LM
Laberinto Musical

Carlos Alberto Fuentes Linares
20142020107
Juan Camilo Navarro Quiroga
20162021317
Julian Mateo Sanchez Ovalle
20162020443

2020

Índice general

I PROYECTO **9**

1. Caso de Estudio **11**
1.1. Introducción 11
1.2. Problema 12
1.3. Objetivo General 12
1.4. Objetivos Específicos 12
1.5. Alcance del proyecto 13

2. Proceso - Metodología **15**
2.1. El modelo en V 16
2.1.1. Descripción 16
2.1.2. Cronograma GANTT 16
2.2. Metodología FDD 17
2.2.1. Descripción 17
2.3. Open SOURCE 18

II DISEÑO **19**

3. Requerimientos **21**
3.1. Introducción 21
3.2. Casos de uso 22
3.2.1. Caso de uso cliente 22
3.2.2. Caso de uso empleado 22

4. Interacción **25**
4.1. Introducción 25
4.2. Diagrama Secuencia desarrollo representacion espacial 26
4.3. Diagrama de Comunicación interacciòn 26
4.4. Diagrama de Secuencia Gestión laberinto 27

4.5. Diagrama de Comunicación Gestión Laberinto 27

5. Diagramas de clases y workflow **29**
5.1. Introducción 29
5.1.1. Diagrama de clases Laberinto Musical 30
5.1.2. Diagrama de WorkFlow Laberinto Musical 31

6. Patrones **33**
6.1. Introducción 33
6.2. Fabrica abstracta 34
6.3. Adaptador 35
6.4. Fachada 36
6.5. Cadena de responsabilidad 38
6.6. Iterador 39

7. Estados **41**
7.1. Introducción 41
7.2. Diagramas de estado 41
7.2.1. Componentes: Estados 42
7.2.2. Componentes: Transición 42
7.3. Diagramas de estado del juego 42

8. Componentes **45**
8.1. Introducción 45
8.2. Interfaces 46
8.3. Diagrama de Componentes 47

9. Metricas **49**

10. Nodos **59**
10.1. Introducción 59
10.2. Diagrama de Nodos 60

11. Actividades **61**
11.1. Introducción 61
11.2. Diagrama de Actividades 62

III REFLEXIONES **63**

12. Conclusiones **65**

13.Bibliografia **67**

Índice de figuras

2.1. Modelo en V 16
2.2. Diagrama GANTT 17
2.3. Diagrama de metodología FDD 17

3.1. Caso de uso cliente 22

4.1. Diagrama Secuencia desarrollo representacion espacial 26
4.2. Diagrama de Comunicación laberinto 27
4.3. Diagrama de Secuencia Gestión Laberinto 27
4.4. Diagrama de Comunicación Gestión Laberinto 28

5.1. Diagrama de clases 30
5.2. Diagrama workFlow 31

6.1. Diagrama de clases patrón fabrica abstracta 34
6.2. Diagrama de secuencia patrón fabrica abstracta 35
6.3. Diagrama de clases patrón adaptador 36
6.4. Diagrama de secuencia patrón adaptador 36
6.5. Diagrama de clases patrón fachada 37
6.6. Diagrama de secuencia patrón fachada 37
6.7. Diagrama de clases patrón cadena de responsabilidad 38
6.8. Diagrama de secuencia patrón cadena de responsabilidad . . 39
6.9. Diagrama de clases patrón iterador 40
6.10. Diagrama de secuencia patrón iterador 40

7.1. Diagrama de estado del juego 43

8.1. Diagrama de Componentes 48

9.1. numero de clases optima 50
9.2. numero de metodos y clases 51

9.3. Acá se visualiza que la carpeta edu tiene dos paquetes diferentes los cuales son edu.presentacion y edu.cableado. 52
9.4. Visualización de la distribución. 53
9.5. Complejidad ciclomatica del proyecto.. 54
9.6. Líneas estimadas por la clase top. 55
9.7. Vista de la distancia y su respectivamente. 56
9.8. Las respectivas clases. 57

10.1. Diagrama de Nodos 60

11.1. Diagrama de Actividades 62

Parte I

PROYECTO

Capítulo 1

Caso de Estudio

1.1. Introducción

1.2. Problema

Los videojuegos cada vez intentan ser más sociales y que se pueda jugar en grupo a través de redes virtuales, competir contra los amigos y desarrollar habilidades como los reflejos o la agudeza visual. Frente a todos estos avances, un grupo de estudiantes del Instituto Tecnológico de Massachussets (MIT por sus siglas en inglés) se preguntaron qué faltaba. Era el verano de 2007 y después de un año de prácticas y ensayos AudiOdyssey por fin ha visto la luz. Su objetivo es sencillo: que las personas ciegas o con otros problemas de visión y aquellas que vean con normalidad puedan compartir un videojuego sin que uno de los grupos esté en desventaja. Aunque este caso es muy especial es precisamente lo que hace diferente este nicho de mercado, dada la baja demanda existente. Pero esa baja demanda no se debe a pocos usuarios sino mas bien a los pocos desarrollos existenten en este campo. La razon de ser del laberinto musical es crear un videojuego que presente una interfaz auditiva con la que personas con discapacidad visual sean capacez de hacer una inmerciòn profunda en el viddeo juego desarrollado en el proyecto.

1.3. Objetivo General

Analizar, desarrollar, diseñar e implementar un prototipo de software que permita a personas invidentes jugar por medio de comanddos de sonido aplicando y desarrollando a través del proyecto las correspondientes etapas del desarrollo de software.

1.4. Objetivos Específicos

- Definir la problemática que justifica el desarrollo del laberinto musical.
- Realizar un análisis con los correspondientes requerimientos que son esenciales para el desarrollo del proyecto.
- Establecer la documentación teórica dentro de los plazos definidos del análisis e implementación del software.
- Modelar el laberinto musical por medio de componentes acorde a las pautas correctas expuestas durante la clase.

1.5. Alcance del proyecto

El software está diseñado como una aplicación web que permitirá a usuarios invidentes jugar en línea e interactuar con diferentes laberintos que ofrece el juego estableciendo diferentes niveles de dificultad dentro del juego. El alcance de las tecnologias hoy en dia nos permite tener audio de calidaddentro de la aplicacion, por lo que el juego debe ıa (con los debidos estudi osy preparacion) ser extensible para oidos entrenados, por otro lado, a pesar decontar con medios de salida visual, estos en la practica no son un requerimientopara el funcionamiento de el mismo

Capítulo 2

Proceso - Metodología

2.1. El modelo en V

2.1.1. Descripción

El Método-V define un procedimiento uniforme para el desarrollo de productos para las TIC. Es el estándar utilizado para los proyectos de la Administración Federal alemana y de defensa. Como está disponible públicamente muchas compañías lo usan. Es un método de gestión de proyectos comparable a PRINCE2 y describe tanto métodos para la gestión como para el desarrollo de sistemas.

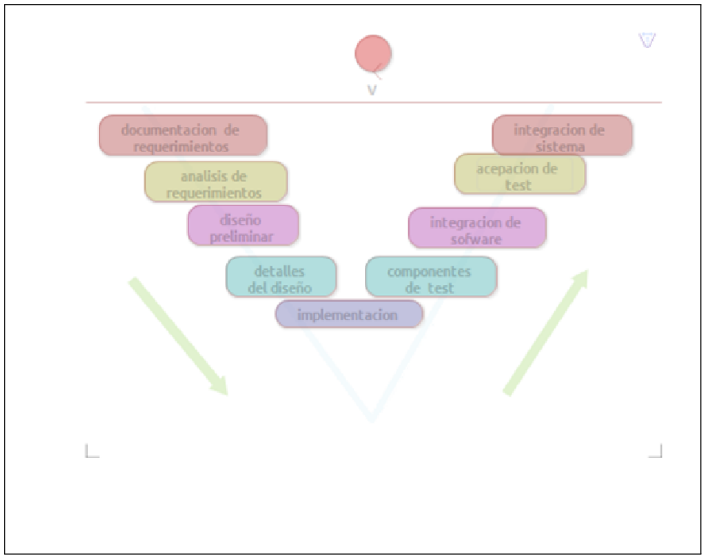


Figura 2.1: Modelo en V

2.1.2. Cronograma GANTT

Mediante la definición de tareas pertinentes al desarrollo del proyecto pudimos establecerlas y fueron organizadas en orden cronológico, así como también se le dio un peso relativo conforme a la importancia de esta misma para determinar su sitio exacto dentro del grupo completo de labores a realizar. Para la elaboración del cronograma se usó una herramienta que permite dibujar un diagrama de GANTT que facilita la visualización y avance dentro del proyecto dando un panorama del estado actual, como también de las futuras actividades además de las que ya fueron realizadas..

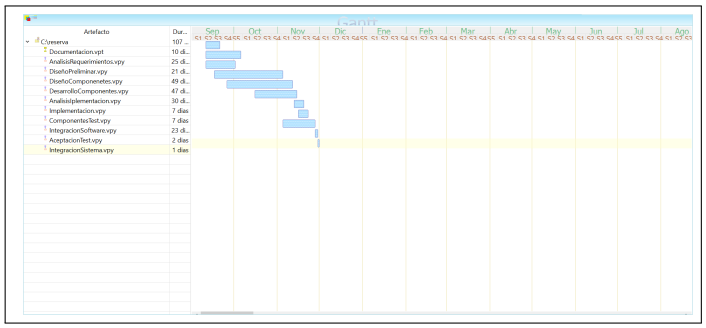


Figura 2.2: Diagrama GANTT

2.2. Metodología FDD

Sus siglas en inglés Feature Driven Development, la metodología del desarrollo impulsado hace parte de las metodologías ágiles, esta fue desarrollada por Jeff De Luca y Peter Coad a mediados de la década de los noventas. Se enfoca en iteraciones cortas, que permiten entregas palpables en un corto período de tiempo (máximo dos semanas).

Entre sus principales características esta la de mantener el proyecto en alta calidad haciendo un análisis constante dentro del mismo, ayuda a su vez a controlar varios riesgos que se puedan presentar dentro del proyecto como los excesos de presupuesto o fallas en el programa, teniendo en cuenta etapas de cierre cada dos semanas y obteniendo así resultados periódicos y tangibles.

