DESARROLLO DE UN CATÁLOGO DE PATRONES DE DISEÑO PARA JAVA

CAMILO ANDRÉS OREJUELA SANDOVAL
MANUEL ALFONSO MORA MONTAGUT

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2016

DESARROLLO DE UN CATÁLOGO DE PATRONES DE DISEÑO PARA JAVA

CAMILO ANDRÉS OREJUELA SANDOVAL
MANUEL ALFONSO MORA MONTAGUT

Trabajo de Grado para optar al título de Ingeniero de Sistemas

Director

FERNANDO ANTONIO ROJAS MORALES

Magister en Ciencias de la Computación

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2016

Dedicado a los que los que enseñan, y a los que quieren aprender.

Manuel Mora

A las personas que entienden que los logros valiosos no se miden por el reconocimiento o estatus que ganan, sino por lo que estos significan para si mismos, para su propósito en la vida, para su lugar en el mundo.

Camilo Orejuela

AGRADECIMIENTOS

Queremos agradecer al profesor Fernando Rojas por confiarnos el tema del proyecto y guiarnos en su desarrollo.

Damos gracias a todas las personas que apoyaron el desarrollo del proyecto, en especial a Javier Fernando Pamplona y Martin Alonso Parra por su ayuda en la interfaz del sitio web, a Crisanto Montagut Martínez por su asesoría durante el desarrollo del proyecto, y al ingeniero Benjamín Pico del DSI, al profesor Manuel Guillermo Flórez y al grupo de investigación GID-CONUSS por brindarnos los recursos necesarios para ponerlo en línea.

También agradecemos a nuestra familia y amigos por su apoyo y consejos.

CONTENIDO

LISTA DE TABLAS

Pág.

LISTA DE GRÁFICAS

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE ANEXOS

RESUMEN

TÍTULO: DESARROLLO DE UN CATÁLOGO DE PATRONES DE DISEÑO PARA JAVA*

AUTORES: CAMILO ANDRÉS OREJUELA SANDOVAL**
MANUEL ALFONSO MORA MONTAGUT**

PALABRAS CLAVE: PATRONES DE DISEÑO, DISEÑO DE SOFTWARE, GENERADOR DE CÓDIGO, JAVA, SITIO WEB.

CONTENIDO:
Los patrones de diseño son un tema del que se empezó a hablar en el mundo del software en la década de los 80, y desde ese entonces hasta la actualidad son un tema que se mantiene vigente y del que se sigue hablando, ya que pertenece a un campo que no pierde importancia, independientemente de las tecnologías del momento o de nuevas metodologías de desarrollo, y es el diseño de software. La banda de los cuatro (GoF, por sus siglas en inglés) definió un conjunto de patrones, que son los más populares en la industria. Este proyecto está enfocado en apoyar el aprendizaje de un subconjunto de dichos patrones GoF; específicamente 13 de ellos: Builder, Factory Method, Singleton, Adapter, Composite, Decorator, Command, Iterator, Memento, Observer, State, Strategy y Template Method.
Para el fin de servir al aprendizaje de estos patrones, se creó un sitio web el cual se compone de contenido teórico escrito por los autores a cerca de estos patrones y una introducción a los mismos, así como ejemplos de aplicación de éstos usando el lenguaje de programación Java, los cuales el usuario puede descargar; además incluye una aplicación que permite al usuario generar código en Java luego de interactuar con el diagrama de clases de un patrón dado, aplicándolo a sus propios proyectos; en este generador de código, el usuario puede elegir el patrón que desea trabajar y enseguida escoger un elemento de ese patrón para modificar su nombre, sus atributos y sus métodos, pudiendo editar así todos los elementos de dicho patrón y finalmente generar el código correspondiente en Java.

* Trabajo de grado

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas. Director: Fernando Antonio Rojas Morales.

ABSTRACT

TITLE: DEVELOPMENT OF A JAVA DESIGN PATTERNS CATALOGUE [*]

AUTHORS: CAMILO ANDRÉS OREJUELA SANDOVAL[**]
MANUEL ALFONSO MORA MONTAGUT[**]

KEY WORDS: DESIGN PATTERNS, SOFTWARE DESIGN, SOURCE CODE GENERATOR, JAVA, WEBSITE.

CONTENT:

Design patterns are a topic which started talking in the software world about the 80s, and since then until now is a subject that remains active and that is still talking about it, since it belongs to a field that does not lose importance, independently of actual technologies or new development methodologies, and it is software design. The Gang of Four (GoF, for its acronym) has defined a set of patterns, which are the most popular in the industry. This project is focused on supporting the learning of a subset of these GoF patterns; specifically 13 of them: Builder, Factory Method, Singleton, Adapter, Composite, Decorator, Command, Iterator, Memento, Observer, State, Strategy and Template Method.
In order to serve the learning of these patterns, it was created a website which consists of theoretical content written by the authors about these patterns and an introduction to it, and examples of application of these using the Java language programming, which the user can download; also it includes an application that allows the user to generate source code in Java after interacting with the class diagram of a given pattern, applying it to their own projects; in this source code generator, the user can choose the pattern that want to work with and then choose an element of that pattern to change its name, its attributes and methods, thus being able to edit all the elements of the pattern and finally generate the corresponding source code in Java.

[*] Bachelor Thesis

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas. Director: Fernando Antonio Rojas Morales.

INTRODUCCIÓN

La adecuada solución a los problemas de diseño es un factor fundamental para el éxito del desarrollo y posterior mantenimiento de un software. Un mal diseño o soluciones ineficientes a los problemas en un software, hace que sea complicado realizar cambios o añadir nuevas funcionalidades. Robert C. Martin habla de los "malos olores" en el software que se "pudre" a causa de un mal diseño y nombra los siguientes síntomas de esta "descomposición": rigidez, inmovilidad, viscosidad, complejidad innecesaria, repetición innecesaria y opacidad [1]. El software empieza a presentar estos síntomas cuando el diseño no anticipó los cambios que era necesario realizar en el código a medida que los requerimientos evolucionaron o aparecieron nuevos. Para evitar que esto suceda es necesario seguir los principios SOLID, los cuales garantizan que el diseño sea fácil de entender, modificar y extender con nuevas funcionalidades.

Para cumplir con los principios SOLID dentro del diseño de un software, el desarrollador debe dar solución a problemas que dificultan su cumplimiento; es necesario valerse de las herramientas que brinda la programación orientada a objeto para superar estos obstáculos. En muchos casos el desarrollador se enfrentará a problemas a los que otros desarrolladores ya se han enfrentado y han encontrado una solución óptima. En 1994 la banda de los cuatro (GoF) formalizó en un libro las soluciones optimas que los desarrolladores dentro de la industria dieron a problemas de diseño que se presentaban en muchos desarrollos; dichas soluciones fueron llamadas patrones de diseño. La idea de los patrones de diseño fue tomada por la banda de los cuatro del arquitecto Christopher Alexander quien definió una estructura para definir soluciones a problemas comunes de diseño dentro de la arquitectura. Un patrón puede aplicarse a un sin número de situaciones y nunca utilizarse de la misma forma dos veces; esto es debido a que los patrones de diseño no son una receta que se sigue al pie de la letra. Los patrones de diseño permiten dar solución a problemas de diseño en un software y poder cumplir con los principios SOLID.

Es indispensable que los estudiantes de Ingeniería de sistemas aprendan del potencial de los patrones de diseño. Estos son una herramienta difícil de dominar para los desarrolladores principiantes, debido a que hacen uso de conceptos básicos de la programación orientada a objetos pero de una forma avanzada. El objetivo del presente proyecto es desarrollar un sitio web con contenido teórico que provea una herramienta software para dar soporte al aprendizaje de los

patrones de diseño a estudiantes con conocimientos básicos de programación orientada a objetos y diseño de software.

1. PLANTEAMIENTO DEL PROBLEMA

Desde que la banda de los cuatro definió algunos de los patrones de diseño más importantes, el tema se ha popularizado en todo el mundo. Y es que al desarrollarse cada vez aplicaciones software más complejas, la cuestión de alcanzar niveles de diseño de software óptimos se hizo cada vez más importante. Un diseño de software realizado de la mejor manera, aunque puede tomar un tiempo considerable, facilitará mucho el desarrollo del sistema en la etapa de implementación, aumentando además la calidad que puede conseguir dicho sistema, consiguiendo una mayor flexibilidad, mantenibilidad y extensibilidad, y en general cumpliendo con los principios SOLID. En cambio, un diseño hecho de forma apresurada, sin la dedicación que la calidad de éste demanda, producirá mayores tiempos en la implementación del sistema, mayores dificultades y errores, y en general un sistema de baja calidad, que será muy difícil mantenerse y extenderse a futuro. En todo este contexto, donde resulta evidente la importancia de invertir un gran esfuerzo en el diseño, los patrones de diseño surgen como unas técnicas absolutamente pertinentes, y ayudan a conseguir ese objetivo tan deseado de construir un diseño óptimo.

Por otro lado, paradójicamente se le ha dado poca importancia a esta temática en programas relacionados con la ingeniería de software y la ingeniería de sistemas en muchas universidades, dejándolo como un tema opcional o un tema más, dentro de un conjunto de temas de una materia en específico, sin tomarse el tiempo de profundizar en su estudio, y menos aún de llevarlos a la practica en proyectos de desarrollo de software en dichas materias. En el caso particular de la UIS, esta temática solo se estudia como uno de los temas de la materia Ingeniería de Software III, la cual es una materia electiva, es decir, la mayoría de estudiantes de la carrera, no alcanzan siquiera a conocer el concepto de patrones de diseño, el cual muchos están obligados a estudiar después de graduados, cuando les demandan de estos conocimientos en el campo laboral.

