UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN

DESARROLLO DEL MÓDULO PRINCIPAL Y ESTADÍSTICAS DE LA LIBRERÍA AUDITORÍAS TURPIAL

Por:
Stefani Carolina Castellanos Torres

INFORME DE PASANTÍA
Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero de la Computación

Sartenejas, Enero 2018

UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN

DESARROLLO DEL MÓDULO PRINCIPAL Y ESTADÍSTICAS DE LA LIBRERÍA AUDITORÍAS TURPIAL

Por:
Stefani Carolina Castellanos Torres

Realizado con la asesoría de:
Tutor Académico: MSc. Angela Di Serio
Tutor Industrial: Ing. Pedro Romero

INFORME DE PASANTÍA
Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero de la Computación

Sartenejas, Enero 2018

RESUMEN

DEDICATORIA

RECONOCIMIENTOS Y AGRADECIMIENTOS

ÍNDICE

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE SÍMBOLOS

LISTA DE ABREVIACIONES

HTML	HiperText Markup Language. Lenguaje de marcado de Hipertexto
HTTP	Hipertext Transfer Protocol. Protocolo de Transferencia de Hipertexto
INVEST	Independent Negotiable Valuable Estimable Small Testable. Independiente Negociable Estimable Small Testable.
MVC	Model-View-Controller. Modelo-Vista-Controlador.
MTV	Model-Template-View. Modelo-Plantilla-Vista.
URL	Uniform Resource Locator. Localizador Uniforme de Recursos

GLOSARIO

Backend

Framework

Frontend

Look-and-feel

Portable

INTRODUCCIÓN

CAPÍTULO 1

Entorno empresarial

1.1 Descripción

Turpial Development es una empresa mediana, con 4 años en el mercado que está enfocada en el desarrollo de sistemas y aplicaciones Web y Móviles. Fundada e integrada por jóvenes venezolanos y ofrece soluciones que cumplen con altos estándares de usabilidad, diseño y funcionalidad.

1.2 Misión

La empresa tiene como misión "prestar servicios y consultoría en diseño y desarrollo de soluciones web, a la medida del cliente, caracterizadas por una alta calidad, excelente soporte y experiencia de usuario". (Manual para desarrolladores, 2016)

1.3 Visión

Su visión es "servir de plataforma para el desarrollo y éxito de nuevos emprendimientos en el área web". (Manual para desarrolladores, 2016)

1.4 Estructura

Dirección de Operaciones

Se encarga del funcionamiento de todos los procesos de soporte de la empresa, como administración, recursos humanos, contabilidad y compras con el fin de garantizar el correcto desarrollo de los procesos principales de la empresa.

Dirección de Proyectos

Se encarga de atender a las necesidades y requerimientos de los clientes, sirviendo de enlace con las direcciones de desarrollo y diseño para garantizar la calidad del producto o entrega.

