Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

ANÁLISIS PARA LA CREACIÓN DE UNA GUÍA PARA LA APLICACIÓN DE BEHAVIOR DRIVEN DEVELOPMENT COMO PARTE DE LA METODOLOGÍA EN LOS CURSOS DEL ÁREA DE DESARROLLO DE SOFTWARE DE INGENIERÍA EN CIENCIAS Y SISTEMAS

Marlon Abraham Fuentes Zárate
Asesorado por el Ing. Marco Tulio Aldana Priwillts

Guatemala, julio de 2023

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ANÁLISIS PARA LA CREACIÓN DE UNA GUÍA PARA LA APLICACIÓN DE BEHAVIOR DRIVEN DEVELOPMENT COMO PARTE DE LA METODOLOGÍA EN LOS CURSOS DEL ÁREA DE DESARROLLO DE SOFTWARE DE INGENIERÍA EN CIENCIAS Y SISTEMAS

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

MARLON ABRAHAM FUENTES ZÁRATE
ASESORADO POR EL ING. MARCO TULIO ALDANA PRIWILLTS

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, JULIO DE 2023

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO a.i.	Ing. José Francisco Gómez Rivera
VOCAL II	Ing. Mario Renato Escobedo Martínez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Ing. Kevin Vladimir Armando Cruz
VOCAL V	Br. Fernando José Paz González
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

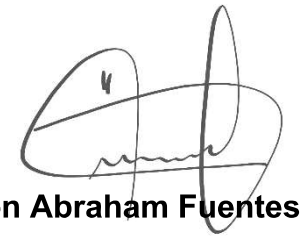
DECANA	Inga. Aurelia Anabela Cordova Estrada
EXAMINADOR	Ing. Marlon Francisco Orellana López
EXAMINADOR	Ing. César Augusto Fernández Cáceres
EXAMINADOR	Ing. Luis Fernando Espino Barrios
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

ANÁLISIS PARA LA CREACIÓN DE UNA GUÍA PARA LA APLICACIÓN DE BEHAVIOR DRIVEN DEVELOPMENT COMO PARTE DE LA METODOLOGÍA EN LOS CURSOS DEL ÁREA DE DESARROLLO DE SOFTWARE DE INGENIERÍA EN CIENCIAS Y SISTEMAS

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha 15 de marzo de 2022.



Marlon Abraham Fuentes Zárate

Guatemala, 7 de junio de 2023

Ingeniero
Carlos Alfredo Azurdia
Coordinador de Privados y Trabajos de Tesis
Escuela de Ingeniería en Ciencias y Sistemas
Facultad de Ingeniería - USAC

Respetable Ingeniero Azurdia:

Por este medio hago de su conocimiento que en mi rol de asesor del trabajo de investigación realizado por el estudiante **MARLON ABRAHAM FUENTES ZÁRATE** con carné **199911132 y CUI 2624 92571 0804** titulado **"ANÁLISIS PARA LA CREACIÓN DE UNA GUÍA PARA LA APLICACIÓN DE BEHAVIOR DRIVEN DEVELOPMENT COMO PARTE DE LA METODOLOGÍA EN LOS CURSOS DEL ÁREA DE DESARROLLO DE SOFTWARE DE INGENIERÍA EN CIENCIAS Y SISTEMAS",** luego de corroborar que el mismo se encuentra finalizado, lo he revisado y doy fé de que el mismo cumple con los objetivos propuestos en el respectivo protocolo, por consiguiente, procedo a la aprobación correspondiente.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Marco Tulio Aldana Prillwitz
Master in Bussiness Intelligence and Data Analytics
Colegio de Humanidades 30747

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala 11 de julio de 2023

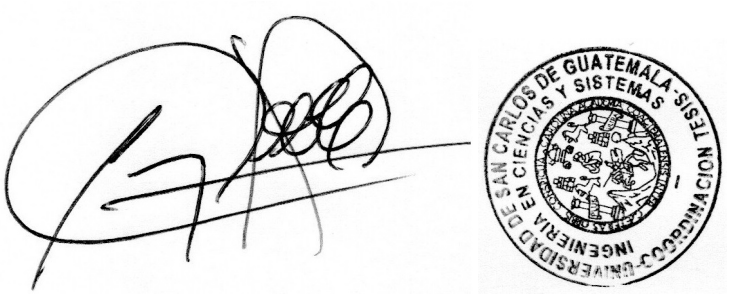
Ingeniero
Carlos Gustavo Alonzo
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Alonzo:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **MARLON ABRAHAM FUENTES ZÁRATE** con carné **199911132** y CUI **2624 92571 0804** titulado **"ANÁLISIS PARA LA CREACIÓN DE UNA GUÍA PARA LA APLICACIÓN DE BEHAVIOR DRIVEN DEVELOPMENT COMO PARTE DE LA METODOLOGÍA EN LOS CURSOS DEL ÁREA DE DESARROLLO DE SOFTWARE DE INGENIERÍA EN CIENCIAS Y SISTEMAS",** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo aprobado.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,



Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

SIST.LNG.DIRECTOR.6.EICCSS.2023

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del Asesor, el visto bueno del Coordinador de área y la aprobación del área de lingüística del trabajo de graduación titulado: **ANÁLISIS PARA LA CREACIÓN DE UNA GUÍA PARA LA APLICACIÓN DE BEHAVIOR DRIVEN DEVELOPMENT COMO PARTE DE LA METODOLOGÍA EN LOS CURSOS DEL ÁREA DE DESARROLLO DE SOFTWARE DE INGENIERÍA EN CIENCIAS Y SISTEMAS**, presentado por: **Marlon Abraham Fuentes Zarate**, procedo con el Aval del mismo, ya que cumple con los requisitos normados por la Facultad de Ingeniería.

"ID Y ENSEÑAD A TODOS"

Ingeniero Carlos Gustavo Alonzo
DIRECTOR
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, julio de 2023

Decanato
Facultad e Ingeniería

24189101- 24189102

LNG.DECANATO.OIE.41.2023

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería En Ciencias Y Sistemas, al Trabajo de Graduación titulado: **ANÁLISIS PARA LA CREACIÓN DE UNA GUÍA PARA LA APLICACIÓN DE BEHAVIOR DRIVEN DEVELOPMENT COMO PARTE DE LA METODOLOGÍA EN LOS CURSOS DEL ÁREA DE DESARROLLO DE SOFTWARE DE INGENIERÍA EN CIENCIAS Y SISTEMAS**, presentado por: **Marlon Abraham Fuentes Zarate** después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Firmado electrónicamente por: José Francisco Goméz Rivera
Motivo: Orden de impresión
Fecha: 24/07/2023 12:39:53
Lugar: Facultad de Ingeniería, USAC.

Ing. José Francisco Gómez Rivera
Decano a.i.

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO a.i.
Facultad de Ingeniería

Guatemala, julio de 2023

ACTO QUE DEDICO A:

Dios	Quien siempre está a mi cuidado.
Mis padres	Milton Fuentes (q. e. p. d.) y Anadély Zárate de Fuentes. Mi ejemplo de vida, amor y superación.
Mi esposa	María Isabel Gálvez. Mi complemento y soporte perfecto.
Mis hijos	Julián, María Emilia y Ana Isabel Fuentes. La mayor bendición que he recibido.
Mis hermanos	Cesar, Alicia y Milton Fuentes. Sin ellos nada sería igual.
Mis tías	Rosalina y Olimpia Zárate. Su amor y compañía nunca me ha faltado.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por abrirme sus aulas y permitir que mi superación pudiera ser realidad.
Facultad de Ingeniería	Por los esfuerzos que a diario se realizan para la formación de ingenieros capaces y de excelente calidad.
Ing. Marco Tulio Aldana	Por su apoyo desinteresado y guía para el final de esta etapa.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

LISTA DE SÍMBOLOS

Símbolo	Significado
$	Dólar
Km	Kilómetro
Mb	Megabyte
Q	Quetzal

GLOSARIO

Agile Es un concepto de gestión de proyectos desarrollado para proporcionar a los gestores de proyectos una forma más flexible y eficiente de comercializar productos más rápido**.**

Automatizar Aplicar procedimientos automáticos en la realización de un proceso.

BDD *Behavior Driven Development*, en español, desarrollo guiado por comportamiento.

Cucumber Es una herramienta de *software* que admite el desarrollo basado en el comportamiento.

DDD *Domain Driven Design*, en español, diseño guiado por dominio.

Escenario Un escenario describe un ejemplo de comportamiento del sistema.

Java Es un lenguaje de programación ampliamente utilizado para codificar aplicaciones *web.*

JBehave	Es un marco de BDD basado en Java de código abierto para escribir pruebas en un estilo de lenguaje natural.
Funcionalidad	La capacidad de un dispositivo o programa de ordenador de llevar a cabo una determinada tarea.
Gherkin	Es el lenguaje que entiende Cucumber, es un DSL legible para gente no técnica, que permite definir el comportamiento del *software* sin detallar cómo está implementado.
Prueba Unitaria	Consisten en verificar el comportamiento de las unidades más pequeñas de su aplicación.
Refactorizar	Es el proceso de modificar el código de un desarrollo para mejorar su estructura interna sin alterar la funcionalidad que ofrece.
Requerimiento	Son las necesidades que se requiere que el Sistema deba de cumplir de manera satisfactoria.
RoI	Son las necesidades que se requiere que el Sistema deba de cumplir de manera satisfactoria.
SpecFlow	Es una herramienta para .NET que nos permite crear nuestras pruebas como una serie de pasos escritos de forma que sean entendidos por cualquier persona.

Stakeholders	Son aquellos individuos o grupos que tienen interés e impacto en una organización y en los resultados de sus acciones.
TDD	*Test Driven Development*, en español, desarrollo guiado por pruebas.

RESUMEN

Actualmente el desarrollo orientado al comportamiento es una técnica de desarrollo muy útil cuando queremos garantizar código de calidad y además representa un cambio en el enfoque del desarrollo que realizamos dando un paso más allá de pensar únicamente en clases o funciones, sino de garantizar que se cumple con la funcionalidad que el usuario espera

Aplicado a los cursos del área de desarrollo de la carrera de Ingeniería en Ciencias y Sistemas podría tener un impacto positivo en los siguientes aspectos:

- Definición clara del comportamiento esperado de los proyectos
- Mejor definición de plan de trabajo de parte de estudiantes
- Definición clara de alcance de funcionalidades
- Mejor definición de pruebas del *software*
- Visibilidad de la correcta aplicación de conocimientos del curso
- Adopción de buenas prácticas como habilidades del estudiante

Esta investigación pretende analizar la posible definición de los proyectos de los cursos como productos de negocio que se construyen mediante la aplicación de los conocimientos adquiridos en los cursos. Se analizarán diferentes escenarios donde se utiliza y se determinará la ventaja que representa su correcta aplicación y su posible adaptación a la enseñanza y finalmente se creará una guía para la aplicación de BDD a los cursos del área de desarrollo de *software* de Ingeniería en Ciencias y Sistemas

OBJETIVOS

General

Analizar y proponer una guía para el uso de *Behavior Driven Development* (BDD) como parte de la metodología de desarrollo en los cursos del área de desarrollo de *software* de Ingeniería en Ciencias y Sistemas.

Específicos

1. Realizar un análisis sobre la combinación a nivel pedagógico y tecnológico de la técnica BDD como parte de la metodología de enseñanza en los laboratorios de los cursos del área de desarrollo de *software* de Ingeniería en Ciencias y Sistemas.

2. Proporcionar una herramienta para una mejor comprensión entre estudiantes y tutores académicos sobre lo que se espera como resultado en los proyectos que se realizan en los laboratorios de los cursos del área de desarrollo de *software* de Ingeniería en Ciencias y Sistemas.

.

INTRODUCCIÓN

El desarrollo guiado por el comportamiento (BDD) es un conjunto de prácticas de ingeniería de *software* diseñado para ayudar a los equipos a crear y entregar *software* más valioso y de mayor calidad más rápido.

Se basa en prácticas ajustadas que incluyen, en particular, el desarrollo basado en pruebas (TDD) y diseño dirigido por dominio (DDD). Pero lo más importante es que BDD proporciona un lenguaje común basado en oraciones simples y estructuradas expresadas en lenguaje natural, que facilitan la comunicación entre todos los miembros de un equipo y las partes interesadas del negocio.

Unos de los aspectos más importantes de BDD es que crea un ambiente colaborativo en el cual pueden participar dueños de negocio, administradores de proyecto, equipos de desarrollo y equipos de control de calidad, con el fin de compartir no sólo las necesidades del negocio, sino también, las posibles formas de solucionarlas.

Entendiendo los beneficios que BDD proporciona, realizaremos un análisis sobre su uso como parte de las actividades de los cursos de desarrollo de *software*, con el fin de establecer las ventajas que puede brindar para conseguir mejores resultados dentro del proceso de enseñanza.

