Reporte
Estudio exploratorio

Resumen

1. Objetivos

El objetivo general de este estudio exploratorio es tratar de tener un primer acercamiento entre desarrolladores y arquitectos de la industria, con la herramienta .jar que permite el uso de anotaciones de código Java para generar reportes y documentar información del Rationale arquitectónico directamente desde el código fuente.

Otro de los objetivos de este estudio exploratorio es refinar el diseño experimental para replicarlo en un posterior experimento controlado. También se pretende obtener resultados a nivel cuantitativo como cualitativo, por parte de los participantes del evento.

2. Planificación

Hipótesis

La hipótesis de este experimento es que la existencia de anotaciones de código fuente con información del Rationale arquitectónico mejora la mantenibilidad de la arquitectura con respecto a la eficiencia, efectividad y comprensión al realizar cambios arquitecturales. Para confirmar esta hipótesis se formula la hipótesis nula y alternativa para cada aspecto de interés.

Eficiencia:

- NULA: La eficiencia al realizar un cambio arquitectural en un sistema con la documentación del Rationale arquitectónico en anotaciones de código es menor o igual a la eficiencia de realizar el cambio sin anotaciones de código.
- ALTERNATIVA: La eficiencia al realizar un cambio arquitectural en un sistema con la documentación del Rationale arquitectónico en anotaciones de código es mayor a la eficiencia de realizar el cambio sin anotaciones de código.

Efectividad:

- NULA: La efectividad al realizar un cambio arquitectural en un sistema con la documentación del Rationale arquitectónico en anotaciones de código es menor o igual a la efectividad de realizar el cambio sin anotaciones de código.
- ALTERNATIVA: La efectividad al realizar un cambio arquitectural en un sistema con la documentación del Rationale arquitectónico en anotaciones de código es mayor a la efectividad de realizar el cambio sin anotaciones de código.

Comprensión:

- NULA: La comprensión de la Arquitectura y su Rationale al realizar un cambio arquitectural en un sistema con la documentación del Rationale arquitectónico en anotaciones de código es menor o igual a la comprensión al realizar el cambio sin anotaciones de código.
- ALTERNATIVA: La comprensión de la Arquitectura y su Rationale al realizar un cambio arquitectural en un sistema con la documentación del Rationale arquitectónico en anotaciones de código es mayor a la comprensión al realizar el cambio sin anotaciones de código.

Variables

Independiente:

En este trabajo la única variable independiente es la existencia o no de anotaciones de código fuente con información del Rationale arquitectónico. Sus únicos valores posibles son “Si” o “No”, esta variable es categórica puesto que representa si un participante debe realizar los cambios a un código con anotaciones de código o sin ellas.

Dependiente:

Para el cálculo de las variables dependientes eficiencia y esfuerzo, es necesario medir otras variables como el tiempo y el nivel de correctitud en la realización de las tareas. Además de definir los valores estimados, con el fin de obtener los resultados de las variables eficiencia y efectividad en términos de porcentajes. Otra variable dependiente es el nivel de comprensión de la Arquitectura y su Rationale, esta se comprende en valores entre 0 y 100. Los participantes deben realizar 3 tareas a las cuales se les califica según el nivel de correctitud de cada tarea, este nivel se describe a continuación:

Nivel Correctitud Tarea 1 (NC1)	
Valor	**Descripción**
100	No realizó el cambio.
200	Realizó cambios, pero no logró terminar la tarea.
300	Completó la tarea, pero no mantuvo el diseño establecido.
400	Completó la tarea manteniendo el diseño, pero tardó más de lo establecido.
500	Completó la tarea manteniendo el diseño y en el tiempo establecido.

Nivel Correctitud tarea 2 (NC2)	
Valor	**Descripción**
100	No realizó el cambio.
200	Realizó modificaciones, pero no logró cambios funcionales.
300	Hizo la tarea cambiando el diseño y manteniéndose dentro de la arquitectura.
400	Hizo la tarea cambiando el diseño y la arquitectura.
500	Hizo la tarea manteniendo el diseño y cambiando la arquitectura.

Nivel Correctitud tarea 3 (NC3)

Valor	Descripción
100	No escribió la documentación.
200	Documentó los cambios a nivel funcional.
300	Documentó los cambios a nivel de diseño.
400	Documentó los cambios a nivel arquitectural sin su Rationale.
500	Documentó los cambios arquitectónicos junto con su Rationale.

La suma de los niveles de correctitud de todas las tareas se define como **'NCT'** y está expresada de la siguiente manera:

$$NCT = \sum_{i=1}^{n} NCi$$

(n: cantidad total de tareas)

El tiempo en que un participante realiza una tarea se denota por la letra **'t'**. El tiempo en que tarda en realizar todas las tareas **'T'** se define como la sumatoria de cada uno de los tiempos tomados en cada tarea, tal como se expresa en la siguiente ecuación:

$$T = \sum_{i=1}^{n} ti$$

Los valores estimados que se definen para el tiempo total y la correctitud total se muestran a continuación: el tiempo estimado para la primera, segunda y tercera tarea es de 1 hora, 50 min y 30min respectivamente, sumando un tiempo total estimado **'Te'** para realizar todas las tareas de 140 min.

$$\boldsymbol{Te = 140}$$

Nivel de correctitud total estimado **'NCTe'** en la realización de todas las tareas asignadas.

$$NCTe = \boldsymbol{NC1\,(500) + NC2\,(500) + NC3\,(500) = 1500}$$

Eficiencia

Valor medido respecto a la correctitud y el tiempo total obtenidos en la realización de las tareas, la eficiencia **'E'** se define a continuación como:

$$E = \frac{NCT}{T}$$

Para dar un valor de la eficiencia en términos de porcentaje es necesario establecer un punto de referencia en la realización de las tareas, el cual se define como:

$$ER = \frac{NCTr}{Te}$$

Donde el nivel de correctitud total referente **'NCTr'** es:

$$NCTr = NC1\,(400) + NC2\,(500) + NC3\,(500) = 1400$$

Con lo cual conseguimos el siguiente valor de eficiencia referente **'ER'**:

$$ER = \frac{1400}{140} = 10\ \frac{NCT}{min}$$

Finalmente obtenemos el porcentaje de eficiencia por cada participante:

$$\%\ eficiencia = \frac{E}{ER} * 100$$

Efectividad

Valor medido mediante la correctitud en la realización de todas las tareas dividido por la correctitud total esperada. En este trabajo se define la efectividad **'EF'** como:

$$EF = \frac{NCT}{NCTe}$$

Finalmente obtenemos el porcentaje de efectividad de cada participante mediante:

$$\%\, efectividad = EF * 100$$

Comprensión

Es un valor cuantitativo evaluado mediante el cruce de información entre las respuestas de una encuesta y la documentación proporcionada en la tarea 3. Para calcular el valor de esta variable se necesita definir el nivel de documentación según el entendimiento expresado por los participantes en los diferentes artefactos del experimento. A continuación, se muestra el diseño para calcular el nivel de comprensión de la arquitectura y su Rationale, mediante la evaluación del contenido en la documentación de las tareas realizadas y lo escrito en las respuestas de la encuesta. Estas variables auxiliares se encuentran en un rango de 0 a 100 sumando un máximo total de 500. El nivel de comprensión denotado como **'C'**, es el promedio de los valores de las variables auxiliares.

Participante	**DC**			**DR**		**C**
	f	**d**	**a**	**d**	**a**	
n	100	100	100	100	100	100

DC: Documentación de los cambios, **DR**: Documentación del Rationale, **f**: Nivel funcional, **d**: Nivel de diseño, **a**: Nivel de arquitectura.

$$C = \frac{DFf + DFd + DFa + DRd + DRa}{5}$$

Participantes

Se hace una convocatoria a través de correos electrónicos, posters y reuniones personales con el fin de obtener la participación de personas que cumplieran con las siguientes características:

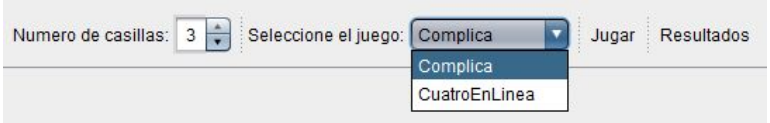
- Conocimientos básicos en Arquitecturas de software.
- Tener experiencia en Java de por lo menos dos años.

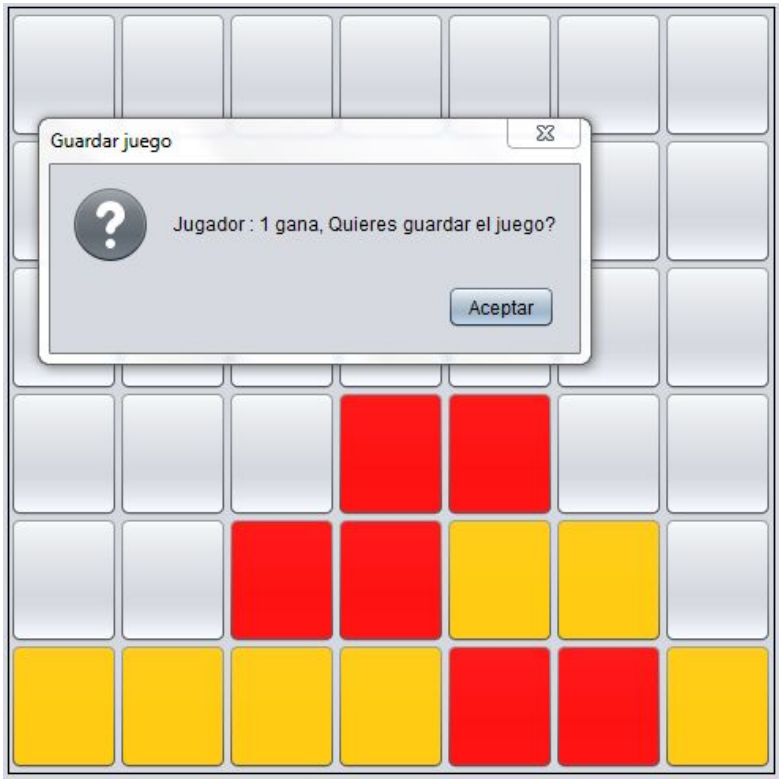
Se realizaron 15 invitaciones a diferentes ingenieros que cumplen con el perfil descrito anteriormente, de cuales 8 de ellos aceptaron. Sin embargo, por diferentes inconvenientes, el día del experimento se presentaron 4 de ellos. Entre los participantes se tienen ingenieros de sistemas y de electrónica que tienen relación en la actualidad con la industria del software. Todos son hombres mayores de 25 años y cuentan con una experiencia entre 2 y 12 años en la industria desempeñando diferentes roles como: analista, ingeniero de producto, desarrollador y

arquitecto de software. Además, algunos participantes tienen una labor activa como docentes en una institución universitaria relacionada con la ingeniería de sistemas. A los participantes se les explicó el objetivo de la investigación y el proceso que se iba a realizar a lo cual se consiguió su consentimiento de forma escrita. Todas las personas seleccionadas tienen conocimientos en campos de la ciencia en computación y afines.

Material experimental

El código fuente experimental es una familia de juegos de 'n' en línea, el cual consiste en un juego en el que cada jugador debe colocar en un tablero 'n' fichas consecutivas en línea de manera vertical, horizontal o diagonal. Este juego tiene muchas variantes, una de ellas se llama 'Cuatro en línea': esta variante es un juego sobre un tablero de siete columnas de ancho y seis filas de alto. Cada jugador usa fichas de un color particular y, alternando el turno, coloca las fichas en el tablero con el fin de lograr ser el primero en lograr que cuatro piezas de su color en una línea, bien sea horizontal, vertical o diagonalmente. Dado que el tablero es vertical, las fichas insertadas en una columna siempre bajarán hasta la posición disponible de la columna.
En las imagenes adjuntas se puede observar la barra de opciones de la interfaz gráfica de usuario, esta tiene como parámetros de entrada el número de casillas contiguas para ganar y el tipo de juego seleccionado y el tablero gráfico se representa mediante una matriz de botones.





Este código fuente se entrega con dos paquetes 'src' y 'lib'. En 'src' se encuentran todas las clases y paquetes que representan la arquitectura del juego y en 'lib' está la librería 'reflections-0.9.9-RC1.jar' y sus dependencias, la cual permite obtener las clases que heredan de algún tipo. Los participantes que tienen las anotaciones de código con información del Rationale Arquitectónico declaradas en el código fuente tienen una librería de más en la carpeta 'lib' llamada 'ARAT.jar' (Architectural Rationale Annotations Tool: por sus siglas en inglés) la cual permite documentar las razones arquitecturales y mostrar un reporte con la ubicación y la información marcada en las anotaciones. El modelo de anotaciones ARAT se muestra a continuación:

```
@Documented
@Retention(RUNTIME)
@Target({METHOD,PACKAGE,TYPE})
public @interface Rationale1 {
        String id();
        String componentName();
        String motivation() default "";
        String justification();
}
```

Esta anotación se conforma de cuatro atributos miembros:

- id: Identificador para el Rationale Arquitectónico.

- componentName: Nombre del componente que se está marcando con la anotación.
- motivation: Texto que describe la necesidad de calidad que se desea alcanzar.
- justification: Texto que describe las razones por las cuales se toma una decisión que cause un impacto arquitectural.

La meta-anotación @Document incluye todos los componentes marcados con la anotación en el documento generado por javadoc, @Retention(RUNTIME) retiene el archivo de clases en tiempo de ejecución para poder acceder a la información a través de reflexión, finalmente con @Target definimos qué tipos de componentes se desean anotar, en este caso se quiere anotar componentes a nivel de método, clase y paquete.

Es necesario instalar el JDK de Java y algún entorno de desarrollo como Eclipse o NetBeans IDE. El código fuente ubicado en 'src' se puede copiar y reemplazar con la carpeta 'src' creada por Eclipse o NetBeans IDE cuando se inicia un nuevo proyecto. La carpeta 'lib' se copia junto a 'src' en el proyecto creado en el IDE y se deben agregar las librerías al Path del proyecto.