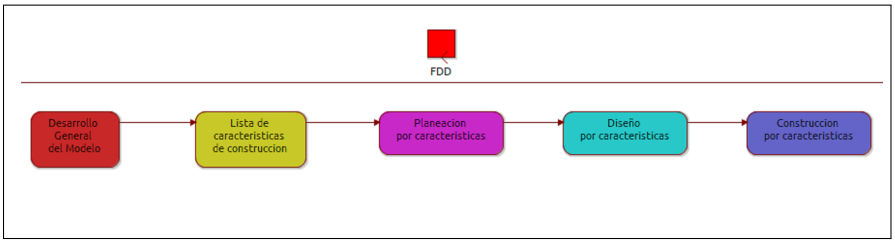


Figura 2.3: Diagrama de metodología FDD

2.2.1. Descripción

Con esta metodología vamos a tener un amplio manejo con el presupuesto y el tiempo para el desarrollo del proyecto, todo esto también nos

da el beneficio de probar todos los componentes con el fin de cumplir los requerimientos del proyecto. Su rápida respuesta a cambios nos da toda la versatilidad a la hora de ampliar reseñadas dentro de la aplicación, lo cual nos brinda una mejora en el rendimiento de los costos a la hora de implementar cambios dentro de nuestro programa y el manejo de los tiempos de diseño.

2.3. Open SOURCE

Si bien el desarrollo de software propietario y privativo es la intención de muchos proyectos, existe una metodología de desarrollo que tomó mucha relevancia en el nacimiento de UNIX/LINUX, y es que a diferencia del software creado a la medida, se propone que el código de los proyectos sea libre (disponible para ser modificado'). Esta forma de trabajo está orientada a una gran comunidad la cual desarrolla dependiendo de sus necesidades, mediante la reutilización de software y además de grandes ventajas como heredar el código desarrollado para que otros lo retomen y realicen los aportes que consideren prudentes atendiendo así una mayor cantidad de necesidades tanto de desarrolladores como de usuarios no especializados. Se puede dar un ejemplo común para nuestra era como lo fue la creación de repositorios mediante GitHub, donde los usuarios comparten sus códigos para que la comunidad tenga una gama amplia de recursos a la hora de adentrarse en algún tema en específico. En síntesis el Open Source es un desarrollo de hiper retroalimentación gestionado por una gran comunidad.

Parte II

DISEÑO

Capítulo 3

Requerimientos

3.1. Introducción

Los casos de uso evitan típicamente el lenguaje técnico, prefiriendo la lengua del usuario final o del experto del campo del saber al que se va a aplicar. Los casos del uso son a menudo elaborados en colaboración por los analistas de requisitos y los clientes.

Cada caso de uso se centra en describir cómo alcanzar una única meta o tarea. Desde una perspectiva tradicional de la ingeniería de software, un caso de uso describe una característica del sistema. Para la mayoría de proyectos de software, esto significa que quizás a veces es necesario especificar decenas o centenares de casos de uso para definir completamente el nuevo sistema. El grado de la formalidad de un proyecto particular del software y de la etapa del proyecto influenciará el nivel del detalle requerido en cada caso de uso.

Los casos de uso pretenden ser herramientas simples para describir el comportamiento del software o de los sistemas. Un caso de uso contiene una descripción textual de todas las maneras que los actores previstos podrían trabajar con el software o el sistema. Los casos de uso no describen ninguna funcionalidad interna (oculta al exterior) del sistema, ni explican cómo se implementará. Simplemente muestran lo que el actor hace o debe hacer para realizar una operación.

3.2. Casos de uso

Un caso de uso es la descripción de una acción o actividad. Un diagrama de caso de uso es una descripción de las actividades que deberá realizar alguien o algo para llevar a cabo algún proceso. Los personajes o entidades que participarán en un diagrama de caso de uso se denominan actores. En el contexto de ingeniería del software, un diagrama de caso de uso representa a un sistema o subsistema como un conjunto de interacciones que se desarrollarán entre casos de uso y entre estos y sus actores en respuesta a un evento que inicia un actor principal. Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema. Una relación es una conexión entre los elementos del modelo, por ejemplo la especialización y la generalización son relaciones. Los diagramas de casos de uso se utilizan para ilustrar los requisitos del sistema al mostrar cómo reacciona a eventos que se producen en su ámbito o en él mismo.

3.2.1. Caso de uso cliente

Se describe la interacción del cliente con la página web

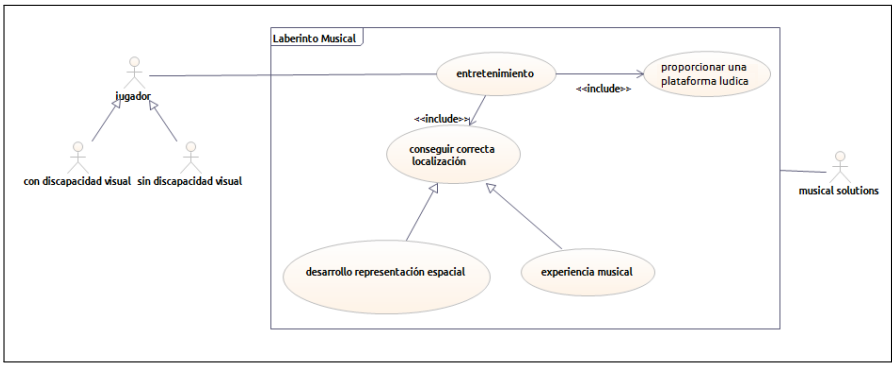


Figura 3.1: Caso de uso cliente

3.2.2. Caso de uso empleado

se describe la interacción del empleado con la página web

identificador		1	nombre	desarrollo representación espacial
descripcion	Mediante la interacción con el juego teniendo en cuenta los factores como la musica y la dimensión del laberinto el usuario va a facilitar un desarrollo espacial.			
condiciones	- Haber ingresado a la plataforma - Seleccionar las condiciones del juego			
flujos	primaria	- Interacción correcta del usuario con el juego puede permitir el desarrollo de la representación espacial según las condiciones del juego.		
	secundaria	- Por características propias de la melodía no concorde con la dimensión del laberinto y no se den las condiciones para desarrollar la representación espacial.		
	excepcionales	- Tener ruido externo que opaque el sonido del juego y genere una mal interpretación del espacio en el que se desarrolla mismo.		

identificador		2	nombre	experiencia musical
descripcion	Según los criterios del usuario puede la aplicacion ser una herramienta que le brinde una experiencia musical satisfactoria y ayude a desarrollar los conocimientos de la misma			
condiciones	- Tener concentración en el desarrollo del juego y su correcto uso - Tener unas condiciones apropiadas en terminos espaciales y de comodidad - Tener una disposición apropiada			
flujos	primaria	- Interacción correcta del usuario con el juego en la que puede lograr un nueva conocimiento sobre la teoría musical que la herramienta le ofrece.		
	secundaria	- Por la diversidad de teoría musical que se va a menejar en la herramienta, en niveles avanzados, se puede dar una mal interpretación de las escalas musicales.		
	excepcionales	- Tener ruido externo que opaque el sonido del juego y genere una mal interpretación del espacio en el que se desarrolla mismo. - Segun los conomientos previos y aprendidos mediante la herramienta, no elegir una dificultad acorde al nivel del usuario		

identificador		3	nombre	proporcionar una plataforma ludica
descripcion	Teniendo en cuenta la problemática a tratar se pretende establecer una plataforma lúdica apropiada con la se puedan adquirir conocimientos divirtiéndose.			
condiciones	- Tener una actitud y disposición apropiada para el desarrollo del juego - Tener unas condiciones apropiadas en terminos espaciales y de comodidad - familiarizarse con las herramientas que la plataforma les proporciona			
flujos	primaria	- Que todos los factores del entorno proporcionen al usuario la experiencia deseada.		
	secundaria	- Debido a la variedad de usuarios a los que va dirigido el juego se pueden presentar varias inconformidades con respecto a las diferentes herramientas que el juego proporciona.		
	excepcionales	- Por algún factor en el desarrollo del juego genere en el usuario cierta apatía lo cual no le va a permitir obtener la experiencia deseada		

identificador		1	nombre	desarrollo representación espacial
descripcion	Mediante la interacción con el juego teniendo en cuenta los factores como la musica y la dimensión del laberinto el usuario va a facilitar un desarrollo espacial.			
condiciones	- Haber ingresado a la plataforma - Seleccionar las condiciones del juego			
flujos	primaria	- Interacción correcta del usuario con el juego puede permitir el desarrollo de la representación espacial según las condiciones del juego.		
	secundaria	- Por características propias de la melodía no concorde con la dimensión del laberinto y no se den las condiciones para desarrollar la representación espacial.		
	excepcionales	- Tener ruido externo que opaque el sonido del juego y genere una mal interpretación del espacio en el que se desarrolla mismo.		

identificador		2	nombre	experiencia musical
descripcion	Según los criterios del usuario puede la aplicacion ser una herramienta que le brinde una experiencia musical satisfactoria y ayude a desarrollar los conocimientos de la misma			
condiciones	- Tener concentración en el desarrollo del juego y su correcto uso - Tener unas condiciones apropiadas en terminos espaciales y de comodidad - Tener una disposición apropiada			
flujos	primaria	- Interacción correcta del usuario con el juego en la que puede lograr un nueva conocimiento sobre la teoría musical que la herramienta le ofrece.		
	secundaria	- Por la diversidad de teoría musical que se va a menejar en la herramienta, en niveles avanzados, se puede dar una mal interpretación de las escalas musicales.		
	excepcionales	- Tener ruido externo que opaque el sonido del juego y genere una mal interpretación del espacio en el que se desarrolla mismo. - Segun los conomientos previos y aprendidos mediante la herramienta, no elegir una dificultad acorde al nivel del usuario		

identificador		3	nombre	proporcionar una plataforma ludica
descripcion	Teniendo en cuenta la problemática a tratar se pretende establecer una plataforma lúdica apropiada con la se puedan adquirir conocimientos divirtiéndose.			
condiciones	- Tener una actitud y disposición apropiada para el desarrollo del juego - Tener unas condiciones apropiadas en terminos espaciales y de comodidad - familiarizarse con las herramientas que la plataforma les proporciona			
flujos	primaria	- Que todos los factores del entorno proporcionen al usuario la experiencia deseada.		
	secundaria	- Debido a la variedad de usuarios a los que va dirigido el juego se pueden presentar varias inconformidades con respecto a las diferentes herramientas que el juego proporciona.		
	excepcionales	- Por algún factor en el desarrollo del juego genere en el usuario cierta apatía lo cual no le va a permitir obtener la experiencia deseada		

Capítulo 4

Interacción

4.1. Introducción

El lenguaje unificado de modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el Object Management Group (OMG).

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano"del sistema (modelo), incluyendo aspectos conceptuales tales como procesos, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

Es importante remarcar que UML es un "lenguaje de modelado"para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

Se puede aplicar en el desarrollo de software gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional, Rational Unified Process o RUP), pero no especifica en sí mismo qué metodología o proceso usar.

UML no puede compararse con la programación estructurada, pues UML significa Lenguaje Unificado de Modelado, no es programación, solo se diagrama la realidad de una utilización en un requerimiento. Mientras que programación estructurada es una forma de programar como lo es la orientación a objetos, la programación orientada a objetos viene siendo un complemento perfecto de UML, pero no por eso se toma UML solo para lenguajes orientados a objetos.