1. BEHAVIOUR DRIVEN DEVELOPMENT

Para comprender mejor las motivaciones y la filosofía que impulsan las prácticas de BDD, es útil para entender de dónde viene BDD

1.1. Origen de BDD

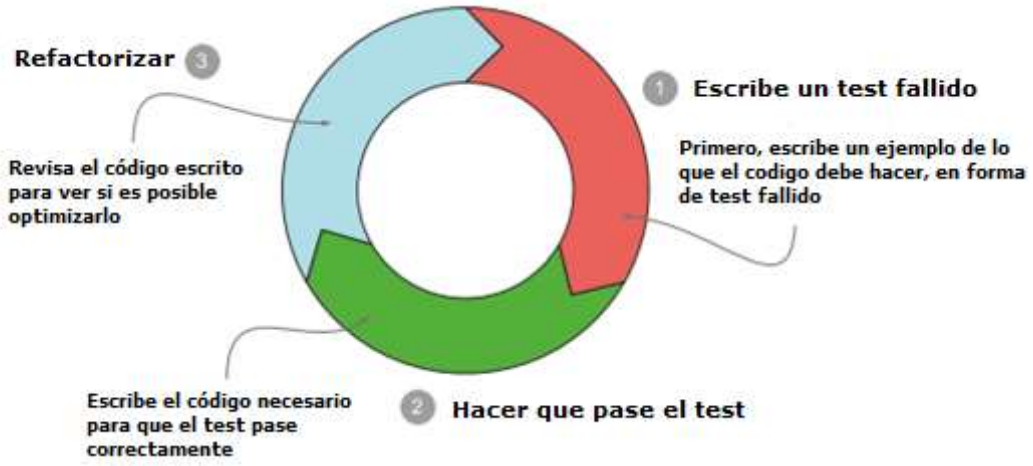
BDD fue inventado originalmente por Dan North a principios o mediados de la década de 2000 como una forma más fácil para enseñar y practicar el Desarrollo Dirigido por Pruebas (TDD). TDD, inventado por Kent Beck en los primeros días de Agile, es una técnica notablemente eficaz que utiliza pruebas unitarias para especificar, diseñar y verificar el código de la aplicación.

Cuando los profesionales de TDD necesitan implementar una función, primero escriben una prueba fallida que describe o especifica esta característica. A continuación, escriben el código suficiente para hacer que el pase de prueba Finalmente, refactorizar el código para ayudar a garantizar que sea fácil de mantener.

Esta técnica simple pero poderosa alienta a los desarrolladores a escribir código más limpio, mejor diseñado y más fácil de mantener y da como resultado una disminución en el recuento de defectos.

Figura 1.

Ciclos de TDD (test driven development)



Nota. Ciclo de *Test Driven Development*. Obtenido de J. Ferguson (2015). *BDD en acción.* (p. 13.) Manning Publications Co.

A pesar de sus ventajas, muchos equipos todavía tienen dificultades para adoptar y usar TDD efectivamente. Los desarrolladores a menudo tienen problemas para saber por dónde empezar o qué pruebas debe escribir a continuación. A veces, TDD puede hacer que los desarrolladores se centren demasiado en los detalles, perdiendo la imagen más amplia de los objetivos comerciales que se supone que deben implementar. Algunos equipos también encuentran que la gran cantidad de pruebas unitarias puede volverse difícil de mantener, ya que el proyecto crece en tamaño.

De hecho, muchas pruebas unitarias tradicionales, escritas con o sin TDD, están estrechamente acopladas a una implementación particular del código. Se centran en el método o función que están probando, en lugar de lo que debería hacer el código en términos comerciales.

Por ejemplo: supongamos que Paul es un desarrollador de Java que trabaja en una nueva aplicación de comercio financiero en un gran banco. Se le ha pedido que implemente una nueva función para transferir dinero de una cuenta a otra. Crea una clase de Cuenta con un método transferir() y un método depositar(), y así sucesivamente. Las pruebas unitarias correspondientes se centran en probar estos métodos:

Figura 2.

Ejemplo de pruebas unitarias

```
public class CuentaBancoTest {
        @Test
        public void testTransferir() {...}
        @Test
        public void testDepositar() {...}
}
```

Nota. Ejemplo de pruebas unitarias para transferir dinero entre cuentas. Elaboración propia, realizado con Canva.

Las pruebas como esta son mejores que nada, pero pueden limitar sus opciones. Por ejemplo, no describen lo que espera que hagan las funciones transferir() y depositar(), lo que las hace más difíciles de entender y arreglar si se rompen. Están estrechamente relacionados con el método que prueban, lo que significa que, si refactoriza la implementación, también debe cambiar el nombre de su prueba. Y porque no dicen mucho sobre lo que en realidad están probando, es difícil saber qué otras pruebas (si las hay) necesita escribir antes de que termines.

North observó que algunas prácticas simples, como nombrar pruebas unitarias como oraciones completas y usar la palabra debería, pueden ayudar a los desarrolladores a escribir pruebas más significativas, lo que a su vez les ayuda a escribir código de mayor calidad de manera más eficiente. Cuando piensas en términos de lo que debe hacer la clase, en lugar de qué método o función se está probando, es más fácil mantener sus esfuerzos enfocados en los requisitos comerciales subyacentes.

Por ejemplo, Paul podría escribir más pruebas descriptivas en las siguientes líneas como se muestra en la figura 3:

Figura 3.

Ejemplo de uso de nombres descriptivos para métodos

```
public class TransferenciadeFondos {
|
        @Test public void Debe_transferir_a_una_cuenta() {...}
        @Test public void debe_transferir_a_una_cuenta_diferente() {...} ...
        @Test public void debe_cobrar_una_comision() {...} ...
}
```

Nota. Ejemplo de uso de nombres descriptivos para pruebas unitarias de aplicación para transferir dinero entre cuentas. Elaboración propia, realizado con Canva.

Las pruebas que están escritas de esta manera se leen más como especificaciones que como pruebas unitarias. Se enfocan sobre el comportamiento de la aplicación, usando pruebas simplemente como un medio para expresar y verificar ese comportamiento. North también señaló que las pruebas escritas de esta manera son mucho más fáciles de mantener porque su

intención es muy clara. El impacto de este enfoque fue tan significativo que empezó a referirse a lo que estaba haciendo ya no como desarrollo basado en pruebas, sino como Desarrollo guiado por el comportamiento.

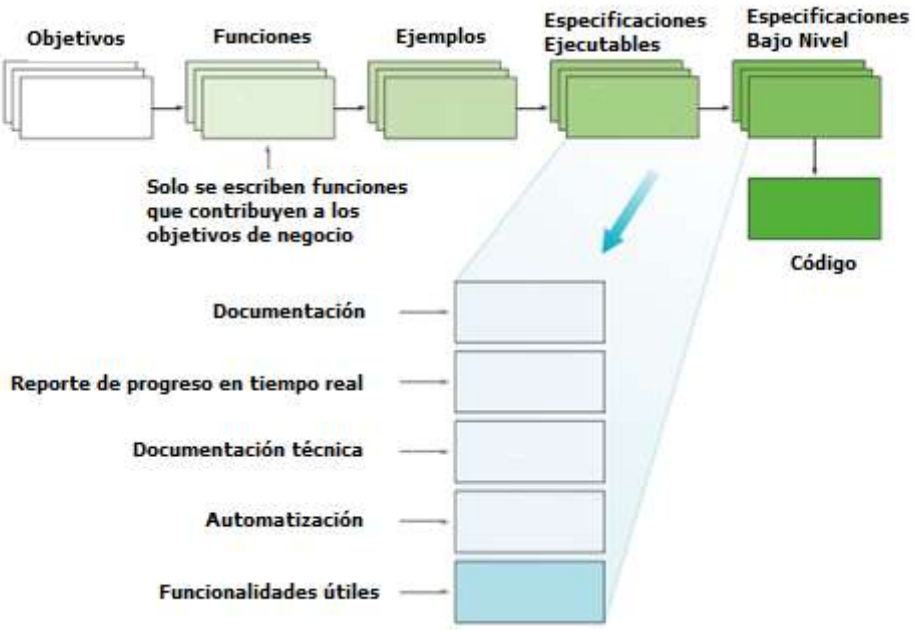
1.2. Principios y prácticas de BDD

Hoy en día, BDD se practica con éxito en una gran cantidad de organizaciones de todos los tamaños en todo el mundo, en una variedad de formas diferentes. La figura 4 ofrece una descripción general de alto nivel de la forma en que BDD ve el mundo. A los profesionales de BDD les gusta comenzar identificando objetivos comerciales y buscando características que los ayuden a entregar estos objetivos.

En colaboración con el usuario, utilizan ejemplos concretos para ilustrar estas funciones. Siempre que sea posible, estos ejemplos se automatizan en forma de especificaciones ejecutables, que validan el *software* y proporcionan automáticamente documentación técnica y funcional actualizada. Los principios de BDD también se utilizan en el nivel de codificación, donde ayudan a los desarrolladores a escribir código de mayor calidad, mejor probado, mejor documentado y más fácil de usar y mantener.

Figura 4.

Principales actividades BDD



Nota. Principales actividades BDD. Obtenido de J. Ferguson (2015). *BDD en Acción.* (p. 16.) Manning Publications Co.

1.3. Beneficio de BDD

A continuación, veamos cuales son los principales beneficios que el uso de BDD puede aportar a nuestros proyectos.

1.3.1. Residuos reducidos

BDD tiene que ver con centrar el esfuerzo de desarrollo en descubrir y entregar las funciones que proporcionarán valor empresarial y evitar las que no

lo hacen. Cuando un equipo construye una característica que no está alineada con los objetivos comerciales subyacentes del proyecto, el esfuerzo es desperdiciado para el negocio. De manera similar, cuando un equipo escribe una característica que la empresa necesita, pero de una manera que no es útil para la empresa, el equipo deberá volver a trabajar la característica para ajustarse a la factura, lo que resulta en más desperdicio. BDD ayuda a evitar este tipo de desperdicio esfuerzo ayudando a los equipos a centrarse en características que están alineadas con los objetivos comerciales. BDD también reduce el esfuerzo desperdiciado al permitir comentarios más rápidos y útiles para los usuarios. Esto ayuda a los equipos a realizar cambios más temprano que tarde.

1.3.2. Costos reducidos

La consecuencia directa de esta reducción de residuos es la reducción de costes. Al enfocarse en construir características con valor comercial demostrable (construir el *software* correcto), y al no desperdiciar esfuerzos en características de poco valor, puede reducir el costo de entregar un producto viable a sus usuarios. Y mejorando la calidad del código de la aplicación (construir el *software* correctamente), reduce la cantidad de errores y, por lo tanto, el costo de corregir estos errores, así como el costo asociado con las demoras que estos errores causarían.

1.3.3. Cambios más fáciles y seguros

BDD facilita considerablemente el cambio y la ampliación de sus aplicaciones. La documentación se genera a partir de las especificaciones ejecutables utilizando términos con los que las partes interesadas están familiarizadas. Esto hace que sea mucho más fácil para las partes interesadas entender lo que la aplicación realmente hace. Las especificaciones ejecutables

de bajo nivel también actúan como documentación técnica para los desarrolladores, facilitándoles la comprensión de la base de código existente y hacer sus propios cambios.

Por último, pero no menos importante, las prácticas de BDD producen un conjunto integral de pruebas unitarias y de aceptación automatizadas, lo que reduce el riesgo de regresiones causadas por nuevos cambios en la aplicación.

1.3.4. Lanzamientos más rápidos

Estas exhaustivas pruebas automatizadas también aceleran considerablemente el ciclo de lanzamiento. Ya no se requiere que los probadores realicen largas sesiones de prueba manual antes de cada nuevo lanzamiento. En su lugar, pueden utilizar las pruebas de aceptación automatizadas como punto de partida, y pasar su tiempo de manera más productiva y eficiente en pruebas exploratorias y otras pruebas manuales no triviales.

1.4. Retos de BDD

Si bien sus beneficios son significativos, introducir BDD en una organización no siempre resulta sin dificultades. En esta sección, veremos algunas situaciones en las que la introducción BDD puede ser más un desafío.

1.4.1. BDD requiere un alto compromiso y colaboración comercial

Las prácticas de BDD se basan en conversaciones y comentarios. De hecho, estas conversaciones impulsan y construyen la comprensión del equipo de los requisitos y de cómo pueden entregar valor comercial basado en estos

requisitos. Si las partes interesadas no están dispuestas o son incapaces de entablar conversaciones y colaboración, o esperan hasta el final del proyecto antes de dar cualquier comentario, será difícil obtener todos los beneficios de BDD.

1.4.2. BDD funciona mejor en un contexto ágil o iterativo

Las prácticas de análisis de requisitos de BDD asumen que es difícil, sino imposible, definir los requisitos completamente por adelantado, y que estos evolucionarán a medida que el equipo (y las partes interesadas) aprenden más sobre el proyecto. Este enfoque es naturalmente más en línea con una metodología de proyecto Agile o iterativa.

1.4.3. BDD no funciona bien en un silo

En muchas organizaciones más grandes, un enfoque de desarrollo en silos sigue siendo la norma.

Las especificaciones detalladas son escritas por analistas de negocios y luego entregadas a equipos de desarrollo que a menudo están fuera del sitio o en el extranjero. Del mismo modo, las pruebas se delegan en otro equipo de control de calidad totalmente separado. En organizaciones como esta, aún es posible practicar BDD a nivel de codificación, y los equipos de desarrollo aún podrán esperar aumentos significativos en la calidad del código, mejor diseño, más código mantenible y menos defectos. Pero la falta de interacción entre los equipos de analistas de negocios y los desarrolladores dificultará el uso de las prácticas de BDD para aclarar y comprender progresivamente los requisitos reales.

Del mismo modo, los equipos de prueba aislados pueden ser un desafío. Si el equipo de control de calidad espera hasta el final del proyecto para intervenir, o lo hace de forma aislada, perderán la oportunidad de contribuir a los requisitos antes, lo que da como resultado un esfuerzo desperdiciado para arreglar problemas que podrían haberse encontrado antes y solucionado más fácilmente. Automatizar los criterios de aceptación también es mucho más beneficioso si el equipo de control de calidad participa en la definición y, posiblemente, en la automatización de los escenarios.