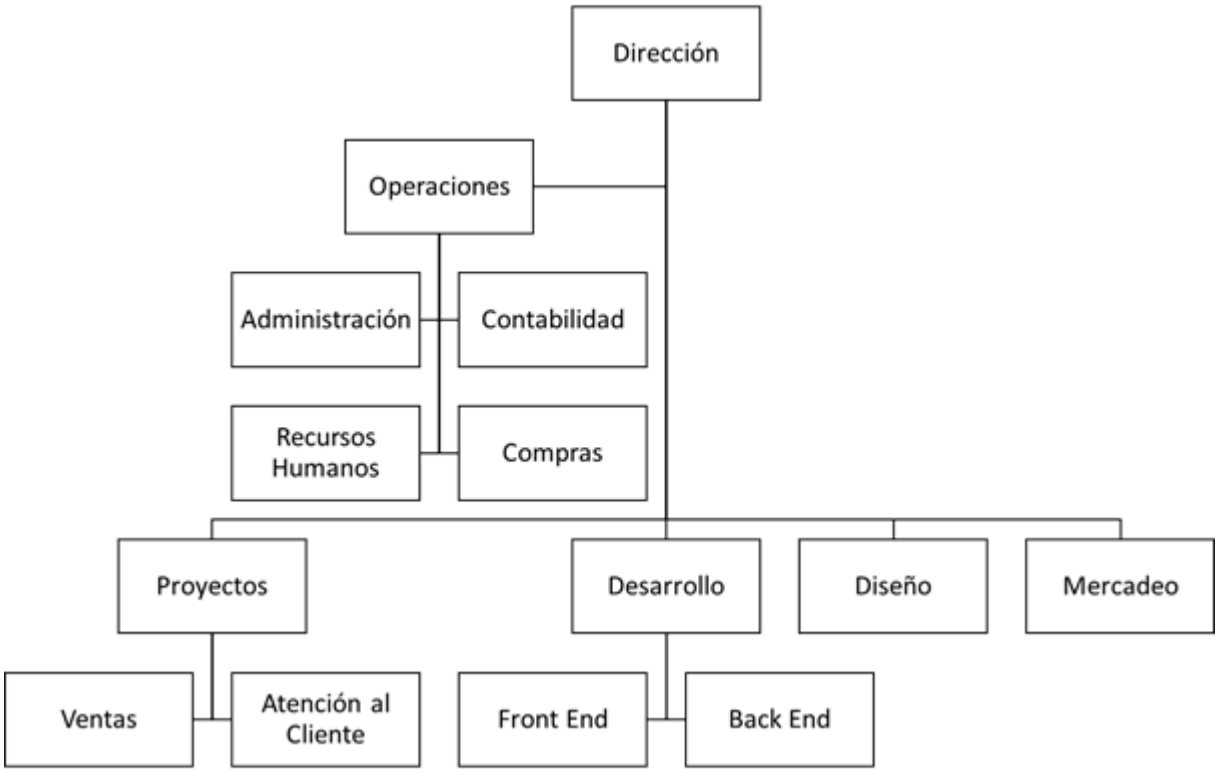
Dirección de Desarrollo

Se encarga de la conceptualización y desarrollo de las necesidades del cliente, implementando e integrando el diseño acordado y las funcionalidades requeridas por dicho cliente. Se divide en tres departamentos: Conceptualización, *Backend* y *Frontend*.

Dirección de Diseño

Se encarga de la conceptualización y diseño de la interfaz gráfica de los proyectos en función de las necesidades del cliente y de la mano con las decisiones tomadas por la Dirección de Desarrollo.

Dirección de Mercadeo



Ubicación del pasante

El desarrollo de la pasantía fue llevado a cabo en la Dirección de Desarrollo de la empresa. El pasante fue asignado al departamento de *Backend* bajo el cargo de Pasante, para cumplir tareas tales como: levantar requerimientos, diseñar y desarrollar las funcionalidades de la solución propuesta para el sistema.

CAPÍTULO 2

Definición del problema

2.1 Antecedentes

2.2 Planteamiento del problema

2.3 Justificación

2.4 Objetivo general

Implementar, probar y presentar las funcionalidades de selección, gestión y listados de auditorías y todas las funcionalidades del módulo Estadísticas de la librería de Auditorías Turpial e implantar un sistema de integración continua con el repositorio.

2.5 Objetivos específicos

CAPÍTULO 3

Marco teórico

3.1 Auditoría

Es una revisión sistemática para determinar si la calidad de las actividades cumplen con los acuerdos planificados y si estos están implementados efectivamente y son adecuados para alcanzar sus objetivos. Ofrecen una oportunidad de mejora al sistema y pueden ser llevadas a cabo para cumplir con normas regulatorias. Las auditorías se pueden aplicar a sistemas, procesos, programas o servicios y pueden ser internas (realizadas por la misma empresa) o externas (realizadas por proveedores). [18]

3.2 Acciones auditables

Se considera una acción auditable todo flujo del sistema que cree, edite o borre alguna información sensible para el mundo de negocio. Son registradas incluyendo información sobre quién realizó la acción, qué acción se intentó realizar y cuándo ocurrió la acción.

3.3 Microservicio

Es un un tipo de arquitectura que consiste en desarrollar una aplicación como un conjunto de pequeños servicios. Dichos servicios son procesos autónomos, cohesivos e independientes que suelen interactuar con otros componentes a través de mensajes [7]. Se enfocan en resolver un único problema y funcionan de manera aislada; si se presentan una falla no se propaga y puede ser atendido más rápidamente. [14]

Estos servicios se manejan de manera descentralizada y hacen uso de un despliegue completamente automatizado. Cada uno de ellos pueden ser escritos en diferentes lenguajes y tecnologías para almacenar información y aún así, interactuar y compartir información. [14]

3.4 Integración Contínua

Es una práctica de desarrollo de software en la que los miembros del equipo combinan su trabajo de forma periódica, usualmente cada día. Cada integración está verificada por una herramienta automatizada que empaquete y pruebe el código para detectar errores lo más rápido posible [10], de esta manera se puede mejorar la calidad del software y reducir el tiempo en validar y publicar actualizaciones.

3.5 Pruebas automatizadas

Las pruebas tienen como objetivo ejercitar el código para detectar errores y verificar que el software satisface los requerimientos especificados para asegurar su calidad. Estas son realizadas desde el punto de vista del usuario y las funcionalidades son probadas ingresando información y examinando la salida. [8]

Esta labor puede ser larga y repetitiva, por lo que es conveniente contar con herramientas que provean métodos que faciliten el proceso de escribir y ejecutar casos de prueba y así, reducir significativamente el esfuerzo y tiempo invertido por los desarrolladores. [8] A este conjunto de casos de pruebas se le conoce como prueba automatizada.