Un beneficio importante de adquirir conocimiento sobre patrones de diseño, es la visión amplia que adquiere el estudiante sobre el desarrollo de software; le permite analizar desde un alto nivel, el funcionamiento de todo un sistema, y cómo las partes deben relacionarse y comunicarse; le permite desarrollar un lenguaje de patrones en su mente, lo que le ayuda a mejorar su habilidad para identificar y resolver problemas. Poder pensar en abstracciones en un determinado algoritmo o

conjunto de tareas, hace al estudiante mucho más capaz de concebir un diseño óptimo y conseguir un software de alta calidad.

Una vez un estudiante está interesado en el tema de los patrones de diseño, existen diferentes fuentes a las cuales puede acudir; algunas son libros, como el popular Design Patterns de la banda de los cuatro, y otras son páginas web que ofrecen información al respecto. Sin embargo existen algunas variables que producen cierta brecha entre estos contenidos y los estudiantes. Para empezar, la mayoría de textos están en el idioma inglés, idioma que no todos los estudiantes dominan, estos textos están escritos principalmente para desarrolladores experimentados, con experiencia de al menos algunos años diseñando y desarrollando software, la mayoría de los textos no ofrecen ejemplos en Java, algunos ofrecen ejemplos muy avanzados, que requieren de otros conocimientos, los ejemplos suelen venir incompletos en los libros y algunas veces es imposible conseguir todo su código, ninguno ofrece una herramienta interactiva en la cual el estudiante pueda poner en práctica lo que aprende sobre los patrones creando sus propios programas de forma guiada, y en general, ninguno es una plataforma que ofrezca de manera conjunta un texto sencillo y claro, con programas de ejemplo listos para descargar y ejecutar, y la posibilidad de crear sus propios programas usando una herramienta interactiva.

Por último mencionar que los profesores de las materias en las que se podría estudiar esta temática, tampoco tienen a su disposición un sitio con las características antes mencionadas, el cual puedan usar de guía para ellos mismos y para sus estudiantes, facilitando la enseñanza, así como la proposición de ejercicios prácticos para los estudiantes.

2. JUSTIFICACIÓN

Los patrones de diseño son una herramienta indispensable en el desarrollo de software para lograr aplicaciones que lidien de forma adecuada con la complejidad y el acoplamiento. Permite a los diseñadores de un software hacer uso de las soluciones que otros programadores idearon y pulieron al aplicarlas a problemas de diseño que aparecen muchas veces en diferentes desarrollos. Aun así, los patrones de diseño son un tema difícil de asimilar para los estudiantes que empiezan su uso. La mayoría de los materiales disponibles abordan el tema con un enfoque específico: usuarios experimentados en el desarrollo de software, con un buen dominio de los principios de la programación orientada a objetos y con ejemplos que tratan temas con los que el autor está familiarizado en su trabajo o su entorno. Todo lo anterior no indica que los materiales existentes sean inútiles o equivocados en su enfoque, solo que la búsqueda de abarcar de forma completa los patrones de diseño los hacen complicados como primer paso para los estudiantes inexpertos. Por este motivo es necesario crear un contenido teórico que permita iniciar el estudio de los patrones de diseño limitando su profundidad, teniendo en cuenta al estudiante que no domina a cabalidad los principios de la programación orientada a objetos.

Para cumplir con las características anteriormente expuestas, el nuevo contenido debe ser complementado por ejemplos con los que esté familiarizado el estudiante y una aplicación que facilite la creación de la estructura de los patrones de diseño. El hecho de que este nuevo contenido sea desarrollado por estudiantes que acaban de pasar por el proceso de aprender los patrones, facilita detectar y simplificar los aspectos más difíciles de aprender, y transmitir esta información a estudiantes no expertos. Asimismo, que los ejemplos sean sencillos de entender, no solamente en su estructura, sino también en el dominio en que se aplican, hace más fácil que un estudiante se concentre en los conceptos y características del patrón que está aprendiendo. Por último, una aplicación dinámica que facilite al estudiante la aplicación de los patrones a sus propios programas, permitiéndole interactuar con la estructura de éstos y generar código en el lenguaje Java, ayuda a complementar el entendimiento de los patrones de forma práctica, aplicando los patrones a casos específicos. Todas estas características hacen que este proyecto sirva para facilitar la primera aproximación de los estudiantes a los patrones de diseño.

3. OBJETIVOS

3.1 OBJETIVO GENERAL

Desarrollar un catálogo de un subconjunto de los patrones GoF, en el lenguaje de programación Java, que provea a los estudiantes de Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander de una herramienta informática que les permita abordar esta temática.

3.2 OBJETIVOS ESPECIFICOS

- Seleccionar los patrones de diseño más usados en la industria del software, que serían desarrollados. Los seleccionados hasta el momento son:
  - Abstract Factory.
  - Adapter.
  - Composite.
  - Decorator.
  - Factory Method.
  - Observer.
  - Strategy.
  - Template Method.
- Recopilar y modificar un conjunto de casos que permitan ver el funcionamiento de los patrones antes mencionado, para facilitar la comprensión y aprendizaje de los mismos.
- Desarrollar un sitio web que permita tener fácil acceso a la información de los patrones de diseño, debe contener los enunciados y el código de los casos seleccionados, además de permitir ver su funcionamiento de forma dinámica y contener un generador de código genérico que pueda ser extendido para uso específico.
- Utilizar diagramas UML y de secuencia de los patrones de diseño y en los casos seleccionados.
- Desarrollar y realizar un plan de pruebas para evaluar el sitio web, prestando especial atención a la usabilidad.

4. MARCO TEÓRICO

A continuación se definen una serie de conceptos utilizados a lo largo del presente proyecto de grado, los cuales se encuentran ligados al propósito del mismo.

4.1 DISEÑO DE SOFTWARE

El diseño de software es la etapa que sigue a la obtención y análisis de requerimientos y precede a la implementación en el proceso de desarrollo de software; consiste en definir de forma general y utilizando un conjunto de diagramas o elementos gráficos, la especificación de cómo funcionará el sistema, el conjunto de sus componentes, sus elementos y sus relaciones; dicho de otro modo, es el proceso por el cual se crea una especificación de un artefacto software, orientado a conseguir metas, usando un conjunto de componentes y sujeto a restricciones [2].

Un diagrama como representación del diseño de un software, comúnmente muestra un conjunto de elementos representados por rectángulos u óvalos, cuyas relaciones se indican a través de líneas o flechas que los unen. El estándar en la representación de diagramas es el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés), el cual define diferentes tipos de diagramas con un conjunto de características bien definidas. UML está orientado a proveer una forma estándar para visualizar los diseños de un sistema [3].

El proceso de diseño de software sigue una serie de principios. Grady Booch en su modelo de objetos, menciona la abstracción, encapsulación, modularidad y herencia, como los principios de diseño fundamentales [4]. Por otro lado, Robert C. Martin menciona los principios SOLID [1], un conjunto de principios muy bien definidos, orientados a evitar malas prácticas en el proceso de diseño. Todos estos principios sientan la base para conseguir calidad en el diseño de software.

En un nivel más alto y aplicando todos los principios, están los patrones de diseño, un conjunto de soluciones bien definidas a problemas comunes en el software.

4.2 PRINCIPIOS SOLID

Las fallas en la etapa de diseño, repercuten negativamente en las etapas de desarrollo posteriores, así como en la calidad del producto final. Además, el software por naturaleza tiende a deteriorarse a medida que se introducen modificaciones por la aparición de nuevos requerimientos o cambios en los existentes. La combinación de un mal diseño y las modificaciones subsecuentes pueden hacer imposible la tarea de mantener el software [5]. Para evitar esta situación es importante hacer uso de los principios SOLID. El conjunto de principios busca disminuir el acoplamiento y la complejidad, y si son bien aplicados el software será fácil de extender y modificar. SOLID se compone de cinco principios [1]:

- Single-Responsibility Principle (SRP): El principio de única responsabilidad enuncia que una clase debe tener una sola razón para cambiar. Una clase que posee más de una razón para cambiar obliga a un desarrollador a entender todo su funcionamiento y no solo lo que le interesa en un momento dado para poder realizar algún cambio. Además, se corre un mayor riesgo de introducir errores en sectores que no tienen una relación lógica con el cambio que se realizó.

- Open/Closed Principle (OCP): Este principio define que el software debe estar abierto para extensión y cerrado para modificación; dicho de otro modo, debe ser flexible al cambio permitiendo agregar nuevo código sin modificar el existente. Si el software fue diseñado correctamente los elementos que lo componen deben permitir agregar otros elementos sin la necesidad de cambios en los existentes ni desencadenar cascadas de cambios por todo el sistema.

- Liskov Substitution Principle (LSP): Este principio se basa en uno de las características más potentes de la programación orientada a objetos, la herencia. Además de ayudar a evitar repetir código innecesariamente, la herencia también es fundamental para dar soporte al polimorfismo y la abstracción. El principio se basa en la idea de asegurar que los subtipos puedan ser reemplazados por su tipo base. Puede parecer un principio sencillo, pero es vital para el desarrollo de un software capaz de cumplir con el principio de abierto/cerrado.

- Dependency-Inversion Principle (DIP): El principio de inversión de dependencias plantea que los módulos de mayor nivel no deben depender

de los de menor nivel, de hecho, ambos deberían depender de las abstracciones. También plantea que las abstracciones no deben depender de los detalles, en vez de eso, los detalles deben depender de las abstracciones. En otras palabras, no dependa de las implementaciones, dependa de las abstracciones. El principio evita que cambios en los módulos de menor nivel tengan efectos en los de mayor nivel. Es evidente que los módulos de mayor nivel que contienen las reglas del negocio no deben depender de los módulos que contienen detalles de implementación, el diseño de un software siempre debe ser independiente de la implementación.