1.4.4. Las pruebas mal escritas pueden conducir a mayores costos de mantenimiento de pruebas

Crear pruebas de aceptación automatizadas, particularmente para aplicaciones web complejas, requiere una cierta habilidad, y muchos equipos que comienzan a usar BDD consideran que esto es un desafío importante. De hecho, si las pruebas no están cuidadosamente diseñadas, con los niveles adecuados de abstracción y expresividad, corren el riesgo de ser frágiles. Y si hay un gran número de pruebas mal escritas, sin duda serán difíciles de mantener. Muchas organizaciones han implementado con éxito pruebas de aceptación automatizadas para aplicaciones web complejas, pero se necesita conocimiento y experiencia para hacerlo bien.

2. IDENTIFICACIÓN DE OBJETIVOS DE NEGOCIO

Identificar de forma correcta los objetivos de negocio es un aspecto de vital importancia y para el éxito de un proyecto, para lograrlo haremos uso de las siguientes técnicas que nos ayudarán durante este proceso.

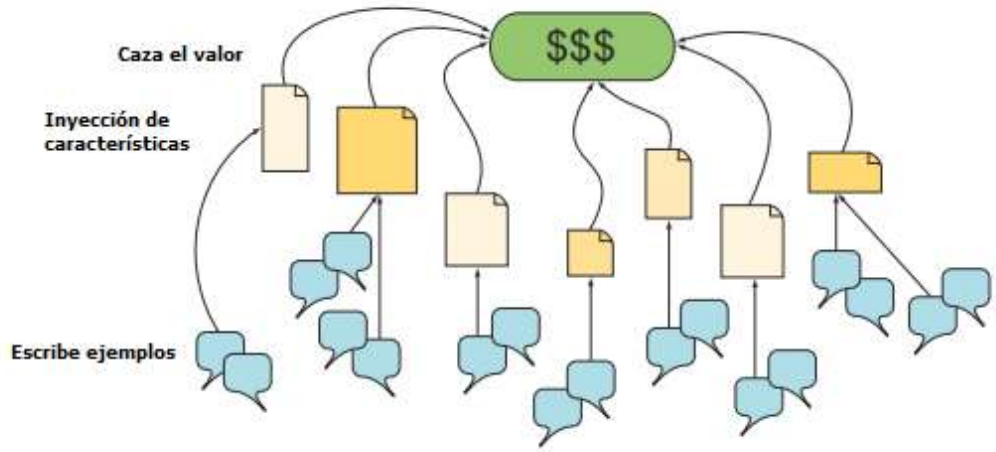
2.1. Inyección de características

El enfoque general que discutiremos en este capítulo se llama inyección de características. La inyección de características es un enfoque que trata de identificar el valor que un proyecto pretende entregar y las características que podrán entregar este valor. Inicialmente elaborado por Chris Matts, y desde entonces defendido por Liz Keogh y otros miembros de la comunidad BDD, inyección de características destila muchas técnicas y enfoques que los practicantes de BDD aplican y han encontrado útiles en la naturaleza. Inyección de Características proporciona a los equipos un marco que ayuda a centrar los esfuerzos de BDD en aquellas funciones que ofrecerán valor de negocio.

El objetivo de Inyección de características es desarrollar el conjunto mínimo de funciones que proporcione el mayor beneficio a las partes interesadas en términos de lograr sus objetivos comerciales. La inyección de características enfatiza la importancia de participar en una conversación en curso con las partes interesadas para explorar, elaborar y ampliar progresivamente un entendimiento compartido de lo que se necesita entregar y por qué. Gira en torno a un proceso simple de tres pasos.

Figura 5.

Proceso de inyección de características



Nota. Proceso de inyección de características. Obtenido de J. Ferguson (2015). *BDD en Acción.* (p. 64.) Manning Publications Co.

- Caza el valor

Muchos equipos ágiles crean una gran lista de historias de usuarios (a menudo denominada *backlog*), y luego priorizar las historias para decidir qué características realmente necesitan construir en la próxima iteración. Pero filtrar las historias para encontrar las de mayor valor lleva tiempo y esfuerzo, especialmente a medida que la lista de historias crece. ¿No sería más fácil si las historias? ¿Realmente no necesitas que no te colocaran en la lista en primer lugar?

La inyección de características hace las cosas al revés. En inyección de características, comienza tratando de aclarar exactamente cómo espera que el

sistema entregue valor comercial, y solo entonces identifica las características que podrían ofrecer mejor este resultado. Esto hace que sea más fácil centrar su trabajo en las funciones que ofrecerán el mejor retorno de la inversión. Esto es lo que se llama cazar el valor. El objetivo es comprender el valor comercial que subyace a una función para que pueda decidir objetivamente qué funciones vale la pena crear.

Los equipos que usan Inyección de características usan una serie de técnicas que ayudan a modelar el valor relativo de las características en términos del valor comercial que pueden proporcionar. No tengas miedo de la palabra modelo: los mejores modelos son bastante simples. En *impact mapping*, por ejemplo, utiliza un enfoque simple de mapas mentales para identificar y comparar el valor relativo de las características, asignándolas a sus objetivos comerciales subyacentes.

El modelo de alineación basado en el propósito es otra técnica gráfica que puede ayudar a decidir cuánto esfuerzo se debe invertir en cada función. Los modelos hacen más fácil discutir los supuestos que subyacen a la propuesta de valor de cada característica, y evitan que las características de bajo valor pasen desapercibidas.

- Inyectar las características

Cómo ha visto, una vez que haya identificado los objetivos de su proyecto y cómo se espera ofrecer valor al negocio, puede decidir objetivamente qué características valen la pena, pero, puedes hacerlo mejor que eso. Un equipo que practica Inyección de características no sólo decide qué características en la cartera de productos valen la pena hacer; buscan proactivamente funciones que aporten valor comercial.

Uno de los inquilinos fundamentales de inyección de características es que el valor de una pieza del *software* generalmente proviene de las salidas. Por ejemplo, el valor de una solicitud proviene de la información que proporciona, las ventas que resultan de su uso, o los informes que genera que permiten a los usuarios tomar decisiones. Viene de lo financiero, transacciones que procesa o de los productos vendidos que envía. El valor se encuentra no en las entradas, sino en las salidas.

Cuando desee saber qué características ofrecerán el mayor valor o el mayor retorno de la inversión, debe comenzar con los productos o resultados. Toma las salidas que ofrecerán el mayor valor y se trabajará para encontrar el conjunto mínimo de características necesarias para producir estos resultados.

Por ejemplo, puede identificar las siguientes características que podrían contribuir al objetivo de aumentar los ingresos por ventas:

- Aumentar las ventas al permitir que los miembros de viajero frecuente compren vuelos con millas canjeadas.

- Aumentar las ventas al permitir que los viajeros reserven vuelos en línea.

Nuevamente, esto es bastante similar al enfoque de Mapeo de Impacto, donde comienza con un objetivo de negocio y vuelva a trabajar para descubrir las características que podrían lograr este objetivo.

- Encuentra los ejemplos

Cuando inyecta características de esta manera, solo obtiene una vista parcial de cómo una característica podría funcionar. Una descripción de función de alto nivel como comprar vuelos con millas canjeadas o reservar un vuelo en línea solo da una idea superficial de cómo debería comportarse una característica. Cuando se trata de implementar una característica de este tipo, obviamente se necesita saber más. Una vez que tenga un conjunto de características de alto nivel que desea implementar, debe ampliar su comprensión para que pueda atender a la variedad de entradas o comportamientos que pueden afectar los resultados. En otras palabras, necesita ampliar el alcance de las características que construirá hasta que esté satisfecho de que le brindarán el valor comercial esperado.

En Inyección de características, utiliza ejemplos de alto nivel para desarrollar este alcance faltante. Como verás, los ejemplos son una forma muy efectiva de comunicar ideas sobre el dominio de un problema y para descubrir y aclarar lo que no sabe. Los ejemplos son intuitivos y fáciles de entender y ayudan a eliminar suposiciones y malentendidos. Pueden ayudar a descubrir áreas que necesitan más investigación. También actúan como la base para un análisis más profundo, usando técnicas tradicionales de análisis de negocios y enfoques más colaborativos.

2.2. Identificar el beneficio al negocio

Una vez que tenga una idea clara de la visión del proyecto, necesita definir los objetivos comerciales que impulsan el proyecto y contribuyen a hacer realidad esta visión. El objetivo define sucintamente cómo el proyecto beneficiará a la organización o cómo alinearse con las estrategias o la vocación de la

organización. Todos los proyectos tienen objetivos comerciales; de lo contrario, la gerencia no habría aprobado ellos en primer lugar. Pero no todos los proyectos tienen objetivos de negocio claramente definidos y visibles. Y los proyectos con objetivos comerciales bien definidos y comunicados tienen muchas más posibilidades de éxito que aquellos que no lo hacen.

Comprender estos objetivos es aún más importante cuando surgen problemas imprevistos, cuando los desafíos técnicos hacen que la implementación de una solución en particular sea más difícil de lo que se pensó inicialmente, o cuando el equipo se da cuenta de que no entendieron los requisitos y hay que hacer las cosas de otra manera. Si se espera que los desarrolladores respondan apropiadamente a este tipo de desafío, necesitan tener una comprensión sólida de qué valor comercial se espera del sistema.

Al final del día, los empresarios quieren que el *software* se construya para ayudarlos a lograr sus objetivos comerciales. Si el *software* cumple en este sentido, la empresa va a considerarlo un éxito, incluso si el alcance y la implementación varían considerablemente de lo que originalmente se imaginó. Pero si el *software* no cumple con los objetivos de negocio subyacentes, entonces con razón se considerará un fracaso, incluso si cumple con los requisitos proporcionados por los clientes al pie de la letra.

2.3. Identificar a los *stakeholders*

Como hemos visto, todos los proyectos en última instancia tienen como objetivo beneficiar a la organización de alguna manera. Pero las organizaciones están formadas por personas, y serán las personas dentro de esta organización (o personas que interactúan con esta organización) que se verán afectados por los resultados de tu proyecto. Estas personas son los interesados o actores. Los

mapas de impacto que analizamos anteriormente enfatizan la importancia de las partes interesadas en la identificación de características valiosas.

Veamos esto con más detalle. Tenga en cuenta que incluso cuando el efecto general de un proyecto está diseñado para ser beneficioso para la organización, el impacto del proyecto a nivel individual puede ser negativo. Por ejemplo, agregar pasos adicionales en un proceso de solicitud de hipoteca puede percibirse como engorroso y molesto para el banquero que vende la hipoteca y para el cliente que solicita un préstamo hipotecario. Pero el efecto neto de reducir el riesgo de malos préstamos puede considerarse lo suficientemente beneficioso para el banco en su conjunto como para compensar estas desventajas. Es importante ser consciente de estos impactos negativos y tratar de minimizar ellos donde sea posible.

Muchos tipos diferentes de partes interesadas estarán interesados en los resultados de su trabajo. Los futuros usuarios de su producto son probablemente el ejemplo más obvio; su trabajo diario se verá afectado por su proyecto, para bien o para mal. Para el sitio web de los viajeros frecuentes, esta categoría de partes interesadas incluirá el viajero frecuente actual y futuro miembros, también puede incluir personal del centro de llamadas, que tendrá que responder preguntas de miembros de viajero frecuente sobre el nuevo programa y poder ver la información actual de un cliente, estado y millas. También pueden necesitar realizar tareas más complicadas que no son aún admitidas en el sitio web, como la transferencia de millas entre miembros de la familia o el abono de millas acumuladas de forma manual.

Tenga en cuenta que es posible que los futuros usuarios no siempre estén directamente interesados en los objetivos comerciales. A veces lo que el usuario quiere y pide tendrá poco o ninguna relación teniendo en cuenta los objetivos

comerciales originales. Cuando piensas en la característica que debe entregar para lograr los objetivos comerciales, debes pensar en términos de cómo puede motivar a estas partes interesadas a comportarse de una manera que le ayude a lograr los objetivos. Por ejemplo, querrá proporcionar funciones que alienten a los usuarios a reservar vuelos con usted en lugar de con un competidor para aumentar los ingresos por venta de boletos.

A menudo se pasa por alto el papel real de las diversas partes interesadas en el logro de los objetivos comerciales y la entrega de valor comercial. Si los usuarios no utilizan la aplicación en la forma esperada, es posible que el *software* no obtenga los beneficios que esperaba. Mirando las cosas desde otro ángulo, una forma muy efectiva de identificar las características que son más probable que respalden los objetivos comerciales y brinde valor real es pensar en términos de cambiar el comportamiento de las partes interesadas. ¿Cómo puede animar a los usuarios a comportarse de una manera que apoya los objetivos de negocio?

Una vez que haya identificado a las principales partes interesadas y entendido sus objetivos, puede pensar en las capacidades que necesita proporcionar para alcanzar estos objetivos.

2.4. Ordenando funcionalidades

La inyección de funciones puede ayudarlo a decidir qué funciones debe tener realmente para entregue el valor comercial que necesita entregar. Pero hay otra dimensión que también vale la pena considerar. No todas las características son iguales. Algunas características serán áreas de innovación, que requerirán conocimientos y experiencia en un dominio especializado y agregarán un valor significativo. Otros, como el pago en línea con tarjetas de crédito, pueden ser necesarios en un mercado, pero no distinguirán su producto de la competencia

de manera significativa. Saber dónde invertir su tiempo y esfuerzo es esencial para garantizar que su producto no solo proporcione valor, sino que también proporcione un alto retorno de la inversión. Una forma conveniente de medir esto es usar el modelo alineación basada en el propósito, inventado por Niel Nickolaisen (Pollyanna, Nickolaisen, Little & McDonald, 2009).