3.6 Patrón Modelo-Vista-Controlador

Este patrón de diseño asigna objetos en una aplicación a uno de tres roles: modelo, vista o controlador y define cómo se comunican entre ellos. La colección de objetos de cada tipo puede ser referido como una "capa".[1]

- Modelo: controla el comportamiento y la data de la aplicación, responde las solicitudes de información (usualmente desde la vista) y las instrucciones de cambio de estado (usualmente desde el controlador).
- Vista: maneja la presentación de la información.
- Controlador: interpreta las entradas del usuario y le informa al modelo y/o vista para cambiar lo que sea apropiado.[12]

Figura 1. Diagrama del patrón MVC

Es importante notar que la vista y el controlador dependen del modelo, sin embargo el modelo es independiente. Esta separación permite probar el modelo aparte de la presentación visual.

3.7 Patrón Modelo-Vista-Plantilla

Es una adaptación del patrón MVC, en el cual la "vista" define cual dato es presentado y su comportamiento, no como se muestra. El dato es obtenido a través de la función de callback para pedir una URL en particular. Por otro lado, la "vista" delega a la "plantilla" la presentación de la información.

En el caso de Django, el controlador es el *framework* en sí: "la maquinaría que envía una petición a la vista apropiada, de acuerdo a la configuración de URL de Django" [5]

3.8 Señales

Permiten a las aplicaciones ser notificadas cuando ocurra una acción en algún otro lugar, es decir, las señales permiten a ciertos emisores notificar a un conjunto de receptores que alguna acción está siendo ejecutada. [6] En términos de software, son el análogo a interrupciones de hardware. [17]

3.9 Mixins

Es un estilo de programación en el cual las unidades de funcionalidad son creadas en una clase y se incorporan en otras [9]. Pueden entenderse como "un subclase abstracta que puede ser usada para especializar el comportamiento de una variedad de padres". [3]

Usualmente, los Mixins, definen nuevos métodos que realizan alguna acción y luego llaman a los métodos del padre correspondiente; pueden utilizarse en varias clases de la jerarquía y sin importar en qué clases sean usadas.

Hay varias razones para usar Mixins: extienden las clases existentes sin tener que editar, mantener o combinar el código fuente; mantienen el proyecto en componentes separados; facilitan la creación de nuevas clases con funcionalidades pre-fabricadas y que se puede combinar según sea la necesidad.[9]

CAPÍTULO 4

Marco tecnológico

4.1 Python

Es un lenguaje de programación interpretado y fácil de entender. Posee estructuras de datos de alto nivel y una aproximación sencilla al paradigma de programación orientado a objetos. Cuenta con una amplia variedad de librerías que fomentan la reutilización de código; también posee tipos de datos dinámicos y una sintaxis simple para facilitar el rápido desarrollo de aplicaciones. [16]

4.2 Django

Es un *framework* de código abierto, escrito en Python y está basado en el patrón Modelo-Plantilla-Vista. Proporciona diversas funcionalidades reutilizables para desarrollar, rápidamente, aplicaciones Web escalables. [5]

4.3 HyperText Markup Language (HTML)

El lenguaje de marcado de hipertexto es un formato de datos simple, usado para crear documentos portables de una plataforma a otra y ha sido utilizada ampliamente en la World Wide Web desde 1990. [2]

Existe comportamiento que es comúnmente utilizado en las aplicaciones (i.e crear un elemento), por esto, Django está equipado con un conjunto de Clases y Mixins que pueden ser heredados y proveen el comportamiento estándar de alguna acción. Adicionalmente, el programador puede crear otros nuevos para extender estas funcionalidades.

También incluye un despachador de señales que ayuda a las aplicaciones desacopladas a recibir notificaciones cuando alguna acción ocurra en algún otro lugar en el framework. Algunos de los eventos sobre los que notifica Django son la inserción, actualización y eliminación de elementos de la base de datos y migración de la misma.