4.2. Diagrama Secuencia desarrollo representacion espacial

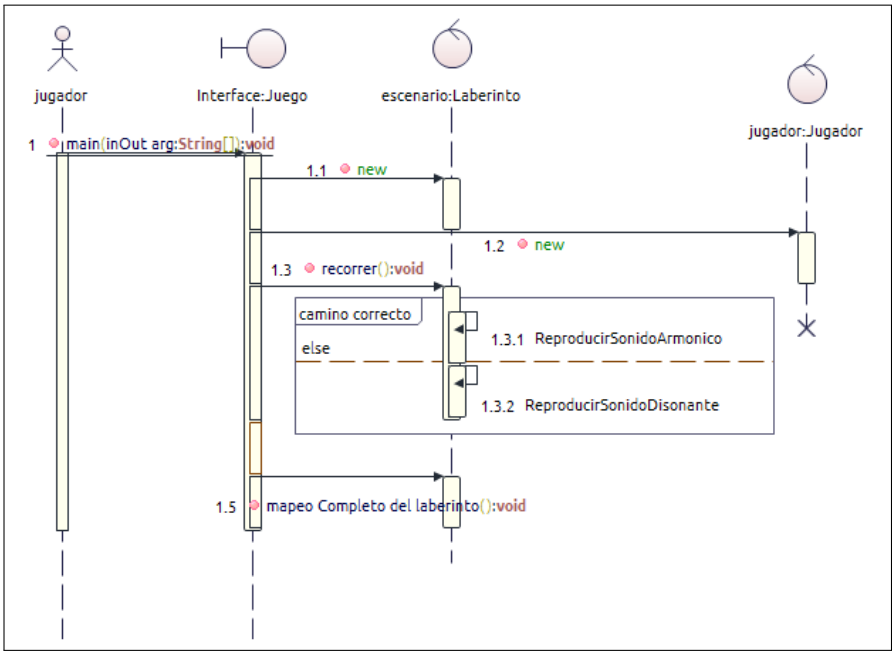


Figura 4.1: Diagrama Secuencia desarrollo representacion espacial

4.3. Diagrama de Comunicación interacciòn

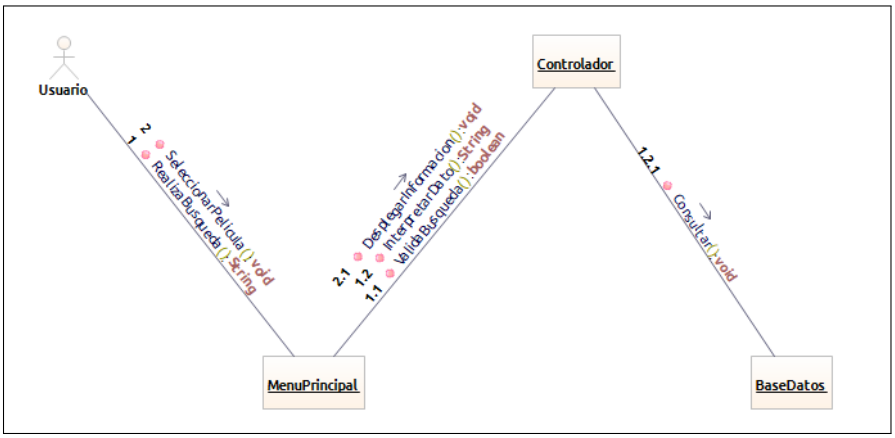


Figura 4.2: Diagrama de Comunicación laberinto

4.4. Diagrama de Secuencia Gestión laberinto

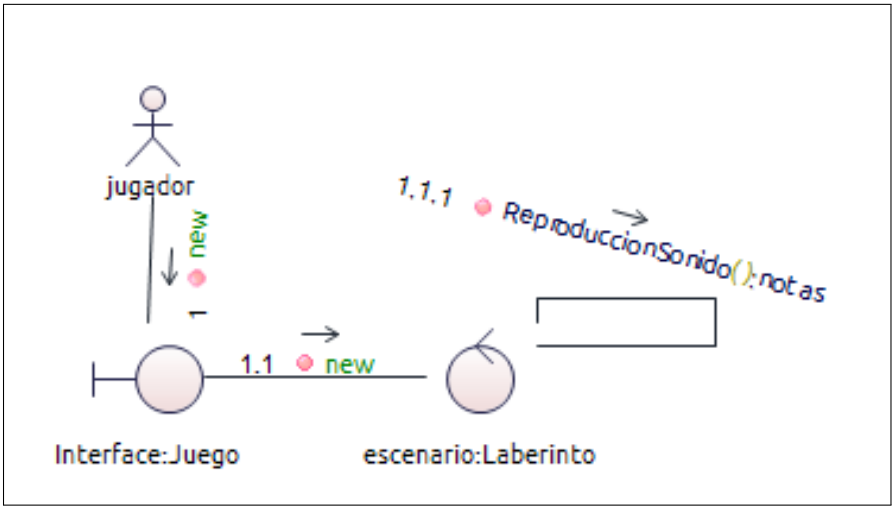


Figura 4.3: Diagrama de Secuencia Gestión Laberinto

4.5. Diagrama de Comunicación Gestión Laberinto

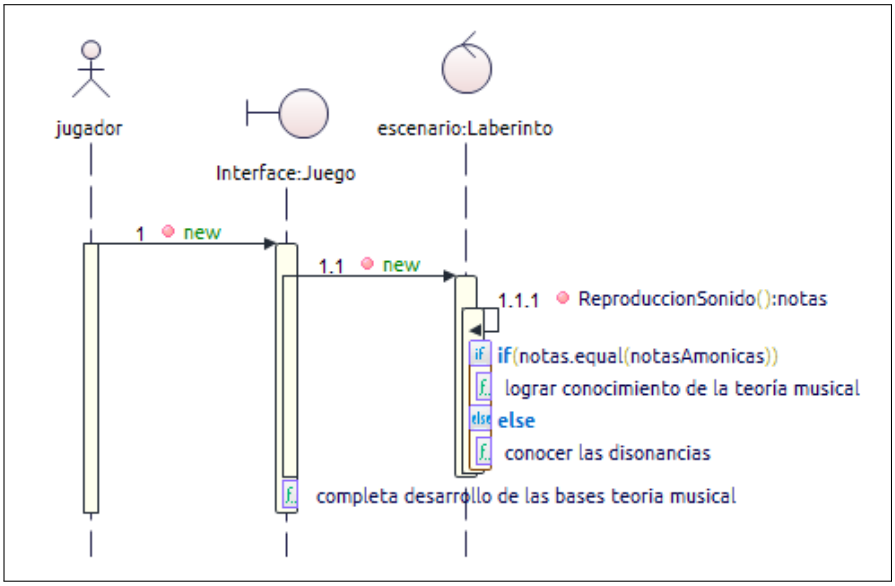


Figura 4.4: Diagrama de Comunicación Gestión Laberinto

Capítulo 5

Diagramas de clases y workflow

5.1. Introducción

El diagrama de clases es un diagrama puramente orientado al modelo de programación orientado a objetos, ya que define las clases que se utilizarón cuando se pase a la fase de construcción y la manera en que se relacionan las mismas. Las clases son el elemento principal del diagrama y representa, como su nombre indica, una clase dentro del paradigma de la orientación a objetos. Este tipo de elementos normalmente se utilizan para representar conceptos o entidades del "negocio". Una clase define un grupo de objetos que comparten características, condiciones y significado. La manera más rápida para encontrar clases sobre un enunciado, sobre una idea de negocio o, en general, sobre un tema concreto es buscar los sustantivos que aparecen en el mismo.

5.1.1. Diagrama de clases Laberinto Musical

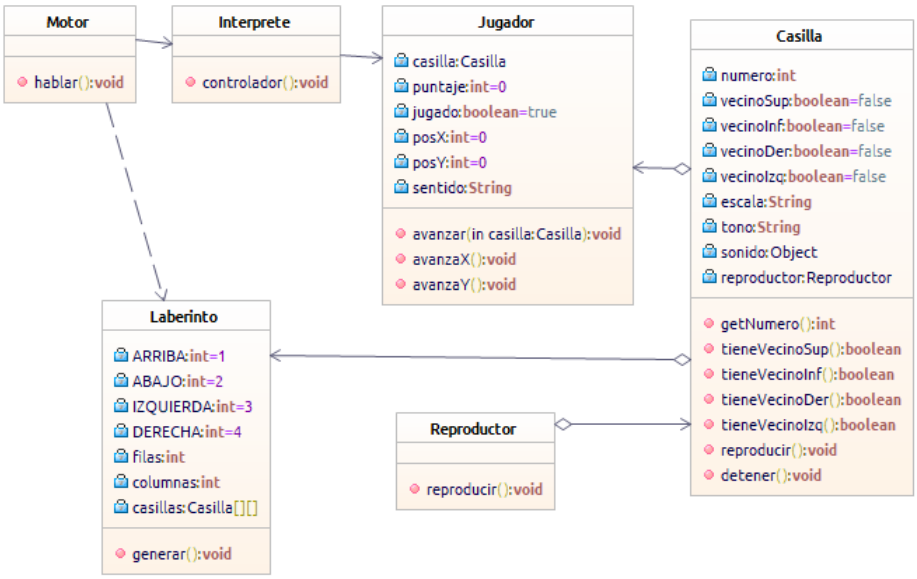


Figura 5.1: Diagrama de clases

5.1.2. Diagrama de WorkFlow Laberinto Musical

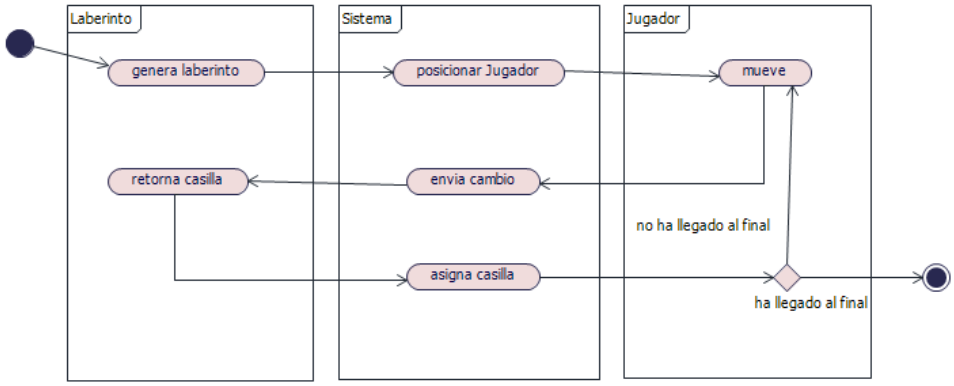


Figura 5.2: Diagrama workFlow

Capítulo 6

Patrones

6.1. Introducción

Los patrones de diseño son unas técnicas usadas para resolver problemas comunes en el desarrollo de software. Un patrón de diseño es una solución a un problema, para ser considerado como tal debe; haber comprobado su efectividad resolviendo problemas similares en ocasiones pasadas, debe también ser reutilizable.