Usando el modelo de alineación basado en el propósito, clasificar las características en cuatro cuadrantes de un diagrama, como el de la figura 6, usando dos criterios simples:

- Misión crítica: ¿Qué tan esencial es la característica para su capacidad de operar como un negocio?

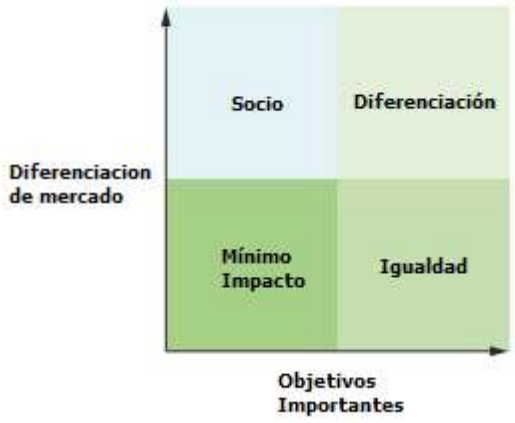
- Diferenciación de mercado: ¿Es significativamente diferente de lo que hacen sus competidores? ¿Te hará destacar en el mercado?

Dependiendo de dónde se ubique una característica en este diagrama, puede decidir cuál es la mejor centrar sus esfuerzos en aquellas características que marcarán la diferencia. Una característica caerá en una de cuatro categorías:

- Diferenciar
- Paridad
- Socio
- Impacto mínimo

Figura 6.

Modelo de alineación basado en el propósito



Nota. Modelo de Alineación Basado en el Propósito. Obtenido de J. Ferguson (2015). *BDD en Acción.* (p. 85.) Manning Publications Co.

- Características diferenciadoras

Una característica diferenciadora es crítica para la misión y marca su producto aparte de la competencia. Estas son las características que proporcionan un alto retorno de la inversión o que sirven para ganar cuota de mercado, o ambas cosas. Le dan a su organización una ventaja competitiva. Un ejemplo podría ser una herramienta de análisis de riesgo para un banco de inversión o una nueva forma innovadora de conciliar extractos de cuentas bancarias en un paquete de contabilidad en línea.

Estas características generalmente se basan en conocimientos y habilidades de dominio especializado, por lo que tiene sentido construirlos con

recursos internos. Las funciones de esta categoría deben ser altamente innovadoras para obtener y mantener una ventaja competitiva.

- Funciones de paridad

Una función de paridad es fundamental para la misión, pero no ofrece mucha diferenciación en el mercado. Estas son características que no son particularmente glamorosas pero que son necesarias para que el producto ser viable. Todos sus competidores ofrecen estas características y se espera que usted lo haga también. Un ejemplo de una función de paridad podría ser el pago en línea con tarjetas de crédito por un sitio de comercio electrónico o la capacidad de ver extractos de cuenta antiguos de un banco minorista en línea.

Como sugiere el nombre, debe implementar estas características para proporcionar aproximadamente el mismo nivel de servicio que se encuentra en los productos de la competencia. Si haces algo menos que eso, pondrá a su organización en desventaja competitiva. Pero sería un desperdicio gaste demasiado esfuerzo en estas características, ya que el retorno de la inversión será marginal.

- Funciones de socios

Las características de los socios no son críticas para la misión, pero aun así pueden diferenciar su producto de la competencia. Estas características no suelen formar parte de la experiencia principal de su organización, por lo que podría resultar ineficaz desarrollar esta experiencia para un proyecto. En este caso, tiene sentido asociarse con un socio que esté más especializado en esta área.

- Impacto mínimo

Algunas características no son críticas para la misión ni diferenciadoras del mercado. Deberías intentar dedicar el menor tiempo y esfuerzo posible a las características de esta categoría. Estas características de bajo valor podrían ser buenas candidatas para la subcontratación.

3. DEFINICIÓN DE REQUERIMIENTOS CON BDD

3.1. ¿Qué es un requerimiento?

En los proyectos Agile, los desarrolladores usan muchas palabras diferentes para describir lo que quieren construir. Capacidades, temas, requisitos, características, casos de uso, historias de usuario, tareas. ¿Confuso? Así es. La comunidad Agile ha hecho un trabajo relativamente pobre al definir estos conceptos de una manera clara y universalmente entendida.

Los términos que utilizamos pretenden simplificar la discusión sobre los requisitos del usuario. Esencialmente, estamos tratando de hacer dos cosas:

- Entregar valor tangible y visible al negocio a intervalos regulares
- Obtenga retroalimentación periódica para saber si vamos en la dirección correcta

La forma en que describimos y organizamos los requisitos debe respaldar estos objetivos. Las diferentes formas en que los diferentes equipos organizan y estructuran historias, epopeyas, características, entre otros. En adelante son simplemente formas de descomponer los requisitos de nivel superior en tamaños manejables, descríbalos en términos que los usuarios puedan entender y permítales proporcionar retroalimentación en cada nivel.

En aras de la simplicidad y la coherencia, nos ocuparemos principalmente de cuatro términos: capacidades, características, historias de usuarios y ejemplos.

Las capacidades brindan a los usuarios o partes interesadas la capacidad de realizar algún objetivo comercial o realizar alguna tarea útil. Una capacidad representa la habilidad de hacer algo; eso no depende de una implementación en particular. Por ejemplo, la capacidad de reservar un vuelo es una capacidad.

Las características representan la funcionalidad del *software* que crea para admitir estas capacidades. Reservar un vuelo en línea sería una característica.

Cuando crea y ofrece estas funciones, puede usar historias de usuarios para desglosar el trabajo en partes más manejables y para planificar y organizar su trabajo.

Puede usar ejemplos para comprender cómo las características ayudarán a sus usuarios y para guíe su trabajo en las historias de usuario. Puedes usar ejemplos para entender ambas características e historias de usuarios individuales.

3.2. Ilustrando requerimientos

Los ejemplos están en el corazón de BDD. En conversaciones con usuarios y partes interesadas, los profesionales de BDD usan ejemplos concretos para desarrollar su comprensión de las características y las historias de usuario de todos los tamaños, sino también para limpiar y aclarar áreas de incertidumbre.

Estos ejemplos, expresados en un lenguaje que las empresas pueden entender, ilustran cómo el *software* debe comportarse en términos muy precisos e inequívocos.

De acuerdo con las teorías de aprendizaje experimental de David Kolb, el aprendizaje efectivo es un proceso de cuatro etapas (Kolb, 1984).

En el modelo de Kolb, todos empezamos a aprender de experiencias concretas de algunas situaciones o eventos del mundo real (experiencia). Cuando observamos y pensamos en una experiencia (reflexión), analizamos y generalizamos este ejemplo, formando un modelo que representa nuestra comprensión actual del espacio del problema (conceptualizar).

Finalmente podemos probar este modelo mental contra otras experiencias del mundo real para verificar o invalidar todo o parte de nuestro entendimiento (prueba).

BDD utiliza un enfoque muy similar (ver la figura 7), donde los ejemplos y la conversación con los usuarios, las partes interesadas y los expertos en el dominio impulsan el proceso de aprendizaje.

- Usted discute ejemplos concretos de cómo debe comportarse una aplicación
- Reflexionar sobre estos ejemplos
- Construir una comprensión compartida de los requisitos
- Busca ejemplos adicionales para confirmar o ampliar su comprensión

Figura 7.

Teoría experimental de aprendizaje de Kolb

Nota. Modelo de Kolb. Obtenido de J. Ferguson (2015). *BDD en Acción.* (p. 101.) Manning Publications Co.

3.3. Requerimientos reales

A mediados de la década de 2000, Chris Matts identificó un principio fundamental que subyace a muchas prácticas ágiles: aplazar las decisiones hasta el último momento responsable, una idea que proviene del desarrollo de *software* eficiente. Llamó a este principio opciones reales. Comprender este principio cambia la forma de pensar acerca de muchas prácticas ágiles y abre la puerta a algunos nuevos.

En finanzas, una opción te da la posibilidad, pero no la obligación, de comprar un producto en algún momento en el futuro al precio de hoy. Por ejemplo, imagina que hay una alta probabilidad de que necesite comprar una gran cantidad de acero en los próximos tres meses, y que el precio del acero está actualmente al alza. No quieres comprar el acero ahora, porque no estás completamente

seguro de que lo necesitará; esperas estar seguro en algún momento de los próximos dos meses. Pero si espera unos meses más, el precio del acero podría haber subido, lo que significa que perderá dinero. Para salir de este acertijo, puede comprar una opción para comprar el acero en algún momento dentro de los próximos tres meses, al precio de hoy. Si el precio del acero sube, todavía se puede comprar al precio de hoy Y si el precio baja, o si no necesita el acero, puede elegir no usar la opción. Debe pagar por esta opción, pero solo cuesta una fracción del precio total del acero: vale la pena porque le permite no comprometerse en comprar el acero hasta que esté seguro de que lo necesita.

Este principio también se aplica en la vida cotidiana. Cuando compras un boleto de avión, estás en realidad comprando una opción para viajar: el boleto no lo obliga a viajar. Pero el precio que paga por esta opción varía. Imagina que tu aerolínea favorita está ofreciendo boletos por solo $ 600 para ir de Sydney a Wellington, pero estos boletos más baratos son no reembolsables si decide no viajar. No estás seguro de poder hacer el viaje, por lo que opta por un boleto más caro de $800, que tiene una tarifa de cancelación de $ 25. Veamos las matemáticas aquí. La opción de cancelar el vuelo te cuesta $ 200 extra. Si es bastante probable que viaje, esto podría ser mucho para pagar por una opción que es poco probable que vayas a utilizar, por lo que es posible que prefiera el billete más barato. Pero si crees que hay un 50 % de posibilidades que no podrás volar, es posible que esté feliz de pagar los $ 200 adicionales. Si puede cancelar, solo perderá $ 225 (los $ 200 adicionales, más la tarifa de cancelación de $ 25), mientras que, si cancela después de optar por el vuelo más barato, perderá $ 600.

Real options (opciones reales) es una aplicación de estos principios al desarrollo de *software* inventado por Chris Matts resumidos en tres simples puntos:

- Las opciones tienen valor.
- Las opciones caducan.
- Nunca se comprometa antes de tiempo a menos que sepa por qué.

4. ESCRIBIENDO ESCENARIOS

4.1. Creación de escenarios

Imagina que estás trabajando en la aplicación de viajeros frecuente para la empresa Flying High. El trabajo consiste en implementar una función que permitirá a los miembros del programa de viajero frecuente ganar puntos cuando vuelan.

Un documento de especificación de requisitos tradicional podría incluir algo como esto: Los miembros ganarán puntos de viajero frecuente de vuelos Flying High y de vuelos asociados.

Esto puede capturar la esencia de lo que necesita construir, pero es un poco vago. Cómo ¿Cuántos puntos debe ganar un miembro por vuelo? ¿Los miembros ganarán el mismo número de puntos en vuelos Flying High como en vuelos asociados? ¿Los vuelos en compañías asociadas deben reservarse a través de Flying High, o se aplica cualquier vuelo en una aerolínea asociada? ¿Los miembros ganarán más puntos si vuelan en Premium, Económico o Business? Y así.

Si deja estas preguntas sin responder ahora, el equipo de desarrollo tendrá que tomar decisiones y juicios sobre las soluciones más adecuadas. Que pueden tener que hacer preguntas adicionales durante el desarrollo, lo que ralentizará el proyecto mientras esperan respuestas. O pueden suponer incorrectamente que han entendido lo que se necesita e implementar una

solución que no se corresponde con lo que realmente necesita el negocio. En ambos casos, se pierde tiempo y esfuerzo.

Como hemos visto antes, discutir ejemplos concretos con los usuarios y partes interesadas es una excelente manera de aclarar y eliminar este tipo de ambigüedad, asegurándonos que todos están en la misma página. Usas el lenguaje y el vocabulario de tus *stakeholders* para aclarar aspectos que no te quedan claros y muchas veces descubres cosas que las partes interesadas no habían pensado originalmente, habían asumido que usted sabía o habían olvidado mencionar. Por ejemplo, Sarah, analista de negocios, y Paul, del equipo de marketing, podrían tener una conversación como la que muestra la figura 8 para profundizar en cómo se ganan los puntos de viajero frecuente:

Figura 8.

Conversación descriptiva con ejemplos

Sara: ¿Puedes darme un ejemplo? Cuantos puntos de viajero frecuente ¿Qué ganaría si volara de Sídney a Melbourne en Económico?

Paul: Bueno, la distancia de Sydney a Melbourne es de 878 km, y la base los puntos se calculan a medio punto por kilómetro, por lo que, si vuela con de Sydney a Melbourne en Económico, ganaría 439 puntos.

Sara: está bien. ¿Hay alguna forma de ganar más o menos puntos volando este viaje?

Paul: Bueno, si fueras un viajero frecuente Plateado, obtendrías un 50 % de bonificación también, por lo que ganaría 659 puntos.

Sarah: ¿Y si fuera un viajero frecuente Dorado?

Paul: En ese caso, ganaría un bono de estado del 75 %, pero también sería derecho a las Ganancias de Puntos Mínimas Garantizadas, que para Económico es 1000 puntos, entonces ganarías 1000 puntos.

Nota. Ejemplo de una conversación con ejemplos para describir comportamiento del sistema. Elaboración propia, realizado con Canva.