4.4 Javascript

Es un lenguaje interpretado, multi-paradigma y dinámico que soporta estilos de programación funcional, orientado a objetos e imperativa, así como funciones de primera clase. Es comúnmente utilizado como el lenguaje de *script* para páginas Web. [13]

4.5 Pytest

Es un framework escrito en Python, que facilita la escritura de complejas pruebas de funcionalidad para aplicaciones y librerías y así, asegurar la calidad del software que será entregado. Permite controlar la ejecución de pruebas automatizadas y comparar los resultados obtenidos con los esperados [15]

4.6 Amcharts o Chart.js

4.7 PostgreSQL

4.8 MySQL

4.9 SQLite

4.10 JSON

Es un formato para intercambiar datos, ligero, independiente del lenguaje y está basado en textos. Define un pequeño conjunto de reglas para la representación de datos estructurados que sean portables. JSON puede representar cuatro tipos primitivos (cadenas de caracteres, números, booleanos y null) y dos tipos estructurados (objetos y arreglos). [4]

4.11 Git

Es un sistema de control de versiones de código abierto, diseñado para administrar cualquier tipo de proyecto con rapidez y eficiencia. Dispone de facilidades para llevar el seguimiento de los cambios realizados, soportar el desarrollo no-lineal, cambiar fácilmente de contexto y realizar experimentos sin afectar las versiones entregables del proyecto. [11]

4.12 Jenkins

CAPÍTULO 5

Marco metodológico

5.1 Descripción de la metodología TAUP

TAUP es una metodología ágil creada por la empresa Turpial Development basada en los principios del "manifiesto ágil" y brinda a su equipo de trabajo una manera eficaz de llevar a cabo el desarrollo de software. En el apéndice A, se encuentra mayor detalle sobre el manifiesto ágil.

TAUP considera que la prioridad es la satisfacción del cliente mediante entregas continuas, las cuales deben realizarse en lapsos entre 1 semana y 1 mes. Está conformada por tres fases: concepción , construcción y transición. En el apéndice B se especifica los diferentes aspectos relacionados a la metodología.

5.1.1 Fase de Concepción

El objetivo de esta etapa es que el equipo de desarrollo profundice la comprensión de los requerimientos del sistema. Para ello, se escriben las Historias de Usuario (HU) con su respectiva prioridad y nivel de dificultad. Estas son establecidas por el cliente y el equipo, respectivamente, haciendo uso de la siguiente escala: Alta, Media alta, Media, Media baja y Baja. También, el equipo, asigna Story Points (puntos) a cada HU que ponderan el esfuerzo que se requiere para culminarla.

Las HU cuentan con criterios de aceptación bien definidos que brindan más detalle respecto a los aspectos que validan su completitud y poseen pruebas de aceptación para asegurar que el producto cumpla con los estándares establecidos. Adicionalmente, deben cumplir con ciertos requisitos para que se comience a desarrollar (Definition of Ready o DoR) y otros para que sea considerada como culminada (Definition of Done o DoD).

Por último, se construye la lista de todas las Historias de Usuarios (Backlog) en el orden que se van a ejecutar tomando en cuenta la prioridad y el riesgo de la tarea y se elabora un

calendario en el cual se agregarán los Sprints que sean necesarios para que todas las Historias de Usuarios sean cumplidas a cabalidad, llamado Release Plan (Apéndice D).

5.1.2 Fase de Construcción

El enfoque de está fase es desarrollar todo lo planteado en la Concepción, validando así, la arquitectura planteada. La metodología se basa en Sprints o iteraciones que son bloques de tiempo de duración corta y fija en los que al final, se puede obtener un producto potencialmente entregable.

El primer día de cada Sprint se lleva a cabo la Reunión de Planificación, en la que todos los miembros del equipo revisan lo que se tiene planteado en el Release Plan, se verifica que cada una de esas Historias de Usuario estén en estado DoR para que sean divididas en tareas y se aclare cualquier duda acerca de lo que se va a realizar.

Durante el Sprint, diariamente, se lleva a cabo una reunión corta llamada Daily Stand Ups Meeting, en la cual se discute el estado de las tareas para el momento, que se va a hacer y cuáles obstáculos se han presentado en el desarrollo de alguna HU.

El último día de cada Sprint, los miembros del equipo deben reunirse para realizar la Revisión de Iteración, en la cual estará presente el cliente para mostrarle todos los avances. Luego, se realiza la Reunión de Retrospectiva, en donde se discute que hizo bien, que se puede mejorar y a que se compromete.

5.1.3 Fase de Transición

En esta fase se probará todo lo desarrollado en la fase de Construcción. Estas pruebas funcionales y no funcionales, verifican que el proyecto pueda ser utilizado en un ambiente de producción, además se creará un manual para el usuario, el cual le dará la capacitación necesaria al cliente para poder utilizar sin problemas el sistema. También se validará la documentación, comprobando que todo el código esté correctamente explicado para que pueda ser entendido con facilidad en caso de que sea necesario realizar cambios.

Luego de que el sistema esté completo, se tendrá que desplegar en el ambiente de producción, al terminar este proceso se otorgará un tiempo prudencial al cliente para efectuar pruebas y se llevan a cabo las correcciones pertinentes.