Los patrones principales del diseño se denominan Gof (Gang of Four) debido a que fueron cuatro los autores que desarrollaron el conocido libro "Design Patterns: Elements of Reusable Object-Oriented Software", en el cual desarrolan 23 patrones de diseño y los clasifican en tres: -Patronces Creacionales (Abstract Factory, Builder, Factory Method, Prototype, Singleton), -Patronces Estructurales (Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy, Module) y -Patronces de Comportamiento (Chain of Responsability, Command, ,Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor)

6.2. Fabrica abstracta

Patrón perteneciente a los patrones creacionales de Gang of Four, este patrón permite trabajar con objetos de distintas familias haciendo que éstas no se mezclen entre si y por lo tanto haciendo transparente el tipo de familia concreta que se esté usando. Su creación va de la mano de la solución al problema de crear diferentes familias de objetos.

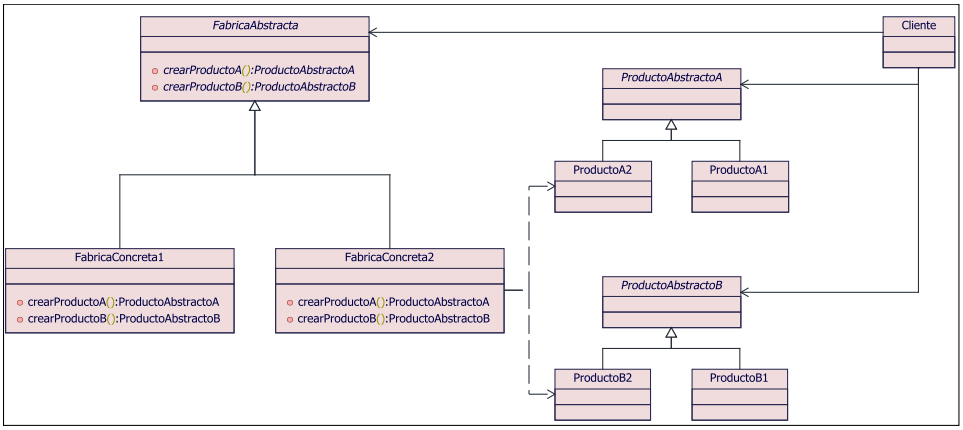


Figura 6.1: Diagrama de clases patrón fabrica abstracta

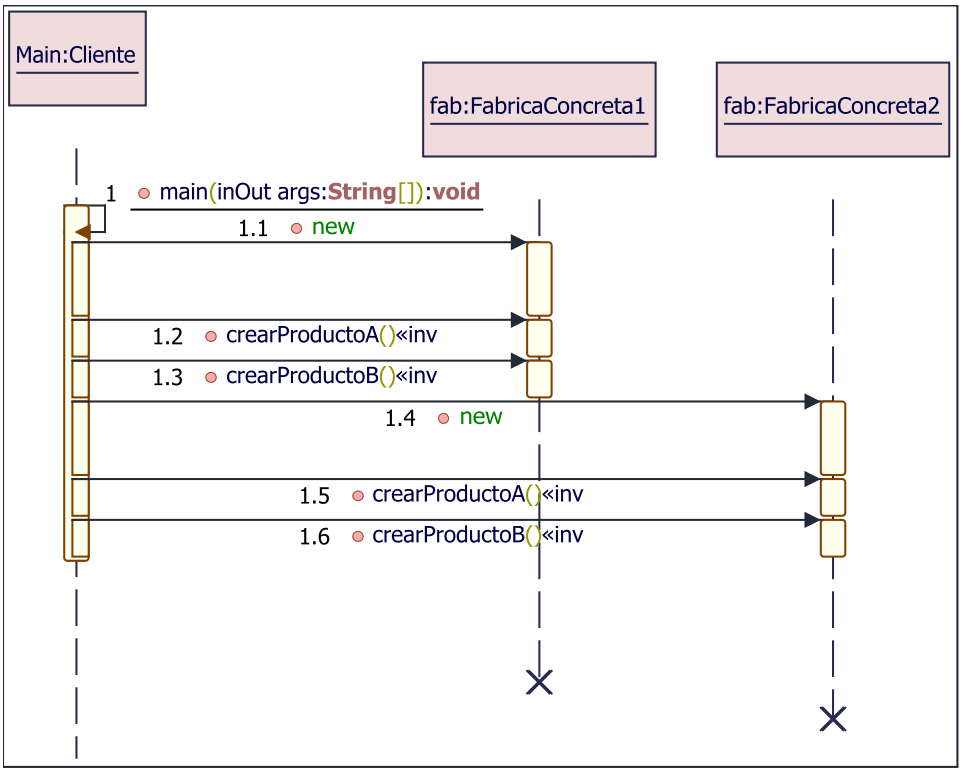


Figura 6.2: Diagrama de secuencia patrón fabrica abstracta

6.3. Adaptador

Patrón perteneciente a los patrones estructurales de Gang of Four, este patrón tambien conocido como Wrapper se usa para adaptar una interfaz así esta podrá ser utilizada por una clase que de uno u otro modo no podría utilizarla.

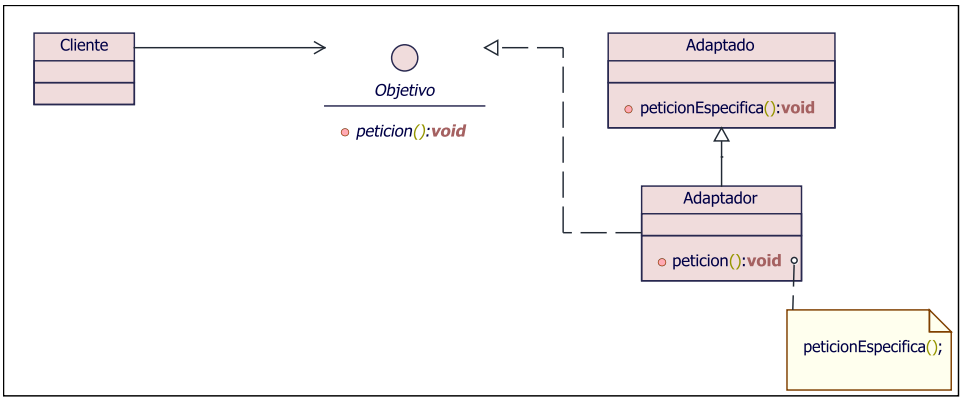


Figura 6.3: Diagrama de clases patrón adaptador

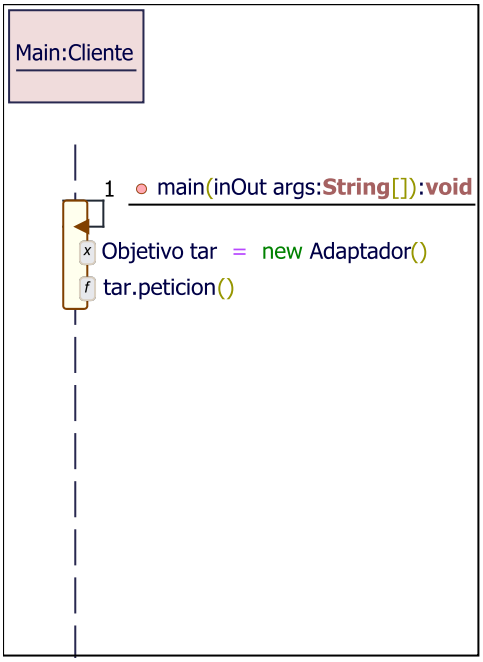


Figura 6.4: Diagrama de secuencia patrón adaptador

6.4. Fachada

Patrón perteneciente a los patrones estructurales de Gang of Four, este patrón se encarga de proveer de una interfaz unificada simple a una interfaz para poder acceder a una interfaz o a un grupo de interfaces de un subsis-

tema.

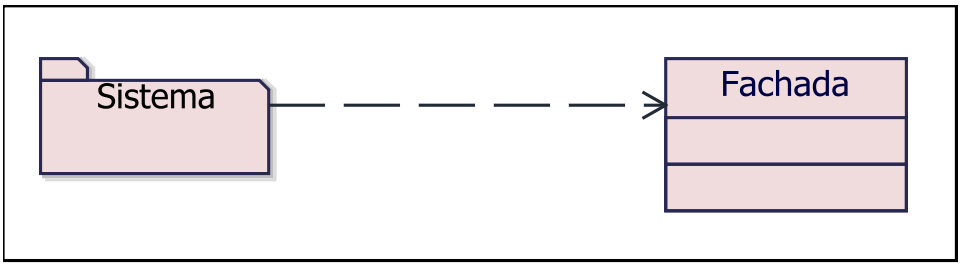


Figura 6.5: Diagrama de clases patrón fachada

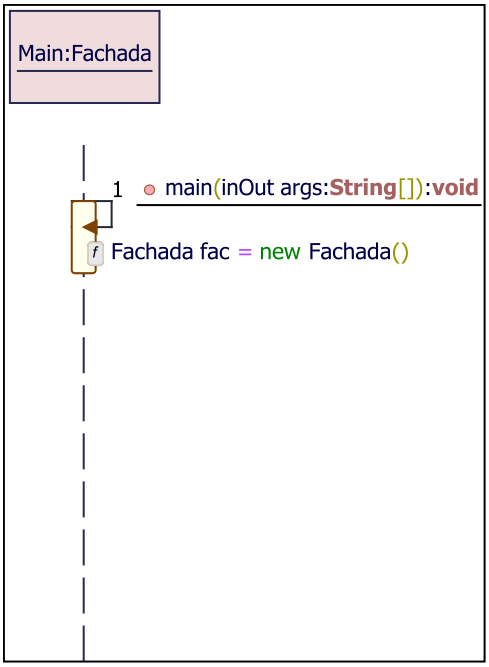


Figura 6.6: Diagrama de secuencia patrón fachada

6.5. Cadena de responsabilidad

Patrón perteneciente a los patrones de comportamiento de Gang of Four, este patrón permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.