Finalmente, todos los participantes cuentan con un Documento de Arquitectura de Software SAD (Software Architectural Document: por sus siglas en inglés), en donde se representa la arquitectura desde diferentes vistas: escenarios, lógica, de desarrollo, de proceso y la vista física, con diferentes modelos como: Diagramas de casos de uso, de clases, de secuencia, de paquetes, de componentes y de despliegue. Los cuales permiten representar la arquitectura para diferentes audiencias.

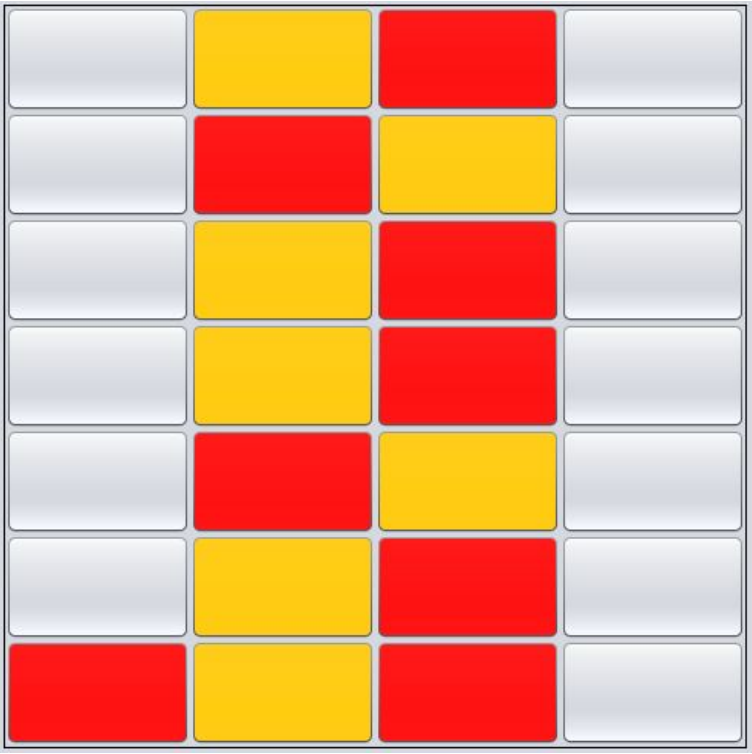
Tareas

El experimento consiste en la realización de tres tareas para cada participante:

1. Mantenimiento aditivo (Nuevo juego): Es una variante del juego N en línea llamada Complica, en donde el tablero está constituido por cuatro columnas y siete filas.
2. Mantenimiento correctivo (Cambio arquitectural)
3. Documentación del Rationale Arquitectónico (Documentación)

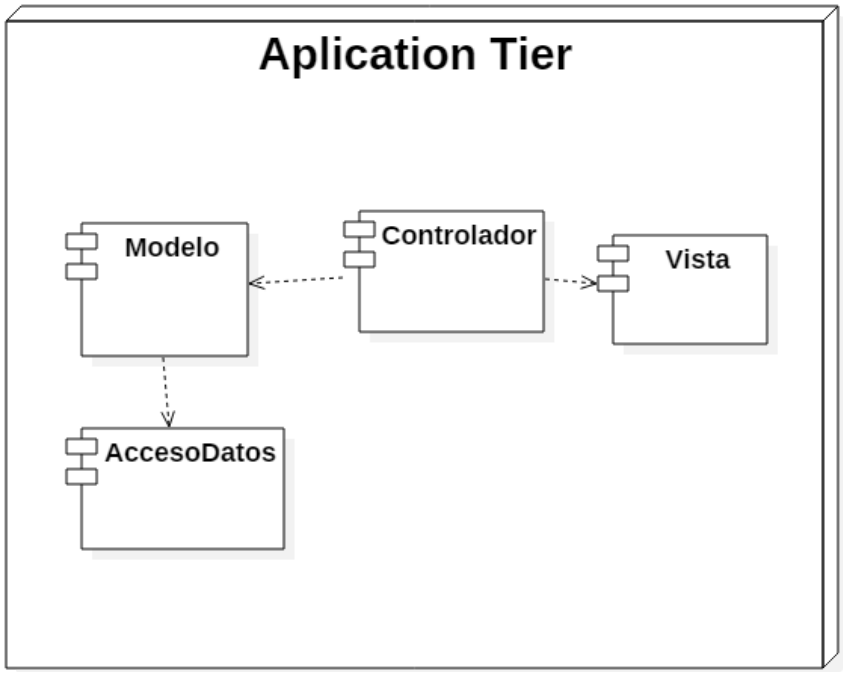
Tarea 1 Mantenimiento aditivo (Nuevo juego)

COMPLICA: Las fichas pueden ser colocadas en una columna llena, en este caso la ficha se inserta y todas las fichas descienden una posición, eliminando las fichas de abajo. Como consecuencia de esta regla, se presentan situaciones más complejas, por ejemplo, ambos jugadores pueden en una misma jugada lograr colocar n de sus fichas en línea, y entonces el juego deberá continuar. También el jugador que coloca una ficha podría perder el juego. Esto sucedería si la pieza es ubicada en una columna llena y consigue que su oponente quede con n de sus fichas en línea. En la siguiente figura se muestra el tablero con los cambios esperados:

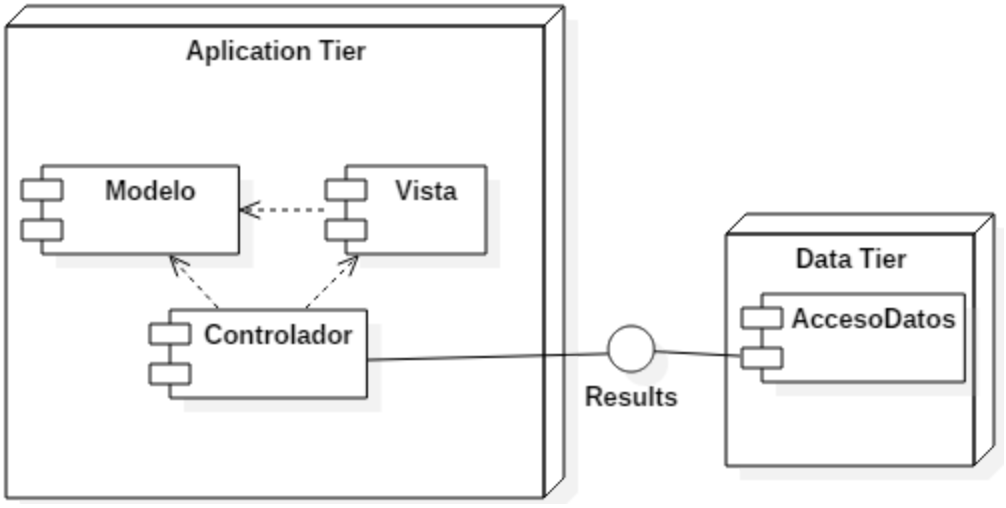


Tarea 2 Mantenimiento correctivo (Cambio arquitectural)

El resultado de un juego se guarda localmente en un archivo ubicado en un directorio del sistema, este directorio puede ser accedido por cualquier persona que sepa dónde se ubican los archivos del proyecto. Se requiere que el sistema tenga la capacidad de delegar la responsabilidad de gestionar los archivos con la información de los resultados a un servidor que esté activo escuchando las peticiones de las instancias del juego. Esto con el objetivo de mantener seguros los datos en un sistema aparte del juego para que no puedan ser modificados y estos puedan ser accedidos por varias instancias del juego al mismo tiempo. En la figura siguiente se puede apreciar un diagrama de despliegue con la arquitectura inicial del sistema, en la cual podemos ver que el componente de acceso a datos está en el nivel de aplicación.



En la siguiente imagen se muestra cómo deben ubicarse los componentes, de tal forma que la responsabilidad de guardar los datos caiga sobre un nuevo nivel de datos.



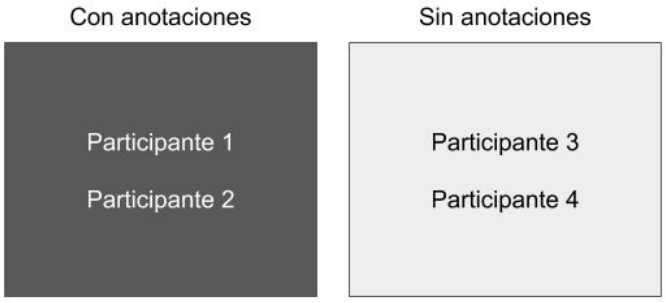
Tarea 3 Documentación del Rationale

Con el objetivo de verificar la comprensión de la arquitectura y su Rationale, es necesario que los participantes documenten los cambios realizados en las tareas anteriormente mencionadas de acuerdo a la estrategia de documentación que le haya correspondido, documento de texto/word o con el modelo de anotaciones ARAT. Los participantes también deben escribir cuál es la necesidad de calidad que se quiere alcanzar y cuales técnicas o estrategias se utilizan. Para

terminar esta tarea los participantes se pueden documentar en los diferentes artefactos del software como el Documento de Arquitectura Software SAD (Software Architecture Document: por sus siglas en inglés), el código fuente en bruto y las anotaciones de código si le corresponde.

Diseño del experimento

Los participantes se agrupan inicialmente en 2 parejas, el procedimiento de agrupación se realiza utilizando la herramienta web https://echaloasuerte.com/draw/new/groups/, la cual permite crear grupos aleatorios adjuntando los nombres de los participantes separados por coma. En la siguiente imagen se puede observar la distribución de los grupos:



El directorio del experimento se describe a continuación: en primer lugar, tiene el código fuente en src, con anotaciones para un grupo y sin las anotaciones para el otro; en lib tiene las librerías para reflexión y el modelo de anotaciones si le corresponde, en EjemploSockets se tiene un ejemplo de una implementación de sockets que se sugiere para la tarea 2 y finalmente se cuenta con el Documento de Arquitectura Software en PDF.

Procedimiento

El experimento inicia haciendo una introducción al Rationale Arquitectónico con el objetivo de familiarizar el término a los participantes, seguido de esto se hace la presentación de las anotaciones de código fuente en Java, realizando un ejemplo para comprender la declaración, el uso y algunas restricciones. Después, se muestra el modelo de anotaciones con información del

Rationale Arquitectónico y se procede a explicar el sistema y las tareas que se deben realizar sobre este. Finalmente se hace la selección de los grupos de manera aleatoria, se entrega el material experimental adecuado y se empiezan a tomar tiempos en la terminación de cada tarea. Por último, después de realizar todas las tareas y entregar los resultados, cada participante debe completar una encuesta con las siguientes preguntas:

1. Describa las razones por las cuales el código fue organizado bajo la estructura sobre la cual trabajó.
2. Describa las razones por las cuales la estructura arquitectónica sobre la cual trabajó tuvo que ser alterada para satisfacer los cambios solicitados.
3. ¿La información del Rationale brindada en esta experiencia fue de utilidad para realizar las modificaciones?
4. ¿Por qué cree usted que es importante documentar el Rationale Arquitectónico en el desarrollo de software?
5. ¿Qué información considera es la más relevante para documentar el Rationale de la Arquitectura?

Ejecución

A los participantes se les da una introducción del término Rationale Arquitectónico en donde se mencionan algunos problemas que se causan no documentarlo. También se les da una capacitación sobre el uso de las anotaciones de código en Java, algunos casos de uso populares y un ejemplo en el cual se busca aclarar los conceptos de anotaciones de código. Después, se explica el modelo de anotación con el que se realiza el experimento. Seguido de esto, se les explica el funcionamiento del sistema y algunas partes de su arquitectura. Finalmente se explican las tareas que se deben realizar, cómo crear un nuevo proyecto en NetBeans IDE y cómo agregar las clases y paquetes del sistema al nuevo proyecto. El desarrollo del sistema se implementó en Eclipse IDE, por lo cual se pensaba en realizar los cambios del experimento en el mismo IDE. Sin embargo, se separan las clases y los paquetes del proyecto en una carpeta aparte ‘src’ junto con las librerías necesarias para su correcto funcionamiento ‘lib’. Esto permitió desarrollar el experimento en el entorno de desarrollo NetBeans, el cual se encuentra preinstalado en las salas de la Universidad del Cauca.

Después de capacitar a los participantes se les hace entrega del material experimental. Se hace la configuración del proyecto en el entorno de desarrollo NetBeans y se copian los archivos descargados del código fuente del material experimental. Los participantes empiezan a realizar las tareas y se capturan los tiempos de realización de cada tarea para cada participante. A medida que cada participante finaliza las tareas se le entrega la encuesta, con el objetivo de

tener resultados cualitativos que aporten en la definición de la estructura final del modelo de anotaciones para el Rationale Arquitectónico.

A continuación, se muestran algunas de las respuestas más acertadas a la encuesta realizada a los participantes, las evidencias de estas respuestas se encuentran el ítem de Materiales y recursos al final de este reporte:

1. Describa las razones por las cuales el código fue organizado bajo la estructura sobre la cual trabajó.

   RTA: El sistema fue organizado así para facilitar la mantenibilidad y la extensibilidad de la arquitectura.

2. Describa las razones por las cuales la estructura arquitectónica sobre la cual trabajó tuvo que ser alterada para satisfacer los cambios solicitados.

   RTA: Porque se requirió una nueva funcionalidad en un nuevo componente donde se guardarán los datos por seguridad.

3. ¿La información del Rationale brindada en esta experiencia fue de utilidad para realizar las modificaciones?

   RTA: En términos generales los participantes respondieron que sí fue de utilidad la estrategia de documentación que les correspondió.

4. ¿Por qué cree usted que es importante documentar el Rationale Arquitectónico en el desarrollo de software?

   RTA: A nivel general, los participantes respondieron a esta pregunta, es importante para facilitar la mantenibilidad de un sistema a personas subsiguientes a los que lo desarrollan.