5.2 Adaptación de la metodología a la pasantía

En la sección anterior se explicó la metodología TAUP, sin embargo, dependiendo del proyecto que se desea desarrollar, se puede realizar algunas modificaciones que mejoren la dinámica y la velocidad del equipo o porque el cliente así lo requiera. A continuación se presentan los cambios realizados en cada fase para ajustarlos a la pasantía en cuestión.

5.2.1 Fase de Concepción

En cuanto a las Historias de Usuario, se ideó un código compuesto por una letra y un número que facilita referenciarla. La letra representa el módulo a la que pertenece, Core (Principal) o Estadísticas, y el número denota el orden que fue concebida. Adicionalmente, se utilizó una modificación para las escalas de prioridad y riesgo, que está conformada por tres valores: Alta, Media y Baja. Para más información sobre las HU desarrolladas, leer el Apéndice C

Con respecto a la Definition of Ready se tiene que debe cumplir con los siguientes criterios:

- Debe ubicarse dentro de uno de los módulos del proyecto.
- Debe de tener asignado una prioridad por el cliente.
- Debe de tener asignado un riesgo por el equipo de desarrollo.
- Debe de tener asignado su respectivo puntaje.
- (Opcional pero deseado) Debe de tener una breve descripción, aclaratoria o criterio adicional según sea el caso.

Mientras que para la Definition of Done posee los siguientes estatutos:

- Debe realizarse la codificación respectiva
- El código generado debe estar debidamente documentado para facilidad de programador
- Debe de realizarse la documentación respectiva (de ser necesaria) de todos los aspectos de configuración asociados al desarrollo y buen funcionamiento de la historia de usuario.
- Deben realizarse pruebas automatizadas a la codificación generada.
- Debe presentarse la nueva funcionalidad al cliente.
- Debe estar disponible en el servidor.

5.2.2 Fase de Construcción

Para el desarrollo de este proyecto se planificaron ocho Sprints con una duración de dos semanas laborales (cinco días) cada uno. Al finalizar esta etapa, se debe haber culminado el desarrollo del módulo principal y estadísticas de Auditorías Turpial.

5.2.3 Fase de Transición

CAPÍTULO 6

Desarrollo

6.1 Fase de investigación

6.2 Fase de concepción

6.3 Fase de construcción del núcleo

6.4 Fase de construcción del módulo de estadísticas

6.5 Fase de transición

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

Recomendaciones

REFERENCIAS

[1] Apple Inc. Model-view-controller. https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html, 2015. Consultado el 27/08/2017.

[2] Berners, L. Hypertext markup language - 2.0. https://tools.ietf.org/html/rfc1866, 1995. Consultado el 15/07/2017.

[3] Bracha, G.Cook, W. Mixin-based inheritance. http://www.bracha.org/oopsla90.pdf, 1990. Consultado el 29/09/2017.

[4] Bray, Ed. Json. https://tools.ietf.org/html/rfc7159, 2014. Consultado el 17/07/2017.

[5] Django Software Foundation. Faq: General. https://docs.djangoproject.com/es/1.11/faq/general/, 2017. Consultado el 27/08/2017.

[6] Django Software Foundation. Faq: General. https://docs.djangoproject.com/en/1.11/topics/signals/, 2017. Consultado el 28/08/2017.

[7] Dragoni, N.Giallorenzo, S. Microservices: yesterday, today, and tomorrow. https://arxiv.org/pdf/1606.04036.pdf, 2017. Consultado 28/07/2017.

[8] Esmite, I. Farías, M. Automatización y gestión de las pruebas funcionales usando herramientas open source. http://sedici.unlp.edu.ar/bitstream/handle/10915/21787/Documento_completo.pdf?sequence=1. Consultado el 25/07/2017.

[9] Esterbrook, C. Using mix-ins with python. http://www.linuxjournal.com/article/4540, 2001. Consultado el 29/09/2017.

[10] Fowler, M. Continuous integration. https://www.martinfowler.com/articles/continuousIntegration.html, 2006. Consultado el 28/07/2017.

[11] Git. Pytest. https://git-scm.com/, 1. Consultado el 17/07/2017.

[12] Microsoft. Model-view-controller. https://msdn.microsoft.com/en-us/library/ff649643.aspx. Consultado el 27/08/2017.

[13] Mozilla. Javascript. https://developer.mozilla.org/es/docs/Web/JavaScript, 1995. Consultado el 15/07/2017.

[14] Sam Newman. *Building Microservices*. O'Reilly Media, Inc., 1st edition, 2015.

[15] Pytest. Pytest. `https://docs.pytest.org/en/latest/`, 2017. Consultado el 17/07/2017.

[16] Python Software Foundation. The python tutorial. `https://docs.python.org/3/tutorial/index.html`, 2017. Consultado el 15/07/2017.