- Interface Segregation Principle (ISP): El principio de segregación de interfaces plantea la necesidad de que las clases posean interfaces claras y orientadas a un solo cliente. Si una clase posee una interfaz que le permite atender a varios tipos de clientes con un grupo de sus métodos y a otros clientes con otro grupo, posiblemente esa clase tenga una interfaz saturada. Algunas clases necesitan de interfaces que interactúan con varios grupos de clientes, pero cada grupo debe comunicarse con la clase por medio de una abstracción que contenga la parte de la interfaz que le interesa.

4.3 PATRONES DE DISEÑO

4.3.1 Definición. En el mundo del desarrollo de software es muy común enfrentarse a problemas que ya otros desarrolladores han enfrentado. Así que siempre ha existido una necesidad por que las soluciones a dichos problemas, que resultan comunes, se puedan compartir entre desarrolladores y todos se vean beneficiados. Algunos desarrolladores expertos dedicados a estudiar estos problemas y soluciones, han hecho un esfuerzo por documentarlos, estandarizarlos y publicarlos, de modo que estén al alcance de todos en un formato que les permita sacar el mayor provecho. Fue de esta manera precisamente que surgieron los patrones de diseño [6].

Los patrones de diseño son soluciones a problemas comunes en el desarrollo de software. Un patrón de diseño describe un problema específico que se da en unas condiciones dadas, y define una solución para dicho problema, la cual puede aplicarse siempre que un problema de este tipo aparezca; la definición de un

patrón a menudo también describe las consecuencias de aplicar este patrón, así como algunos detalles sobre su implementación o variantes que pueda tener.

4.3.2 Historia. Todo empezó en la década de los 70 cuando el profesor de arquitectura de Berkeley, Christopher Alexander, empezó a estudiar los problemas comunes en la construcción de edificios, casas, entre otros, y las soluciones que se aplicaban a estos problemas, para luego introducir el concepto de patrones, y crear el concepto de "lenguaje de patrones". Sus libros, A Pattern Language: Towns, Buildings, Construction (Oxford University Press, 1977) y The Timeless Way of Building (Oxford University Press, 1979), describen una serie de patrones bien definidos, los cuales siguen una plantilla o formato, que incluye la descripción del problema y la solución apropiada para dicho problema. El trabajo de Christopher sirvió de inspiración para otros campos del conocimiento, entre estos, principalmente, el desarrollo de software; algunos otros campos fueron la pedagogía en educación, los sistemas de control de aviónica, y la interacción persona-computadora (HCI, por sus siglas en inglés), entre otros.

En el año 1987, Kent Beck y Ward Cunningham, llevaron las ideas de Christopher al diseño y desarrollo de software, definiendo un conjunto de patrones para hacer interfaces de usuario elegantes en Smalltalk. Ellos presentaron su trabajo titulado Using Pattern Languages for Object-Oriented Programming en la conferencia Object-Oriented Programming Systems, Languages, and Applications (OOPSLA) '87.

En el año 1991 aparece un trabajo inicial del primero del popular grupo conocido como "La banda de los cuatro" ("Gang of Four"; a veces mencionado por sus siglas "GoF"), Erich Gamma, quien presentó su trabajo de tesis doctoral en OOPSLA '91, en el cual hacía un primer acercamiento a los patrones que conocemos hoy día. A partir de ese momento se fueron uniendo a él Richard Helm, luego John Vlissides y por último Ralph Johnson, y mostraron su trabajo nuevamente y con mayor avance en OOPSLA '92 y ECOOP '93. Una de las cosas que hicieron esos años fue tratar de describir y explicar los patrones de una forma en que no solo los desarrolladores que ya hubieran tenido estos problemas y soluciones pudieran utilizarlos, sino que también pudieran entenderlos personas menos experimentadas y con conocimiento apenas de los fundamentos de la programación orientada a objetos, por lo cual hicieron una explicación más extensa de cada patrón, incluyendo código de ejemplo, entre otras cosas. Algunos patrones tuvieron un nombre diferente inicialmente y luego cambiaron de nombre; tal es el caso de Wrapper (envoltorio) que se convirtió en Decorator (decorador),

Glue (pegamento) se convirtió en Facade (fachada), Solitaire (solitario) se convirtió en Singleton (único) y Walker (caminante) en Visitor (visitante). La banda de los cuatro finalmente decidió convertir el catálogo en un libro, publicando Design Patterns: Elements of Reusable Object-Oriented Software en el año 1994. Este libro se convirtió en un hito en la industria y en un libro de referencia que sigue estando vigente y protagonista a pesar del tiempo que ha pasado desde su publicación.

Desde ese momento se han publicado muchos otros libros y artículos, los cuales han profundizado en el estudio de estos patrones, los han explicado de distintas formas y también han definido nuevos patrones más allá de los que definió la banda de los cuatro; además se han creado patrones para dominios específicos del software como procesamiento distribuido, aplicaciones empresariales, seguridad en los sistemas y videojuegos.

4.4 LENGUAJES DE PROGRAMACIÓN

Para el desarrollo del proyecto se hizo uso de los siguientes lenguajes de programación:

- HTML (Hyper Text Markup Languaje): *"Es un lenguaje de marcas (etiquetas) que se emplea para dar formato a los documentos que se quieren publicar en la WWW. Los navegadores pueden interpretar las etiquetas y muestran los documentos con el formato deseado."* [7].

- CSS3 (Cascading Style Sheets): *"es un lenguaje utilizado para describir la presentación de un documente en HTML o XML. CSS describe cómo los elementos estructurados deben de ser presentados en pantalla, papel o en otro medio"* [8].

- Javascript: "es un lenguaje ligero e interpretado, dialecto del estándar ECMAScript, orientado a objetos con funciones de primera clase, más conocido como el lenguaje de script para páginas web, pero también usado en muchos entornos sin navegador, tales como node.js o Apache CouchDB. Es un lenguaje script multi-paradigma, basado en prototipos, dinámico, soporta estilos de programación funcional, orientada a objetos e imperativa." [9]

- Java Server Pages (JSP): es una tecnología que provee una forma rápida y simple de crear contenido web dinámico y el desarrollo de aplicaciones web que son independientes de servidor y la plataforma. [10]

- Hypertext Procesor (PHP): es un lenguaje open source de propósito general que es especialmente útil para el desarrollo web y puede ser embebido en HTML. Su sintaxis está basada en C, Java y Perl, y es fácil de aprender. Su principal propósito es permitir a los desarrolladores web a escribir páginas web generadas dinámicamente de forma rápida [11].

4.5 PRUEBAS DE SOFTWARE

Las pruebas de software son el conjunto de actividades que durante el desarrollo de una aplicación permite evaluar si el software está llevando a cabo las tareas que debe realizar y si las está cumpliendo correctamente [12]. Hay muchas formas de probar un software y una gran variedad de metodologías que guían su uso durante el diseño y desarrollo. La implementación de pruebas automatizadas permite verificar de forma rápida y sencilla que una modificación tuvo los efectos deseados sobre el software y no introdujo errores. Las pruebas unitarias son un tipo de pruebas que se basan en probar pequeños segmentos del código para verificar su correcto funcionamiento, haciendo uso de los datos necesarios, el resultado obtenido y el resultado esperado [13]. Existen numerosos frameworks para dar soporte a las pruebas unitarias en diversos lenguajes, como por ejemplo JUnit para Java, y Jasmine o QUnit para Javascript [14][15]. Por último, realizar pruebas no solo se limita a la necesidad de verificar el correcto funcionamiento del software, sino que también juega un papel importante en el diseño y la documentación de la aplicación [1].

4.6 CONTROL DE VERSIONES

El control de versiones es un sistema que facilita la gestión de cambios en documentos, código software, sitios web y otras colecciones de información. Este guarda cambios de un archivo o conjunto de archivos a través del tiempo de tal forma que se puedan visualizar versiones específicas de estos después [16].

El manejo del control de versiones es una actividad que resulta indispensable hoy día en el desarrollo de software, ya que comúnmente una aplicación tiene cientos

o miles de líneas de código y es creada por un equipo de varios desarrolladores. Sincronizar los cambios que realizan unos con los que realizan otros sin utilizar un sistema de versiones puede ser bastante complicado, incluso si los desarrolladores están trabajando en módulos diferentes de la aplicación. Aún un solo programador puede tener muchas dificultades si no tiene una manera de visualizar las modificaciones que ha hecho al software y en llegado caso revertirlas y tomar otro camino.

Los sistemas de control de versiones actuales permiten manejar todos los cambios en el software de forma muy sencilla, sincronizar los cambios con los realizados por otros desarrolladores, y en general, tener un mayor conocimiento y control de lo que se va añadiendo o modificando a un software en un proceso de desarrollo. En la actualidad existen varios sistemas de control de versiones; algunos de los más populares son Git, Subversion, CVS, Mercurial, entre otros.