5. ¿Qué información considera es la más relevante para documentar el Rationale de la Arquitectura?

   RTA: En esta pregunta los participantes nombraron los siguientes aspectos que les parecieron importantes:

   - Tipo de requerimiento a satisfacer
   - Prioridad del requerimiento
   - Historial de modificaciones al Rationale

En las Tablas a continuación, se muestran los tiempos capturados en cada tarea para cada uno de los participantes de los dos grupos, el nivel de correctitud evaluado en las tareas de cada participante y los valores para el nivel de comprensión de la Arquitectura y su Rationale.

Tiempos

Grupo Sin Anotaciones GSA				
Participante	**t1**	**t2**	**t3**	**T(min)**
1	101	32	16	149
2	54	35	46	135
Grupo Con Anotaciones GCA				
1	52	90	2	144
2	104	45	2	151

Niveles de Correctitud

Grupo Sin Anotaciones GSA				
Participante	**NC1**	**NC2**	**NC3**	**NCT**
1	300	300	200	800
2	500	500	400	1400
Grupo Con Anotaciones GCA				
1	500	300	400	1200
2	400	400	500	1300

Comprensión en documentación

	DC			**DR**		**C**
GSA	**F**	**D**	**A**	**D**	**A**	
1	100	100	50	0	0	50
2	100	30	100	30	50	62
GCA						

1	100	100	100	100	50	90
2	100	100	100	50	50	80

Resultados

Luego de tomar los tiempos de cada participante se reciben los sistemas con las modificaciones con el objetivo de evaluar el nivel de correctitud en la realización de las tareas. Para determinar los resultados del experimento se realiza una prueba de hipótesis con la cual se busca confirmar que el uso de las anotaciones de código con información del Rationale Arquitectónico mejora la mantenibilidad de la Arquitectura en términos de eficiencia, efectividad y comprensión de la Arquitectura y su Rationale, cuando se realizan cambios con un impacto a nivel arquitectural. La prueba de hipótesis que se utiliza es una prueba t student, esta permite determinar si existe una significancia estadística entre dos variables, esto quiere decir que la presencia de la variable independiente afecta positivamente los resultados de las variables dependientes. Para esta prueba es necesario tener dos muestras iguales y se debe suponer igual varianza entre las muestras de cada grupo. En este trabajo los grupos se dividen en: Grupo Con Anotaciones (GCA) y Grupo Sin Anotaciones (GSA).

Cada una de las hipótesis considera una variable independiente en una escala nominal con dos niveles (con anotaciones, sin anotaciones) y una variable dependiente (efectividad, eficiencia o comprensión de la Arquitectura y su Rationale). Para confirmar las hipótesis se utiliza una prueba de t student suponiendo una distribución normal, varianzas y tamaños de muestras iguales, para dos medidas independientes. Debido a que nos interesa saber si los resultados están por encima de la distribución normal se hace uso de una prueba t student de una cola.

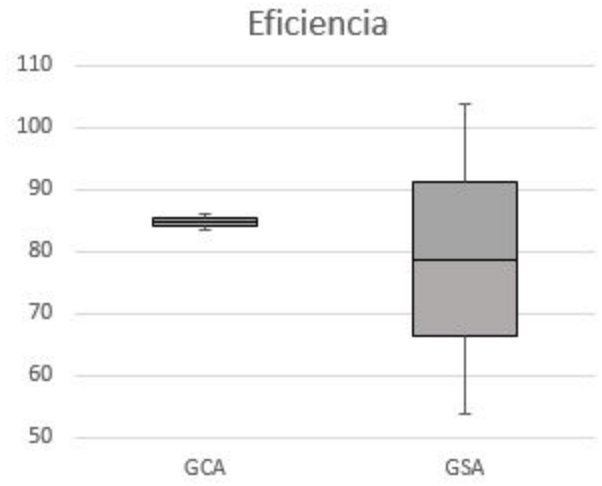
% Eficiencia		
Participante	**GCA**	**GSA**
1	83,3	53,7
2	86,1	103,7
% Efectividad		
1	80,0	53,3
2	86,7	93,3
Comprensión		

1	90,0	50,0
2	80,0	62,0

Una vez calculadas las variables dependientes se hace el contraste de hipótesis mediante la prueba de t student con un nivel de significancia igual a 0.05.

Eficiencia

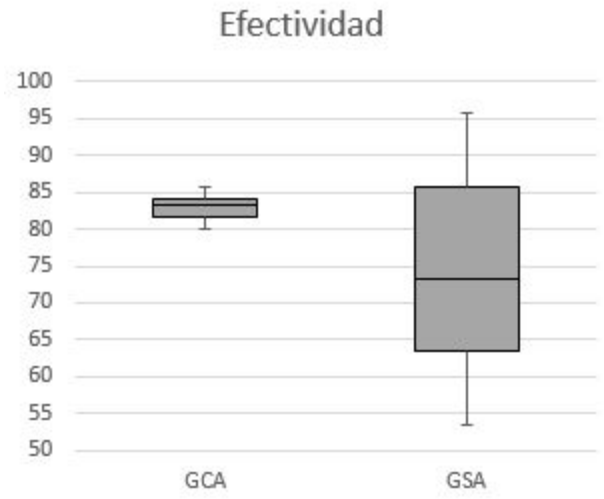
El porcentaje de eficiencia promedio para el grupo con anotaciones de código fue del 84.7% a diferencia del grupo sin anotaciones de código el cual tiene un porcentaje promedio de eficiencia del 78,7%. Sin embargo, el p-valor es de 0.4913 el cual es mucho mayor que alfa (0.05), por lo tanto, aceptamos la hipótesis nula y rechazamos la hipótesis alternativa. Esto significa que el uso de anotaciones de código con información del Rationale Arquitectónico, no mejora la eficiencia en la realización de cambios con un impacto arquitectural en este experimento. En la imagen siguiente, se puede observar la distribución de los datos con respecto a la mediana de la eficiencia de cada grupo, en donde podemos ver que la mediana de eficiencia es mayor en el grupo con anotaciones de código, sin embargo, los resultados se mantienen dentro del rango del grupo de sin anotaciones de código, por lo cual podemos afirmar que el aumento en la mediana para el grupo con anotaciones se debe a otros factores y no al uso de anotaciones de código con información del Rationale Arquitectónico.



Efectividad

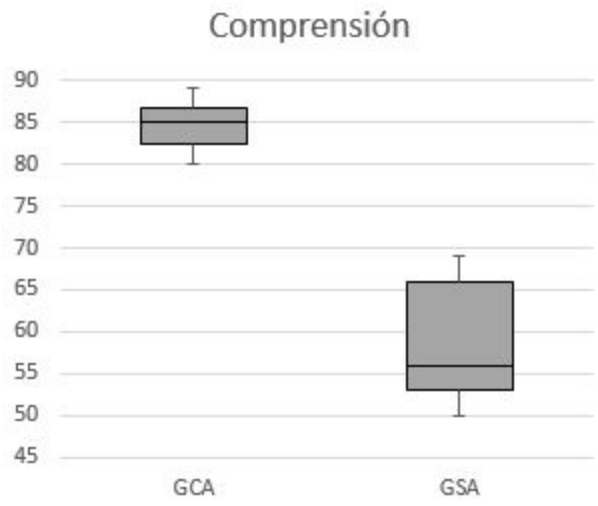
Los resultados muestran que el promedio de efectividad para el grupo con anotaciones de código es de 83,3% y el porcentaje promedio para el grupo sin anotaciones de código es del 73,3%. Sin embargo, con el p-valor de 0.4369 mayor a 0.05 se debe rechazar la hipótesis

alternativa y aceptar la hipótesis nula, esto significa que el uso de las anotaciones de código con información del Rationale Arquitectónico no tiene una significancia estadística sobre la efectividad al realizar cambios con un impacto arquitectural en este experimento. En la siguiente imagen, tenemos un resultado similar al de la eficiencia, en donde los resultados para el grupo de anotaciones son mayores, sin embargo, estos se mantienen en el rango de valores que el grupo de sin anotaciones de código, por lo tanto, confirmamos que el aumento de efectividad para el grupo de anotaciones de código no se debe al uso de anotaciones de código fuente.



Comprensión

El promedio de comprensión para el grupo con las anotaciones de código es de 85%, a diferencia del grupo sin anotaciones el cual es de 56%. Cuando se calcula el p-valor da como resultado 0.0327, el cual es menor a nuestro alfa 0.05, por lo tanto, aceptamos la hipótesis alternativa y rechazamos la hipótesis nula. Esto significa que el uso de anotaciones de código tiene una significación estadística sobre la comprensión de la arquitectura y su Rationale al realizar cambios con un impacto arquitectural en este experimento. Finalmente, podemos ver en la siguiente imagen que los valores para el grupo con anotaciones de código son mayores y están por fuera del rango de los resultados del grupo sin anotaciones de código, por lo cual, podemos afirmar que el aumento en la comprensión de la Arquitectura y su Rationale se debe al uso de anotaciones de código fuente con información estructurada del Rationale.



Materiales y recursos

Para este experimento se hace uso de los siguientes recursos informáticos:

- Alguno de estos IDE (Integrated Development Environment):
  - NetBeans
  - Eclipse
- El modelador de Software: StarUML
- Las herramientas del Sistema Operativo
  - Visor de PDF
  - Visor de imágenes
- Documentación
  - Diagramas en imágenes .png
  - Diagramas en .mdj(StarUML)
  - SAD: Software Architecture Document
- Código fuente Con anotaciones
  - com: Paquete con el código fuente y las anotaciones de código
  - lib: 7 librerías .jar utilidades para Reflexión y 1 .jar con el modelo de anotaciones ARAT
- Código fuente Sin anotaciones
  - com: Paquete con el código fuente
  - lib: 7 librerías .jar utilidades para Reflexión
- Ejemplo de una implementación de Sockets

Los códigos fuente de prueba junto con la documentación de cada sistema se encuentran en el siguiente repositorio en GitHub: https://github.com/zahydo/experimento-ARAT.git. En el final de este documento se anexa el Documento de arquitectura con todos los diagramas relacionados.

FInalmente, se muestran los resultados de la encuesta realizada a los participantes del estudio exploratorio.

Encuesta Rationale Arquitectónico

En nombre del grupo IDIS agradecemos su participación en este experimento, los datos individuales no serán de conocimiento público (sólo usaremos su perfil), ya que son los resultados que la experiencia nos brindará la información para responder nuestras preguntas de investigación. A continuación, haremos un conjunto de preguntas que nos permitirán indagar algunos aspectos relevantes a la documentación del Rationale arquitectónico, luego haremos una sección de discusión conjunta para tener una perspectiva más amplia de sus apreciaciones.

Nombre: Sebastián Felipe Londiñez García

Experiencia en desarrollo de software

Tiempo: Profesional (1.5 años), estudiantil (5 años).

Lenguajes y plataformas: Ruby on Rails, C, Python, Android, Java, Javascript, Node, R, Matlab, PHP, sistemas embebidos, etc.

Roles desempeñados: Ingeniero de producto.

.

Tipos de organizaciones (y nombres si Usted lo desea):

Codesaurus.

.

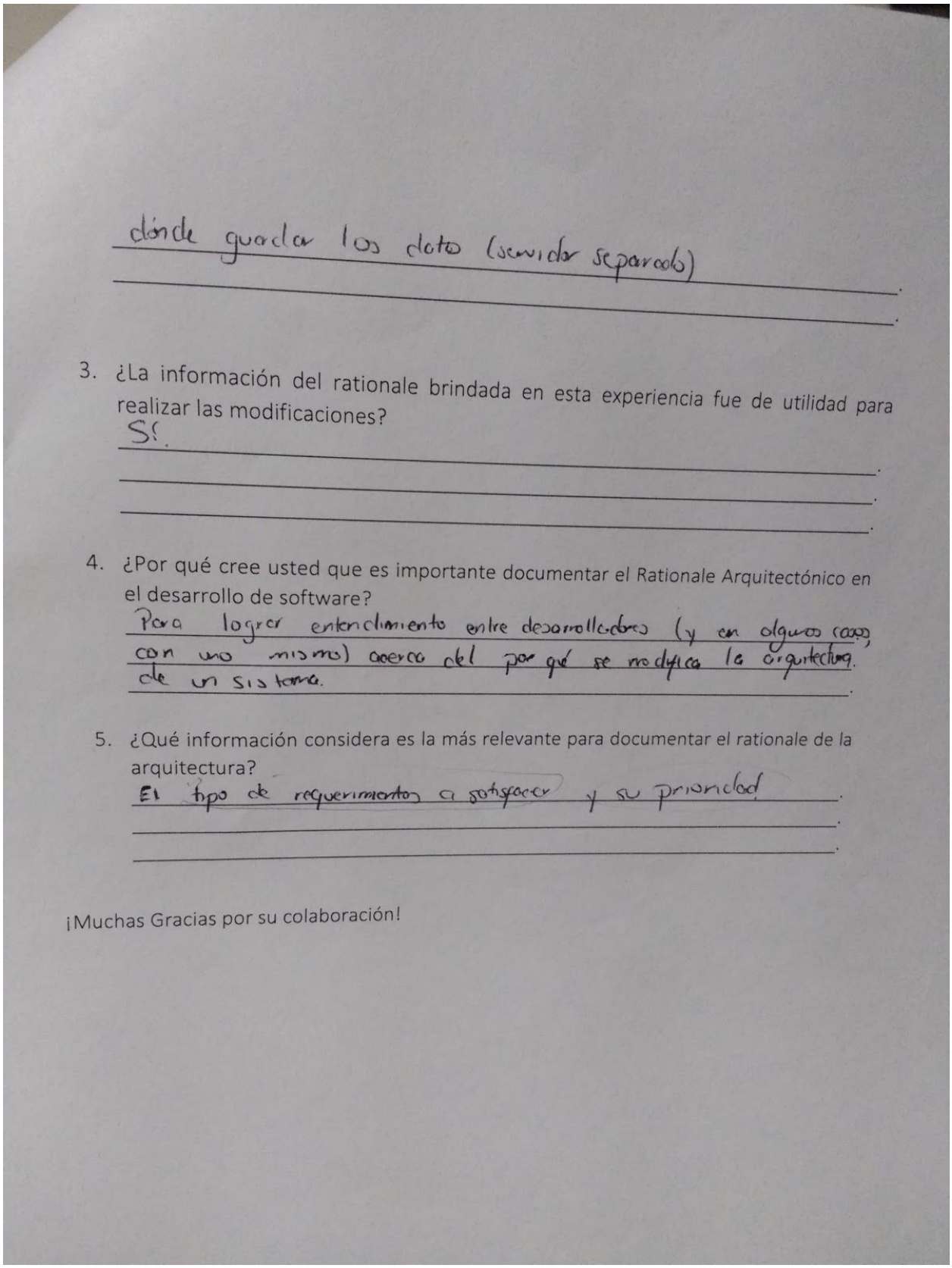
Preguntas abiertas:

1. Describa las razones por las cuales el código fue organizado bajo la estructura sobre la cual trabajó.

   Porque su principal requerimiento no funcional es la mantenibilidad.

