución de alguna o todas las pruebas realizadas.

Involucra reutilizar las pruebas, de forma que se pueda volver a probar con esas pruebas luego del cambio. Hay tres tipos de pruebas de regresión:

- Se puede hacer regresión de bugs arreglados cuando se reporta un bug y vuelve una nueva versión luego de arreglado. El objetivo es probar que no fue arreglado.

- Se puede hacer regresión de bugs viejos, que muestra que un cambio en el software causó que un bug arreglado se vuelva a romper
- Se puede hacer regresión de efectos secundarios. Implica volver a probar una parte del producto. El objetivo es probar que el cambio ha causado que algo que funcionaba ya no funcione.

Pruebas del Sistema

Son pruebas de integración del sistema completo, permiten probar el sistema en su conjunto y con otros sistemas con los que se relaciona para verificar que las especificaciones funcionales y no funcionales se cumplen.

Pruebas de Desempeño

Consisten en determinar que los tiempos de respuesta del sistema están dentro de los intervalos establecidos en las especificaciones del sistema.

Pruebas de Aceptación

El objetivo de las pruebas de aceptación es permitir al usuario validar que el sistema cumple con el funcionamiento esperado y que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento.

3.1.1. Prueba de integridad de los datos y la base de datos

3.1.1.1. Objetivo de la prueba

Asegurar que los métodos y procesos de acceso a la base de datos funcionan correctamente y sin corromper datos.

3.1.1.2. Técnica

A través de scripts sql se ejecutarán a modo de prueba en un ambiente controlado cada store procedure, asegurando el estado final con lo esperado en tablas concretas o en caso alternativo si el store procedure modifica un amplio espectro de la base de datos entonces el estado final será comparado contra backups con estado esperado.

3.1.1.3. Criterio de aceptación

Se dará por aceptada la verificación si cada store procedure definido (que implicitamente pueden hacer uso de triggers también) tengan al menos un caso de uso para las principales clases de equivalencia.

3.1.1.4. *Consideraciones especiales*

Como fue mencionado es preciso montar un ambiente de base de datos independiente del de producción.

3.1.2. Prueba de Funcionalidad

La prueba de funcionalidad se enfoca en requerimientos para verificar que se corresponden directamente a casos de usos o funciones y reglas del negocio. Los objetivos de estas pruebas son verificar la aceptación de los datos, el proceso, la recuperación y la implementación correcta de las reglas del negocio. Este tipo de prueba se basa en técnicas de caja negra, que consisten en verificar la aplicación y sus procesos interactuando con la aplicación por medio de la interfase de usuario y analizar los resultados obtenidos.

3.1.2.1. Objetivo de la prueba

Asegurar la funcionalidad apropiada del objeto de prueba, incluyendo la navegación, entrada de datos, proceso y recuperación.

3.1.2.2. Técnica

Ejecutar cada caso de uso, flujo de caso de uso, o función usando datos válidos y no válidos, para verificar lo siguiente:

- Se obtienen los resultados esperados cuando se usan datos válidos.
- Cuando se usan datos no válidos se despliegan los mensajes de error o advertencia apropiados.
- Se aplica apropiadamente cada regla del negocio.

3.1.2.3. Criterio de aceptación

Todas las pruebas planificadas se realizaron. Todos los defectos encontrados han sido debidamente identificados.

3.1.2.4. Consideraciones especiales

Guardar un historico de tipos de pruebas funcionales que se hicieron a modo de ejemplo para hacer pruebas de regresión y no repetir pruebas funcionales.

3.1.3. Prueba de Interfase de Usuario

Esta prueba verifica que la interfase de usuario proporcione al usuario el acceso y navegación a través de las funciones apropiada. Además asegura que los objetos presentes en la interfase de usuario se muestren como se espera y conforme a los estándares establecidos por la empresa o de la industria.

3.1.3.1. Objetivo de la prueba

Verificar que: la navegación a través de los elementos que se están probando reflejen las funciones del negocio y los requerimientos, incluyendo manejo de ventanas, campos y métodos de acceso; los objetos de las ventanas y características, como menúes, tamaño, posición, estado funcionen de acuerdo a los estándares.