4.7 USABILIDAD

Cuando un nuevo software se pone en funcionamiento o se realizan mejoras a uno existente, es pertinente realizar pruebas para medir la usabilidad de este. Algunas veces se utiliza la expresión “amigable con el usuario”, pero esta no es la más adecuada ya que define que un sistema es amigable o no es amigable, ignorando que dentro de la usabilidad hay varias variables a considerar y no una sola variable. Por este motivo los profesionales en el desarrollo de interfaces de usuario han optado por otros nombres, como por ejemplo HCI (human-computer interaction). El término usabilidad, se refiere a todos los aspectos de un sistema con los cuales un humano puede interactuar, lo que también incluye la instalación y el mantenimiento. Pero por encima de la usabilidad está la aceptabilidad del sistema, que compone todos los elementos a tener en cuenta para poder afirmar que un sistema cumple con todos los requerimientos del usuario. La aceptabilidad está compuesta por varios factores, entre esos la usabilidad, y la usabilidad a su vez está conformada por cinco dimensiones [17]:

- Facilidad de aprender: Debe ser fácil para el usuario aprender a utilizar el sistema, de tal forma que pueda empezar a trabajar con él tan pronto como sea posible.
- Eficiente de usar: Una vez el usuario ha aprendido a utilizar el sistema, debe ser capaz de obtener el mayor nivel de productividad posible.

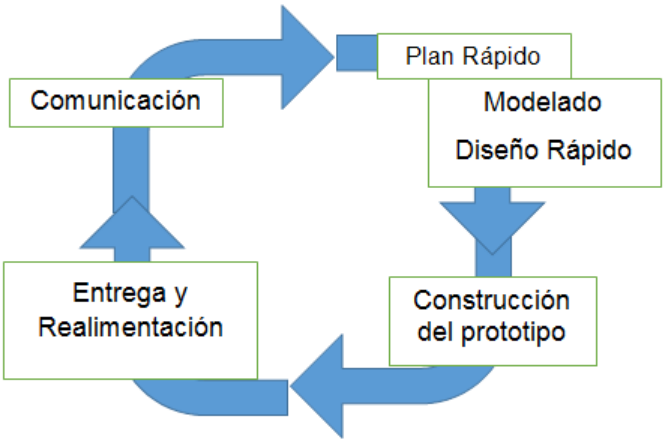
- Fácil de recordar: Una vez el usuario conoce el sistema, después de un periodo de tiempo de no interactuar con él, debe ser capaz de volver a utilizarlo sin necesidad de tener que volver a aprender su funcionamiento.
- Pocos errores: El usuario debe cometer pocos errores con el sistema, en caso de que uno suceda, debe ser capaz de reponerse de él. No deben suceder errores graves por ningún motivo.
- Agradable: El sistema debe ser agradable de utilizar. Los usuarios deben estar subjetivamente satisfechos cuando usan el sistema.

5. METODOLOGÍA

5.1 METODOLOGÍA PARA LA CREACIÓN DEL CONTENIDO Y DEL SITIO WEB

A pesar de que la creación del contenido no era un desarrollo de software, se decidió que era necesaria una metodología adecuada para este fin. La creación del sitio por su parte, iría de la mano con la construcción del contenido, ya que están fuertemente ligados siendo el contenido el que define la estructura del sitio. Los principales criterios para seleccionar una metodología fueron: que permitiera ir evaluando versiones intermedias del contenido para poder ir aplicando los cambios antes de llegar a una versión completa; que facilitara contar con una realimentación a lo largo de la creación del contenido por parte del director del proyecto; y que garantizara alcanzar las características y la calidad deseadas a pesar de que la forma exacta de cómo se presentarían los textos aún estaba definiéndose. Por los criterios antes mencionados, se eligió la metodología de desarrollo por construcción de prototipos [14], que por su estructura (ver Figura 1), orientada a la creación de productos incompletos que se modifican hasta alcanzar el estado deseado, es perfecto para satisfacer las necesidades de la creación del contenido. La metodología se ajusta a las necesidades del proyecto, puesto que no era posible tener un plan de creación totalmente definido, y ésta posibilita realizar cambios según sea solicitado y facilita adaptarse a las necesidades que se detecten posteriormente. Cada ciclo evolutivo consta de cinco pasos:

Figura 1. Paradigma de construcción de prototipos[1]



- Comunicación: Aclaración de dudas y discusión acerca lo que se espera del producto.
- Plan rápido: Se definen los objetivos y la forma en la que se alcanzarán, basados en el paso anterior.
- Modelado y diseño rápido: Análisis de los temas a trabajar y un diseño simple de cómo se desarrollarían.
- Construcción del prototipo: Desarrollo del plan antes mencionado haciendo uso del análisis y diseño propuestos.
- Entrega y realimentación: Entrega del prototipo y recepción de recomendaciones o correcciones al mismo.

En la fase de comunicación se estipulan los objetivos del prototipo y de todo el proceso. En el plan rápido se definen los patrones a estudiar y las modificaciones al contenido existente. Luego en el modelado y diseño rápido se deciden las características específicas del patrón y qué fuentes son las más adecuadas para tener como base. Durante la construcción del prototipo se estudian las fuentes seleccionadas y se toman ideas o características que se consideren útiles de otras fuentes, para finalmente escribir el contenido y realizar las modificaciones necesarias. Luego de construir el prototipo, este se presenta al director de proyecto para revisión, de la cual se reciben una serie de correcciones, así como recomendaciones para los próximos prototipos. Al final de cada iteración se tiene como resultado un incremento en el contenido o una mejora en sus características

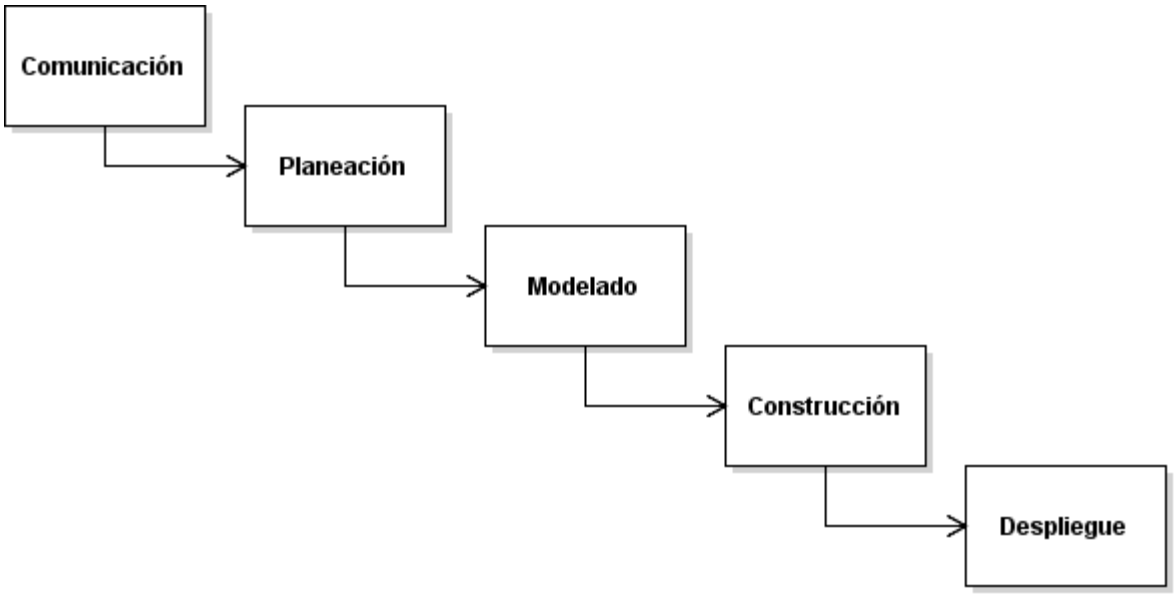
[1] Adaptado de PRESSMAN, Roger. Ingeniería de software un enfoque práctico sexta edición. Editorial Mc Graw Hill 2005. Modelo de construcción de prototipo. 55 p.

en base a la realimentación. De esta forma el contenido no solo incluirá más patrones en cada iteración, sino que además podrá tener correcciones en los existentes. Cada prototipo, producto de un ciclo, sirve como mecanismo para identificar y clarificar los requerimientos que debe cumplir el contenido [18].

5.2 METODOLOGÍA PARA EL DESARROLLO DEL GENERADOR DE CÓDIGO

Teniendo en cuenta la claridad de los requerimientos que debía cumplir el generador de código y lo poco probable que estos cambiaran en algún momento, se optó por desarrollar el software siguiendo el modelo en cascada. Por tanto, el desarrollo siguió las siguientes etapas [18]:

Figura 2. Modelo en cascada[2]



- Comunicación: En esta etapa se definieron y clarificaron los requerimientos que debía cumplir el generador de código.

[2] Adaptado de PRESSMAN, Roger. Ingeniería del software un enfoque práctico sexta edición. Editorial Mc Graw Hill 2005. Modelo en cascada. 50 p.

- Planeación: En esta etapa se definió el plan de desarrollo, los riesgos y las estimaciones necesarias.
- Modelado: En esta etapa se realizó el análisis del software y el diseño del mismo.
- Construcción: En esta etapa se implementó el diseño, realizándose toda la codificación.
- Despliegue: Etapa centrada en la puesta en funcionamiento del sitio web con el generador en un servidor web, y en la realización de las pruebas necesarias.

6. CONSTRUCCIÓN DEL CONTENIDO Y DEL SITIO WEB

6.1 GENERALIDADES

La teoría que sustenta el proyecto, proviene fundamentalmente del trabajo desarrollado por la banda de los cuatro (GoF) en el libro Design Patterns: Elements of Reusable Object-Oriented Software [6], el cual recopila un conjunto de patrones de diseño obtenido a partir de la experiencia de muchos desarrolladores en el área de diseño y desarrollo de software orientado a objetos. También se tomaron en cuenta otros textos dedicados al aprendizaje de los patrones de diseño, que abordan el aprendizaje de los patrones de diseño GoF para programadores que inician en su estudio. Cada uno de ellos ofrece una forma diferente de abordar el aprendizaje de estos patrones, con lo cual se obtiene la ventaja de tener diferentes puntos de vista acerca de cómo deberían ser el contenido teórico y los ejemplos acerca del tema. Algunos de los libros que abordan los patrones de diseño GoF fueron publicados más de diez años después que el libro publicado por la banda de los cuatro.