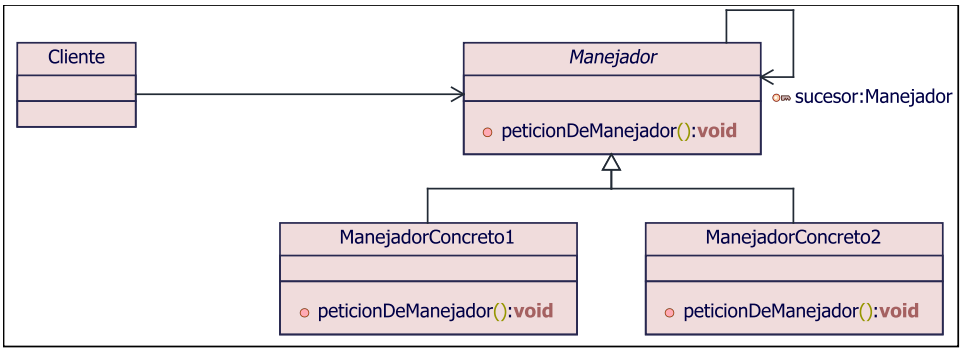


Figura 6.7: Diagrama de clases patrón cadena de responsabilidad

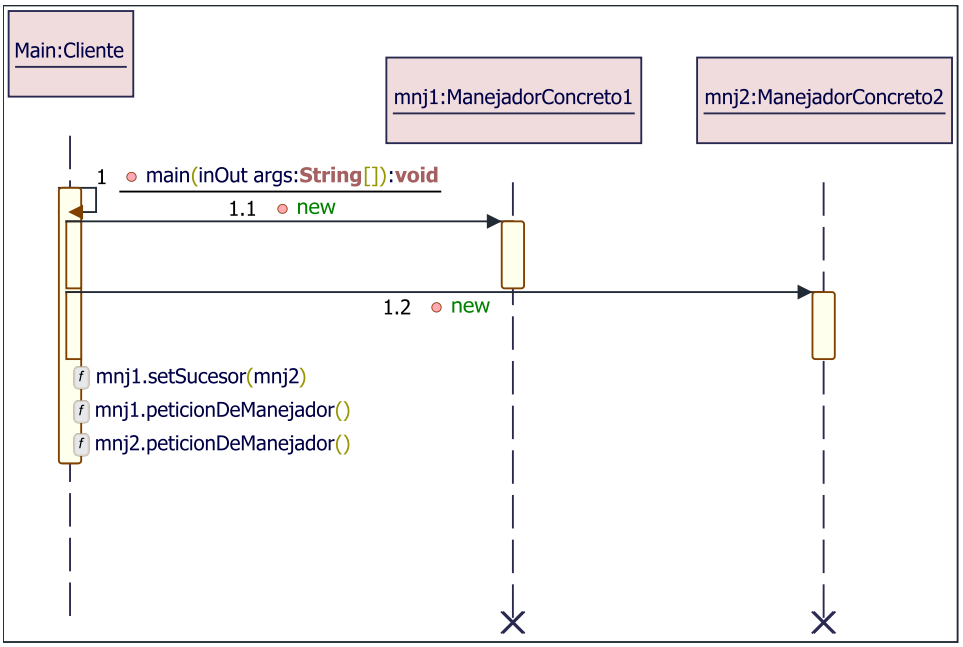


Figura 6.8: Diagrama de secuencia patrón cadena de responsabilidad

6.6. Iterador

Patrón perteneciente a los patrones de comportamiento de Gang of Four, este patrón permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.

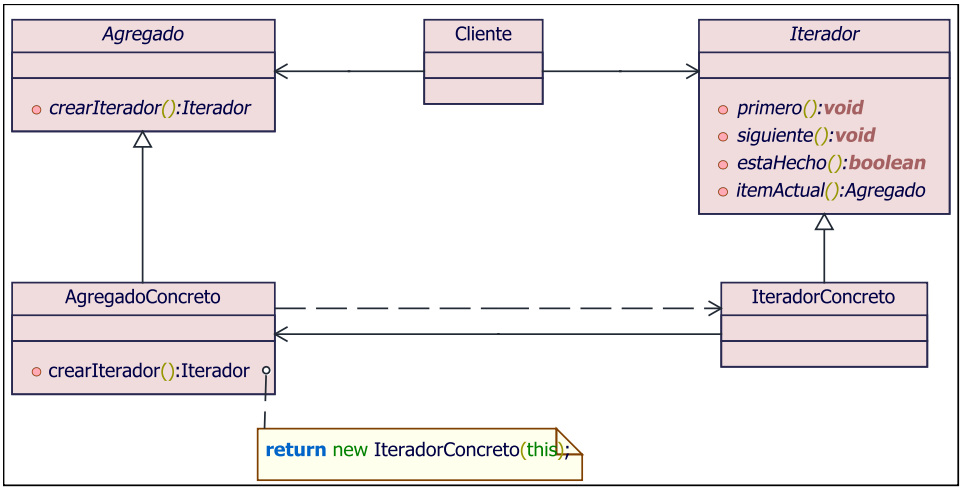


Figura 6.9: Diagrama de clases patrón iterador

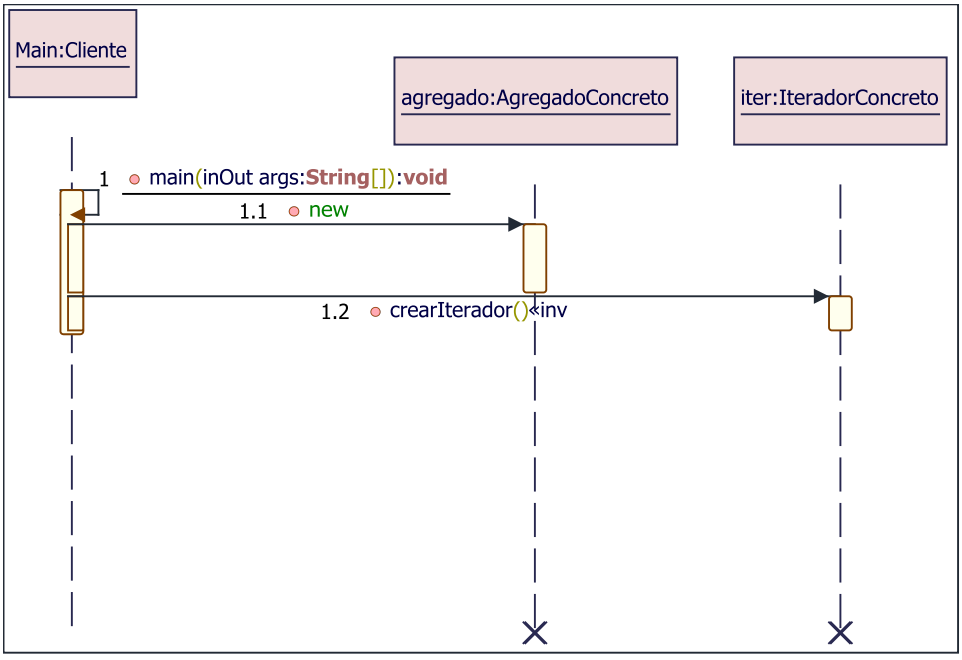


Figura 6.10: Diagrama de secuencia patrón iterador

Capítulo 7

Estados

7.1. Introducción

Es importante poder tanto conocer como predecir el comportamiento de un sistema durante su ejecución, a partir de diversas interacciones y eventos que realice, aparece la posibilidad de dar solución y de controlar de la mejor manera cada uno de los procesos. Un estado se refiere a la condición o valor que asume una variable o un objeto en general a lo largo o en un momento determinado de su ciclo de vida. Para resolver esto surge una herramienta única y veloz como lo son los diagramas de estado, los cuales se transforman en poderosas herramientas puesto que con ellos nos es permitido trazar un comportamiento predecible de un sistema de software durante su ciclo de vida.

7.2. Diagramas de estado

Es una forma de representación gráfica más intuitiva de los autómatas finitos basadas en dígrafos con arcos acotados llamados transiciones en los cuales se ponen los símbolos de tránsito entre un vértice (estado) y otro y se identifican los estados de partida y los de aceptación del resto. Son también representaciones más cómodas para su elaboración, legibilidad y comprensión de distintos tipos de abstracciones computacionales de reconocimiento.

7.2.1. Componentes: Estados

Un estado se refiere a la condición o valor que asume una variable o un objeto en general a lo largo o en un momento determinado de su ciclo de vida, existen distintos tipos de estados; entre los cuales tenemos algunos como:

- Iniciado
- Juegando
- Bloqueado
- Pausado
- Finalizado

7.2.2. Componentes: Transición

Se refiere al paso de un estado a otro estado, el cual se encuentra caracterizado por un disparo, por una condición y una acción y esto constituye a una firma. La transición es motivada por un evento de los reportes que el sistema le da a uno ante la interacción con el mismo, por ejemplo: reacción ante un click, teclado, navegación, entre otros). Las transiciones se representan mediante flechas dirigidas en las cuales se detalla que tipo de transmisión es.

7.3. Diagramas de estado del juego

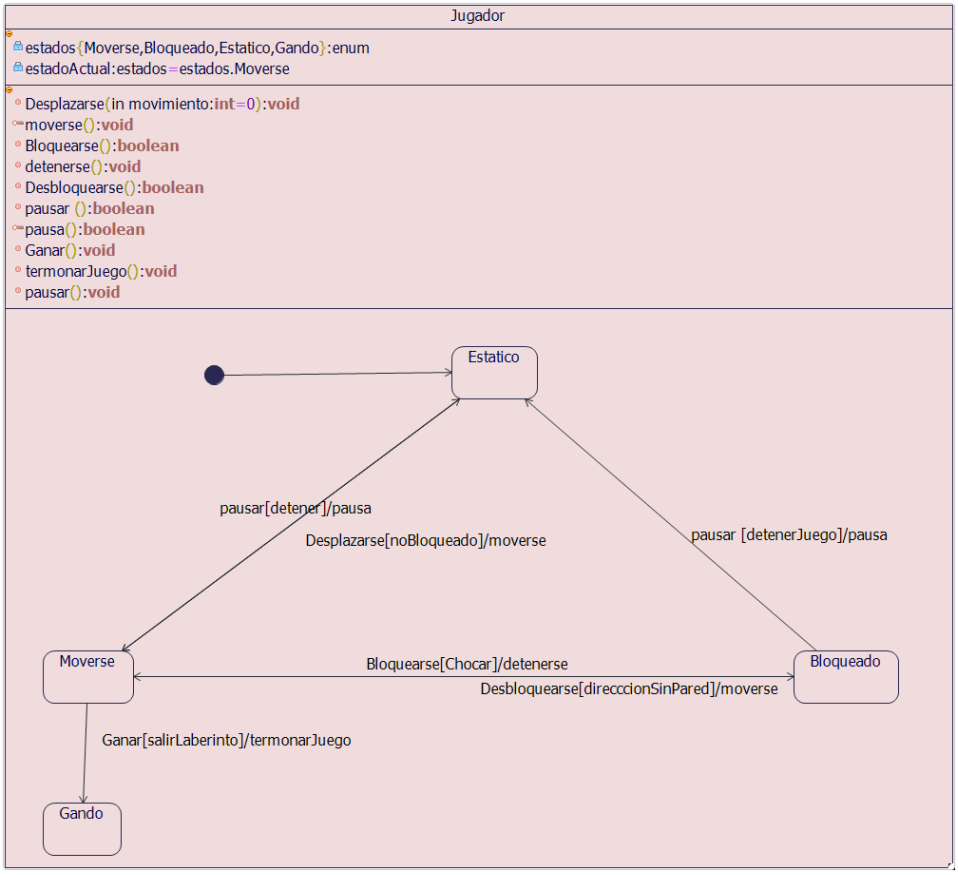


Figura 7.1: Diagrama de estado del juego

Capítulo 8

Componentes

8.1. Introducción

Los componentes se pueden definir como un paquete de software que contiene todo lo necesario para cumplir cierta finalidad dentro de un sistema, es decir, dentro de un desarrollo de software cada componente tiene una funcionalidad que directamente no depende de otro, estos se comunican entre sí mediante interfaces.