2. Describa las razones por las cuales la estructura arquitectónica sobre la cual trabajó tuvo que ser alterada para satisfacer los cambios solicitados.

   Porque se requirió una nueva funcionalidad y un nuevo componente.



dónde guardar los datos (servidor separado)

3. ¿La información del rationale brindada en esta experiencia fue de utilidad para realizar las modificaciones?
Si.

4. ¿Por qué cree usted que es importante documentar el Rationale Arquitectónico en el desarrollo de software?
Para lograr entendimiento entre desarrolladores (y en algunos casos con uno mismo) acerca del por qué se modifica la arquitectura de un sistema.

5. ¿Qué información considera es la más relevante para documentar el rationale de la arquitectura?
El tipo de requerimientos a satisfacer y su prioridad

¡Muchas Gracias por su colaboración!

Encuesta Rationale Arquitectónico

En nombre del grupo IDIS agradecemos su participación en este experimento, los datos individuales no serán de conocimiento público (sólo usaremos su perfil), ya que son los resultados que la experiencia nos brindará la información para responder nuestras preguntas de investigación. A continuación, haremos un conjunto de preguntas que nos permitirán indagar algunos aspectos relevantes a la documentación del Rationale arquitectónico, luego haremos una sección de discusión conjunta para tener una perspectiva más amplia de sus apreciaciones.

Nombre: Alejandro Bolaños Jssa.

Experiencia en desarrollo de software

Tiempo: 5 años.

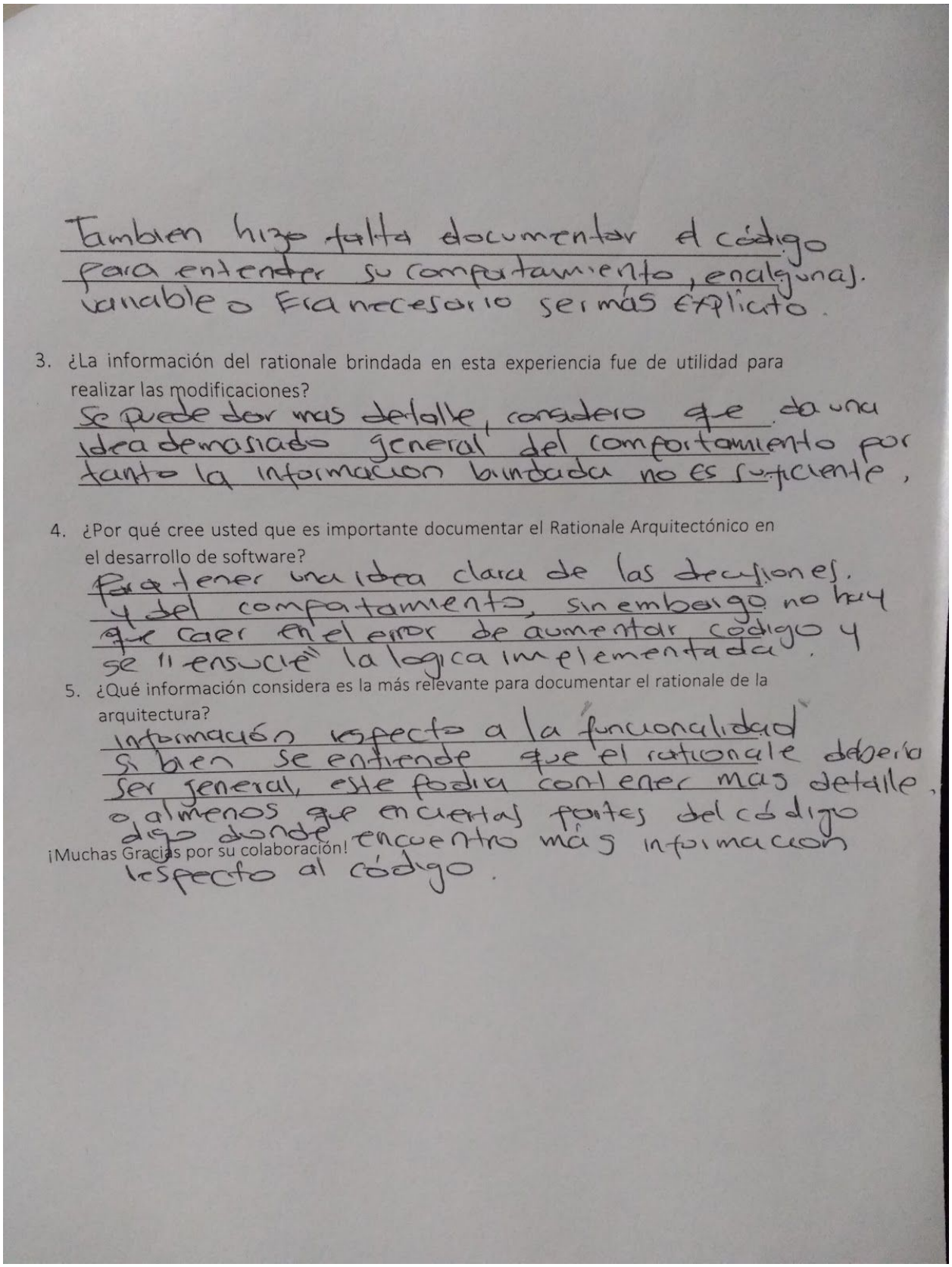
Lenguajes y plataformas: Java ispring, JEE, angular. maven, eclipse.

Roles desempeñados: Analista-Dllador, Programador.

Tipos de organizaciones (y nombres si Usted lo desea): HDS, mayorgafm, Sunset Software House,

Preguntas abiertas:

1. Describa las razones por las cuales el código fue organizado bajo la estructura sobre la cual trabajó.
   Existe una buena división de la lógica. esta muy bien separado funcionalmente. los nombres entienden el concepto de lo que se desea hacer. (En algunos).
2. Describa las razones por las cuales la estructura arquitectónica sobre la cual trabajó tuvo que ser alterada para satisfacer los cambios solicitados.
   Existe una buena separación sin embargo. para la parte de integrar la conexion con los sockets se hizo necesario alterar la estructura.



También hizo falta documentar el código para entender su comportamiento, en algunas variable o Era necesario ser más explícito.

3. ¿La información del rationale brindada en esta experiencia fue de utilidad para realizar las modificaciones?

Se puede dar mas detalle, considero que da una idea demasiado general del comportamiento por tanto la información brindada no es suficiente,

4. ¿Por qué cree usted que es importante documentar el Rationale Arquitectónico en el desarrollo de software?

Para tener una idea clara de las decisiones, y del comportamiento, sin embargo no hay que caer en el error de aumentar código y se "ensucie" la logica implementada.

5. ¿Qué información considera es la más relevante para documentar el rationale de la arquitectura?

Información respecto a la funcionalidad Si bien se entiende que el rationale debería ser general, este podría contener mas detalle, o almenos que encuentras partes del código digo donde encuentro más información respecto al código.

¡Muchas Gracias por su colaboración!

Encuesta Rationale Arquitectónico

En nombre del grupo IDIS agradecemos su participación en este experimento, los datos individuales no serán de conocimiento público (sólo usaremos su perfil), ya que son los resultados que la experiencia nos brindará la información para responder nuestras preguntas de investigación. A continuación, haremos un conjunto de preguntas que nos permitirán indagar algunos aspectos relevantes a la documentación del Rationale arquitectónico, luego haremos una sección de discusión conjunta para tener una perspectiva más amplia de sus apreciaciones.

Nombre: Libardo Pantoja

Experiencia en desarrollo de software

Tiempo: 5 años y luego acorte.

Lenguajes y plataformas: Java [illegible].

Roles desempeñados: [illegible].

Tipos de organizaciones (y nombres si Usted lo desea):

Preguntas abiertas:

1. Describa las razones por las cuales el código fue organizado bajo la estructura sobre la cual trabajó.

   Se organizó así para cumplir los requisitos.

2. Describa las razones por las cuales la estructura arquitectónica sobre la cual trabajó tuvo que ser alterada para satisfacer los cambios solicitados.

   [illegible]. El codigo actual ya tenia patrones que obligaban a hacerlo así

3. ¿La información del rationale brindada en esta experiencia fue de utilidad para realizar las modificaciones?

El docu. de arquitectura fue importante

4. ¿Por qué cree usted que es importante documentar el Rationale Arquitectónico en el desarrollo de software?

Para luego hacer cambios

5. ¿Qué información considera es la más relevante para documentar el rationale de la arquitectura?

las clases y sus metodos

¡Muchas Gracias por su colaboración!

Encuesta Rationale Arquitectónico

En nombre del grupo IDIS agradecemos su participación en este experimento, los datos individuales no serán de conocimiento público (sólo usaremos su perfil), ya que son los resultados que la experiencia nos brindará la información para responder nuestras preguntas de investigación. A continuación, haremos un conjunto de preguntas que nos permitirán indagar algunos aspectos relevantes a la documentación del Rationale arquitectónico, luego haremos una sección de discusión conjunta para tener una perspectiva más amplia de sus apreciaciones.

Nombre: Óscar Andrés López

Experiencia en desarrollo de software

Tiempo: 12 años.

Lenguajes y plataformas: Java, Python, Ruby, Scheme.

.

Roles desempeñados: Desarrollador, arquitecto de software.

Tipos de organizaciones (y nombres si Usted lo desea):

Tradicionales y startups (Heinsohn, Tappsi).

Preguntas abiertas:

1. Describa las razones por las cuales el código fue organizado bajo la estructura sobre la cual trabajó.

   Fue organizado así para facilitar su mantenibilidad y extensibilidad.

2. Describa las razones por las cuales la estructura arquitectónica sobre la cual trabajó tuvo que ser alterada para satisfacer los cambios solicitados.

   los cambios requeridos implicaban agregar nuevas.

funcionalidades ó modificar existente. Las alteraciones fueron mínimas y puntuales.

3. ¿La información del rationale brindada en esta experiencia fue de utilidad para realizar las modificaciones?

   En parte me ayudó a ubicar los puntos donde debía modificar, aunque lo que más usé fue la funcionalidad de búsqueda del IDE.

4. ¿Por qué cree usted que es importante documentar el Rationale Arquitectónico en el desarrollo de software?

   Para facilitar la vida de las personas que llegan a modificar el código después de uno.

5. ¿Qué información considera es la más relevante para documentar el rationale de la arquitectura?

   Que los cambios queden documentados en el sitio y momento en que ocurren. Sería útil tener el historial de las versiones de cada vez que se modifica el rationale

¡Muchas Gracias por su colaboración!

Universidad del Cauca

Documento de Arquitectura de Software

Juego N en línea

Santiago Hyun Dorado
8-3-2018

Historial de cambios

Fecha	Versión	Descripción	Autor
06/04/2018	1.0	Definición de la introducción y la representación de la arquitectura	Santiago Hyun Dorado
05/04/2018	1.2	Refactorización de los diagramas del Modelo y el Controlador	Santiago Hyun Dorado

Tabla de contenido

Tabla de Figuras

1. Introducción

Esta sección provee un resumen de todo el *Documento de Arquitectura de Software (SAD)* para el sistema Juego N en Línea. Brinda información sobre el propósito, el alcance, las referencias, las definiciones, acrónimos y abreviaturas utilizadas en todo el documento.

1.1. Propósito

El propósito principal del Juego N en línea es servir como base experimental en una investigación que tiene como finalidad determinar el valor de *ARAT* en el mantenimiento de una aplicación software desarrollada en Java. Este documento tiene como finalidad expresar en diferentes vistas el conjunto de decisiones arquitecturales que se han realizado en el desarrollo del sistema, con el objetivo de mostrar de distintas maneras los componentes y las conexiones que deben establecerse para cumplir con los escenarios requeridos.

1.2. Alcance

El alcance de este documento es explicar la arquitectura del Juego N en línea y las decisiones consideradas para realizar su diseño. La información contenida en este documento permite entender la estructura del sistema desde diferentes vistas, además, permite también describir explícitamente las razones de las decisiones arquitecturalmente importantes implementadas en el proceso de desarrollo. La representación de la arquitectura se expresa en el modelo 4+1 vistas ya que este permite observar la arquitectura desde diferentes perspectivas y para diferentes audiencias, además para cada vista de este modelo existen diagramas bien definidos que facilitan la descripción de cada una de ellas.

1.3. Definiciones, acrónimos y abreviaturas

SAD: Software Architecture Document

GUI (Graphic User Interface): Interfaz Gráfica del Usuario

POO: Programación Orientada a Objetos

ISO (International Organization for Standardization): Organización Internacional de Normalización.

ARAT (Architectural Rationale Annotations Tool): Es una herramienta software basada en anotaciones de código fuente para documentar las razones arquitecturales de un sistema desarrollado en Java.

Java: Es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems.