Durante el desarrollo de los prototipos para la gestión de los contenidos y sus versiones hicimos uso de Google Drive, el cual nos facilitaba llevar un control de los cambios individuales realizados, tener acceso a los documentos por parte de los dos miembros y poder ir compartiendo de forma fácil y segura el contenido con el director de proyecto para recibir de este sugerencias y correcciones.

Para la creación de los diagramas de clases y de secuencias se utilizó la herramienta Violet[3], la cual fue elegida de entre varias opciones por la simplicidad de su uso y aplicación adecuada de los estándares de UML.

6.2 PROTOTIPO I

Comunicación:

En conjunto con el director de proyecto, se definió que objetivo de este primer ciclo y prototipo era realizar una primera aproximación a la manera como se podría ver

[3] Violet es un editor UML libre, desarrollado por Cay S. Horstmann y Alexandre de Pellegrin violet.sourceforge.net/

el sitio web, así como definir qué patrones conformarían el contenido y cuál sería la forma como se describirían los mismos.

Plan rápido:

Respecto al desarrollo del sitio web, el enfoque estaba en el estudio de las distintas herramientas y tecnologías disponibles para desarrollar un sitio que cumpliera con las características deseadas.

Respecto a la creación del contenido, continuaríamos el estudio de los patrones de diseño (el cual habíamos comenzado desde el inicio del proyecto), para lo cual era necesario definir las fuentes que servirían de base para todo la creación del contenido.

Modelado y diseño rápido:

Se decidió utilizar el framework Foundation[4] para la realización de este prototipo.

La estructura del sitio contendría los patrones, el generador de código, las relaciones entre patrones y un acerca de.

Para la estructura del contenido de cada patrón se propuso contener los siguientes elementos: Propósito, También conocido como, El problema, La solución y Consecuencias.

Luego de revisar distintas fuentes y su relevancia en el campo, decidimos utilizar estos 6 libros como fuentes base:

- Design Patterns: Elements of Reusable Object-Oriented Software [6]
- Head First Design Patterns [19]
- Object-Oriented Design & Patterns - Second Edition [20]
- GoF Design Patterns - with examples using Java and UML2 [21]
- The Design Patterns Java Companion [22]
- Agile Software Development, Principles, Patterns, and Practices [1]

[4] Foundation es un framework front-end responsive. http://foundation.zurb.com/

Construcción del prototipo:

El primer prototipo del sitio web quedó como se muestra a continuación:

Figura 3. Página de inicio del primer prototipo

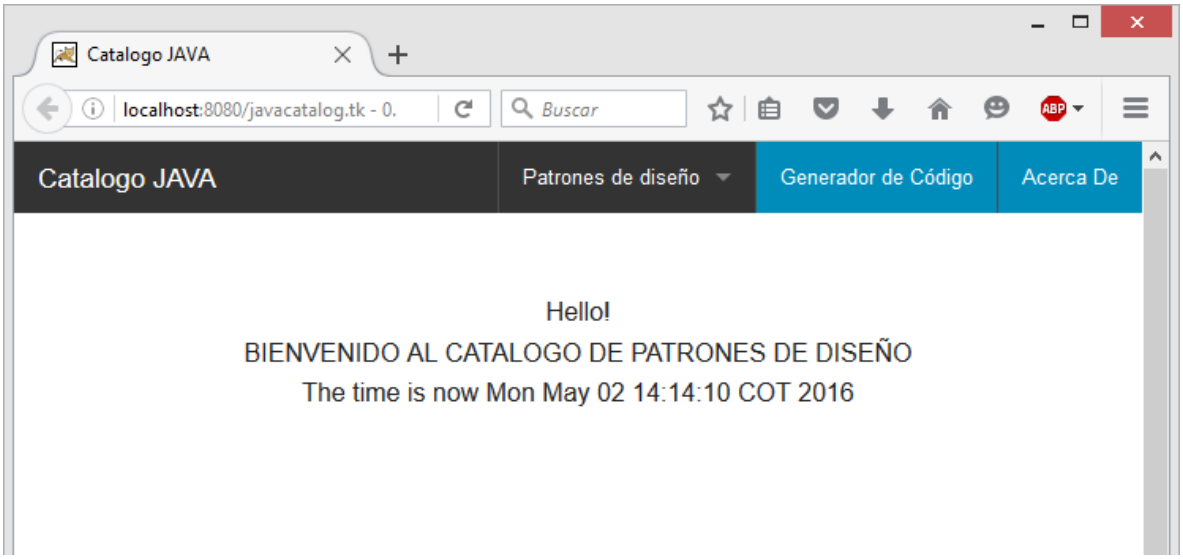
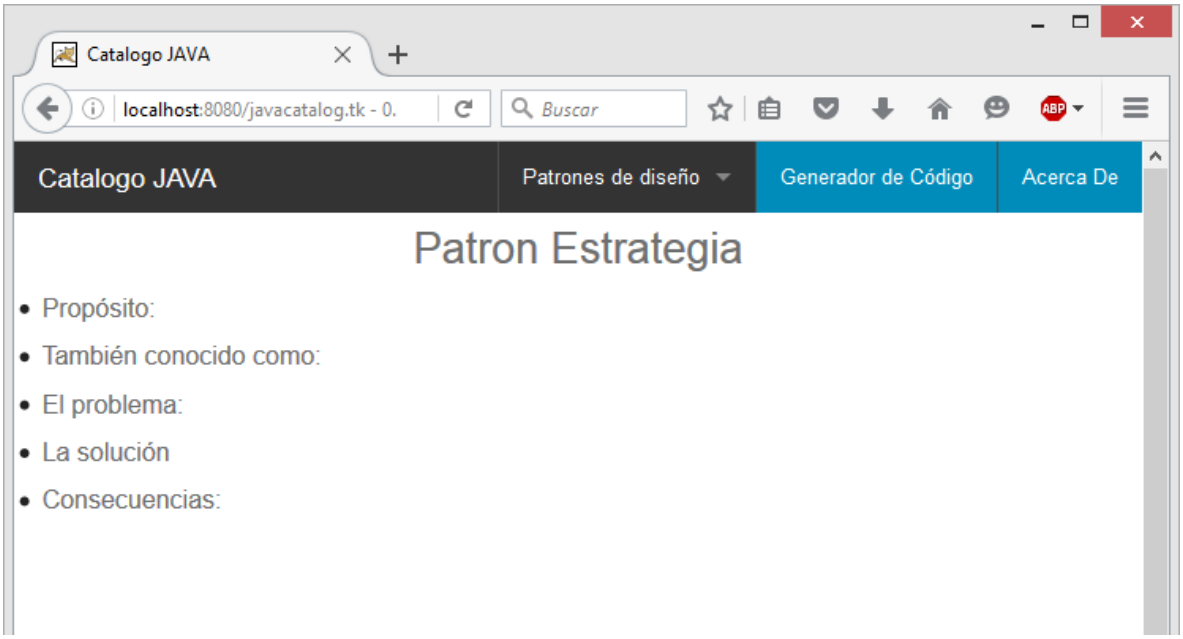


Figura 4. Estructura del contenido en el primer prototipo



Entrega y realimentación:

Este primer prototipo, cumplió su objetivo de ser un acercamiento a la manera como podría lucir el sitio y su contenido, permitiéndonos tener una visión más clara del proyecto y de los pasos a seguir.

El uso del framework Foundation, aunque este es robusto y potente, presentaba unas dificultades técnicas que por el alcance del proyecto no podíamos abordar, por lo que se decidió no utilizar para los siguientes prototipos y dejar la parte visual del sitio en pausa, para enfocarnos en la creación del contenido de los patrones.

6.3 PROTOTIPO II

Comunicación:

Se decidió que se trabajarían los patrones Decorator, Observer, Composite, Strategy y State, por ahora. Quedaban por definir los demás y el número total de patrones a desarrollar.

Se presentó un boceto de contenido para el patrón Strategy proponiendo la estructura de contenido que utiliza la banda de los cuatro [6]: Nombre, Propósito, También conocido como, Motivación, Aplicabilidad, Estructura, Participantes, Colaboraciones, Consecuencias, Código de ejemplo, Usos conocidos y Patrones relacionados. Luego de discutir sobre la mejor forma de organizar el contenido, decidimos utilizar la estructura sugerida por el profesor: Contexto, Problema, Solución y Ejemplos.

Plan rápido:

La tarea a realizarse sería aplicar la estructura definida al patrón Strategy, teniendo el primer contenido completo de un patrón, para evaluar a través de este la forma más adecuada de explicar el patrón. El objetivo principal que se quería

alcanzar era ser lo suficientemente didácticos para que los contenidos fueran fáciles de entender para los estudiantes, pero concisos, de modo que no se convirtieran en textos demasiado largos.

Al mismo tiempo profundizábamos en el estudio de los patrones Decorator y Observer, tomando todos los apuntes pertinentes para la posterior construcción del contenido.

Decidimos también que crearíamos un contenido para la historia de los patrones de diseño, así como para explicar algunos conceptos como los principios SOLID y una vista general de los patrones de diseño.

Modelado y diseño rápido:

Se definió crear archivos de HTML sin ningún tipo de estilo, para empezar a visualizar la estructura de los contenidos. También que empezaríamos a subir los prototipos a un hosting temporal, para ir viendo los prototipos del sitio en funcionamiento.