[17] Andrew S. Tanenbaum and Herbert Bos. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 4th edition, 2014.

[18] M.B. Weinstein. *Total Quality Safety Management and Auditing*. Taylor & Francis, 1997.

APÉNDICE A

Manifiesto Ágil

Es un estatuto que promueve modelos organizacionales basados en la colaboración entre personas sin documentar de manera excesiva el proceso. Posee sistema que se adapta a las necesidades del cliente y lo incluyen de manera activa en el proceso de desarrollo. Se rige por los siguientes 12 principios.

1. La mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. La mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
3. Aceptar que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
4. Entregar software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
5. Los responsables de negocio y los desarrolladores trabajan juntos de forma cotidiana durante todo el proyecto.
6. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
7. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
8. El software funcionando es la medida principal de progreso.
9. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
10. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
11. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.

12. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.

13. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

APÉNDICE B

Manual de metodología de Turpial (resumen)

Descripción de la metodología TAUP

TAUP o Turpial Agile Unified Process es una metodología ágil creada por la empresa Turpial Development basada en los principios del "manifiesto ágil" que brinda a su equipo de trabajo una manera eficaz de llevar a cabo el desarrollo de software.

Fase de Concepción

El objetivo de esta etapa es que el equipo de desarrollo profundice la comprensión de los requisitos del sistema y validación de la arquitectura propuesta.

Historias de Usuario (HU)

Es una técnica para expresar requerimientos que se caracteriza por ser sencilla de escribir, leer y evaluar. El usuario final o cliente le dará valor a esta HU ya que describe una funcionalidad que él necesita.

Las historias de usuarios se escriben en oraciones que poseen tres componentes: el rol del usuario que quiere realizar la acción, que desea hacer y por qué. Una buena HU está caracterizada por ser independiente, negociable, valiosa, estimable, pequeña y comprobable (INVEST).

Adicionalmente, deben contar con criterios de aceptación bien definidos que brinden más detalle respecto a los aspectos de la Historia de Usuario que validan su completitud. Deben ser escritos en lenguaje natural y cada oración debe decir una funcionalidad que tendrá que ser ejecutada por la HU.

Por último, una HU, debe contar con pruebas de aceptación ya que van a permitir que el producto cumpla con los estándares y que se satisfagan los requerimientos establecidos.

Corresponden a una condición que se pueda verificar de los criterios de aceptación y una vez que han sido validadas, se puede dar por culminada la Historia de Usuario.

Priorizar Historias de Usuario

El cliente establecerá una prioridad para las Historias de usuario, la cual es uno de los factores que influyen en el orden de ejecución de las mismas. Se puede utilizar la siguiente escala de cinco valores: Alta, Media alta, Media, Media Baja y Baja.

Análisis de riesgo de Historias de Usuario

El equipo de desarrollo debe establecer un nivel de dificultad para cada HU, para esto, los desarrolladores deben tomar en cuenta diversos factores como: aprender nuevas tecnologías, manejar herramientas desconocidas o investigar un tema en particular. La escala cuenta con cinco valores: Alto, Medio Alto, Medio, Medio Bajo, Bajo.

Para este proyecto se utilizó, al igual en la sección anterior, una escala reducida para puntuar el riesgo: Alto, Medio y Bajo.

Análisis de Esfuerzo de Historias de Usuario

Representa una estimación de cuánto esfuerzo se requiere para culminar una Historia de Usuario, a esto se le denominará Story Points. Al momento de asignar los Story Points se deben considerar diversos factores como el riesgo, complejidad y desconocimiento de la tarea a realizar, entre otros. La escala de ponderación viene dada por la secuencia de Fibonacci.

Cuando una Historia de Usuario consigue un Story Point muy alto (usualmente un valor mayor a 8), deberá ser revisada nuevamente ya que podría dividirse en varias Historias de Usuario que puedan más manejables. Cada Story Point corresponde a lo que se le denomina un Login Point que posee el valor de uno y corresponde a un módulo de autenticación de usuarios de un
sistema.

El equipo de desarrollo debe estimar de manera consensuada los Story Point a través de una dinámica denominada Planning Poker; cada miembro puntuará las Historias de Usuario, se eliminarán el valor más alto y el más bajo, el resto se promedian.

Backlog

A la lista de todas las Historias de Historia de Usuarios construidas a partir de toda la información suministrada en los puntos anteriores se le denominará Backlog, al construirla se determina el orden se van a ejecutar tomando en cuenta la prioridad y el riesgo de la tarea, sin embargo, puede cambiar.

Release Plan

Es un calendario en el cual se agregarán los Sprint que sean necesarios para que todas las Historias de Usuarios sean cumplidas a cabalidad.