Factory Method: Método de fábrica, patrón de diseño creacional [1].

Observer: Observador, patrón de diseño de comportamiento [1].

Stakeholder: Según N. Rozanski et al [2] un stakeholder en una arquitectura de software "es una persona, grupo o entidad con intereses o preocupaciones sobre la realización de la arquitectura".

MVC (Model View Controller): Es uno de los patrones de diseño arquitectural más utilizado en el desarrollo web, empezó como un Framework desarrollado por Trygve Reenskaug alrededor de 1970 [3].

OMG (Object Management Grpup): Es un consorcio internacional de estándares tecnológicos sin fines de lucro, de membresía abierta [4].

UML (Unified Modeling Language): El lenguaje de modelado unificado (UML) es un lenguaje gráfico creado por el OMG para visualizar, especificar, construir y documentar los artefactos de un sistema de software [5].

1.4. Referencias

DAS: https://www.csun.edu/engineering-computer-science

Java: https://www.java.com/es/download/faq/whatis_java.xml

Stakeholder: https://www.viewpoints-and-perspectives.info/home/stakeholders/

MVC: https://msdn.microsoft.com/en-us/library/ff649643.aspx

ISO: https://www.iso.org/home.html

OMG: https://www.omg.org/

UML: http://www.uml.org/

1.5. Resumen

Este documento se compone de las siguientes secciones:

- Sección 1: Provee una introducción relacionada con la arquitectura de software del Juego N en línea.
- Sección 2: Describe los objetivos a nivel general de la Arquitectura del sistema.
- Sección 3: Describe la representación de la arquitectura en diferentes vistas.
- Sección 4: Desglosa la representación de la arquitectura en diagramas para cada vista.
- Sección 5: Referencias bibliográficas usadas en la creación de este documento.

2. Objetivos y restricciones de la Arquitectura

El Juego N en línea es un activo para un experimento el cual trata de comprobar si el uso de anotaciones de código como herramienta de documentación del *Rationale Arquitectónico* ayuda a mejorar la eficiencia y la efectividad en el mantenimiento de un sistema. El principal objetivo de la arquitectura del Juego N en línea es permitir la extensibilidad del sistema en términos de tipos de juegos y tableros, con el fin de conseguir aumentar la modificabilidad del sistema, de tal forma que se pueda dar cumplimiento a uno de los principios SOLID denominado “Abierto/Cerrado”. Se utiliza el estilo arquitectural de tres capas para mantener separadas las responsabilidades en cuanto a persistencia, lógica del negocio y presentación de vistas; dado que la complejidad del sistema no es alta, se requiere de un estilo arquitectural simple en el que se puedan efectuar cambios sin desestabilizar el sistema y con el que se puedan detectar en menor tiempo los errores. El patrón de diseño arquitectural MVC separa la interacción del usuario con la interfaz gráfica permitiendo que se actualicen las vistas y los datos del juego de manera dinámica a través de un controlador. Este patrón favorece la Mantenibilidad mediante

la separación de conceptos, con el objetivo de reutilizar código y organizar de manera adecuada las responsabilidades en la interacción con el usuario.

3. Representación de la Arquitectura

3.1. Vistas arquitecturales

La representación del sistema se puede ver desde diferentes puntos de vista, en el Modelo 4+1 Vistas propuesto por Kruchten p. et al [6] se definen 5 perspectivas que permiten extender el entendimiento de la arquitectura a las diferentes personas involucradas en el desarrollo de un sistema. En la Figura 1 se puede observar la organización de las vistas y su relación.

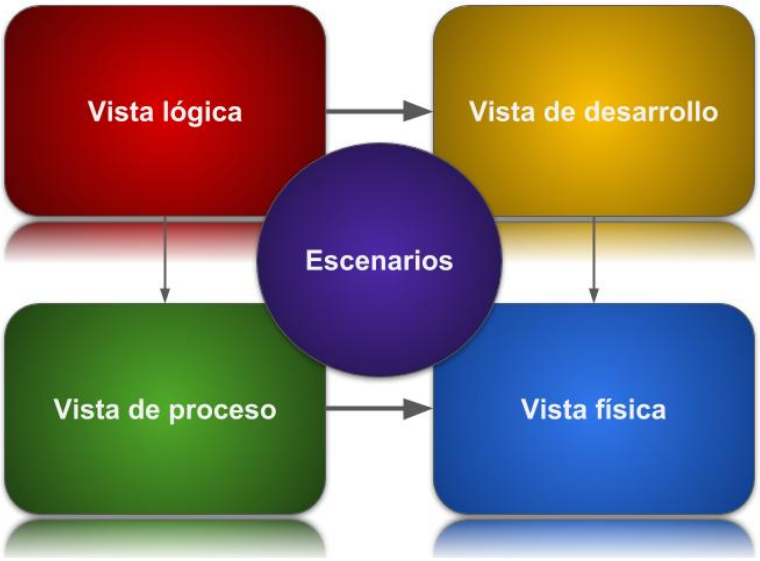


Figura 1 Modelo 4+1 vistas

3.1.1. Vista de escenarios

Conocida también como Vista de Casos de Uso, brinda información a nivel gráfico sobre los requerimientos funcionales de un sistema. Esta vista permite establecer las acciones y los roles que existen en el desarrollo de un sistema. Cada rol definido en el sistema debe tener una cantidad de casos de uso correspondientes a los requerimientos establecidos.

Audiencia: Stakeholders

Artefactos relacionados: Diagrama de Casos de Uso

3.1.2. Vista lógica

La Vista Lógica describe un sistema mediante el paradigma orientado a objetos, esta vista muestra las abstracciones diseñadas para estructurar el sistema junto con sus atributos y las operaciones que definen su comportamiento. Además, muestra cómo se relacionan cada una de ellas para formar partes más grandes de un sistema complejo.

Audiencia: Diseñadores

Artefactos relacionados: Diagrama de clases

3.1.3. Vista de desarrollo

Conocida también como Vista de Componentes, representa la arquitectura en términos de componentes software generalmente constituidos por una o más clases, estos componentes se relacionan entre sí para definir el comportamiento esperado por los usuarios finales. Estos componentes suelen ser software, sin embargo, se pueden dar casos donde los algunos de los componentes sean físicos o de hardware.

Audiencia: Desarrolladores

Artefactos relacionados: Diagrama de componentes, Diagramas de paquetes

3.1.4. Vista de proceso

Esta vista permite observar las tareas individuales que realizan las abstracciones software para cumplir con un determinado proceso. Los requisitos del usuario se pueden comprender como tareas que un sistema debe cumplir, esta vista permite desglosar esas actividades en partes más pequeñas con el objetivo de determinar los procesos de manera gráfica y ordenada. La complejidad de los procesos determina el nivel de abstracción adecuado para realizar el diseño del modelo.

Audiencia: Integradores, Desarrolladores

Artefactos relacionados: Diagrama de actividades, Diagrama de secuencia

3.1.5. Vista física

Es importante conocer la distribución de los componentes físicos que constituyen un sistema, la Vista Física permite observar la topología y las comunicaciones que se establecen entre los nodos físicos que conforman la arquitectura hardware del sistema.

Audiencia: Líderes de despliegue

Artefactos relacionados: Diagrama de despliegue

En la sección Descomposición de la arquitectura se muestran los diagramas utilizados en el Juego N en línea para cada tipo de vista.

3.2. Patrones de diseño arquitectónicos

El patrón de diseño arquitectural implementado en el desarrollo del Juego N en línea es el patrón *Modelo Vista Controlador MVC*. Por medio de este patrón podemos realizar una separación de la de interacción del usuario con la Interfaz Gráfica en tres diferentes componentes. En la Figura 2 se puede observar la interacción entre los componentes del patrón.

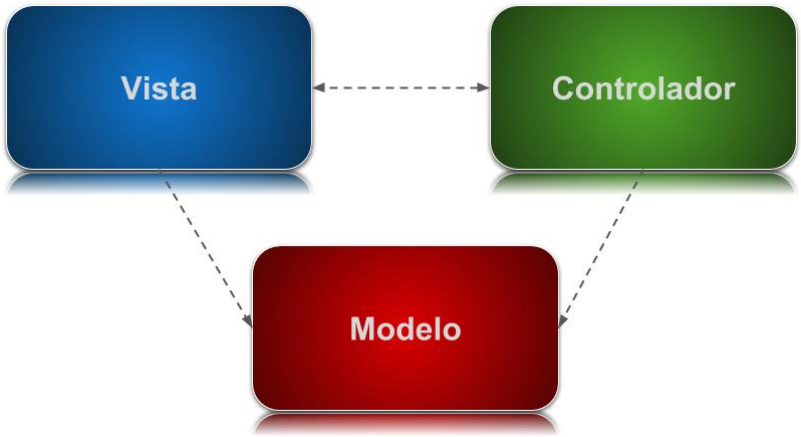


Figura 2 Patrón MVC

Modelo: Representa la información sobre el dominio del negocio, los datos y el comportamiento del sistema.

Vista: Es un componente que representa la *Interfaz Gráfica del Usuario GUI* con los datos del modelo, este componente captura las acciones y los datos y acude al controlador adecuado para procesarlos.

Controlador: Captura los datos ingresados por el usuario, manipula el modelo y actualiza la vista de forma correcta.

En la sección *Vista lógica* se muestra con más detalle las clases y los paquetes que componen este patrón de diseño arquitectural.

3.3. Estilo arquitectural

Los estilos arquitecturales son soluciones para resolver un problema relacionado con requerimientos NO funcionales o atributos de calidad, cada estilo arquitectural representa la experiencia de los diseñadores en un dominio de un problema específico. Las necesidades de calidad cambian de un sistema a otro por lo tanto los estilos arquitecturales son muy diversos y es muy difícil que contemplen todos los atributos de calidad requeridos [7]. Para el desarrollo del Juego N en Línea el requisito NO funcional de mayor prioridad es la mantenibilidad. Esta se traduce según la Norma ISO 25000 [8], en la facilidad para efectuar cambios, reutilizar componentes o agregar funcionalidad al sistema.

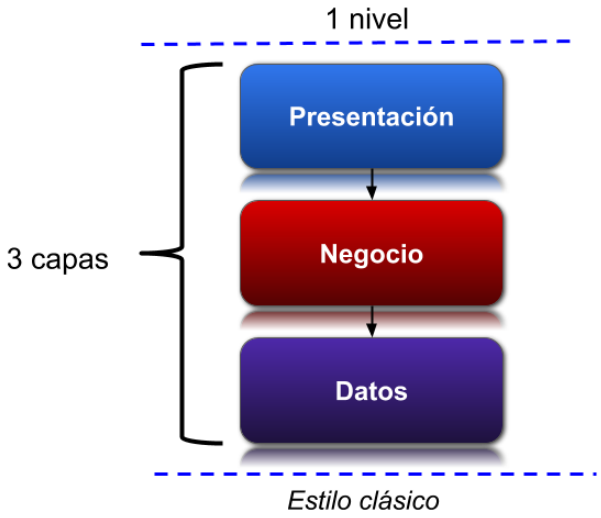


Figura 3 Estilo arquitectural clásico

En la Figura 3 se puede observar que el Juego N en línea sigue el estilo clásico de un nivel y tres capas propuesto por G. Florijn et al [7].

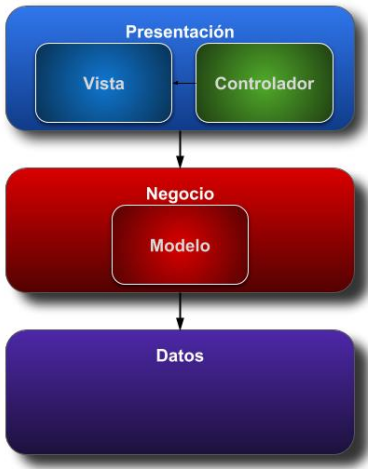
Capa de Presentación: Se encarga de controlar el flujo de trabajo entre el usuario y la GUI, también es responsable por mostrar la información del modelo. En el Juego N en línea esta capa contiene los componentes *Vista* y *Controlador* del *Patrón MVC.*

Capa de Negocio: Define los atributos y el comportamiento de las abstracciones de software en términos del contexto del negocio. En esta capa se puede encontrar las entidades que representan el *Modelo* del *Patrón MVC*.

Capa de datos: Esta capa es la encargada de acceder a los datos y establecer la persistencia de la información en el sistema.

En la Figura 4 se puede observar la organización de las capas y los componentes del *MVC.*

Figura 4 Organización Capas y componentes MVC



4. Descomposición de la arquitectura

La representación de la arquitectura se puede definir desde diferentes vistas adecuadas para una determinada audiencia. Estas vistas están relacionadas con uno o más diagramas UML que detallan con más información la arquitectura del sistema.

4.1. Vista de escenarios (casos de uso)

Las funcionalidades a nivel general se describen en la Figura 5. El jugador en el contexto del juego real es la persona que utiliza un tablero con n filas por m columnas y una cantidad x de fichas con el objetivo de realizar una secuencia de fichas continua del mismo color. Esta persona es la encargada de ubicar la ficha en una columna del tablero y soltarla, la evaluación del ganador se hace de manera visual, rectificando que haya una cantidad x de fichas contiguas del mismo color.