Los módulos poseen tres características particulares:

- Un componente es una caja negra (es decir, algo que no se ve, totalmente abstracto) la cual sirve para conectar una abstracción de la cual no nos interesan los detalles de su construcción interna. Ésta construcción interna bien puede estar formado por un modelo de clases, además con organización de paquetes y que finalmente es un sistema que está funcionando allá adentro. Como mencionamos anteriormente la única forma de comunicación entre los componentes es una capa abstracta, es decir, a través de interfaces. Entre las grandes ventajas está que es posible remover los componentes y la aplicaión seguirá funcionando siempre. A través del trabajo por componentes es fácil planear toda un arquitectura.

- Un componente además es un modulo de implementación de interfaces, (refiriéndonos a la elaboración de la abstracción en su mejor nivel, pura (es decir, que va a tener el mínimo de capas posibles), sin contaminación de ninguna elaboración, solo ellas tienen expuesto lo que se

supone uno debe implementar a nivel modular).

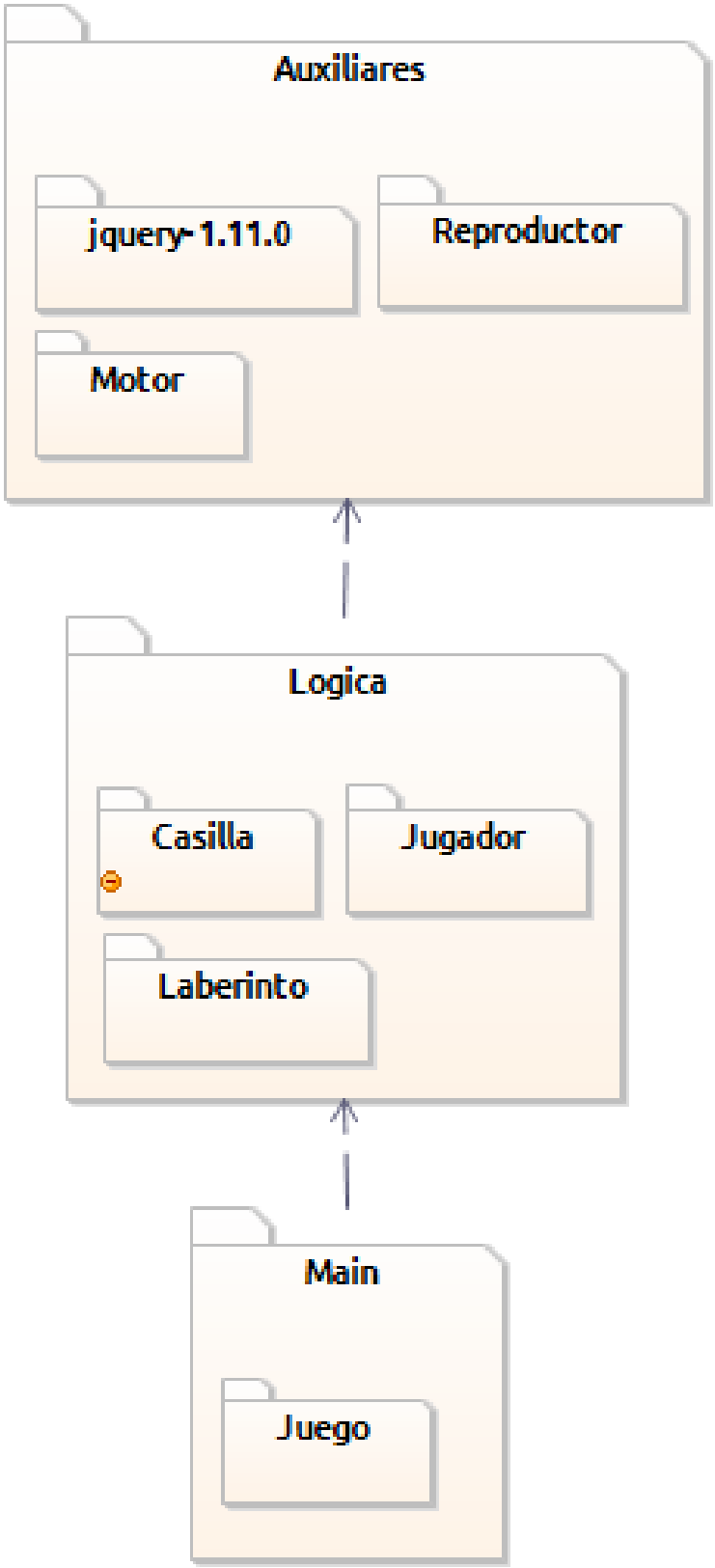
- Los componentes poseen reubicabilidad binaria, la cual tiene que ver con el hecho de que el módulo que ya está elaborado binariamente.

8.2. Interfaces

las interfaces es un término que es utilizado para nombrar a la conexión funcional entre dos sistemas, programas, dispositivos o componentes de cualquier tipo. Además las interfaces proporcionan una comunicación de distintos niveles permitiendo el intercambio de información. Un componente puede tener mezcla de interfaces o s"olo una, es decir, puede tener cualquier combinación. Las interfaces que los componentes usan son:

- interfaz proveída: Es aquella interfaz que suministra el componente y lo exporta para que sea usado por otros componentes.
- Interfaz media luna requerida

8.3. Diagrama de Componentes


Auxiliares
jquery-1.11.0
Reproductor
Motor
Logica
Casilla
Jugador
Laberinto
Main
Juego