Construcción del prototipo:

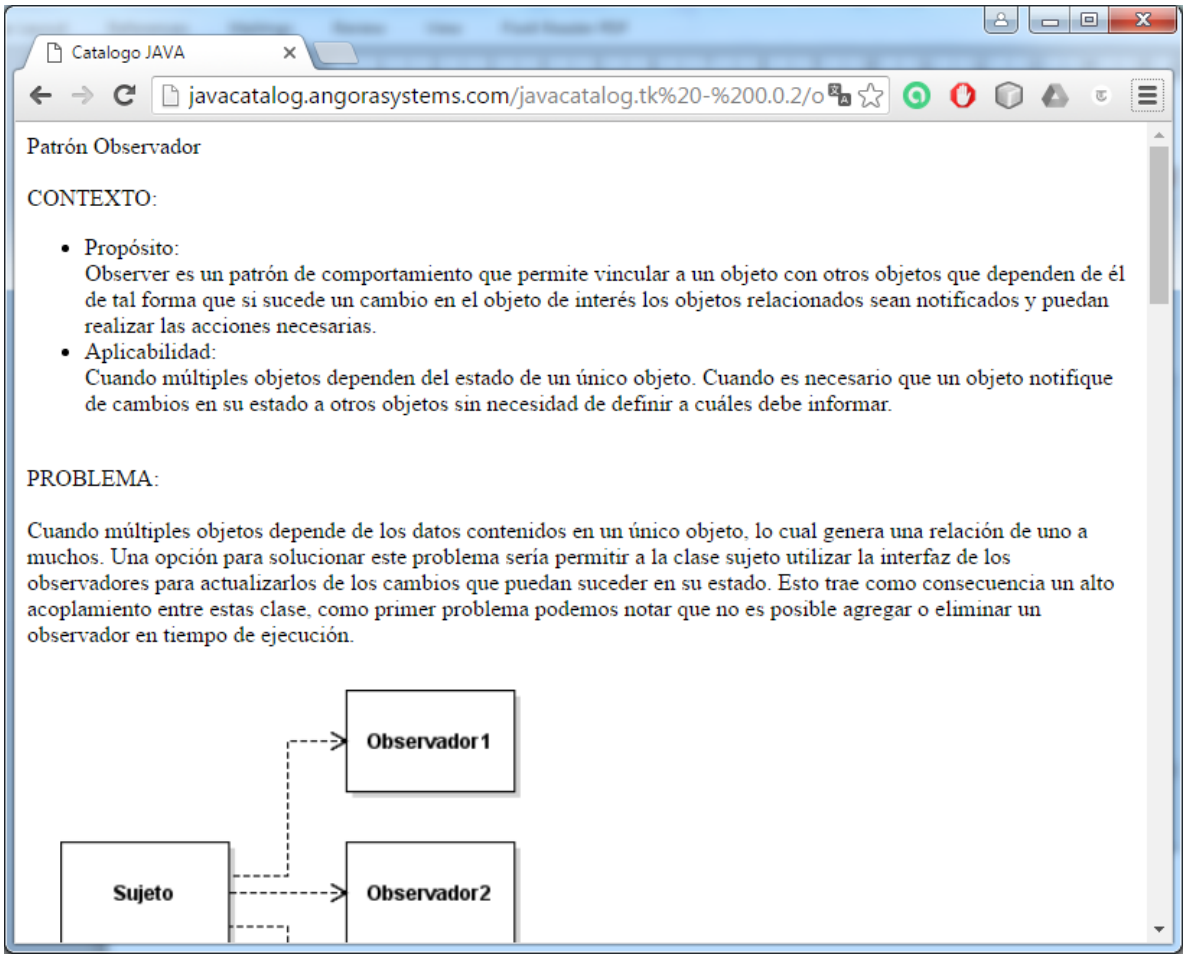
Se completó el contenido para el patrón Strategy aplicando la estructura definida.

Utilizamos un hosting con soporte para PHP y MySQL para albergar temporalmente los prototipos.

Así luce este segundo prototipo:

Figura 5. Inicio del segundo prototipo

Figura 6. Ejemplo de la estructura del contenido en el segundo prototipo



Entrega y realimentación:

El director del proyecto nos hizo varias correcciones al contenido del patrón Strategy; algunas correspondientes a la forma como se escribió el texto y otras correspondientes a los conceptos propios del patrón. Estuvo de acuerdo con los contenidos que quisimos agregar de Historia de los patrones y Vista general de los patrones de diseño.

6.4 PROTOTIPO III

Comunicación:

Se discutió con el director sobre la lista definitiva de los patrones que desarrollaríamos. Finalmente decidimos que realizaríamos un total de 13 patrones, que serían los siguientes:

Patrones creacionales:
- Factory Method
- Singleton
- Builder

Patrones estructurales:
- Decorator
- Composite
- Adapter

Patrones de comportamiento:
- Observer
- Strategy
- State
- Command
- Iterator
- Template Method
- Memento

Decidimos también que los otros contenidos que no corresponden a los patrones, que incluiríamos, serían: Aspectos a tener en cuenta (sobre el sitio web), Principios SOLID, Historia de los patrones de diseño y Relaciones entre patrones.

Plan rápido:

El objetivo de este ciclo era avanzar lo más posible la creación de los contenidos de los patrones.

Por otro lado, reanudamos la tarea de definir cómo se vería el sitio a nivel de diseño, por lo que iniciaríamos a estudiar y hacer pruebas con HTML5, CSS3 y JavaScript.

Modelado y diseño rápido:

Decidimos agregar dos nuevos libros a la lista de libros base:

- Design Patterns Java Workbook [23]
- Software Architecture Design Patterns in Java [24]

Adicionalmente, investigamos una manera de hacer interactiva la descripción de los elementos del diagrama de clases de cada patrón; queríamos que la descripción apareciera al pasar el cursor por encima del elemento. Para realizar esto, encontramos una solución utilizando CSS.

Construcción del prototipo:

Desarrollamos los contenidos correspondientes a Decorator, Composite, Observer, State y Factory Method.

Implementamos lo necesario para que mostrara la descripción de los elementos al pasar el cursor por encima de ellos; aplicamos esto a algunos de los patrones desarrollados.

Así luce el tercer prototipo:

Figura 7. Inicio del tercer prototipo

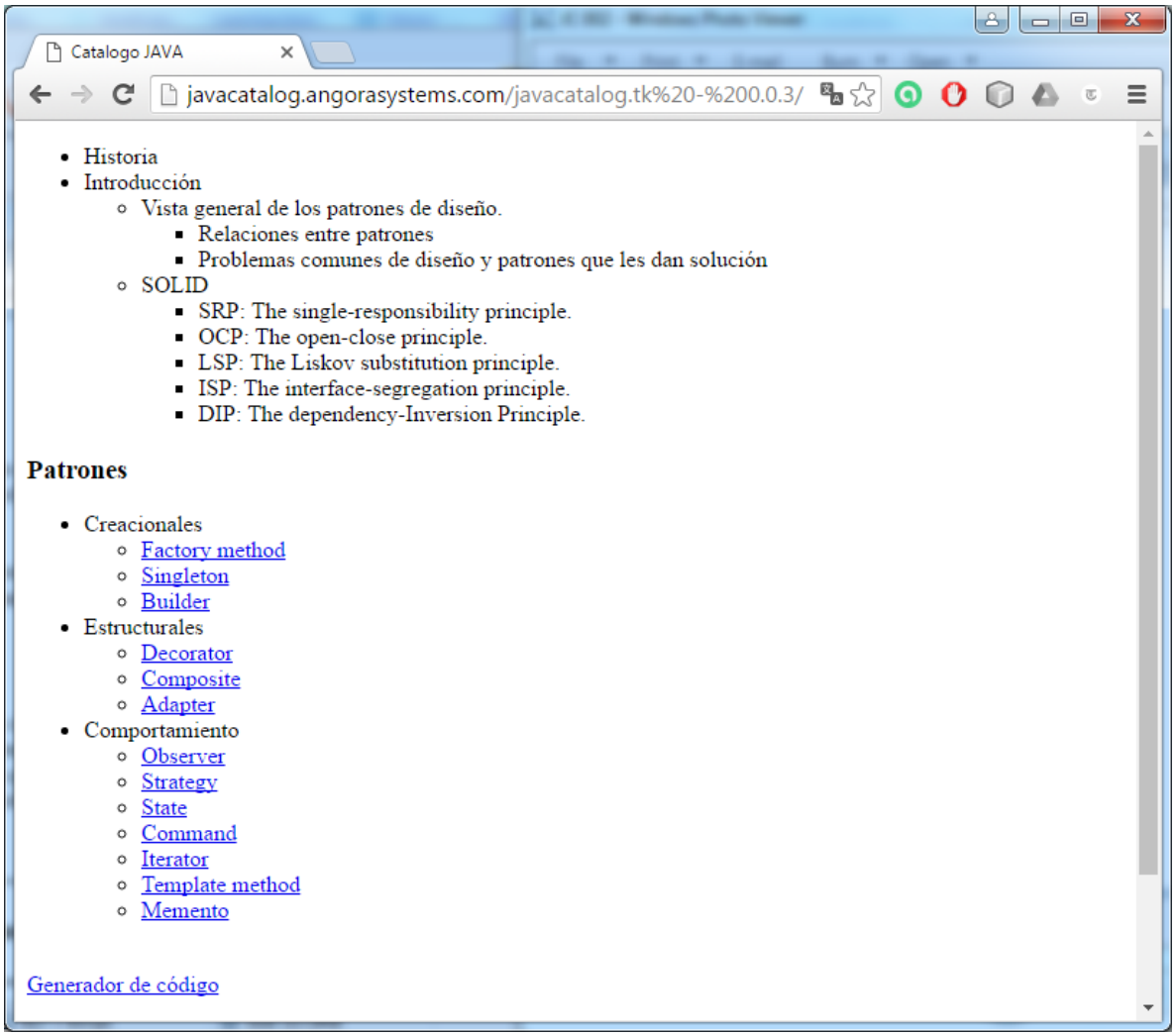
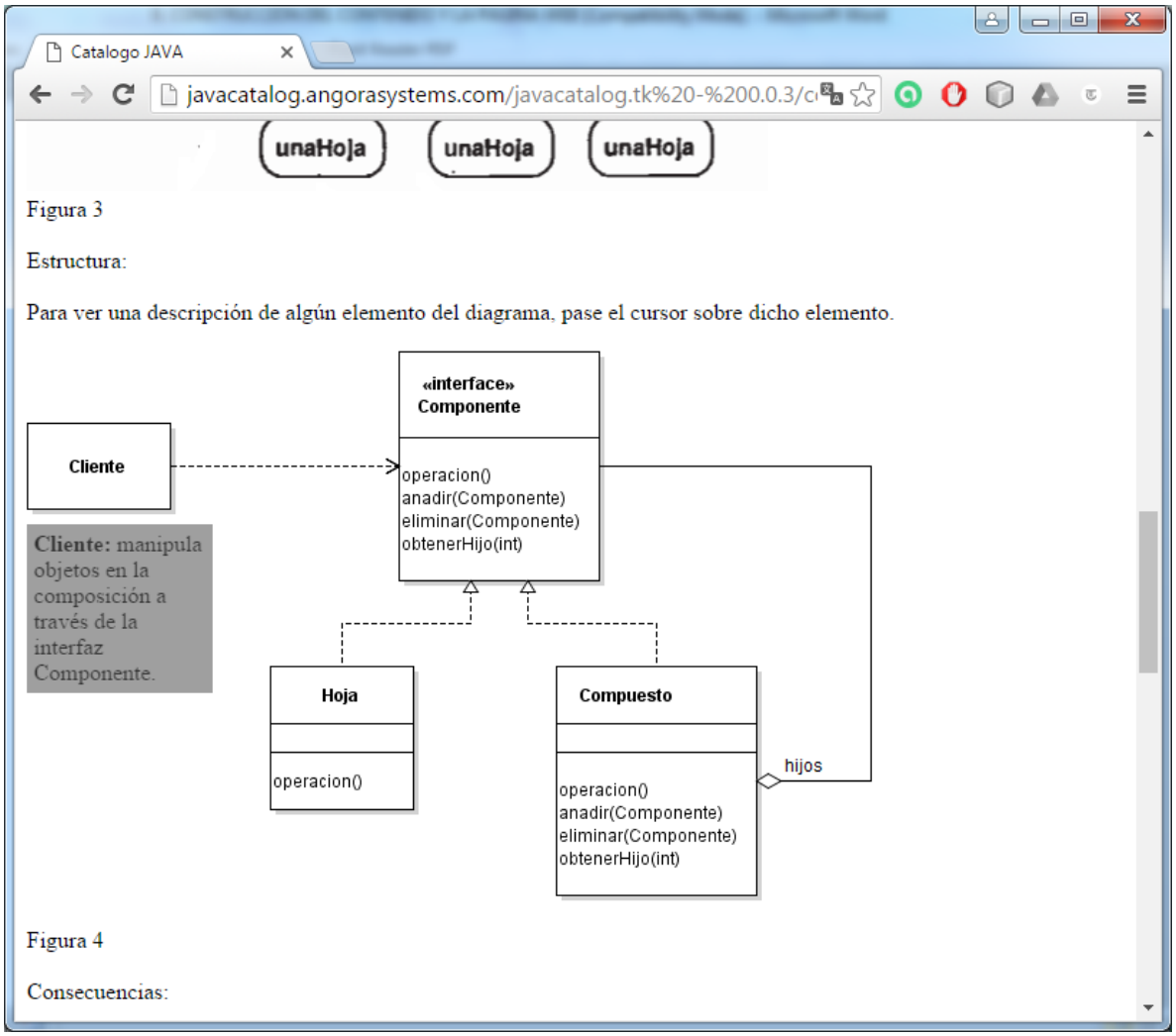


Figura 8. Mejoras a la estructura del contenido del tercer prototipo



Entrega y realimentación:

El director nos realizó las correcciones y sugerencias pertinentes respecto a los contenidos desarrollados. Asimismo, estuvo de acuerdo en mostrar la descripción de los elementos de una forma interactiva.

Por otro lado, decidimos que usaríamos una plantilla para el diseño del sitio web, de modo que se nos facilitara un poco este aspecto, pudiendo enfocar nuestros

esfuerzos en el centro del proyecto: crear el contenido y desarrollar el generador de código.

6.5 PROTOTIPO IV

Comunicación:

Uno de los objetivos principales de este ciclo era terminar el contenido de los patrones y los demás contenidos.

Plan rápido:

Decidimos buscar y comprar una plantilla (template) y adaptarla a las necesidades de nuestro sistema. Asimismo, buscamos una persona con experiencia en desarrollo web para asesorarnos en este aspecto.

Modelado y diseño rápido:

Para este ciclo hicimos tres prototipos no funcionales, antes de pasar al diseño final usando la plantilla. El primero de ellos fue realizado utilizando la herramienta Axure[5]:

[5] herramienta para realizar prototipos/mockups de aplicaciones o páginas web. http://www.axure.com/

Figura 9. Prototipo no funcional del inicio del sitio web

Figura 10. Prototipo no funcional del contenido del sitio web

Y dos prototipos adicionales evaluando un diseño más detallado:

Figura 11. Prototipo no funcional del inicio del sitio web realizado con más detalle 1

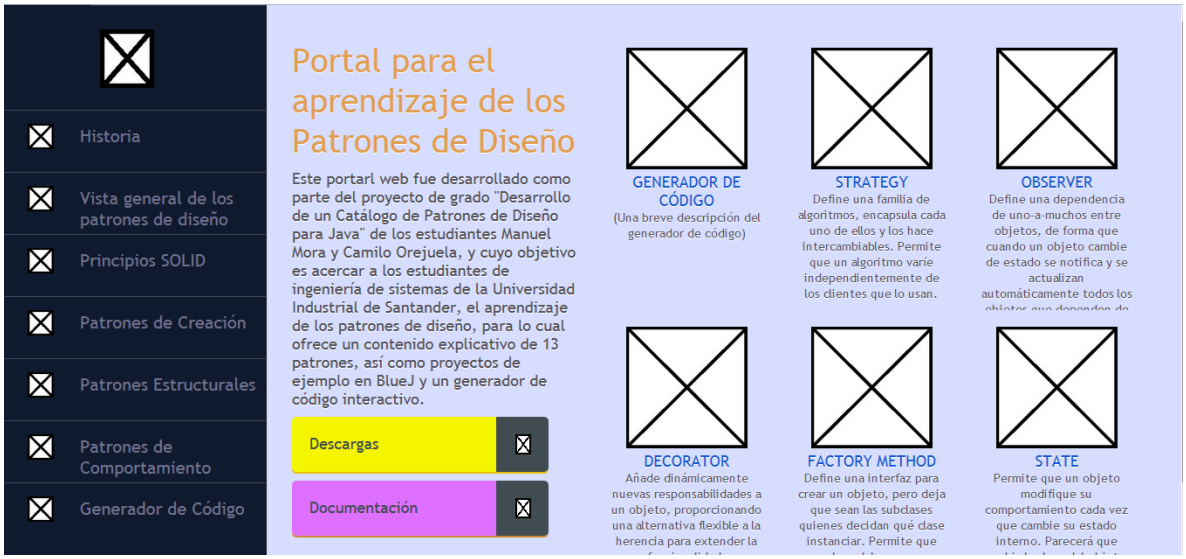
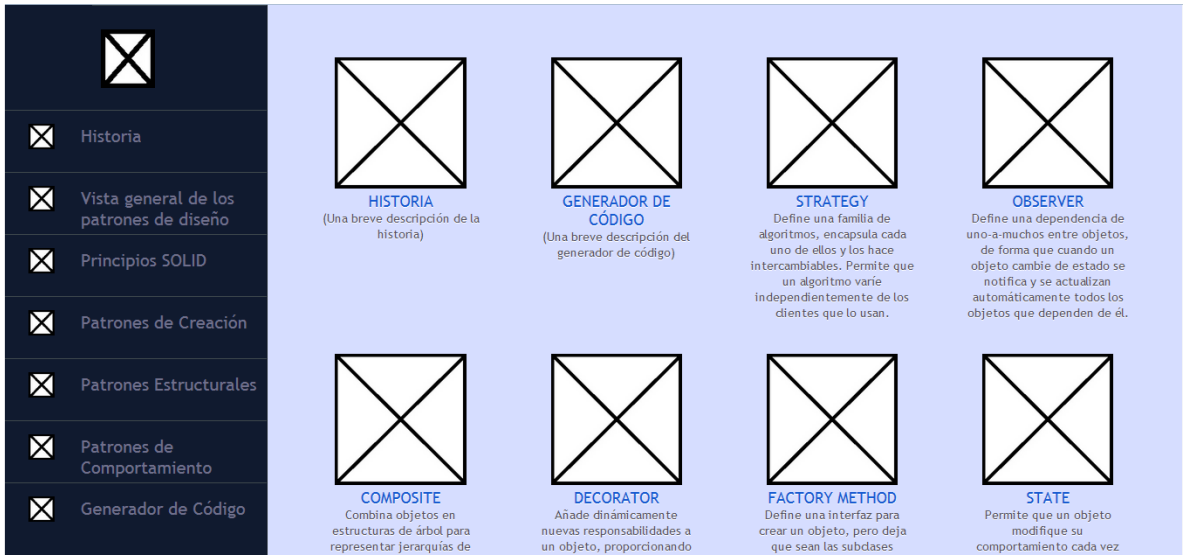


Figura 12. Prototipo no funcional del inicio del sitio web realizado con más detalle 2



Construcción del prototipo:

Logramos terminar la creación de todos los contenidos del sitio.