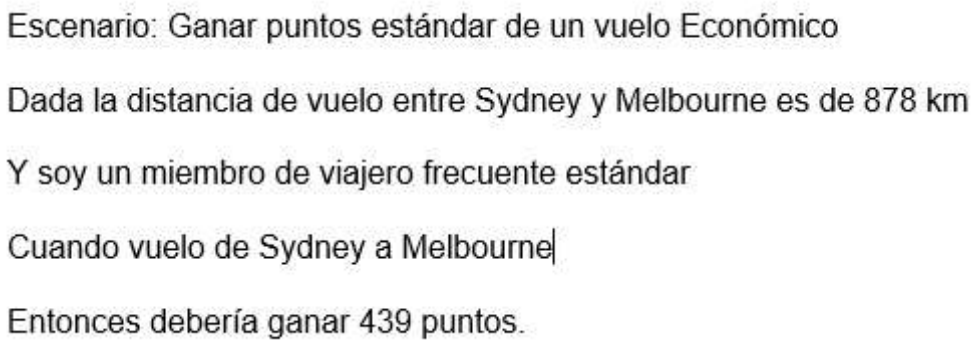
Al igual que con muchos requisitos del mundo real, las cosas se ponen más complicadas cuando empiezas a mirar en los detalles.

Uno de los conceptos centrales detrás de BDD es la idea de que puede expresar ejemplos concretos en un formato que sea legible para las partes interesadas y ejecutable como parte de su conjunto de pruebas automatizadas. Escribirá especificaciones ejecutables en el idioma nativo de sus usuarios, y producirá resultados de prueba que informen el éxito o el fracaso no en términos de clases y métodos, sino en términos de las características que los interesados solicitaron. Las partes interesadas podrán ver sus propias palabras aparecer en la documentación, lo que hace maravillas para aumentar su confianza en que ha entendido sus problemas. Esto es lo que aportan las herramientas BDD como Cucumber, JBehave y SpecFlow.

Cuando automatiza sus criterios de aceptación usando este tipo de herramienta BDD, usted expresa sus ejemplos en una forma un poco más estructurada, a menudo denominada escenarios. Dan North definió una forma canónica para estos escenarios a mediados de la década de 2000, construida en torno a una estructura simple dado ... cuando ... entonces, y este formato ha sido ampliamente adoptado por los practicantes de BDD desde entonces. Podría escribir el primer ejemplo que discutimos anteriormente como se muestra en la figura 9.

Figura 9.

Creando escenarios



Nota. Ejemplo de descripción de comportamiento del sistema utilizando escenarios. Elaboración propia, realizado con Canva.

Aunque el ejemplo es un poco más estilizado que un párrafo de texto libre, todavía estamos hablando el idioma de los interesados. Con un poco de práctica, las partes interesadas rápidamente van a familiarícese lo suficiente con el formato para poder proponer y discutir ejemplos como este.

Este tipo de escenario no solo es bastante legible, sino que también es ejecutable: las herramientas BDD como JBe have y Cucumber pueden leer y ejecutar estos escenarios para verificar el estado de su aplicación, comportamiento y generar informes de prueba significativos.

4.2. Escribiendo escenarios ejecutables

Los escenarios escritos en este formato constituirán el núcleo de sus especificaciones ejecutables. Pero para que sean realmente ejecutables, debe integrarlos en sus proyectos.

En esta sección, verá cómo hacer esto tanto en JBehave como en Gherkin. Gherkin es el lenguaje utilizado por Cucumber y la gran mayoría de las herramientas BDD basadas en Cucumber, incluidos SpecFlow (para .NET), Behave (para Python) y muchos otros. JBehave utiliza un formato muy similar.

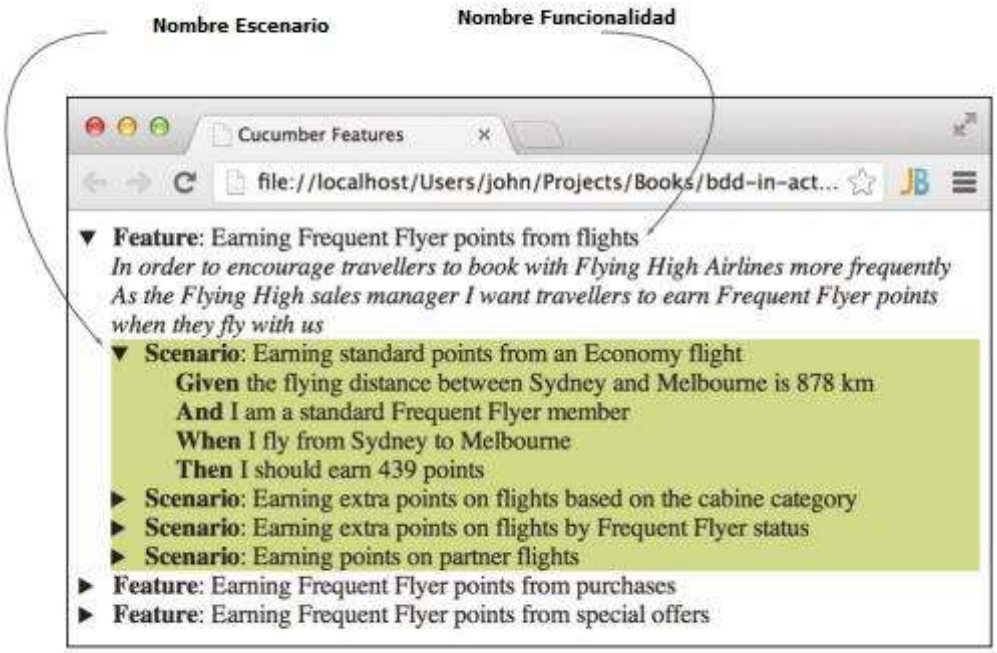
Los escenarios se almacenan en archivos de texto simples y se agrupan por características. Estos archivos son llamados, lógicamente, archivos de características. Tanto en Gherkin como en JBehave, un escenario comienza con la palabra clave escenario y un título descriptivo: escenario: <un título>

El título es importante. Como con la mayoría de las cosas en BDD, la buena comunicación es esencial. El título del escenario debe resumir lo que tiene de especial este ejemplo en una breve oración declarativa, un poco como un subtítulo para un libro. Debe enfatizar cómo difiere de los otros escenarios. Por ejemplo, diría: obtener puntos estándar de un vuelo económico o obtención de puntos extra en clase ejecutiva, en lugar de un miembro de viajero frecuente gana puntos estándar cuando vuela en clase económica".

Los títulos de los escenarios juegan un papel clave en los informes, haciendo de la documentación informes más fáciles de leer y navegar. Tener una lista sucinta de títulos de escenarios hace que sea más fácil de entender lo que se supone que debe hacer una función en particular, sin tener que estudiar los detalles del texto dado... cuando... entonces. También hace que sea más fácil aislar problemas cuando las pruebas se rompen. La figura 10 muestra un ejemplo de un informe de Cucumber que muestra las características y los encabezados de escenarios de esta manera.

Figura 10.

Ejemplo de un informe de Cucumber



Nota. Ejemplo de un informe de Cucumber. Obtenido de J. Ferguson (2015). *BDD en Acción.* (p. 121.) Manning Publications Co.

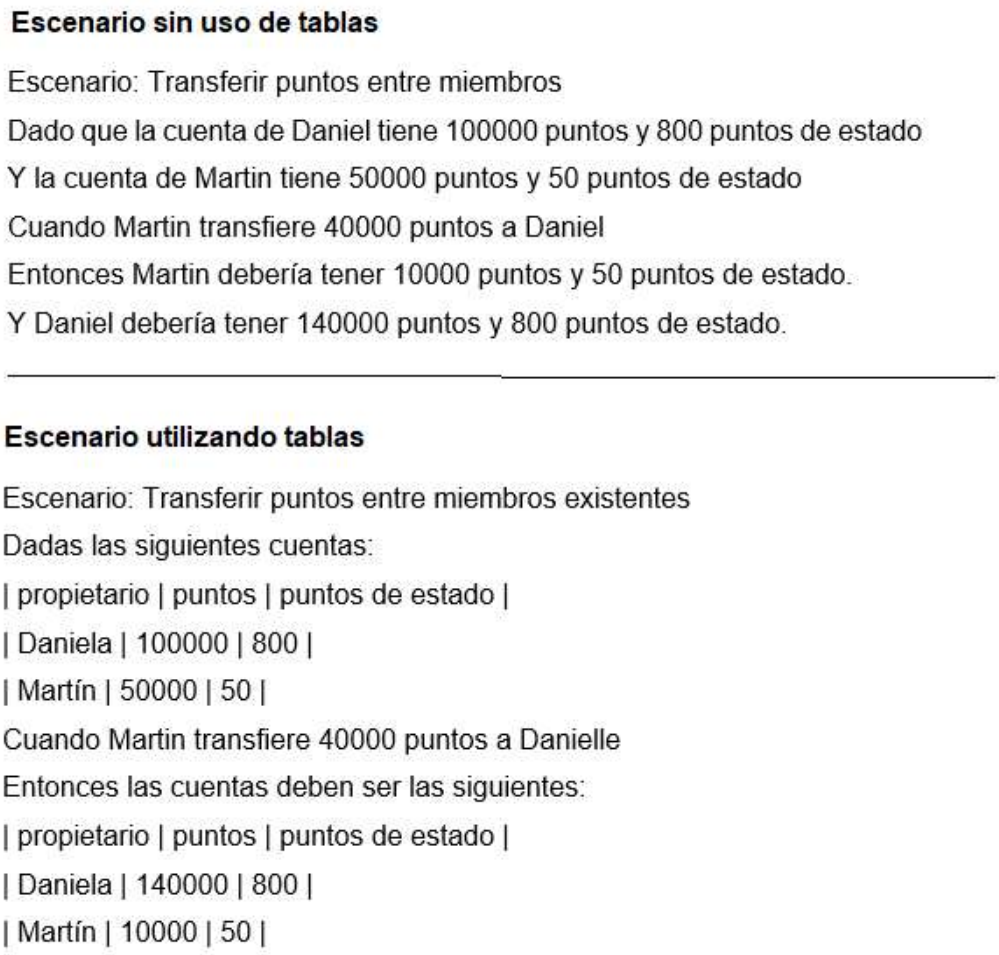
4.3. Uso de tablas en escenarios

Suponga que está trabajando en una función que permite a los miembros de viajero frecuente transferir puntos a otros miembros. Por ejemplo, suponga que Daniel y Martin son frecuentes miembros volantes. Tanto Daniel como Martin han acumulado muchos puntos en el año. Quieren irse de vacaciones juntos usando sus puntos, pero ninguno de los dos tiene suficientes puntos para comprar los boletos directamente. Martín necesita poder transferir algunos de sus

puntos a Daniel para que pueda comprar los vuelos para ambos con sus puntos. Podría expresar este escenario como se muestra en la figura 11:

Figura 11.

Uso de tablas en escenarios



Nota. Ejemplo de uso de tablas en la creación de escenarios. Elaboración propia, realizado con Canva.

El problema al no utilizar tablas es que hay mucha repetición y desorden, y el significado se pierde en todas las palabras. Una forma mucho mejor de escribir este escenario es expresar los datos en un formato tabular más conciso. Las tablas comienzan directamente después de los pasos Dado y Entonces los valores separados por conductos (|). Los encabezados en la parte superior de cada columna son útiles en este caso, pero son opcionales. Por ejemplo, en el siguiente paso, podría proporcionar una lista de valores:

Entonces debería poder ascender a una de las siguientes clases de cabina: | económica *premium* | negocios |

Incrustar datos tabulares es una excelente manera de expresar las condiciones previas y los resultados esperados de una manera clara y concisa.

4.4. Escenarios expresivos: patrones y anti-patrones

Comencemos con el paso dado. Este paso debe apuntar a que la aplicación esté en el estado apropiado tan pronto como sea posible. En este sentido, está bien tomar algunos atajos.

Por ejemplo, si está probando una aplicación web, debe intentar imitar al usuario experiencia y acciones, y hacer tanto como sea posible a través de la interfaz web, pero si tu necesitas configurar datos de prueba en la base de datos, está bien omitir la interfaz web y usar un servicio *backend* para actualizar la base de datos. El paso *dado*, como los demás, debe reflejar la intención comercial, o el qué, no la implementación técnica, o el cómo. Por ejemplo, considere lo siguiente:

- Dado que una cuenta de administrador está configurada en la base de datos y estoy registrado como administrador

Se exponen demasiados detalles sobre cómo se configura la prueba. Esto debería ser hecho en silencio, entre bastidores, tanto para evitar saturar el escenario con detalles innecesarios como para permitirle configurarlo como mejor le parezca. Todo lo que te interesa en este nivel es el contexto, que en este caso es que el usuario está conectado con el rol de administrador. La siguiente versión centra la atención en la condición previa en términos comerciales:

- Dado que estoy conectado como administrador

El paso *dado* también debe contener todas las condiciones previas o pasos que deben ocurrir antes de la acción que estás probando: ni más, ni menos. Por ejemplo, la figura 12 muestra un escenario un poco confuso:

Figura 12.