Figura 5 Diagrama de Casos de uso

ID	CU01
Nombre	Iniciar Juego
Actores	Jugador
Sinopsis	Iniciar los valores del juego y del tablero
Pre-condición	Se deben cargar los tipos del juego al componente gráfico por medio de reflexión
Curso típico de eventos	1. Seleccionar el número de casillas contiguas para ganar 2. Seleccionar el tipo de juego que quiere jugar 3. Presionar el botón jugar para crear el juego y el tablero gráfico
Cursos alternativos	Si no presiona el botón jugar no se deben mostrar los datos del juego ni del tablero.
Extensiones	Ninguno
Prioridad	Alta
% Completado	100

ID	CU02
Nombre	Cambiar Juego
Actores	Jugador
Sinopsis	Cambiar de tipo de juego, volver a seleccionar el número de casillas y el tipo de juego al que se quiere cambiar
Pre-condición	Se debe haber iniciado el juego (CU01)
Curso típico de eventos	1. El sistema crea la barra de opciones con el número de casillas para ganar y los tipos de juego que se pueden jugar. 2. El jugador selecciona el número de casillas contiguas para ganar 3. El jugador selecciona el tipo de juego al que quiere cambiar 4. El jugador presiona el botón reiniciar o el botón jugar para crear el juego y el tablero gráfico con los nuevos valores seleccionados
Cursos alternativos	Si no presiona el botón reiniciar el juego continuará con los mismos valores
Extensiones	Ninguno
Prioridad	Baja
% Completado	100
ID	CU02
Nombre	Cambiar Juego
Actores	Jugador
Sinopsis	Cambiar de tipo de juego, volver a seleccionar el número de casillas y el tipo de juego al que se quiere cambiar
Pre-condición	Se debe haber iniciado el juego (CU01)
Curso típico de eventos	1. El jugador selecciona el número de casillas contiguas para ganar 2. El jugador selecciona el tipo de juego al que quiere cambiar 3. El jugador presiona el botón reiniciar o el botón jugar para crear el juego y el tablero gráfico con los nuevos valores seleccionados
Cursos alternativos	Si no presiona el botón reiniciar el juego continuará con los mismos valores
Extensiones	Ninguno
Prioridad	Baja
% Completado	100

ID	CU03
Nombre	Ver tablero
Actores	Jugador
Sinopsis	Vista gráfica con los datos del juego y del tablero. Esta vista del tablero se actualiza de forma automática cada vez que el jugador inicia un juego o realiza una jugada.
Pre-condición	Se debe haber iniciado el juego (CU01)
Curso típico de eventos	1. El sistema crea el juego con la información que suministra el usuario en el caso de uso Iniciar Juego. 2. El sistema crea el tablero con la información del juego seleccionado y el número de casillas contiguas para ganar. 3. El sistema muestra la información del tablero en una matriz de botones.

	4. El sistema muestra la información del juego en una barra informativa.
Cursos alternativos	El usuario no suministra la información para crear el Juego y presiona Jugar: El sistema debe tomar los valores que están por defecto en la barra de opciones
Extensiones	Ninguno
Prioridad	Baja
% Completado	100

ID	CU04
Nombre	Realizar Jugada
Actores	Jugador
Sinopsis	Se selecciona la ubicación en la cual el jugador decide realizar su jugada
Pre-condición	Debe poderse observar la matriz del tablero (CU03)
Curso típico de eventos	1. El jugador selecciona un botón de la matriz del tablero para indicar cuál es la posición en la que quiere jugar. 2. El sistema determina si la ubicación es permitida para realizar la jugada. 3. El sistema ubica la ficha y actualiza el tablero con la nueva información. 4. El sistema muestra la información del juego en una barra informativa.
Cursos alternativos	El sistema determina que la ubicación no es permitida: El sistema debe ignorar las posiciones en las cuales no sea válida la jugada.
Extensiones	Ninguno
Prioridad	Baja
% Completado	100

ID	CU05
Nombre	Guardar Resultado
Actores	Jugador
Sinopsis	Se desea guardar el resultado del juego junto con la matriz del tablero en un archivo externo
Pre-condición	Debe existir un ganador
Curso típico de eventos	1. El sistema pregunta al jugador si desea guardar 2. El sistema guarda la información a través de un controlador, el cual se encarga de obtener el número del juego de un archivo que tiene un contador el cual representa el identificador de cada juego guardado en count-game.txt. 3. El controlador le notifica al tablero gráfico que ya se guardaron los resultados.
Cursos alternativos	El jugador no quiere guardar: El sistema debe reiniciarse con los valores por defecto de la barra de opciones. Error al guardar: El sistema le notifica al jugador que no se puede guardar el juego
Extensiones	Ninguno
Prioridad	Baja

% Completado	100

ID	CU06
Nombre	Ver Resultados
Actores	Jugador
Sinopsis	Lista con todos los resultados de los juegos guardados anteriormente
Pre-condición	Se debe haber iniciado el juego (CU01)
Curso típico de eventos	1. El jugador presiona el botón Resultados 2. El sistema carga la información de todos los resultados guardados previamente 3. El sistema crea el componente gráfico para mostrar los resultados 4. El jugador observa los resultados
Cursos alternativos	Error al cargar resultados: El sistema le notifica al jugador que hay un error al cargar los resultados
Extensiones	Ninguno
Prioridad	Baja
% Completado	100

4.2. Vista lógica (diseño)

En esta vista se muestra la organización de las clases dentro del patrón de diseño arquitectural *MVC*. El paquete juegos contiene una abstracción que representa las diferentes variantes del juego y las clases que heredan de esta abstracción, estas clases subtipos de Juego se crean por reflexión a través del FactoryMetodh en el método del JuegoController 'iniciarJuego' especificando la dirección de este paquete en una variable. El paquete tableros contiene una abstracción que representa las diferentes implementaciones de un tablero y las clases que heredan de esta abstracción, estas clases subtipos de Tablero también se crean por reflexión a través del FactoryMetodh en el método del Juego 'iniciarJuego' especificando la dirección de este paquete en una variable. Esto permite que el sistema se pueda extender en funcionalidades únicamente haciendo herencia de las abstracciones e implementando sus métodos abstractos, contribuyendo con la capacidad para ser modificado. La clase abstracta Juego es un conjunto de servicios comunes para los sub tipos que hereden de ella, esto con el fin de tener mayor capacidad para detectar cambios y conseguir aumentar la modificabilidad del sistema. Los servicios para el acceso de datos y la validación del ganador deben ser transparentes para el juego, esto quiere decir que los sub tipos que heredan de juego no deben saber cómo se están implementando estos servicios. Esto permite que se puedan realizar cambios a futuro sin afectar lo que ya está construido, mejorando la capacidad para detectar y ejecutar los cambios.

En la Figura 6 se muestra el paquete correspondiente al componente del *Modelo.*

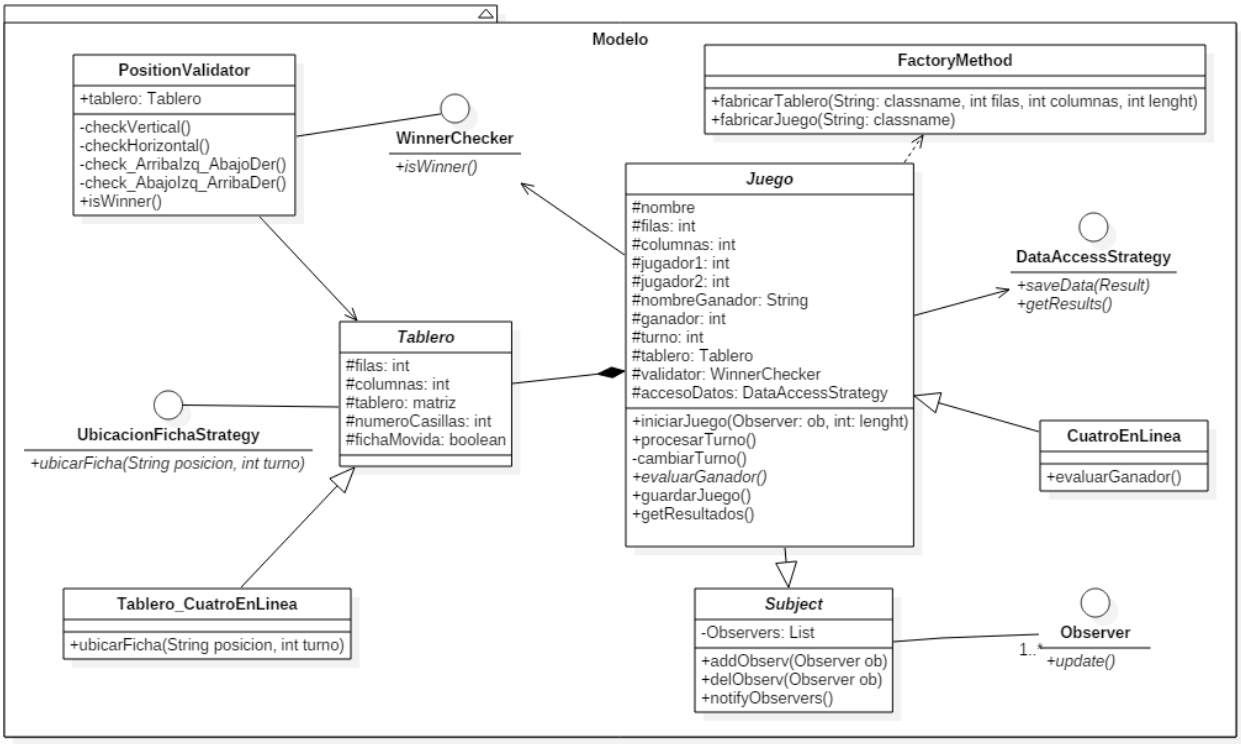


Figura 6 Diagrama de Clases Modelo

Clase Juego: Es una clase abstracta que representa las reglas de negocio del Juego N en línea, esta clase es encargada de iniciar el juego, procesar el turno y evaluar el ganador. Los atributos protegidos de esta clase permiten compartir información común entre los diferentes tipos de juego. El método evaluar ganador es abstracto ya que la forma de evaluar depende de las reglas para definir un ganador de cada tipo de juego, es por eso que debe ser implementado por las sub clases derivadas. Esta clase está compuesta por un tablero y tiene una relación directa con la clase PositionValidator que se encarga de hacer la verificación del tablero y determinar si hay un ganador.

Clase Tablero: Es una clase abstracta que representa los tipos de tableros, sus atributos protegidos permiten que en las clases derivadas de esta solo se desarrolle la implementación del método abstracto ubicarFicha.

Clase PositionValidator: Es una clase auxiliar que se encarga de verificar la existencia de un ganador en un tablero. Para la verificación se realiza un recorrido por toda la matriz identificando si existe un ganador. Se realiza una verificación horizontal, vertical, diagonal hacia arriba y diagonal hacia abajo para determinar si existe un ganador.

Interface WinnerChecker: Es la interfaz que contiene el método abstracto para verificar si un jugador es ganador.

Clase FactoryMethod: Es una clase que representa el patrón de diseño *Factory Method*. Esta clase contiene un método llamado fabricarTablero, el cual recibe el nombre de un tablero, las filas, las columnas y el número de casillas para ganar, instancia el tablero con esta información y la retorna al solicitante. También tiene otro método para crear un Juego a través del nombre de la clase que se quiere instanciar. Esta clase utiliza la reflexión propia de Java a través de Class.forName().

Clase Subject: Es una clase abstracta que representa al Observado del patrón de diseño *Observador*. Esta clase contiene una lista de los observadores que están pendientes de los cambios del Observado. Este se encarga de agregar, eliminar y notificar a todos los observadores que tiene en la lista.

Clase CuatroEnLinea: Es una clase derivada de Juego, esta clase redefine los atributos filas y columnas en 6 y 7 respectivamente. Esta clase realiza la implementación del método abstracto evaluarGanador. La regla del juego para determinar un ganador es: si un jugador marca las n fichas del mismo color de manera consecutiva.

Clase Tablero_CuatroEnLinea: Es una clase derivada de Tablero, esta clase implementa el método abstracto ubicarFicha heredado de Tablero. La ubicación de las fichas en este tablero es igual a la forma de ubicar en el juego físico, se debe seleccionar una columna y posicionar la ficha en la última posición disponible de la columna seleccionada.

Interface Observer: Es una interfaz con un método abstracto que realiza las acciones pertinentes después de haber sido notificado. Este método abstracto puede ser implementado por diferentes clases, sin embargo, este no se realiza si el Observador no ha sido agregado a una lista de observadores de la clase Subject (Observado).

Interface UbicaciónFichaStrategy: Es la interfaz en la cual se define el método abstracto para ubicar la ficha en el tablero. El comportamiento de este método varía dependiendo de las reglas del juego y es desarrollado de diferentes maneras por las clases que realizan su implementación.

El paquete Reflection permite obtener los tipos de las clases que heredan de Juego utilizando la librería reflections-0.9.9-RC1.jar. Esto permite que TableroGráficoController cree la barra gráfica de opciones con un ComboBox con el nombre de todas las clases que heredan de Juego, causando que en el proceso de agregación de tipos de juegos no se tenga que reescribir el código fuente actual y la vista se modifique a sí misma cuando se creen las nuevas implementaciones del Juego y el Tablero, contribuyendo en la capacidad para ser modificado. El JuegoController se encarga de recibir las peticiones del usuario y manipular los datos del modelo y los componentes de las vistas. Esta clase permite que se realice la separación de la interacción del uso de la interfaz gráfica por parte del usuario, permitiendo que el flujo de las peticiones sea más limpio y se pueda hacer refactorización del código en menos tiempo.

En la Figura 7 se observan las clases que componen el *Controlador* del patrón *MVC*.