3.1.3.2. Técnica

A través de captura de pantallas tanto gráficas (fotos) como en leguaje descriptivo (ejemplo xml) comparar con el resultado esperado de manera de controlar cambios no deseados, paralelamente se puede controlar consistencia en estilos como por ejemplo tamaño y tipo de fuentes, matices de colores, layouts o cualquier otro tipo de atributo de objetos de la interfaz.

3.1.3.3. Criterio de aceptación

Cada ventana ha sido verificada exitosamente siendo consistente con una versión de referencia o estándar establecido.

3.1.3.4. Consideraciones especiales

No todas las propiedades de los objetos se pueden acceder, quedando su verficación sujeta a inspección visual.

3.1.4. Prueba de Performance

En esta prueba se miden y evalúan los tiempos de respuesta, los tiempos de transacción y otros requerimientos sensitivos al tiempo. El objetivo de la prueba es verificar que se logren los requerimientos de performance. La

prueba de performance es implementada y ejecutada para poner a punto los destinos de pruebas de performance como función de condiciones de trabajo o configuraciones de hardware.

3.1.4.1. Objetivo de la prueba

Verificar la performance de determinadas transacciones o funciones de negocio bajo ciertas condiciones:

- condiciones de trabajo normales conocidas.
- peores casos de condiciones de trabajo conocidas.

3.1.4.2. Técnica

Usando procedimientos de prueba desarrollados para verificar funciones o ciclos de negocio.
Cada prueba de integración registrará instante de comienzo y final, verificando así que no se supere 3 segundo en lo que conlleva: comunicación entre componentes, extracción de datos a través de store procedures o no, procesamiento de datos, generación de resultados para persistir y/o presentar al usuario.

3.1.4.3. Criterio de aceptación

Éxito completo de las pruebas sin fallas y dentro del tiempo esperado o requerido.

3.1.4.4. Consideraciones especiales

Las bases de datos usadas para las pruebas de performance deben tener un tamaño similar a las reales.

3.1.5. **Prueba de Configuración**

La Prueba de Configuración verifica el funcionamiento del software con diferentes configuraciones de software y hardware.

3.1.5.1. Objetivo de la prueba

Verificar que el software funcione apropiadamente en dispositivos móviles con sistema operativo Android a partir de la versión 4.0 o Windows 7.1.

3.1.5.2. Técnica

Usar las pruebas de Funcionalidad bajo la diversidad de hardware disponible en el equipo de desarrollo, hemos constatado amplia variedad de versiones de sistema operativo, así como resoluciones de pantalla, capacidad de memoria RAM y procesamiento.

3.1.5.3. Criterio de aceptación

Por cada combinación de software objeto de prueba y software que no es objeto de prueba, todas las operaciones son completadas exitosamente sin fallas.

3.1.5.4. Consideraciones especiales

Los sistemas, red, servidores de red, bases de datos, etc., deben ser documentados como parte de esta prueba.

3.1.6. **Prueba de Instalación**

La Prueba de Instalación tiene dos propósitos. Uno es asegurar que el software puede ser correctamente descargado de las tiendas virtuales, instalandolo en diferentes condiciones. El otro propósito es verificar que, una vez instalado, el software opera correctamente. Esto significa normalmente

ejecutar un conjunto de pruebas que fueron desarrolladas para Prueba de Funcionalidad.

Respecto al servidor web, se proponé ante una nueva publicación de los Web Services Rest ofrecidos la reejecución de todo las pruebas de integración correspondientes, en lo posible de manera automática.

3.1.6.1. Objetivo de la prueba

Verificar que el software objeto de prueba se instala correctamente en cada configuración de hardware requerida bajo las siguientes condiciones:

- instalación nueva, una nueva máquina, nunca instalada previamente con BeatIt!
- actualización, máquina previamente instalada con BeatIt!, con la misma versión
- actualización, máquina previamente instalada con BeatIt!, con una versión anterior.

3.1.6.2. *Técnica*

Manualmente se para validará la condición de la máquina destino (nueva, nunca instalado, misma versión, versión anterior ya instalada).

Realizar la instalación.

Ejecutar un conjunto de pruebas funcionales ya implementadas para la Prueba de Funcionalidad.

3.1.6.3. Criterio de aceptación

Las pruebas de funcionalidad de BeatIt! se ejecutan exitosamente sin fallas.