Escenarios confusos

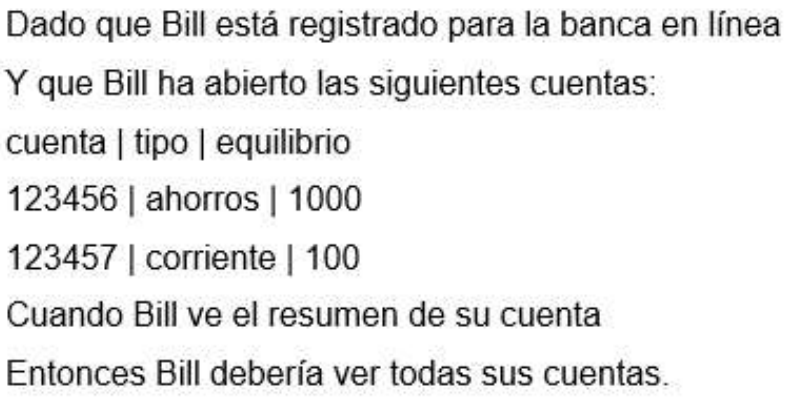
Dado que Bill se registra para la banca en línea Y que Bill abre las siguientes cuentas:
cuenta | tipo | equilibrio
123456 | ahorros | 1000
123457 | corriente | 100
Cuando Bill inicia sesión Y Bill va a la página de inicio. Y Bill ve sus cuentas
Entonces Bill debería ver una lista de sus cuentas:
cuenta | tipo | equilibrio
123456 | ahorros | 1000
123457 | corriente | 100

Nota. Ejemplo de un escenario confuso. Elaboración propia, realizado con Canva.

En este ejemplo, no está claro lo que está probando. ¿Estás comprobando que Bill puede iniciar sesión con éxito?, o está más interesado en las cuentas que puede ver? En el último caso, podría reescribir este escenario de la manera en que se muestra en la figura 13:

Figura 13.

Escenarios expresivos



Nota. Ejemplo de un escenario expresivo. Elaboración propia, realizado con Canva.

Las condiciones previas aquí son que Bill esté registrado para la banca en línea y que tenga dos cuentas. Si la autenticación se ha especificado en otro lugar, probablemente puede asumir con seguridad que Bill necesita iniciar sesión para ver sus cuentas. Las condiciones previas son también expresadas en tiempo pasado para que sea más obvio que se supone que estas acciones ya han ocurrido. La forma en que Bill llega a la página de resumen de cuentas tampoco es el enfoque principal de esta prueba, por lo que está oculto dentro del paso dado. La acción bajo prueba es ahora una sola línea que representa con mayor precisión la acción comercial que está especificando.

La cláusula *entonces* también se ha simplificado aquí asumiendo que puede reutilizar la lista de cuentas proporcionada en las condiciones previas. Esto reduce el desorden y se centra en la esencia del resultado esperado.

5. AUTOMATIZANDO ESCENARIOS

5.1. Introducción a la automatización de escenarios

Gran parte del valor de BDD proviene de las conversaciones sobre los escenarios. Por eso es tan importante colaborar para escribir los escenarios. No todos los escenarios necesitan ser automatizados; algunos pueden ser demasiado complicados para automatizar de manera rentable y pueden dejarse para la prueba manual. Otros pueden ser sólo de interés marginal para el negocio, y podría ser mejor implementarlo como pruebas unitarias o de integración. Aún otros pueden ser experimentales y pueden no entenderse lo suficientemente bien como para definir escenarios claros; en este caso, podría valer la pena hacer algunos prototipos iniciales para tener una mejor idea de lo que realmente se necesita.

Pero cuándo se puede automatizar un escenario, cuándo tiene sentido hacerlo y cuándo se hace bien, la automatización del escenario trae su propio conjunto de beneficios innegables:

- Los evaluadores dedican menos tiempo a las pruebas de regresión repetitivas. Cuando los criterios de aceptación y los escenarios correspondientes se escriben en estrecha colaboración con los probadores, las versiones automatizadas de estos escenarios brindan a los probadores más confianza en nuevos comunicados, los probadores pueden comprender y relacionarse más fácilmente con lo que verifican las pruebas automatizadas, porque participaron en su definición.

- Las nuevas versiones se pueden lanzar de forma más rápida y fiable. Porque menos pruebas manuales son necesarias, los nuevos lanzamientos se pueden impulsar de manera más eficiente.

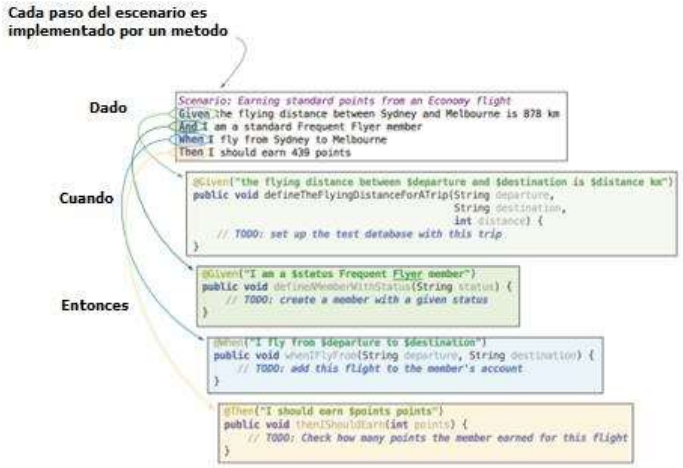
- Los escenarios automatizados dan una visión más precisa del estado actual del proyecto. Puede usarlos para crear un panel de progreso que describa qué características han sido entregados y cómo han sido probados, en base a los resultados de los escenarios automatizados.

5.2. Definición de pasos

Las definiciones de pasos son fragmentos de código que interpretan el texto en archivos de características y saben qué hacer para cada paso.

Figura 14.

Definición por pasos de un escenario



Nota. Definición por pasos de un escenario. Obtenido de J. Ferguson (2015). *BDD en Acción.* (p. 144.) Manning Publications Co.

Un método de definición de pasos es como el director de una orquesta: sabe a un alto nivel cómo realizar una tarea, y coordina las llamadas a otras bibliotecas más específicas de la aplicación que hacen el trabajo más detallado. Las definiciones de los pasos deben ser limpias, simples y descriptivas: deben describir lo que debe hacer con la aplicación o lo que debe preguntarle. Para cualquiera, incluso la más trivial de las aplicaciones, debe reagrupar el código que manipula la aplicación en una capa separada de las propias definiciones de los pasos. Una arquitectura típica implica un mínimo de tres capas:

- Los escenarios describen el requisito de alto nivel:

    - Escenario: Ganar puntos estándar de un vuelo económico
    - Dada la distancia de vuelo entre Sydney y Melbourne es de 878 km
    - Y soy un miembro de viajero frecuente estándar
    - Cuando vuelo de Sydney a Melbourne
    - Entonces debería ganar 439 puntos.

- Las definiciones de pasos interpretan los textos del escenario y llaman a la capa de automatización de pruebas para realizar las tareas reales:

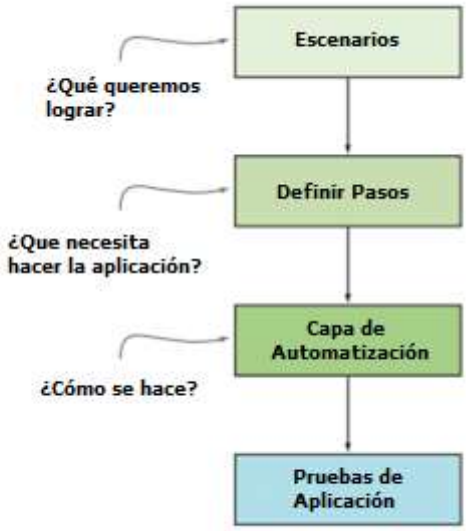
```
@Given("la distancia de vuelo entre $a y $b es $distancia km")
public void defineLaDistanciaDeViaje(...) {...}
```

- La capa de automatización de pruebas interactúa con la aplicación bajo prueba:

```
enBaseDatosPruebas.laDistanciaEntre(salida)
 . y(destino)
 .es(distancia);
```

Figura 15.

Arquitectura de definición por pasos de un escenario



Nota. Arquitectura de definición por pasos de un escenario para test automatizados. Obtenido de J. Ferguson (2015). *BDD en Acción.* (p. 146.) Manning Publications Co.

6. PRUEBAS UNITARIAS CON BDD

6.1. BDD, TDD, y pruebas unitarias

En este capítulo, aprenderá cómo los principios de BDD también pueden ayudar a los desarrolladores a escribir más código de bajo nivel enfocado, más efectivo, más mantenible y mejor documentado, y cómo las pruebas unitarias se pueden usar de manera muy efectiva para expresar, documentar y validar la especificación y el diseño de bajo nivel.

¿Pero no es eso TDD? Escucho preguntar. A menudo hay confusión sobre la distinción entre el desarrollo basado en el comportamiento y el desarrollo basado en pruebas (TDD). Muchos desarrolladores piensan en BDD como una técnica utilizada para pruebas de aceptación y uso TDD para referirse a actividades de prueba primero de nivel inferior que involucran pruebas unitarias. De hecho, las cosas no son tan claras, y las dos técnicas están profundamente entrelazadas.

6.1.1. BDD se trata de escribir especificaciones, no pruebas, en todos los niveles

Cómo ha visto, BDD implica descubrir y especificar el comportamiento de un sistema, pero este concepto funciona tan bien para toda la aplicación como para una clase individual: en ambos casos, estás especificando comportamiento. Hasta ahora hemos visto cómo los practicantes de BDD expresan requisitos de alto nivel en forma de especificaciones ejecutables.

Los requisitos de alto nivel se ocupan del comportamiento del sistema como un todo desde el punto de vista del negocio. Los requisitos de bajo nivel se ocupan del comportamiento de un componente, clase o API desde el punto de vista del desarrollador que trabaja con ellos. En ambos casos estas especificando el comportamiento, y en ambos casos puede describir este comportamiento en términos de especificaciones ejecutables. Solo cambia el público objetivo: los requisitos de alto nivel son típicamente dirigidos al equipo más amplio, mientras que los requisitos de bajo nivel y más técnicos están dirigidos a futuros desarrolladores que deberán comprender y mantener el código de la aplicación.

6.1.2. BDD se basa en prácticas establecidas de TDD

Las prácticas centrales que estamos viendo son esencialmente las de TDD, o están profundamente arraigadas en TDD. De hecho, los profesionales experimentados en TDD han estado usando estas técnicas durante mucho tiempo. En muchos aspectos, BDD ha formalizado la forma en que muchos practicantes exitosos de TDD han estado haciendo las cosas, enfatizando algunos aspectos clave de práctica avanzada de TDD que hace que la técnica sea más efectiva. Estas prácticas incluyen:

- Usar el desarrollo de afuera hacia adentro para asegurarse de que el código que escribe está entregando valor comercial real.

- Usar un lenguaje de dominio compartido (o ubicuo) para fomentar una colaboración y comprensión más estrechas dentro del equipo.

- Usar ejemplos para describir este comportamiento más claramente.

- Describir y especificar el comportamiento del sistema tanto a un nivel alto como a un nivel más detallado.

7. ANÁLISIS DE LA APLICACIÓN DE BDD EN CURSOS DEL ÁREA DE DESARROLLO DE SOFTWARE DE INGENIERÍA EN CIENCIAS Y SISTEMAS

7.1. Diferencias entre BDD y el método tradicional

A continuación, analizaremos tanto BDD como la forma tradicional de trabajo para poder comparar las diferencias principales entre ambos.

7.1.1. Método tradicional

Utilizando el método tradicional seguimos los pasos en el siguiente orden (ver figura 16):

- Realizar es la explicación de la necesidad que se desea cubrir. En el caso de un proyecto de laboratorio estas necesidades están en el contenido del curso que el estudiante debe aprender y dominar. Este contenido ya se encuentra definido en el programa del curso y del laboratorio

- El ingeniero y auxiliar en conjunto definen el proyecto que será asignado a los estudiantes y se crea el enunciado de proyecto, el cual contiene todos los aspectos que deben cubrirse y las restricciones que deben respetarse.

- El auxiliar entrega a los estudiantes el enunciado del proyecto para que lo lean y generalmente se realiza una reunión para explicarlo y para resolver las dudas que puedan existir

- El auxiliar define cuál será el método de calificación, las pruebas que se realizarán y también el valor en puntos que cada funcionalidad tendrá.

- Se realiza la calificación, en el caso de un proyecto no hay espacio para correcciones, las funcionalidades que no funcionan no son tomadas en cuenta o queda a criterio del auxiliar el punteo que pueda recibir según su percepción del avance.

Figura 16.

Método tradicional desarrollo de software

Nota. Gráfica que muestra el proceso general del proceso tradicional de desarrollo de software. Elaboración propia, realizado con Canva.

Un aspecto importante que debemos resaltar del método tradicional es que el estudiante queda fuera de la discusión sobre las necesidades del negocio, ejemplos, funcionamiento esperado y procesos de valor. Por lo tanto, es probable que existan muchas dudas y posibles soluciones que no estén ajustadas a los objetivos del curso.

También podemos ver que las pruebas de calificación no están en conocimiento del estudiante durante el proceso de desarrollo lo que puede ocasionar atrasos por falta de entendimiento y que las pruebas realizadas por el estudiante no están precisamente orientadas a cumplir con los objetivos del proyecto.

7.1.2. Método BDD

Utilizando el método dirigido por el comportamiento seguimos los pasos en el siguiente orden (ver figura 17):

- En el primer paso se realiza la explicación de la necesidad que se desea cubrir. En el caso de un proyecto de laboratorio estas necesidades están en el contenido del curso. Este contenido ya se encuentra definido en el programa del curso y del laboratorio.

- El ingeniero y auxiliar en conjunto definen el proyecto que será asignado a los estudiantes y se crea el enunciado de proyecto, las funcionalidades son definidas en forma de escenarios, presentando mediante ejemplos el comportamiento esperado de cada funcionalidad.