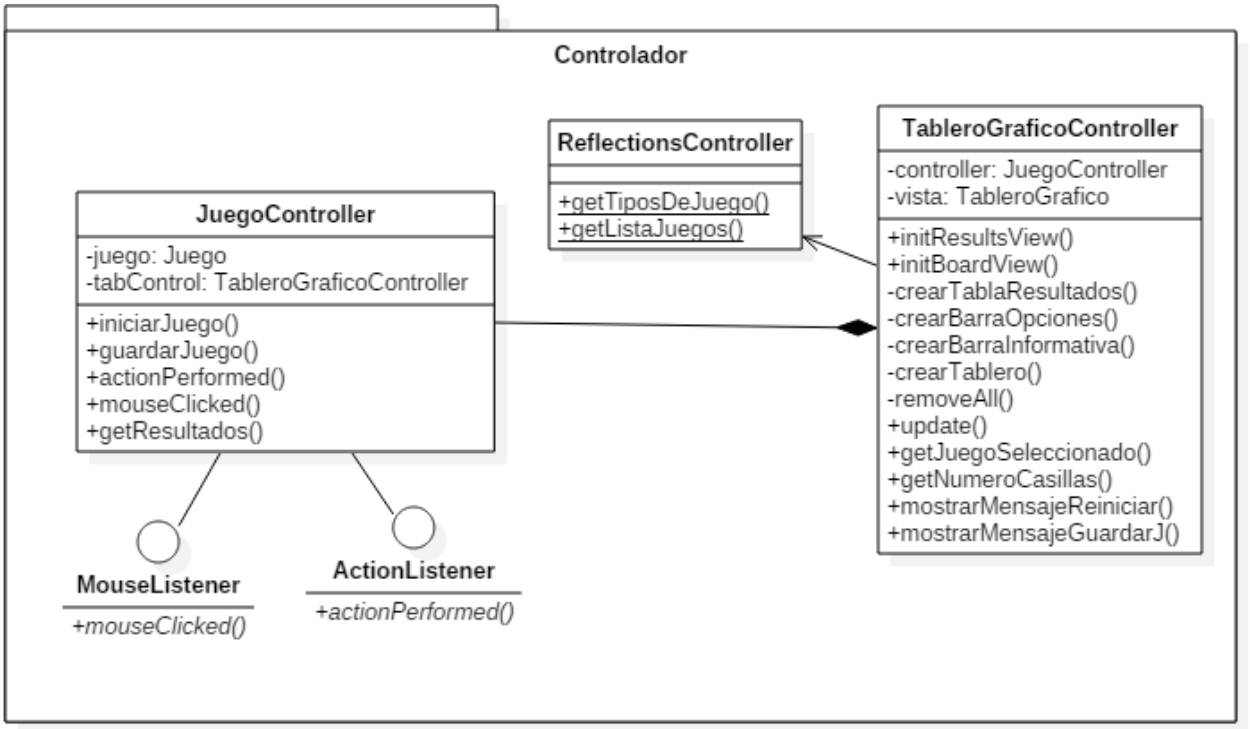


Figura 7 Diagrama de Clases Controlador

Clase JuegoController: Esta clase implementa la interface ActionListener con el objetivo de manejar los eventos de iniciar juego, reiniciar, ver resultados y seleccionar posición. También se encarga de hacer el llamado al Juego para guardar los resultados de una partida. Esta clase se relaciona con el TableroGraficoController para que actualice los componentes de la vista.

Clase TableroGraficoController: Esta clase se encarga de instanciar y actualizar los componentes gráficos del TableroGrafico y de la ListaResultadosGraficos de la vista. Se encarga de definir el controlador que va a manejar los eventos desplegados desde los botones de la vista. Esta clase implementa el método abstracto update del Observador, con el fin de actualizar la vista TableroGrafico cada vez que se inicia un juego o se realiza una jugada.

Clase ReflectionsController: Esta clase tiene métodos estáticos para obtener los tipos de juego derivados de la clase Juego y la lista con los nombres de los juegos. Estos métodos se utilizan en la creación de los componentes gráficos que permiten la selección del juego por parte del usuario en el TableroGrafico. Esta clase hace llamados a la librería reflections-0.9.9-RC1.jar para obtener los subtipos a través de una determinada clase.

Interface ActionListener: Es una clase nativa de Java que permite establecer un canal de comunicación entre los componentes gráficos y las clases que se encargan de manejar los eventos generados por el usuario. Define un método abstracto actionPerdormed para ser desarrollado por las clases que implementen la interface.

Interface MouseListener: Es una clase nativa de Java que permite establecer un canal de comunicación con los botones del componente gráfico. Esta clase define cuatro métodos abstractos mouseClicked, mouseEntered, mouseExited, mousePressed y mouseReleased los cuales permiten manejar los diferentes eventos generados por el usuario en el TableroGrafico.

En la Figura 8 se muestran las clases que componen la *Vista* del patrón *MVC*.

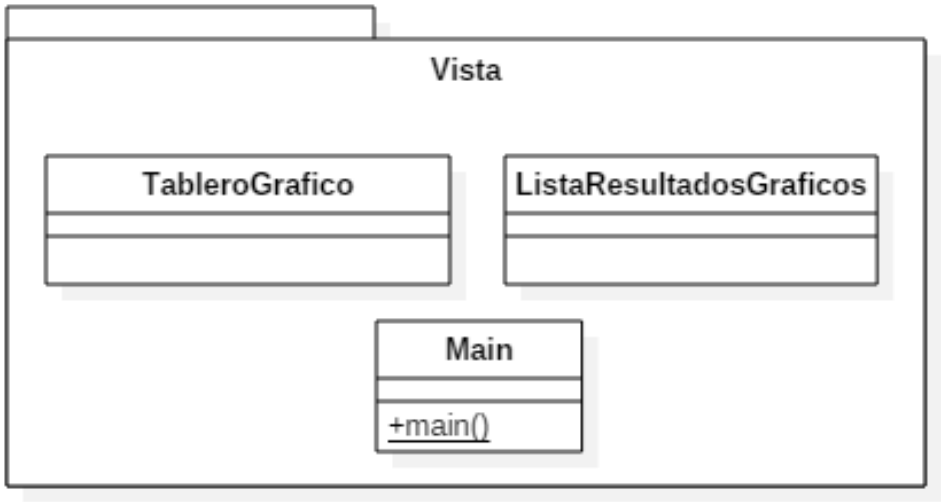


Figura 8 Diagrama de Clases Vista

Clase TableroGrafico: Tiene atributos gráficos que representan la información del juego y del tablero. El tablero se representa mediante una matriz de botones, la selección del juego se hace mediante un Combobox y un Spinner para el número de casillas, los datos del Combobox se cargan por medio de ReflectionController. La información del juego corresponde a una barra informativa la cual describe el juego seleccionado, el número de casillas y el turno de la jugada actual. Las acciones del juego se realizan desde 3 botones (Jugar, Reiniciar, Resultados)

Clase ListaResultadosGraficos: Tiene atributos gráficos que representan los resultados guardados con anterioridad. La información de los resultados se despliega en una tabla con 6 columnas (número del juego, nombre del ganador, número de filas, número de columnas, nombre del juego y el número de casillas para ganar). También tiene un botón para volver al TableroGrafico nombrado Regresar.

Clase Main: Encargada de instanciar el TableroGraficoController e iniciar la vista del TableroGrafico.

En la Figura 9 puede ver las clases que pertenecen al paquete de datos del Juego N en línea.

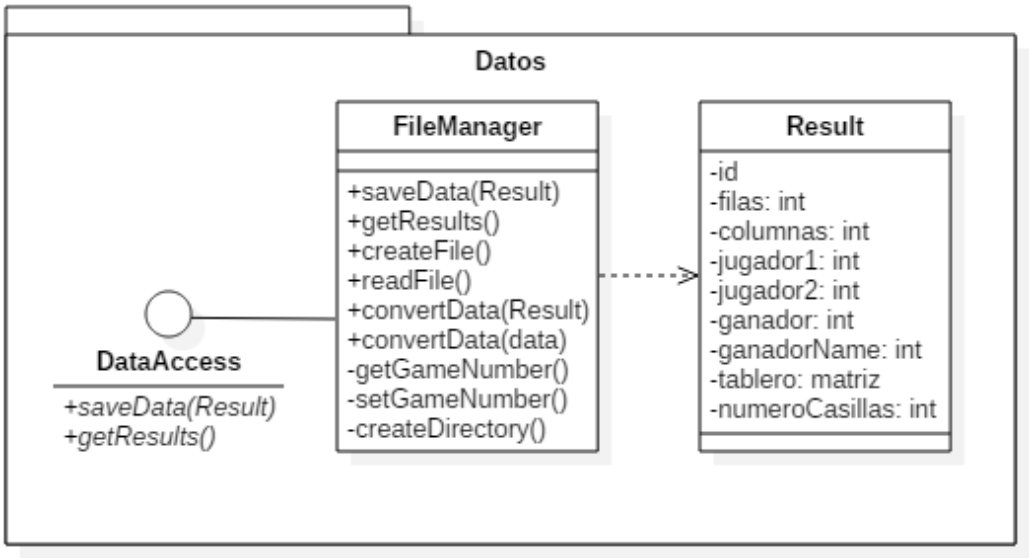


Figura 9 Diagrama de Clases Data

Clase FileManager: Esta clase implementa los métodos de la interfaz DataAccessStrategy. Esta clase tiene como responsabilidad realizar toda la interacción con los archivos de los resultados del juego, permite escribir un archivo con un resultado, leer todos los resultados guardados, convertir una lista de datos en objetos Result y viceversa.

Clase Result: Representa el resultado de un juego, esta clase tiene un identificador por juego y contiene la unión de los atributos del juego y del tablero.

Interface DataAccessStrategy: Es la interfaz con la cual un juego puede delegar el proceso de guardar el resultado de un juego u obtener los resultados de los juegos guardados anteriormente.

4.3. Vista de proceso (actividades)

En la Figura 10 se puede observar el flujo de mensajes y los participantes que están involucrados en iniciar un juego. El TableroGrafico activa un evento que es manejado por el JuegoController, este obtiene la información del juego seleccionado y el número de casillas del TableroGrafico a través del TableroGraficoController, después instancia el tipo de juego seleccionado por medio del método fabricar Juego del FactoryMethod y hace el llamado a Juego para que inicie el juego, este agrega la vista TableroGrafico como observador, hace un llamado al método fabricar tablero, el cual crea un tablero por medio de reflexión y lo retorna de acuerdo al tipo de juego seleccionado; seguido de eso Juego instancia un AccessData y un PositionValitador al cual le pasa el Tablero que le retorna el FactoryMethod. Finalmente, JuegoController le indica al TableroGraficoController que actualice los componentes de TableroGrafico con los datos del juego y el tablero seleccionado.

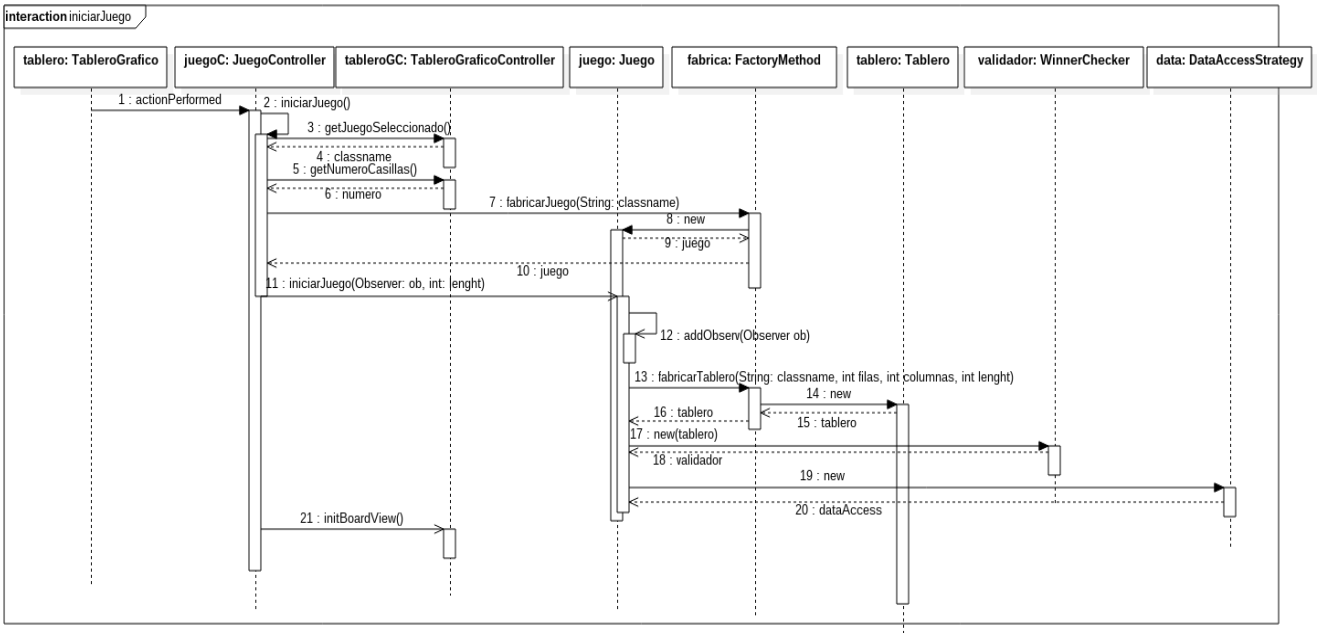


Figura 10 Secuencia Iniciar Juego

La Figura 11 muestra cómo interactúan los participantes del evento RealizarJugada. Esta secuencia comienza cuando la vista TableroGrafico captura el evento de la selección del botón en el cual el jugador quiere realizar la jugada. El JuegoController es el encargado de manejar este evento; obtiene el tablero de Juego, el turno actual y hace un llamado al Tablero para ubicar la ficha en la posición seleccionada y con el número del turno del jugador. Despues el JuegoController notifica a su lista de observadores que se ha modificado el Juego y el TableroGraficoController procede a actualizar la vista TableroGrafico. Finalmente, el JuegoController verifica si existe un ganador y de ser correcto envía un mensaje para informar al usuario y preguntar si desea guardar el juego.

La secuencia para guardar el resultado de un juego se muestra en la Figura 12. Después de que el TableroGraficoController envíe un mensaje al usuario a través del TableroGrafico y obtenga el nombre del jugador, el TableroGraficoController solicita al JuegoController que guarde el juego, este pasa el mensaje al Juego, el cual crea un Result, instancia los datos del resultado y solicita al AccessData que lo guarde.