3.1.6.4. Consideraciones especiales

Únicamente una conjunto de pruebas completas y exaustivas nos darán una garantía de correctitud de sistema.

3.1.7. **Prueba de Documentos**

La Prueba de Documentos debe asegurar que los documentos relacionados al software que se generen en el proceso sean correctos, consistentes y entendible. Se incluyen como documentos los Materiales todo tipo de documento que forme parte del paquete de software y no así para Soporte al Usuario, ya que por tratarse de una aplicación móvil deberá ser intuitiva y sencilla sin necesidad de ofrecer ayuda al momento de operarlo.

3.1.7.1. *Objetivo de la prueba*

Verificar que el documento objeto de prueba sea:

- Correcto, esto es, que cumpla con el formato y organización para el documento establecido en el proyecto.
- Consistente, esto es, que el contenido del documento sea fiel a lo que hace referencia. Y no debe contradecirse respecto a la función del sistema.
- Entendible, esto es, que al leer el documento se entienda correctamente lo que expresa y sin ambigüedades, además que sea fácil de leer.
- Completitud, esto es, que la especificación de requisitos debe incluir requisitos que definan todas las funciones y restricciones propuestas por el cliente.
- Verificabilidad, dado el perfil tecnico del cliente, no supondrá una limitante la utilización de lenguaje tecnico.

3.1.7.2. *Técnica*

Para verificar que el documento es correcto se debe comparar con el estándar definido o con las pautas de documentación y ver que el documento cumple con ellas.

En caso de Documentación Técnica se debe revisar el código al cual corresponde la documentación y comprobar que dicha describe el código.

Para verificar que el documento es entendible, debe comprobar que se entiende correctamente, que no tiene ambigüedades y que sea fácil de leer.

3.1.7.3. Criterio de aceptación

El documento expresa exactamente lo que debe expresar, no hay diferencias entre lo que está escrito y el objeto de la descripción (operación de software, código de programa, decisiones técnicas) y se entiende fácilmente.

3.1.7.4. Consideraciones especiales

Ninguna

3.2. Herramientas

Sistema de integración continua, para Java TravisCI de Travis CI Community, para C# TeamCity de JetBrains.

Inyector de dependencia, para Java Google-Guice de Google, para C# IUnity Container.

Framework de pruebas unitarias, para Java JUnit, para C# UnitTestFramework.

Covertura de código por pruebas, para Java EclElmma, para C# dotCover de JetBrains.

Analizador estático, para Java CodePro AnalytiX de Google, para C# ReSharper de JetBrains.

4. Recursos

En esta sección se presentan los recursos recomendados para el proyecto BeatIt!, sus principales responsabilidades y su conocimiento o habilidades.

4.1. Roles

En la tabla a continuación se muestra la composición de personal para el proyecto BeatIt! en el área Verificación del Software.

Rol	**Cant. mínima de recursos recomendada**	**Responsabilidades**
Responsable de verificación	1	Identifica, prioriza e implementa los casos de prueba. • Genera el Plan de Verificación. • Genera el Modelo de Prueba. • Evalúa el esfuerzo necesario para verificar. • Proporciona la dirección técnica. • Adquiere los recursos apropiados.

		Proporciona informes sobre la verificación.
Asistente de verificación	3	• Ejecuta las pruebas • Registra los resultados de las pruebas. • Recuperar el software de errores. • Documenta los pedidos de cambio.
Administrador de Base de Datos	1	• Realiza la gestión y mantenimiento del entorno de los datos (base de datos) de prueba y los recursos. • Administra la base de datos de prueba.

4.2. Sistema

En la siguiente tabla se establecen los recursos de sistema necesarios para realizar la verificación.

Recurso	Nombre/Tipo
Servidor de base de datos	MongoDB
Red o subred	Azure
Nombre del servidor	No definido
Nombre de la base de datos	No definido
Móvil Cliente para pruebas	Windows Phone y Android
Repositorio	GitHub

5. Hitos del proyecto de Verificación

La verificación de BeatIt! incorpora actividades de prueba para cada verificación identificada en las secciones anteriores. Entre los hitos del proyecto, se identifican:

- Cubrir con test javascript la funcionalidad de los web services (node.js).
- Sistema de integración continua verificando el repositorio y generando estadísticas
- Al cambiar la linea base, los requerimientos alcanzados correspondan con la documentación del momento.
- Ante la libaración de un prototipo, los requerimientos alcanzados correspondan con lo especificado en la documentación también.
- Queda completamente definido el alcance del proyecto por parte del cliente y se entiende factible su desarrollo.
- Al cabo de cada iteración se constata que los logros fueron los estimados.