Capítulo 9

Metricas

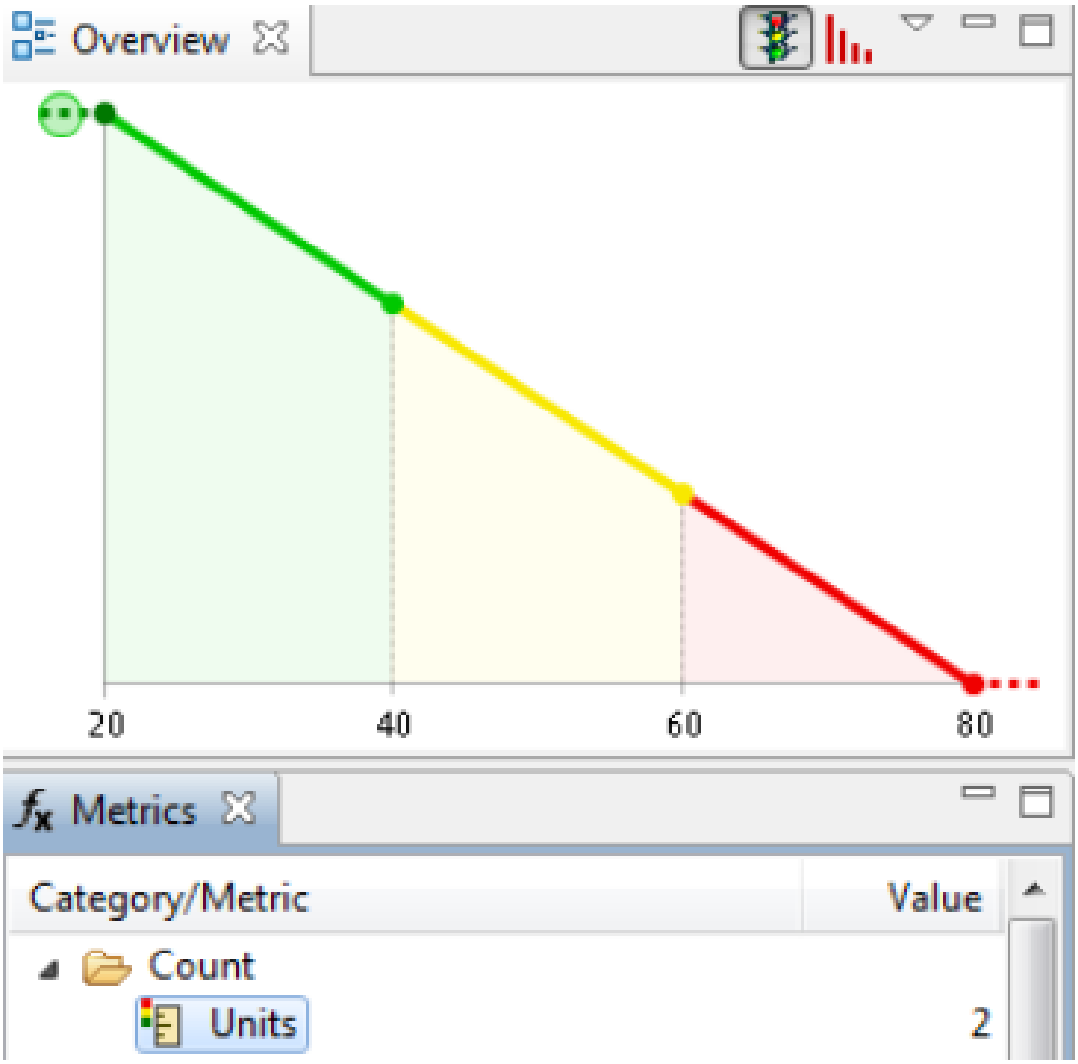


Figura 9.1: numero de clases optima

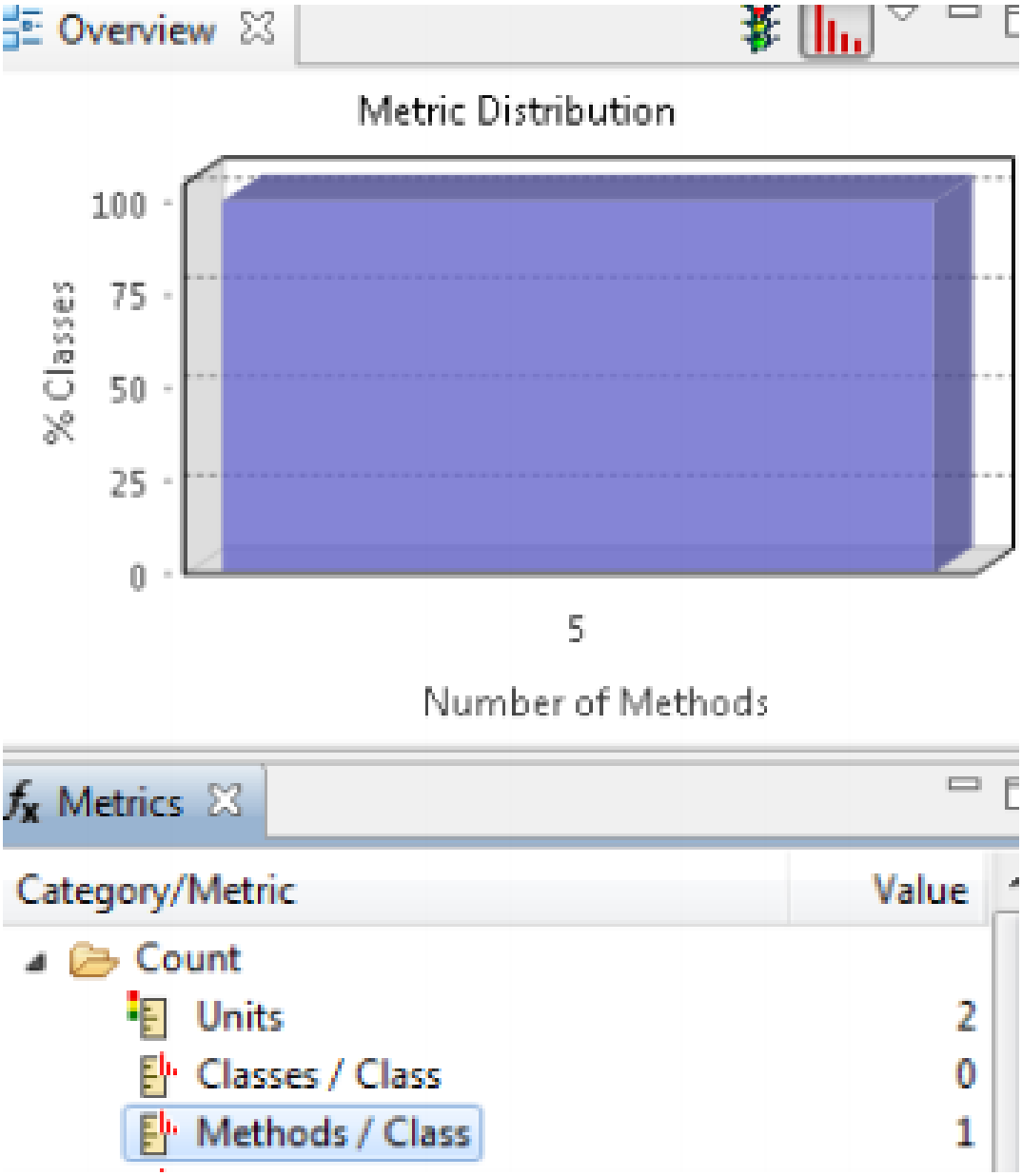
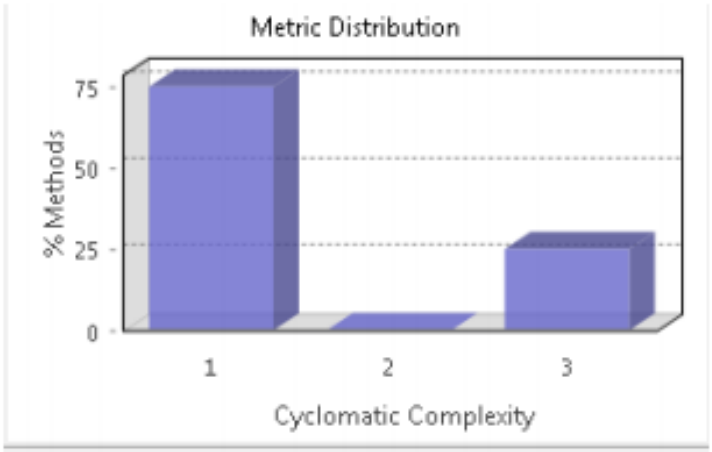


Figura 9.2: numero de metodos y clases

Figura 9.3: Acá se visualiza que la carpeta edu tiene dos paquetes diferentes los cuales son edu.presentacion y edu.cableado.

Figura 9.4: Visualización de la distribución.



Metrics

Category/Metric	Value
Count	
Units	6
Classes / Class	0
Methods / Class	2
Fields / Class	0.5
ELOC	88
ELOC / Unit	14.67
Complexity	
CC	1.5
Fat	1
Tangled	0.00%

Figura 9.5: Complejidad ciclomatica del proyecto..

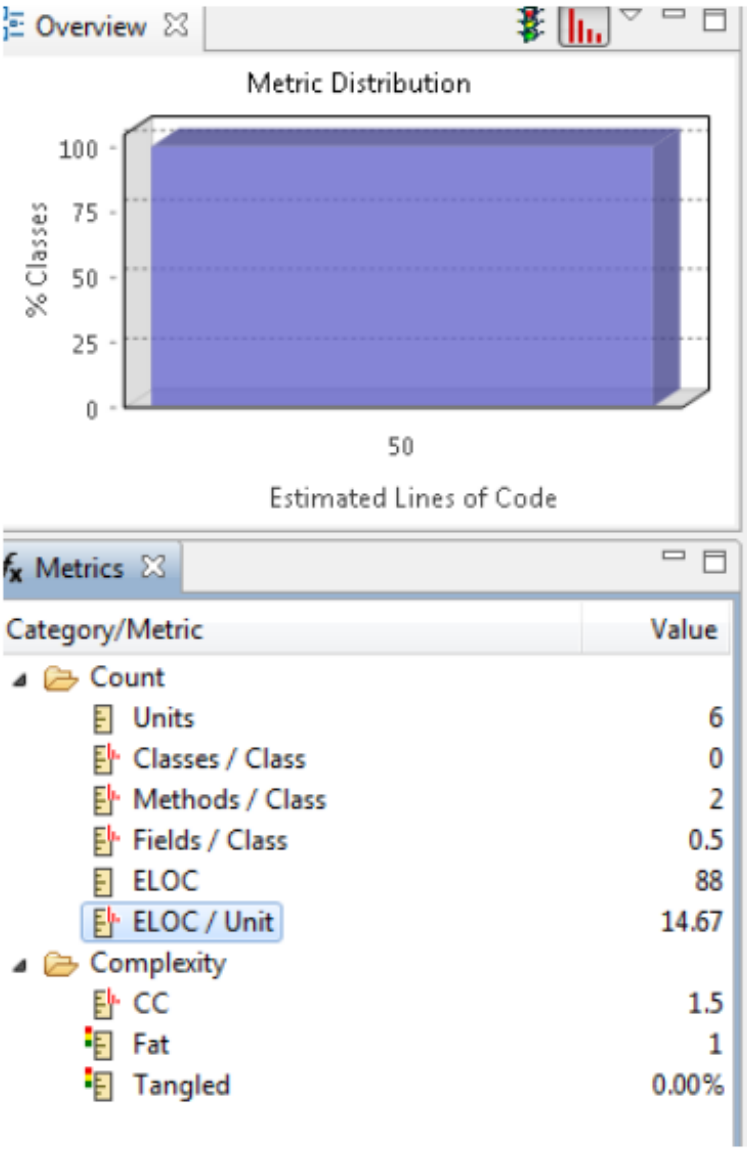


Figura 9.6: Líneas estimadas por la clase top.

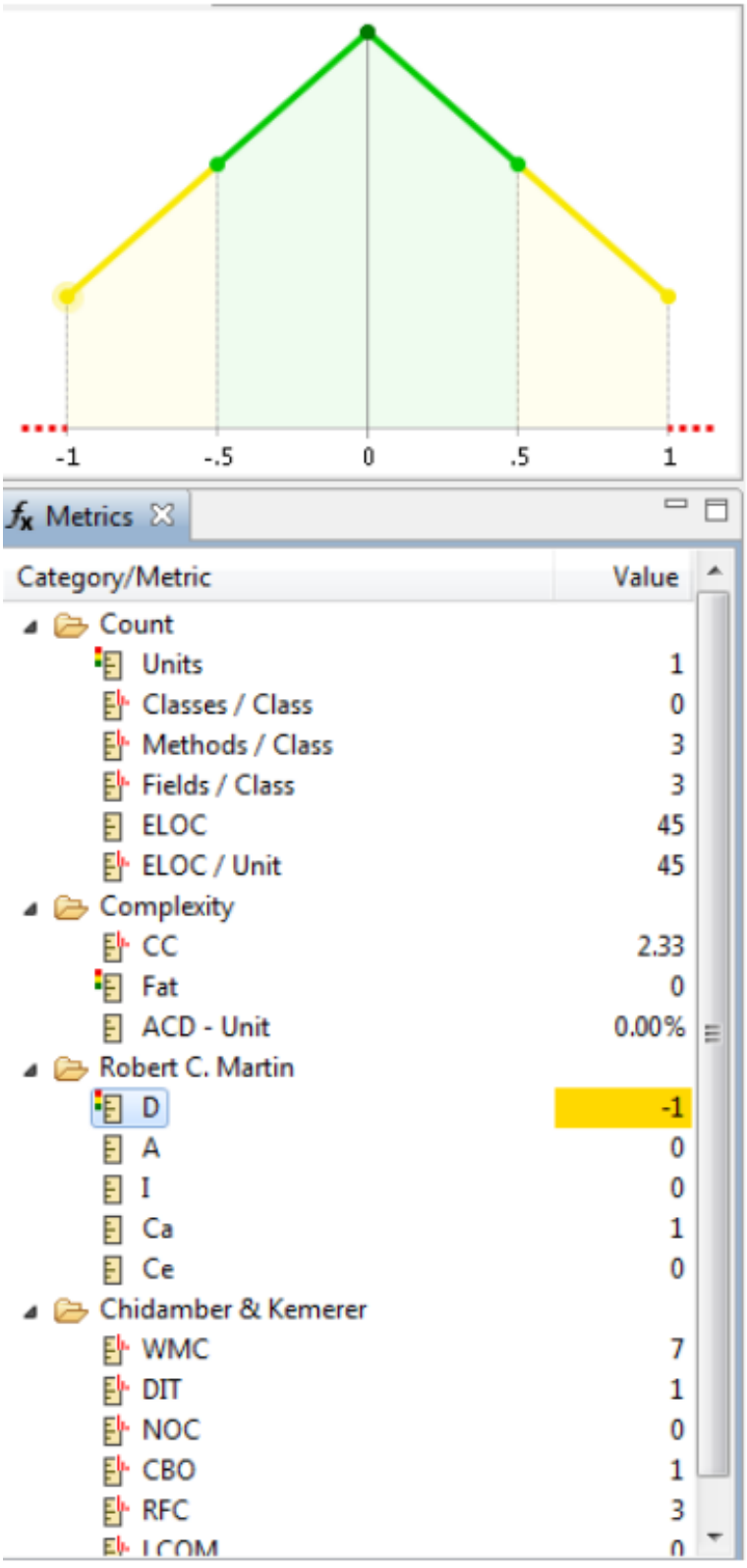


Figura 9.7: Vista de la distancia y su respectivamente.