Definición de Listo "DoR"

DoR (Definition Of Ready) definido así por sus siglas en inglés, es un acuerdo que se establece en el equipo de desarrollo para cada Historia de Usuario que responde a la pregunta '¿Qué tiene que estar listo antes de empezar a trabajar en una Historia de Usuario y la misma pueda ser ejecutada en el próximo Sprint?'.

Se debe tener en cuenta los siguientes aspectos:

- Que sea independiente o de lo contrario que la(s) Historia de Usuario que lo preceden ya están implementadas o van a desarrollarse en el mismo Sprint.
- Que la Historia de Usuario tiene que ser desarrollada en un solo Sprint.
- Que los criterios de aceptación para la Historia de Usuario sean detallados
- Que las pruebas de aceptación puedan realizarse en el Sprint.
- Que la Historia de Usuario tenga asignado su Story Point.
- Tomar en cuenta si existe alguna dependencia externa para la Historia de Usuario.

Definición de Terminado "DoD"

DoD (Definition of Done) definido así por sus siglas en inglés, responde a la pregunta "¿Qué tiene que cumplir una Historia de Usuario para que se considere finalizada?", también puede incluir otros aspectos que sean considerados por el equipo de desarrollo como primordiales para que una HU sea considerada terminada. Este acuerdo explica con detalle los criterios para que toda HU sea considerada terminada.

- La Historia de Usuario ya ha sido analizada.
- El código tiene que estar escrito.
- El código tiene que estar documentado o comentado.
- El código se ha integrado a todo lo desarrollado con anterioridad.
- La Historia de Usuario ha pasado por todas sus pruebas de aceptación con éxito.
- Haberle mostrado la Historia de Usuario al cliente y que este la acepte.

Fase de Construcción

El enfoque de está fase es desarrollar todo lo planteado en la Concepción, validando así, la arquitectura planteada. La metodología se basa en Sprints o iteraciones.

Sprints

El desarrollo del software se tiene que ejecutar por bloques de duración corta y fija, de 5 días (semana laboral) hasta 1 mes. Cada Sprint tiene que proporcionar un resultado concreto, un incremento de producto que sea potencialmente entregable, de tal manera que pueda estar disponible para ser utilizado por el cliente y así, proporcionar observaciones de lo que se le está mostrando.

El primer día de cada Sprint se lleva a cabo la Reunión de Planificación, en la que todos los miembros del equipo revisan lo que se tiene planteado en el Release Plan para ese Sprint, verifican que cada una de esas Historias de Usuario estén en estado DoR para que sean divididas en tareas y se aclare cualquier duda acerca de lo que se va a realizar. Luego de que se conozcan cuáles son las tareas a desarrollar, se distribuyen a cada miembro del equipo y se colocan en un Kanban Board para saber cuáles están por hacer (To Do), cuales se están haciendo (Doing) y cuales están finalizadas (Done).

Durante el Sprint se debe realizar todos los días el Daily Stand Ups Meeting cuyos integrantes son el equipo desarrollador y su líder de proyecto. Tienen un horario (no debería de durar más quince minutos) y lugar fijo y los participantes tendrán tiempo para hablar y responder tres preguntas: qué ha realizado desde el último Daily, qué se está haciendo y qué bloquea su progreso.

El último día de cada Sprint, los miembros del equipo se deben reunir para realizar una Revisión de Iteración, en la cual estará presente el cliente para mostrarle todos los avances

y que dé su opinión para que el equipo realice las correcciones pertinentes lo más pronto posible. Luego, se realiza la Reunión de Retrospectiva, en donde se discute que hizo bien, que se puede mejorar y a que se compromete

Para cada Sprint es necesario ver qué cantidades de Story Points se van a realizar y cuáles se están trabajando y cuales están ya en DoD. Para esto se crea una gráfica que sirve para saber si se puede o no terminar lo planteado para el Sprint, llamada Burndown Chart, en la cual se coloca en el eje X el tiempo que dura el Sprint y en el eje Y la cantidad de Story Points. Esta práctica también sirve para referirse al avance de todo el proyecto, haciendo el mismo procedimiento antes explicado, pero cambiando los ejes de manera que en ellos se encuentre el total de Story Point del desarrollo del proyecto y todos los Sprint del mismo.

Fase de Transición

En esta fase se probará todo lo desarrollado en la fase de construcción. Estas pruebas funcionales y no funcionales verifican que el proyecto pueda ser utilizado en un ambiente de producción, además se creará un manual para el usuario, el cual le dará la capacitación necesaria al cliente para poder utilizar sin problemas el sistema.