Actividad que determina el hito	Esfuerzo	Fecha de comienzo	Fecha de finalización
Planificar la verificación	10 horas	16/8/2014	24/8/2014
Elaborar casos de prueba	6 horas	25/8/2014	7/9/2014
Fase 1 - Iteración 1			
Ajuste y Control de Verificación	21 horas	16/8/2014	30/8/2014
Ejecutar la verificación	10.5 horas	16/8/2014	30/8/2014

Evaluar la verificación	3.5 horas	26/8/2014	30/8/2014
Fase 1 - Iteración 2			
Ajuste y Control de Verificación	21 horas	31/8/2014	13/9/2014
Ejecutar la verificación	10.5 horas	31/8/2014	13/9/2014
Evaluar la verificación	3.5 horas	9/9/2014	13/9/2014
Fase 2 - Iteración 1			
Ajuste y Control de Verificación	24 horas	7/9/2014	20/9/2014
Ejecutar la verificación	12 horas	7/9/2014	20/9/2014
Evaluar la verificación	4 horas	16/9/2014	20/9/2014
Fase 2 - Iteración 2			
Ajuste y Control de Verificación	24 horas	21/9/2014	4/10/2014
Ejecutar la verificación	12 horas	21/9/2014	4/10/2014
Evaluar la verificación	4 horas	30/9/2014	4/10/2014
Fase 3 - Iteración 1			
Ajuste y Control de Verificación	27 horas	5/10/2014	18/11/2014
Ejecutar la verificación	13.5 horas	5/10/2014	18/11/2014
Evaluar la verificación	4.5 horas	14/11/2014	18/11/2014
Fase 3 - Iteración 2			
Ajuste y Control de Verificación	33 horas	19/11/2014	1/12/2014
Ejecutar la verificación	16.5 horas	19/11/2014	1/12/2014
Evaluar la verificación	5.5 horas	26/11/2014	1/12/2014

6. Entregables

6.1. Modelo de Casos de Prueba

Documento	**Modelo de Casos de Prueba**
Creado por	El Responsable de verificación, Emiliano Vázquez.
Para quien	Es la guía para realizar las pruebas del sistema y lo usarán los Asistentes de verificación y el Responsable de verificación cuando se ejecuten las pruebas del sistema.
Fecha de liberación	Será liberado el 7/9/2014.

6.2. Informes de Verificación

Documento	Se genera un documento **Informe de Verificación Unitaria** por cada prueba unitaria que se realice al sistema.
Creado por	Las personas que ejecutan las pruebas.
Para quien	Es el retorno para los implementadores de la tarea de verificación, que detalla los errores encontrados para que puedan ser corregidos.
Fecha de liberación	Será liberado luego de cada verificación unitaria. El informe es generado de inmediato bajo un sistema de integración continua.

Documento	Se genera un documento **Informe Consolidación** por cada consolidación que se realice al sistema.
Creado por	Las personas que ejecutan las pruebas.
Para quien	Es el retorno para los implementadores de la tarea de consolidación, que detalla los errores encontrados

	para que puedan ser corregidos.
Fecha de liberación	Será liberado luego de cada consolidación. El informe es generado de inmediato bajo un sistema de integración continua.

Documento	Se genera un documento **Informe de Verificación de Integración** por cada prueba de integración que se realice al sistema.
Creado por	Las personas que ejecutan las pruebas.
Para quien	Es el retorno para los implementadores de la tarea de verificación, que detalla los errores encontrados para que puedan ser corregidos.
Fecha de liberación	Será liberado luego de cada verificación de integración. El informe es generado de inmediato bajo un sistema de integración continua.

Documento	Se genera un documento **Informe de Verificación de Sistema** por cada prueba de sistema que se realice.
Creado por	Las personas que ejecutan las pruebas.
Para quien	Es el retorno para los implementadores de la tarea de verificación, que detalla los errores encontrados para que puedan ser corregidos.
Fecha de liberación	Será liberado luego de cada verificación de sistema. La fecha de liberación del informe será tre días antes de la fecha de publicación del prototipo en la tienda virtual.