- El auxiliar entrega a los estudiantes el enunciado del proyecto para que lo lean y se realizan reuniones para analizar el proyecto que se desea

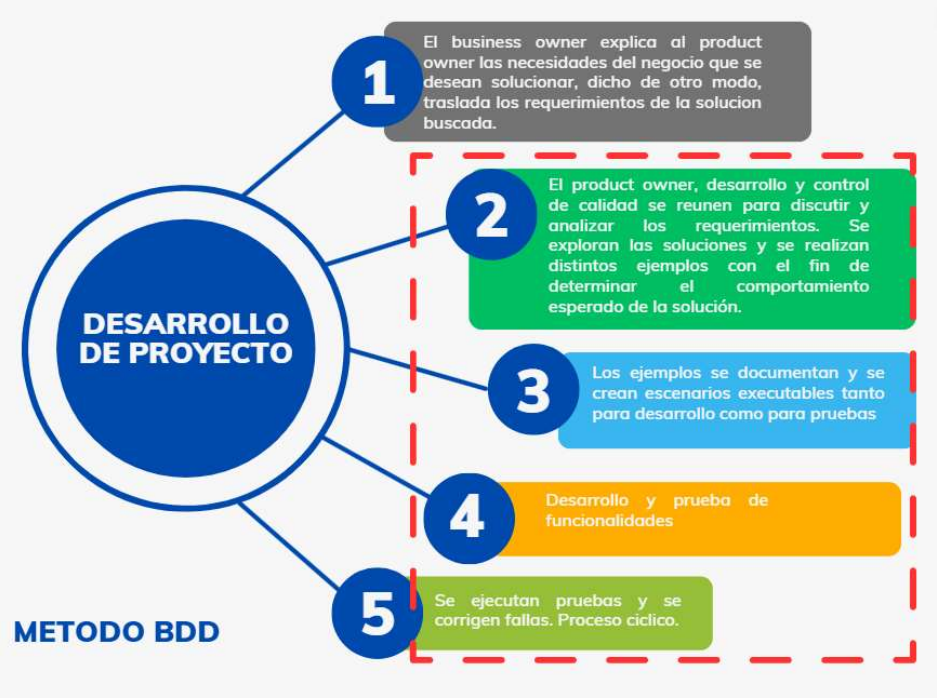
implementar y para poder crear y documentar más escenarios que contribuyan a una comprensión correcta del resultado esperado.

- Los escenarios que se definan ya funcionan como una prueba definida, y también se definen las funcionalidades que más valor aportan al proyecto.

- Se realiza la calificación, aunque al momento de la calificación ya no hay espacio para correcciones, durante la creación de las funcionalidades es posible realizar pruebas que ayudarán a tener mayor seguridad del funcionamiento correcto de las funcionalidades.

Figura 17.

Desarrollo de software utilizando BDD



Nota. Gráfica que muestra el proceso general del proceso de desarrollo de *software* utilizando BDD. Elaboración propia, realizado con Canva.

Como vemos en el diagrama, cuando utilizamos el modelo BDD el proceso (pasos 2,3,4 y 5) desde la definición de requerimientos se realiza en un ambiente colaborativo en el que participan todo los involucrados, lo que facilita una comprensión común tanto del problema como de la solución buscada. Existe también una sola fuente de información, asegurando así el correcto entendimiento del comportamiento esperado del sistema a desarrollar.

Debemos resaltar también que existen mini ciclos de desarrollo, prueba, corrección y validación que ayudan a obtener funcionalidades más completas y que hacen lo que se espera de ellas.

7.2. Ventajas del uso de BDD

Veamos ahora las ventajas que obtenemos con el uso de BDD como metodología de trabajo.

7.2.1. Ambiente colaborativo

Se crea desde el inicio un ambiente en el cual se facilita la comunicación para la explicación de los proyectos y también para la resolución de dudas, ayudando así a que los estudiantes se sientan más confiados del trabajo que realizarán.

7.2.2. Mejor comprensión de lo esperado

La definición de escenarios ayuda a entender de forma clara el comportamiento esperado por cada funcionalidad. Si existen dudas es posible aclararlas y ajustar la definición tan detalladamente como sea posible. Este punto es importante ya que muchas veces por pequeños detalles se puede presentar

fallas al momento de las pruebas o calificación, y esto puede ser por una comprensión inexacta de lo que se espera.

7.2.3. Evitar funcionalidades no importantes

Tener una mejor comprensión de lo esperado e importante del proyecto ayuda a enfocarse únicamente en lo que aporta valor a lo solución y nos ayuda a ver aquello que no es importante. Esta visibilidad puede ayudar también a Ingenieros y Auxiliares a notar si hay muchas tareas que no son de valor para el curso, pero que son necesarias, y tomar decisiones sobre cómo implementarlas.

7.2.4. Mejor uso del tiempo

Un aspecto importante durante el desarrollo de un proyecto es el tiempo, la mejor comprensión de lo esperado y también de lo importante permiten que se haga un mejor uso del tiempo disponible.

7.2.5. Facilidad de probar

Debido a que el proceso de BDD incluye la realización de pruebas, el estudiante podría saber con anticipación si cada funcionalidad opera de forma correcta.

Ingenieros y auxiliares podrían también contar con un set de pruebas definido desde el inicio, ordenado en cuanto a importancia y con facilidad de probarse.

7.3. Identificación de roles

Como hemos visto con anterioridad en BDD es muy importante el conocimiento del negocio y también tener la mayor claridad posible de los resultados esperados. Considerando entonces que quien tiene el conocimiento tanto sobre los contenidos como de los resultados que se esperan al elaborar un proyecto, los roles serían los siguiente:

- *Product owner*: este rol estaría asignado al ingeniero titular de la clase y el tutor académico asignado al curso. Ellos serían los encargados de definir cuáles son los problemas que se desean resolver, así como las reglas que deben respetarse en su resolución.

- *Developer*: el estudiante asignado al laboratorio del curso.

- *Testers*: este rol lo desarrollarían los mismos estudiantes durante el proceso de elaboración del proyecto y al finalizar también el auxiliar del curso cumpliría con este rol.

8. GUÍA DE APLICACIÓN DE BDD EN CURSOS DEL ÁREA DE DESARROLLO DE SOFTWARE DE INGENIERÍA EN CIENCIAS Y SISTEMAS

8.1. Definición de requerimientos

El primero aspecto importante para la correcta aplicación de BDD es entender el negocio, si bien es cierto en nuestro caso los objetivos de los diferentes laboratorios de los cursos están definidos ya, no debemos pasar por alto este punto. Entender cuál es el objetivo del curso es importante para que todos los participantes puedan estar seguros de lo que se persigue y así evitar profundizar en aspectos que no contribuyen al logro de éste.

En esta etapa todos los involucrados deben tener claro lo que se espera de un proyecto y como este puede ayudar a cumplir los objetivos de aprendizaje del curso. Este paso debe ser realizado por ingenieros de clase y auxiliares de laboratorio.

8.2. Descubrimiento de funcionalidades

Más que identificar tantas funcionalidades como sea posible, la técnica BDD propone identificar cuáles son las funcionalidades que son necesarias. Este es un paso importante ya que permite el uso del tiempo creando código que realmente importe y de valor al objetivo deseado. Es importante en este paso que no solo se definen las funcionalidades sino también se identifica “¿cómo?” ayudan a obtener los resultados esperados.

En este paso nos encontramos en una fase que podría ser realizada en dos pasos:

- Realizar primero una definición de funcionalidades por tutores académicos e ingenieros titulares de clase.

- Explicación y análisis de funcionalidades como parte de las actividades del proyecto. En este paso es donde se introduce a los estudiantes desde una temprana etapa en el uso de técnicas y metodologías ágiles, y se podrán también incentivar el análisis para la búsqueda de soluciones a problemas, no solo a nivel de desarrollo sino también a un nivel más alto.

8.3. Creación de escenarios

Para ejemplificar este paso vamos a simular el enunciado de un proyecto del laboratorio del curso Manejo e Implementación de archivos. El proyecto busca la creación de un sistema de archivos EXT3, el cual permita realizar operaciones sobre diferentes archivos y directorios.

- Creación de particiones

Esta función es administrada por medio de la consola, que guía cada paso para crear la partición y formatear el disco.

- Crear nuevo disco duro

Esta función permite la creación de discos duros virtuales que luego podrán ser usados dentro del sistema. Veamos qué ocurre cuando especificamos solo con reglas:

- RF01: permite crear un nuevo disco duro virtual con un nombre y un tamaño específico.

- RF02: el tamaño del disco podrá ser un múltiplo de 4.

Cuando utilizamos reglas pueden existir dudas que estas reglas no resuelven, por ejemplo:

- ¿Puedo utilizar un mismo nombre dos veces?
- ¿Existe límite de tamaño para el disco duro?
- ¿Qué se debe hacer si el tamaño ingresado no es múltiplo de 4?

Si modelamos los requisitos con ejemplos obtenemos los escenarios como se muestra en la figura 18:

Figura 18.

Escenario de creación de un disco duro

Escenario: Creación de un nuevo disco
Dado que usuario está en la pantalla de creación
Cuando escribo un nombre que no existe y un tamaño que es múltiplo de 4 y menor a 128Mb
Entonces se debe crear un disco nuevo

Escenario: Creación de un nuevo disco fallida
Dado que usuario está en la pantalla de creación
Cuando escribo un nombre que ya existe o un tamaño que no es múltiple de 4, o un tamaño mayor a 128Mb
Entonces se debe mostrar un error indicando el criterio no cumplido.

Nota. Ejemplo de escenarios de funcionalidad crear disco duro. Elaboración propia, realizado con Canva.

Como se puede observar mediante la descripción de escenarios se logra ejemplificar de mejor forma, estableciendo criterios y generando pruebas que posteriormente incluso pueden automatizarse.

- Crear partición

Esta función permite crear particiones dentro de un disco duro. Veamos qué ocurre cuando especificamos solo con reglas:

  - RF01: permite crear una partición de cualquier tamaño (menor al tamaño del disco duro).
  - RF02: pueden crearse n particiones.
  - RF03: estas particiones no tendrán formato.

Dudas que estas reglas no resuelven, por ejemplo:

  - ¿Se pueden crear varias particiones al mismo tiempo?
  - ¿Cómo se debe identificar cada partición?

Si modelamos los requisitos con ejemplos obtenemos un escenario como en que muestra la figura 19.

Figura 19.

Ejemplo 1 de escenario de creación de una partición

Escenario: Creación de una partición exitosa
Dado que quiero crear una partición nueva y que ya existe un disco duro y que la letra que identifica las particiones está disponible
Cuando ingreso una letra que identificará la partición y coloco el tamaño menor al tamaño del disco duro
Entonces se debe crear una partición dentro del disco duro, la cual quedara sin formato y regreso al menú principal

Nota. Ejemplo de escenarios de funcionalidad crear disco partición. Elaboración propia, realizado con Canva.

Otra forma de crear el escenario del ejemplo anterior podemos verlo en la figura 20:

Figura 20.

Ejemplo 2 de escenario de creación de una partición

Escenario: Crear una partición
Dado que tenemos un disco duro de 32 Mb sin particiones
Cuando creamos una partición de 16 Mb
Entonces el disco duro queda con una partición de 16Mb sin formato y un espacio libre de 16Mb

Nota. Ejemplo de escenarios de funcionalidad crear disco partición. Elaboración propia, realizado con Canva.

Como se puede observar mediante la descripción de escenarios se logra ejemplificar de mejor forma, estableciendo criterios y generando pruebas que posteriormente incluso pueden automatizarse.

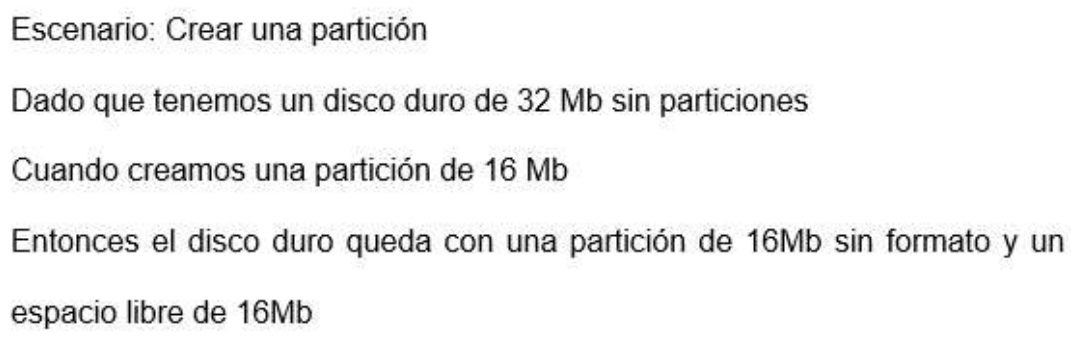
8.4. Formulación

Una vez definidos y validados algunos escenarios, el siguiente paso es la creación de cada escenario de una forma estructurada y ejecutable, esto será útil para asegurarnos la correcta comprensión de lo que se desea construir. Adicional, este paso es la base para posteriormente poder automatizar.

Para el siguiente ejemplo (ver figura 21) utilizaremos el lenguaje Gherking, el cual es un lenguaje de especificación para varias herramientas como por ejemplo Cucumber.

Figura 21.

Formulación de escenario



Nota. Ejemplo formulación de escenario en Gherking. Elaboración propia, realizado con Canva.

Ahora entonces, es posible que esta especificación se convierta en una prueba automatizada y por lo tanto en la guía del proceso de desarrollo e implementación de la solución.

8.5. Automatización

Una vez tengamos las especificaciones ejecutables, podremos realizar la conexión con la aplicación como una prueba. Siempre al inicio la prueba será fallida pues aún no se ha implementado el comportamiento descrito, ahora es el momento de comenzar a escribir el código.