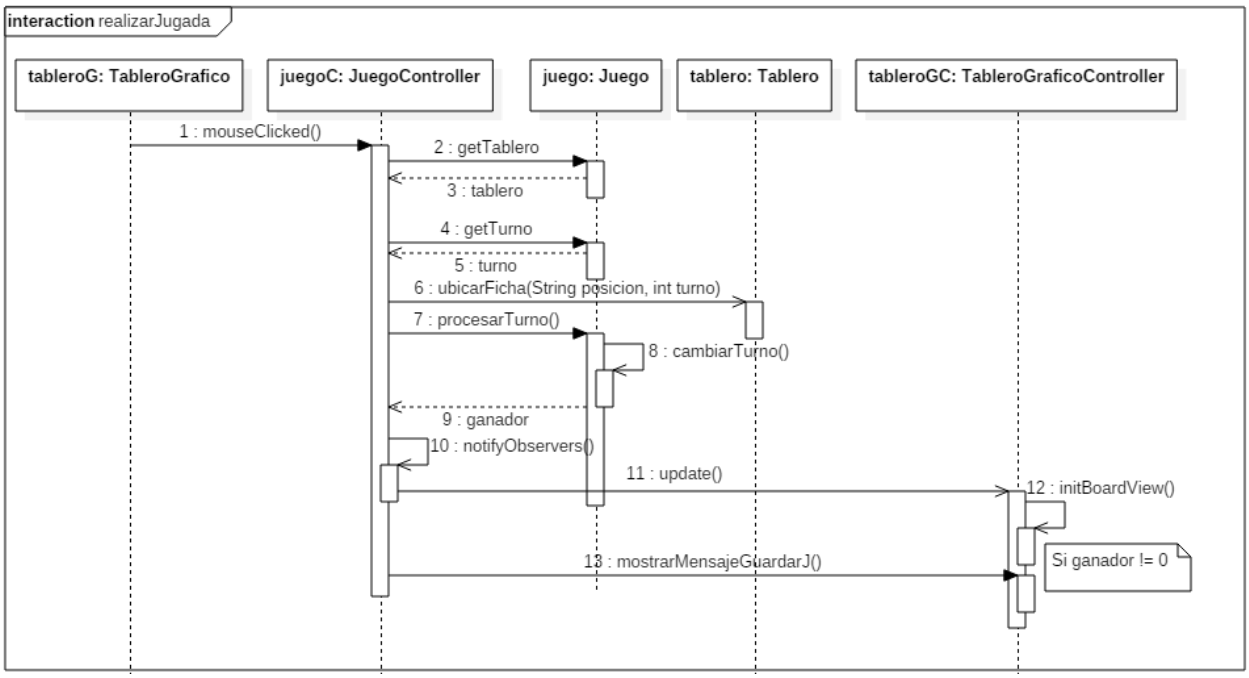


Figura 11 Secuencia Realizar Jugada

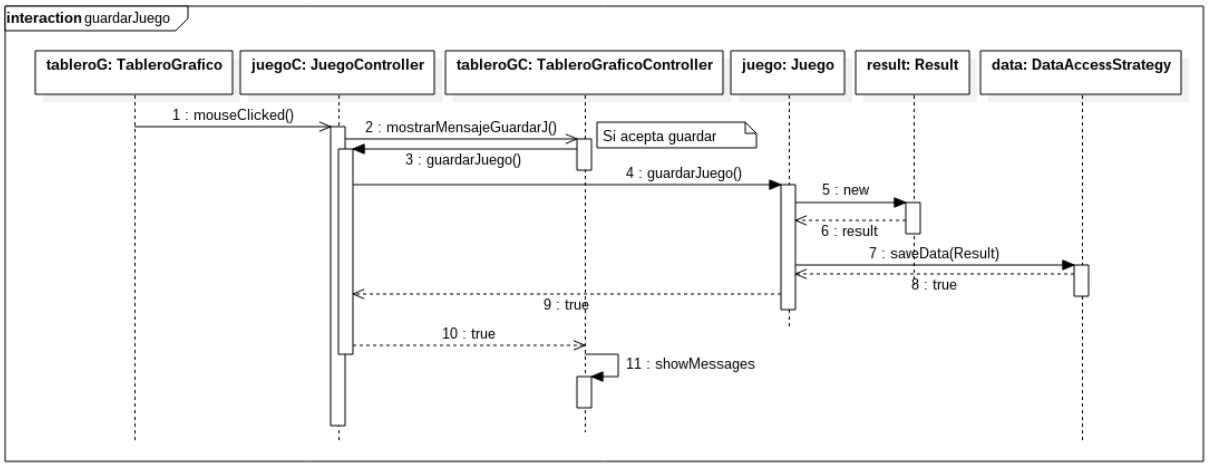


Figura 12 Secuencia Guardar Juego

En la Figura 13 se muestra cómo es el flujo de actividades que se realizan en la tarea de guardar resultado del juego. Este proceso inicia cuando el TableroGraficoController captura el evento de mostrar resultados generados por el botón Resultados de la vista TableroGrafico, este inicializa la vista ListaResultadosGraficos construyendo la tabla con los resultados de los juegos anteriores obtenidos mediante el DataAccess a través del JuegoController.

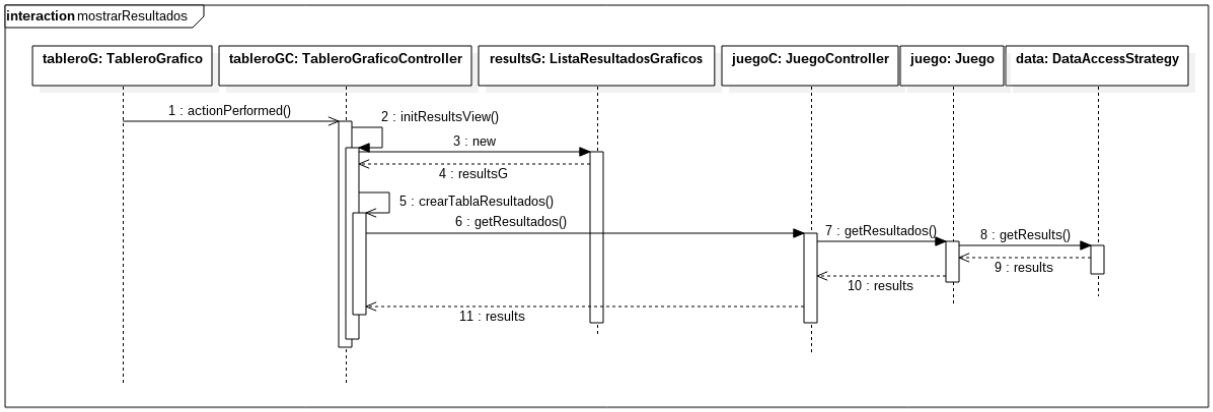


Figura 13 Secuencia Mostrar Resultados

4.4. Vista de desarrollo (componentes)

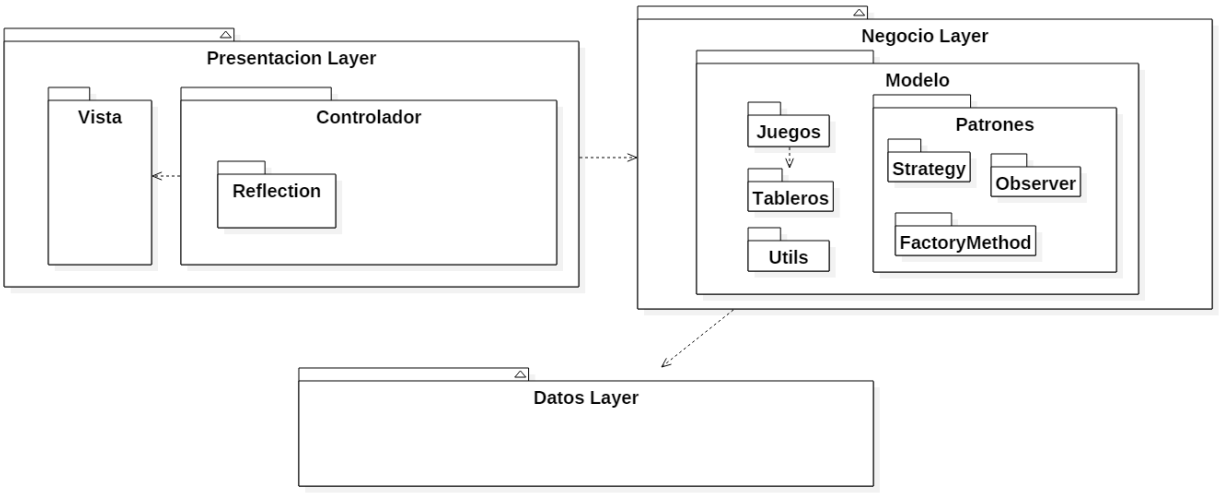


Figura 14 Diagrama de Paquetes

Los componentes del Juego N en línea se pueden observar desde diferentes perspectivas, en la Figura 14 podemos observar la organización de los paquetes entre las capas *Datos, Negocio* y *Presentación* con los componentes del patrón de diseño arquitectural *Modelo Vista Controlador.*

Los patrones de diseño utilizados en la arquitectura le dan estructura y comportamiento a la lógica del negocio, estos se encuentran en el paquete patrones para tener una idea de cuáles patrones se están utilizando en el sistema. En el paquete Utils se encuentra la clase PositionValidator que se encarga de verificar si existe un ganador. En los paquetes Juegos se encuentra la clase abstracta Juego y sus derivadas. En el paquete Tableros se encuentra la clase Tablero y sus clases derivadas. En el paquete Reflection se encuentra el ReflectionsController encargado de obtener los tipos de Juego en tiempo de

ejecución. En la capa de Datos se encuentra la interace de acceso a datos AccessData, una implementación de la interfaz llamada FileManager y la clase que representa los resultados del juego Result.

En la Figura 15 se puede observar la distribución de los componentes de manera que una o más clases conforman un componente. El componente Games representa la lógica de los juegos, este tiene una dependencia directa con el componente Boards el cual representa a los tableros y con el componente Patterns el cual representa los patrones de diseño estructurales y de comportamiento. El componente Boards tiene una relación directa con Patterns debido a que hace llamados a un patrón de diseño creacional (FabricaDeTableros). El componente Controllers representa los controladores encargados de modificar el modelo y actualizar las vistas, estos tienen una relación directa con Patterns debido a que necesita el comportamiento de un Observador para notificar los cambios, también tiene una relación con el componente AccesoDatos el cual representa las clases necesarias para crear persistencia en el sistema.

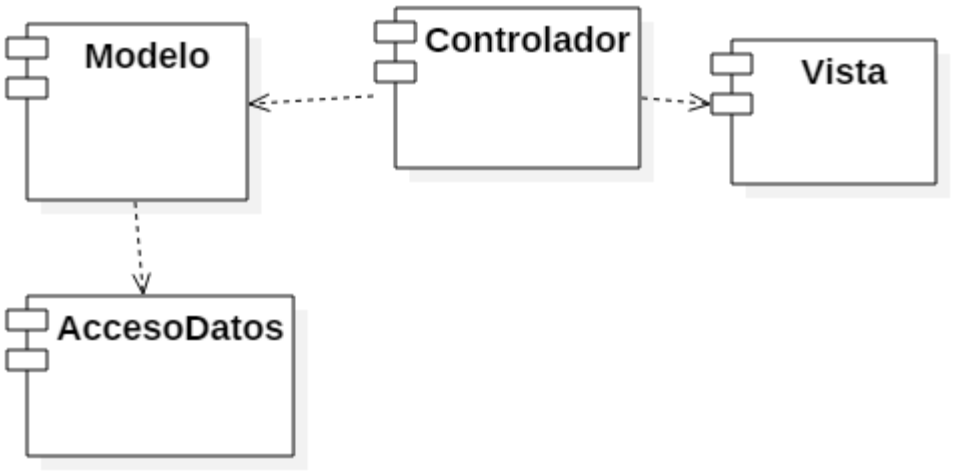


Figura 15 Diagrama de Componentes

Finalmente tenemos el componente View el cual está ligado al componente Controlador, ya que este ejecuta llamados al controlador a través de eventos generados por componentes gráficos de la vista (TableroGrafico).

4.5. Vista física (despliegue)

El sistema Juego N en línea es una aplicación monolítica y su despliegue requiere de un solo nodo, como se muestra en la Figura 16.

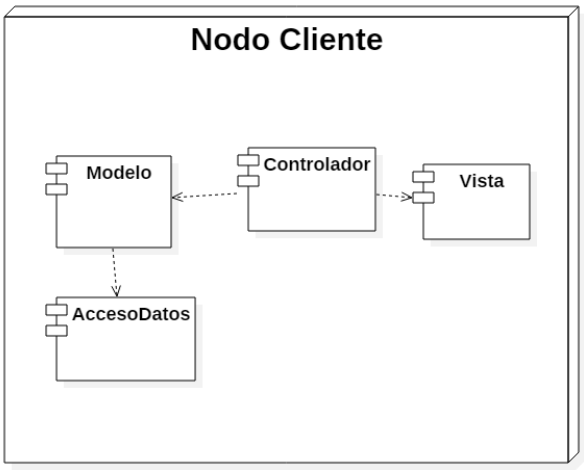


Figura 16 Diagrama de Despliegue

5. Bibliografía

[1] E. Robson and E. Freeman, *Head First Design Patterns Poster*. 2005.

[2] N. Rozanski and E. Woods, *Software Systems Architecture*. 2005.

[3] M. Fowler, *Patterns of Enterprise Application Architecture*, vol. 48, no. 2. 2002.

[4] OMG, "Object Management Group." [Online]. Available: https://www.omg.org/.

[5] Object Management Group, "Unified Modeling Language UML." [Online]. Available: http://www.uml.org/.

[6] P. Kruntchen, "Architectural blueprints–the" 4+ 1" view model of software architecture," *IEEE Softw.*, vol. 12, no. November, pp. 42–50, 1995.

[7] G. Florijn, "Architectural styles and patterns," 2015.

[8] Portal ISO, "ISO/IEC 25010." [Online]. Available: http://iso25000.com/index.php/normas-iso-25000/iso-25010?limit=3&limitstart=0.