Pruebas

Se realiza la validación del sistema a través de pruebas de integración, despliegue y comportamiento y que el mismo pueda ser desplegado en un ambiente de preproducción. También se validará la documentación, en donde se verificará que todo el código esté correctamente documentado para que pueda ser entendido con facilidad en caso de que sea necesario realizar cambios. Se crea el Manual de Usuario, en el que se especifica todas las funcionalidades del sistema para que el cliente pueda consultar si tiene alguna duda con respecto a la funcionalidad.

Por último, se verifica que todo lo anterior se haya cumplido, de no ser así se tendrá que hacer pruebas de regresión y modificaciones a la validación de la documentación hasta que el cliente esté satisfecho con el trabajo realizado.

Puesta en Producción

Luego de que el sistema esté completo, se tendrá que desplegar en el ambiente de producción, al terminar este proceso se otorgará un tiempo prudencial al cliente para efectuar pruebas. El reporte de estos defectos será más formal ya que llevarán un registro, junto con sus detalles para que los desarrolladores puedan corregirlos.

APÉNDICE C

Backlog del proyecto

Núcleo

Código	Historia de usuario	Prioridad	Riesgo	Story Points
C01	Como programador, quiero indicar cuáles elementos de mi base de datos quiero auditar para poder indicar a la librería cuales modelos debe llevarle seguimiento.	Alta	Alta	5
C02	Como programador, quiero poder almacenar cuando un elemento auditable fue creado así podré saber cuándo, quién y qué fue ingresado en el sistema.	Alta	Alta	5
C03	Como programador, quiero poder almacenar cuando un elemento auditable fue editado así podré saber cuándo, quién y cómo fue alterada la información en el sistema.	Alta	Alta	5
C04	Como programador, quiero poder almacenar cuando un elemento auditable fue eliminado así podré saber cuándo, quién y cuál la información fue borrada en el sistema.	Alta	Alta	5
C05	Como usuario, quiero disponer de listas de cada elemento auditable para poder observar el comportamiento de mi información.	Alta	Baja	2
C06	Como usuario, quiero que las listas de mis auditorías puedan ordenarse según autor, fecha de creación, acción efectuada sobre la data	Alta	Baja	1

C07	Como usuario, quiero que las listas de mis auditorías puedan filtrarse según autor, fecha de creación, acción efectuada sobre la data; para así facilitar la búsqueda de auditorías en específico.	Alta	Baja	1
C08	Como programador, quiero poder configurar el orden en que se desplegarán las columnas de mis listados, así mantener los patrones de experiencia de usuario de mi sistema.	Media	Media	2
C09	Como programador, quiero contar con etiquetas personalizadas que me permitan incluir cualquier listado de las auditorías en mi sistema	Alta	Media	3
C10	Como programador, quiero contar con un medio para personalizar el Look and Feel de mis listados de auditorías para ajustarlos al estilo de mi sistema.	Media	Alta	5
C11	Como usuario, quiero contar con un control de acceso a los listados de las auditorías y así solo brindar a un conjunto de usuarios las auditorías(Dividida en C11.1, C11.2 y C11.3)	Media	Baja	8
C11.1	Como usuario, quiero contar con un login para restringir el acceso a usuarios no autorizados a mis auditorías.	Media	Baja	2
C11.2	Como usuario, quiero poder autorizar/desautorizar usuarios al acceso de auditorías.	Media	Baja	5
C11.3	Como usuario, quiero poder listar a todos los usuarios autorizados a acceder a las auditorías.	Media	Baja	1
C12	Como usuario, quiero que el sistema se actualice con la última versión de la librería por medio de integración continua para garantizar el soporte al sistema en donde se instale la librería.	Alta	Alta	5

C13	Como desarrollador quiero que se instalen automáticamente las dependencias necesarias en el sistema que se instale la librería.	Alta	Media	3

Estadísticas

Código	Historia de usuario	Prioridad	Riesgo	Story Points
E01	Como usuario quiero ver las estadísticas globales entre todas las acciones auditables a lo largo de un rango de tiempo.	Alta	Media	2
E02	Como usuario quiero disponer de gráficas con las estadísticas globales.	Media	Media	2
E03	Como usuario quiero ver las estadísticas por acción auditable a lo largo de un rango de tiempo.	Alta	Media	2
E04	Como usuario quiero disponer de gráficas con las estadísticas por acción auditable.	Media	Media	2
E05	Como usuario quiero ver las estadísticas por autor a lo largo de un rango de tiempo.	Alta	Media	2
E06	Como usuario quiero disponer de gráficas con las estadísticas por autor.	Media	Media	1
E07	Como usuario quiero ver las estadísticas de una tabla de auditoría específica de la base de datos a lo largo de un rango de tiempo.	Alta	Media	3
E08	Como usuario quiero disponer de gráficas con las estadísticas de una tabla de auditoría específica.	Media	Media	1