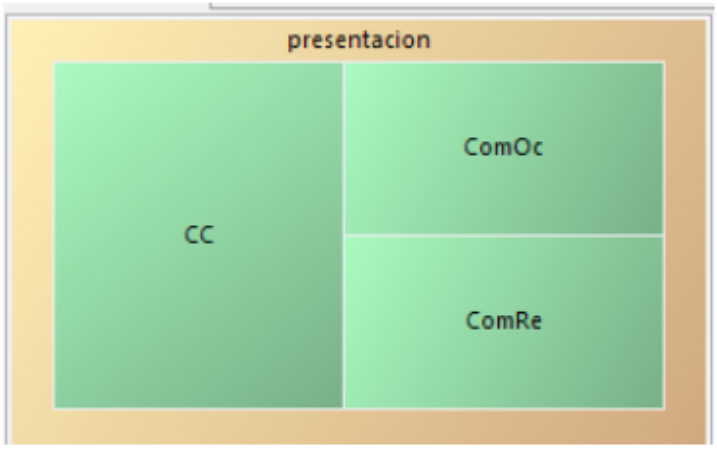


Figura 9.8: Las respectivas clases.

Capítulo 10

Nodos

10.1. Introducción

Los nodos son una forma de representación con los cuales podemos modelar distintos dispositivos como Pc's, procesadores, memorias, distintos dispositivos de red, entre otros. Es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional con memoria y capacidad de procesamiento. Todo aquello que represente un recurso computacional puede ser modelado como un nodo. Por medio de su despliegue los nodos permiten conocer el recurso computacional que se va a usar. Los nodos se pueden utilizar para expresar topologías del hardware sobre el que se ejecuta el sistema, aunque no es el más conveniente para ello. En los nodos pueden estar presentes las relaciones de herencia, asociaciones y dependencias, además los nodos tienen la posibilidad de alojar los componentes.

10.2. Diagrama de Nodos

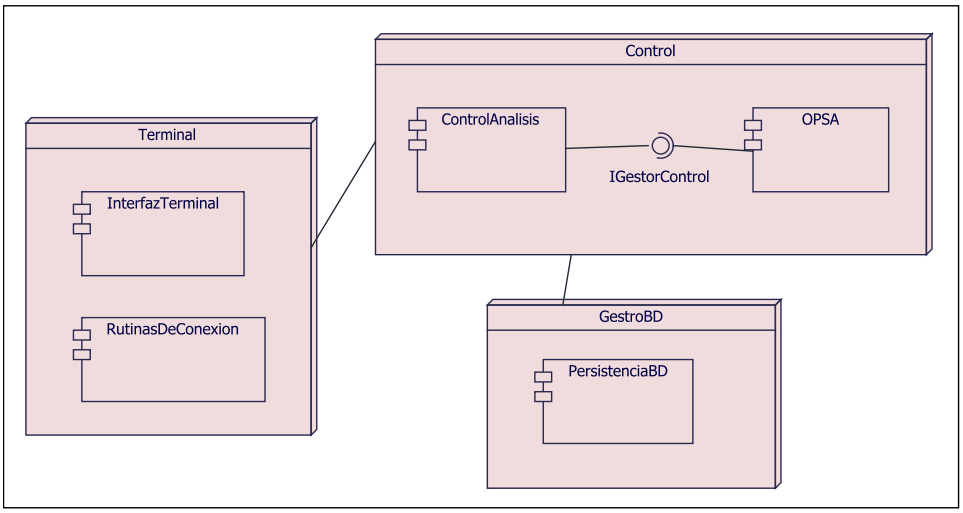


Figura 10.1: Diagrama de Nodos

Capítulo 11

Actividades

11.1. Introducción

Es una herramienta poderosa y muy típica para las representaciones organizacionales, sin embargo, no es la mejor manera de representación pero es una buena opción igualmente. La idea del uso de estos es la de representar actividades, la diferencia está en el nivel de abstracción y en el nivel de abstracción detallado, con estos diagramas podemos modelar el workflow (con el cual se busca establecer cómo la conformación transcurre en estas transmisiones) El work flow modela a partir de particiones y actividades en la cual representamos una máquina de estados sin hacer énfasis en los estados sino en los mensajes que se transmiten, en últimas con estos diagramas buscamos entender como los procesos resuelven algo.

Uno modela a través de estos diagramas: funciones, servicios o tareas de alto nivel, con lo cual nos brinda a nosotros una imagen de como se comporta un determinado sistema. Al igual que los diagramas de estados, hay unos estados en particular, como lo son el estado incial y el final. Estos diagramas además traen consigo situaciones como barras de sincronización que generalmente se utilizan para representar flujos alternativos. También existen las decisiones las cuales permiten hacer ramificaciones.

11.2. Diagrama de Actividades

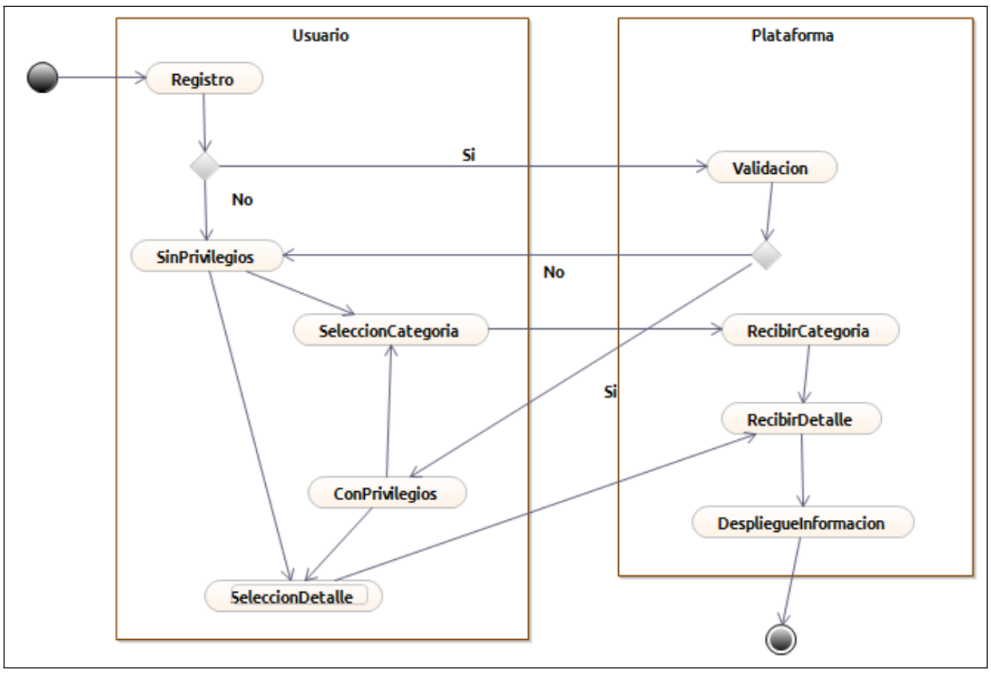


Figura 11.1: Diagrama de Actividades

Parte III

REFLEXIONES

Capítulo 12

Conclusiones

A lo largo del desarrollo de este libro, se ha evidenciado cada paso y cada distinta transformación de una forma de programar, modelar, interpretar y representar todo el proceso de generación de una plataforma denominada Laberinto Musical. En el documento se evidencia paso a paso como se realiza estructuradamente el software a través de resaltar la importancia en el uso de los patrones y las distintas herramientas UML mediante la representación de los diagramas de componentes, de nodos, de actividades y de estados. Además del uso de metodologías es posible llegar a decir que es lo que le da vida y utilidad a todo el sistema y evitar fallas por falta de algún sistema.

La plataforma Lberinto Msical ha tenido una evolución constante tanto en su estructuración, como en su rendimiento y funcionalidad, esto es notable debido a la nueva estructura y fácil manejo de ésta. Es un programa desarrollado por y para las personas invidentes por consiguiente es de fácil uso y accesibilidad, asegurando cumplir con el deseo de alcanzar los diferentes objetivos de entretenimiento. Para el equipo de desarrollo fue totalmente novedoso la reestructuración por componentes y nodos y la creación de un API.

Capítulo 13

Bibliografia

R. Pressman. Ingeniería del Software: Un Enfoque Práctico. 7 Ed. McGraw Hill, 2010. · Sommerville Ian. Ingeniería del Software. Pearson 7a. Edición. 2005.

· S. Sanchez, M. Sicilia, D. Rodriguez. Ingenieria del software. Un enfoque desde la guía SWEBOK. Alfaomega. 2012.

Rumbaugh J., Jacobson I., Booch G. El Lenguaje Unificado de Modelado. Manual de Referencia. Editorial Addison-Wesley – 2000 · Braude Eric. Ingeniería de Software. Una perspectiva orientada a objetos. Alfaomega Grupo Editor S.A. 2003.

· Daniel Bolaños, Almudena Sierra, Idoia Alarcón. Pruebas de software y JUnit. Un análisis en profundidad y ejemplos prácticos. Pearson Education. 2007.

· Booch G; Rumbaugh J; Jacobson I. El Lenguaje Unificado de Modelado. UML 2.0 Addison Wesley. 2006.

· Larman Craig. UML y Patrones. Prentice Hall. 1999. [MEY99] Meyer Beltrand. Construcción de software Orientado a Objetos. Segunda Edición. Prentice Hall. 1999

Bibliografía

Bibliografía

[1] Bolaños Castro,Sandro Javier, *Apuntes de clase Fundamentos de Ingenieria de Software.* 2018.

[2] Bolaños Sandro. Medina Victor Hugo. Ferro Roberto, *MetaProceso de Desarrollo del Software*, 1ra edición, 2016.

[3] Gamma Erich, Helm Richard, Johnson Ralph, Vlissides John, *Design Pattern*, Addison Wesley, 2nd edition, 1994.

[4] Pressman, Roger S, *Ingenieria del software. Un enfoque practico* 2010.

[5] Sommerville, Ian, *Ingenieria del software* 2011.

[6] J. Alfaro, *Code and fix*, Blog.jerti.com, 2018. Disponible: http://blog.jerti.com/2011/07/el-code-and-fix.html.