8.5.1. Probar

Una vez terminado el código correctamente, la prueba creada funcionará correctamente, lo cual nos servirá para monitorear en cualquier momento el funcionamiento de la funcionalidad creada.

8.5.2. Refactorizar

Refactorizar es el proceso en el que se revisa y modifica el código de una funcionalidad con el fin de mejorar su estructura o atributos sin alterar su funcionamiento, por lo tanto, cuando necesitemos regresar más tarde al código y realizar mantenimiento, las pruebas automatizadas serán de mucha utilidad para verificar que todo sigue funcionando bien y no causar fallas no planificadas.

CONCLUSIONES

1. El uso de técnicas como BDD dentro de los proyectos del desarrollo ayudan a asegurar que los resultados finales sean los deseados.

2. El uso de BDD como parte de la metodología de trabajo de los cursos del área de desarrollo de *software* puede contribuir a una mejor comprensión de las funcionalidades más importantes de los proyectos a realizar y de cómo estas contribuyen a los objetivos deseados.

3. Utilizar BDD como metodología de trabajo introduce a los estudiantes al trabajo en ambientes colaborativos y le permite adoptar de forma temprana técnicas para realizar buenas prácticas en las actividades de desarrollo.

4. Es posible utilizar los principios generales de BDD desde etapas tempranas de la carrera, e incluir en etapas posteriores más prácticas y actividades de la técnica.

RECOMENDACIONES

1. Asegurar que quienes dirijan estos cursos cuentan con el conocimiento necesario para poder incluir BDD dentro de las actividades de los laboratorios y hacer su correcto uso.

2. Adoptar metodologías de desarrollo desde las etapas tempranas de los cursos, para garantizar así las mejores prácticas desde el inicio de la formación.

3. Considerar que no es necesario incluir todo el ciclo de BDD en todas las funcionalidades ni desde los primeros cursos de desarrollo.

4. Crear tutoriales sobre BDD incluyendo conceptos, herramientas y procesos, para ayudar al estudiante y evitar que el uso de esta metodología de trabajo le quite la atención al tema central del curso.

REFERENCIAS

Ferguson, J. (2015). *BDD en acción.* Manning Publications Co.

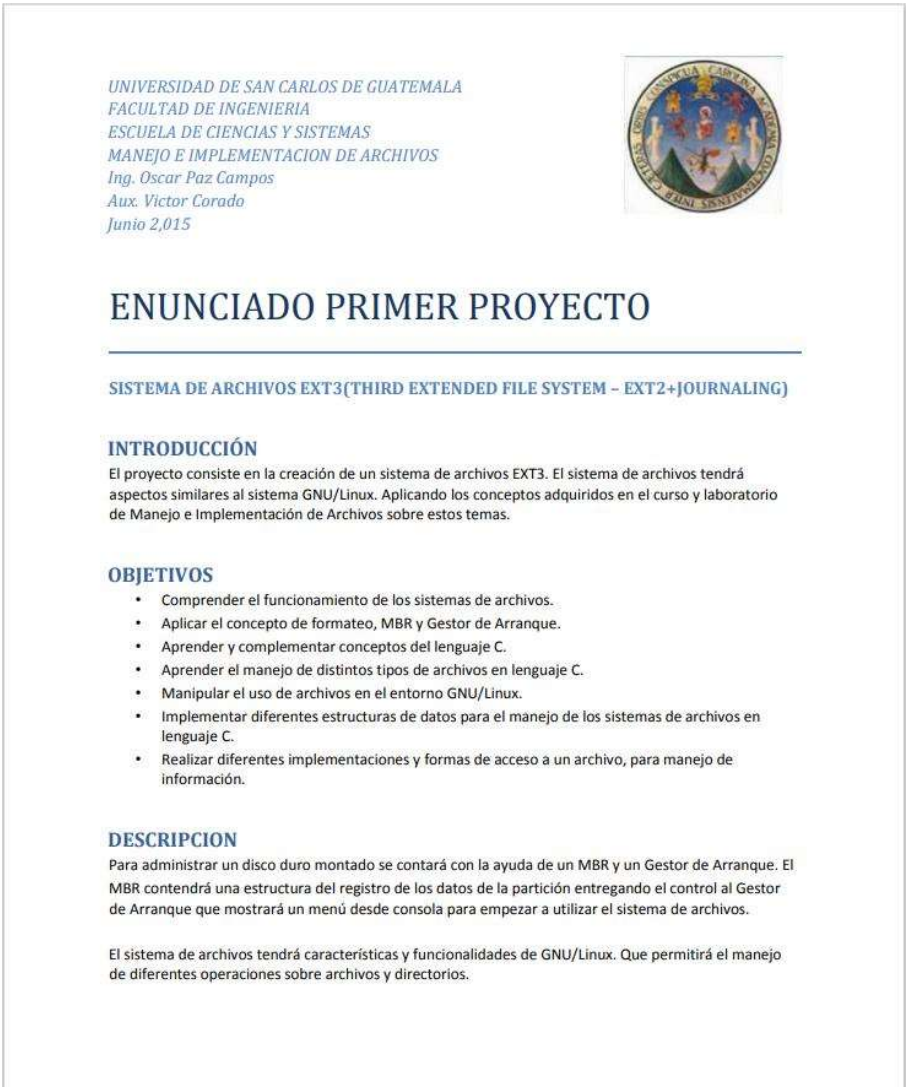
Kolb, D. (1984). *Aprendizaje por experiencia: experiencias como fuente de aprendizaje y desarrollo.* Prentice Hall.

Pollyanna, P., Nickolaisen, N., Little, T. & McDonald, K. (2009). *Retroceder y entregar: acelerando la agilidad del negocio.* Addison-Wesley Professional.

ANEXOS

Anexo 1.

Proyecto 1, manejo e implementación de archivos



UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA
ESCUELA DE CIENCIAS Y SISTEMAS
MANEJO E IMPLEMENTACION DE ARCHIVOS
Ing. Oscar Paz Campos
Aux. Victor Corado
Junio 2,015

ENUNCIADO PRIMER PROYECTO

SISTEMA DE ARCHIVOS EXT3(THIRD EXTENDED FILE SYSTEM – EXT2+JOURNALING)

INTRODUCCIÓN

El proyecto consiste en la creación de un sistema de archivos EXT3. El sistema de archivos tendrá aspectos similares al sistema GNU/Linux. Aplicando los conceptos adquiridos en el curso y laboratorio de Manejo e Implementación de Archivos sobre estos temas.

OBJETIVOS

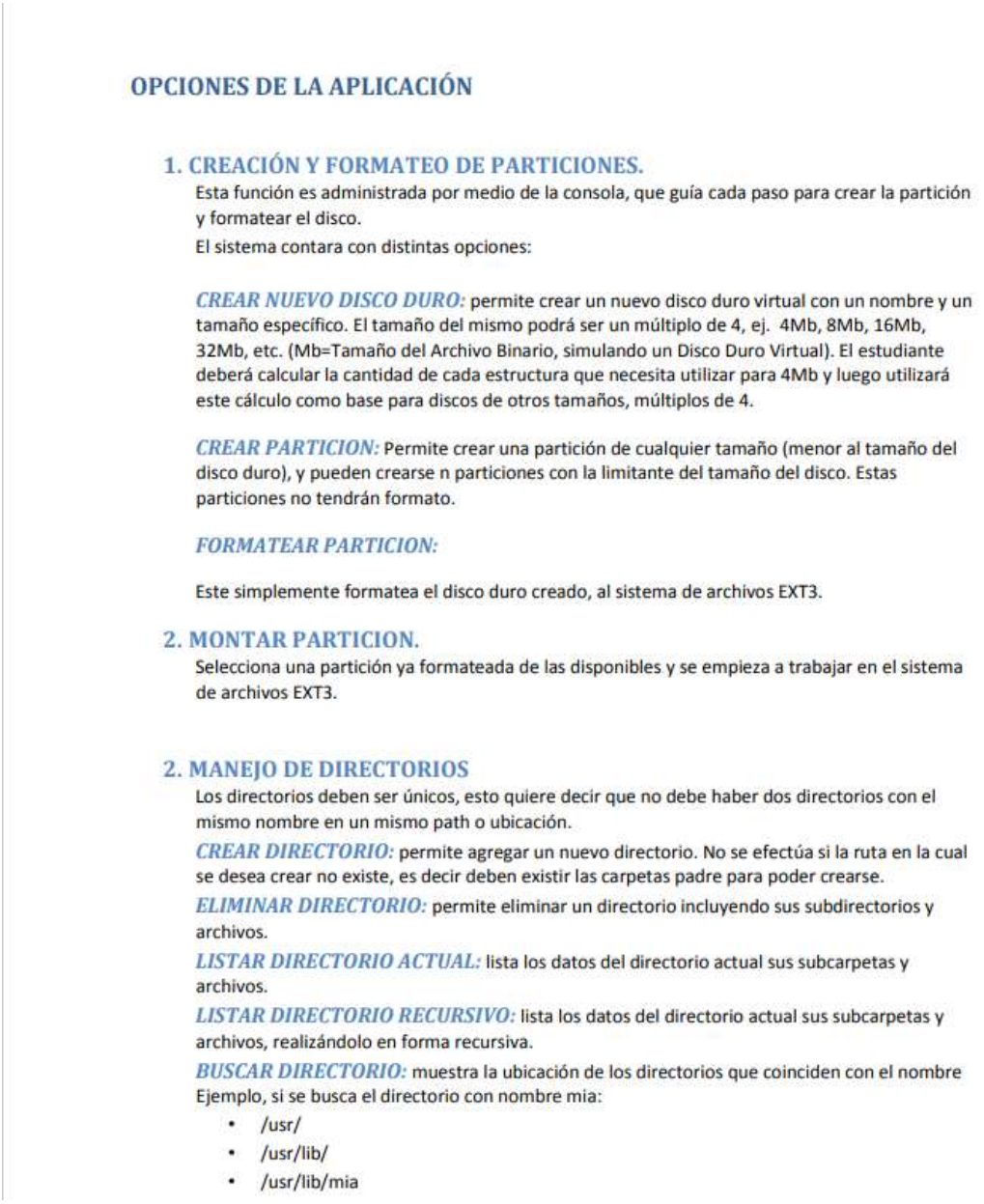
- Comprender el funcionamiento de los sistemas de archivos.
- Aplicar el concepto de formateo, MBR y Gestor de Arranque.
- Aprender y complementar conceptos del lenguaje C.
- Aprender el manejo de distintos tipos de archivos en lenguaje C.
- Manipular el uso de archivos en el entorno GNU/Linux.
- Implementar diferentes estructuras de datos para el manejo de los sistemas de archivos en lenguaje C.
- Realizar diferentes implementaciones y formas de acceso a un archivo, para manejo de información.

DESCRIPCION

Para administrar un disco duro montado se contará con la ayuda de un MBR y un Gestor de Arranque. El MBR contendrá una estructura del registro de los datos de la partición entregando el control al Gestor de Arranque que mostrará un menú desde consola para empezar a utilizar el sistema de archivos.

El sistema de archivos tendrá características y funcionalidades de GNU/Linux. Que permitirá el manejo de diferentes operaciones sobre archivos y directorios.

Continuación del anexo 1.



OPCIONES DE LA APLICACIÓN

1. CREACIÓN Y FORMATEO DE PARTICIONES.

Esta función es administrada por medio de la consola, que guía cada paso para crear la partición y formatear el disco.

El sistema contara con distintas opciones:

CREAR NUEVO DISCO DURO: permite crear un nuevo disco duro virtual con un nombre y un tamaño específico. El tamaño del mismo podrá ser un múltiplo de 4, ej. 4Mb, 8Mb, 16Mb, 32Mb, etc. (Mb=Tamaño del Archivo Binario, simulando un Disco Duro Virtual). El estudiante deberá calcular la cantidad de cada estructura que necesita utilizar para 4Mb y luego utilizará este cálculo como base para discos de otros tamaños, múltiplos de 4.

CREAR PARTICION: Permite crear una partición de cualquier tamaño (menor al tamaño del disco duro), y pueden crearse n particiones con la limitante del tamaño del disco. Estas particiones no tendrán formato.

FORMATEAR PARTICION:

Este simplemente formatea el disco duro creado, al sistema de archivos EXT3.

2. MONTAR PARTICION.

Selecciona una partición ya formateada de las disponibles y se empieza a trabajar en el sistema de archivos EXT3.

2. MANEJO DE DIRECTORIOS

Los directorios deben ser únicos, esto quiere decir que no debe haber dos directorios con el mismo nombre en un mismo path o ubicación.

CREAR DIRECTORIO: permite agregar un nuevo directorio. No se efectúa si la ruta en la cual se desea crear no existe, es decir deben existir las carpetas padre para poder crearse.

ELIMINAR DIRECTORIO: permite eliminar un directorio incluyendo sus subdirectorios y archivos.

LISTAR DIRECTORIO ACTUAL: lista los datos del directorio actual sus subcarpetas y archivos.

LISTAR DIRECTORIO RECURSIVO: lista los datos del directorio actual sus subcarpetas y archivos, realizándolo en forma recursiva.

BUSCAR DIRECTORIO: muestra la ubicación de los directorios que coinciden con el nombre Ejemplo, si se busca el directorio con nombre mia:

- /usr/
- /usr/lib/
- /usr/lib/mia

Nota. Enunciado Proyecto 1, manejo e implementación de archivos. Obtenido de Corado, V. (2015). *Enunciado primer proyecto*. (pp. 1-2).