6.3. Evaluación de la verificación

Documento	Se genera un documento **Evaluación de la verificación** por cada prueba que se realice al sistema. Este documento contiene las fallas encontradas en el sistema, la cobertura de la verificación realizada y el estado del sistema.
Creado por	El Responsable de verificación, que toma como fuente de su trabajo los Informes de verificación.
Para quien	Es el resumen de la tarea de verificación y es el retorno para todo el equipo de trabajo del estado del sistema.
Fecha de liberación	Será liberado luego de cada verificación, unitaria, de integración y de sistema. La fecha estimada será uno o dos días antes de cada iteración.

6.4. Informe final de verificación

Documento	El documento **Informe final de verificación** es el resumen de la verificación final del sistema antes de que sea liberado al entorno del usuario.
Creado por	El Responsable de verificación, que toma como fuente de su trabajo los Informes de verificación.

Para quien	Indica el estado del sistema.
Fecha de liberación	Será liberado luego de la verificación final del sistema. Fecha estimada 1 de diciembre de 2014

7. Dependencias

7.1. Dependencia de personal

Cada implementador estará a cargo de las correspondientes pruebas unitarias.

El responsable de verificación hará las pruebas de integración y sistema.

Los diseñadores de interfaz harán pruebas de sistema también y de diseño de interfaz.

El responsable de SCM gestionará las herramientas de integración continua, la cual generará reportes de verificación.

7.2. Dependencia de software

Será necesario implementar determinada funcionalidad para la carga de datos de prueba en un contexto transaccional. Esta funcionalidad tendrá tambien operadores de verificación para los distintas instancias de clase del sistema.

Por otro lado código sql para la ejecución y verificación de store procedures.

Emuladores de dispositivos móviles Windows Phone.

7.3. Dependencia de hardware

El hardware necesario será un Build Server y un servidor de Base de Datos.

Dispositivos móviles con sistema operatirvo Android.

7.4. Dependencia de datos y base de datos de prueba

Parte fundamental para agilizar la verificación y así reducir costos es tener información en cantidad y calidad semejante a la realidad.

La misma se encontrará en la base de datos propuesta para estas finalidades, asimismo código java y c# que recree la información pertinente para la verificación de los distintos componentes.

No se descarta la posibilidad de usar backups de base de datos por seguridad y para ser utilizados en herramientas de verificación automáticas, tanto sea para comparación de estado como para montaje de las pruebas a un estado inicial conocido.

8. Apéndice

8.1. Niveles de gravedad de error

A continuación se da una sugerencia de cuatro niveles diferentes de gravedad de error para poder capturar de alguna manera su impacto en el sistema. Además para poder evaluar la verificación y el sistema.

- **Catastrófico**: un error cuya presencia impide el uso del sistema.
- **Crítico**: un error cuya presencia causa la pérdida de una funcionalidad crítica del sistema. Si no se corrige el sistema no satisfará las necesidades del cliente.
- **Marginal**: un error que causa un daño menor, produciendo pérdida de efectividad, pérdida de disponibilidad o degradación de una funcionalidad que no se realiza fácilmente de otra manera.

- **Menor**: un error que no causa perjuicio al sistema, pero que requiere mantenimiento o reparación. No causa pérdida de funcionalidades que no se puedan realizar de otra manera.

Adicionalmente gracias a la herramienta de integración continua se podrá identificar responsables y asignar encargado de reparación.

8.2. Niveles de aceptación para lo elementos verificados

Se establecen niveles de aceptación para los elementos verificados para poder establecer el estado en el que se encuentra el proyecto.

Niveles de aceptación:

- **No aprobado**: el elemento verificado tiene errores catastróficos (uno o varios) que impiden su uso o tiene errores críticos (uno o varios) que hacen que el elemento verificado no sea confiable. El usuario no puede depender de él para realizar el trabajo.
- **Aprobado con Observaciones**: el elemento verificado no tiene errores catastróficos, ni errores críticos, pero tiene errores marginales (uno o varios) que hacen que el elemento de software se degrade en algunas situaciones.
- **Aprobado**: el elemento verificado no tiene errores o tiene errores menores que no afectan el normal funcionamiento del elemento.