Y finalmente, la implementación inicial de la plantilla para el sitio, luce de la siguiente manera:

Figura 13. Inicio del cuarto prototipo

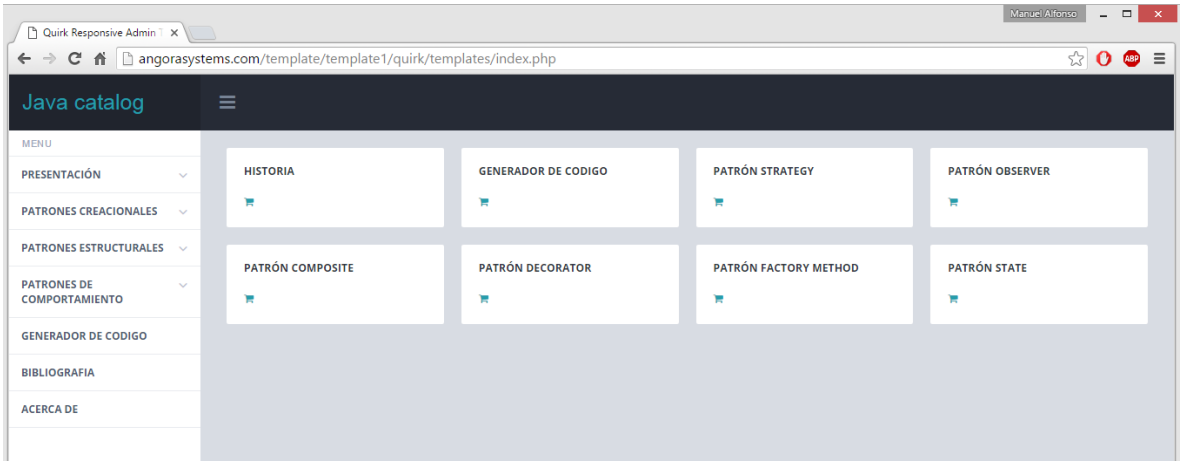


Figura 14. Ejemplo del contenido del cuarto prototipo

Entrega y realimentación:

El director nos dio sus apreciaciones sobre el sitio, sugiriéndonos modificar algunas cosas referentes al diseño y a algunas funcionalidades que presentaron defectos leves. También nos indicó correcciones sobre los últimos contenidos agregados al sistema.

6.6 PROTOTIPO V

Comunicación:

Esta etapa del proceso consistía en mejorar el diseño del sitio y llevar a cabo todas las mejoras indicadas por el director y otras detectadas por nosotros.

Por otro lado, se contempló la posibilidad de tener el portal en un servidor en la universidad. Esto posibilitaría que tuviera un lugar seguro donde hospedarse y que la universidad y la escuela de ingeniería de sistemas en específico, tuviera el sistema en su poder para poder mantenerlo a futuro y aprovecharlo al máximo como herramienta de apoyo para el aprendizaje de los patrones de diseño por parte de los estudiantes.

Plan rápido:

Desde que decidimos utilizar esta plantilla, sabíamos que debíamos crear unas ilustraciones para poner en el inicio, que embellecieran el sitio y acompañaran los enlaces a las páginas más importantes dentro de este. Así que nos pusimos en contacto con una persona idónea para esta labor.

Nos pusimos en contacto con el grupo GID-CONUSS para solicitar sus servicios y albergar el sitio en sus servidores.

Modelado y diseño rápido:

Nos reunimos con el ilustrador y definimos los conceptos a plasmar en las imágenes.

Realizamos toda la gestión para que pusieran a nuestra disposición una máquina virtual en sus servidores. Por otro lado realizamos la gestión con el DSI (Departamento de Sistemas de Información) para que nos proveyeran de una IP pública para acceder desde internet y un subdominio.

Construcción del prototipo:

Obtuvimos el acceso desde internet, y subimos el sitio al servidor.
El dominio por el que se accede ahora a nuestro portal es: http://patdis.uis.edu.co

Y finalmente, después de haber realizado muchas mejoras, así luce el sitio:

Figura 15. Inicio del quinto prototipo

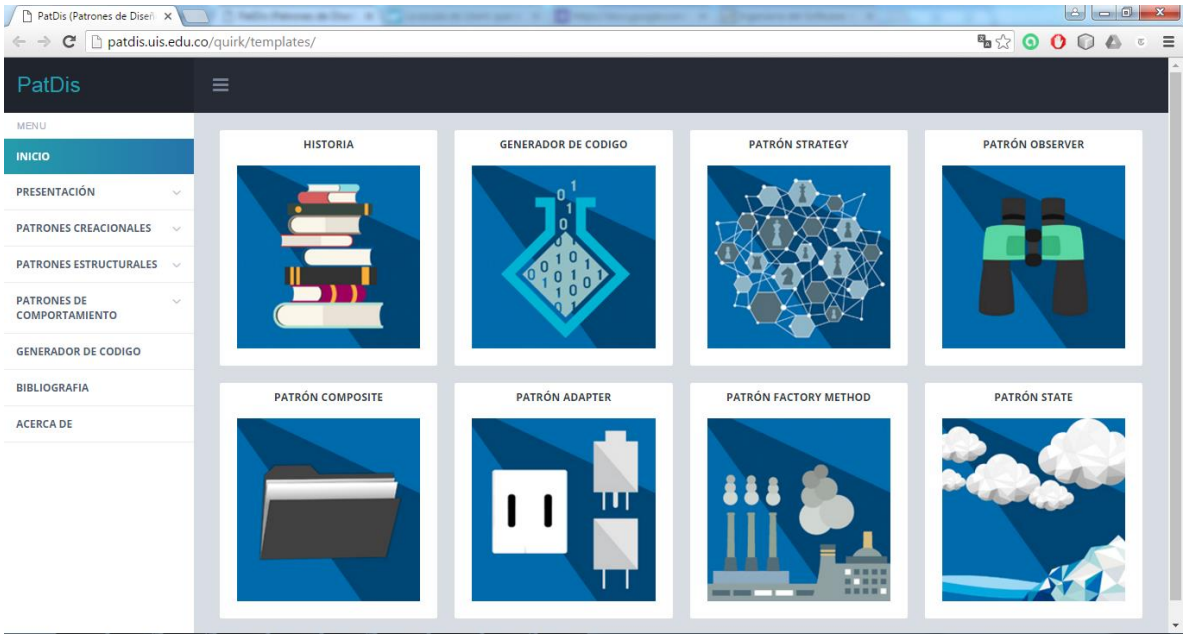
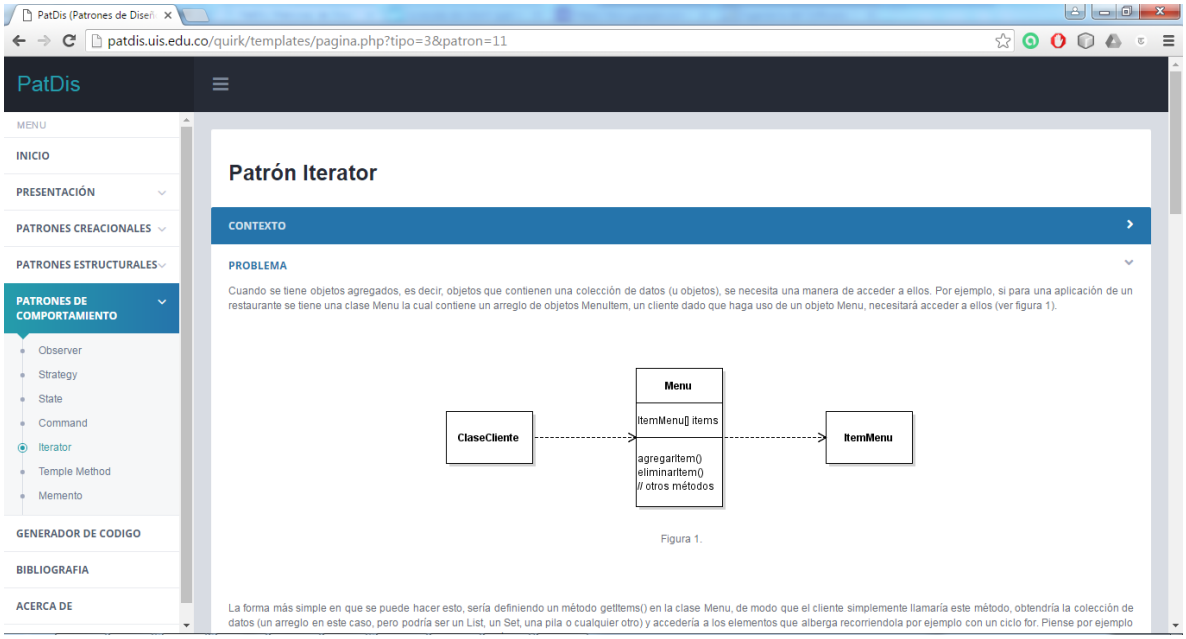


Figura 16. Ejemplo del contenido del quinto contenido



Entrega y realimentación:

Tanto nosotros como desarrolladores del sitio y su contenido, como el director del proyecto, quedamos muy satisfechos con el resultado final. Gracias al refinamiento que se hizo al sistema, prototipo tras prototipo, aplicando correcciones y mejoras una y otra vez, se logró concebir un sistema con una calidad alta, lo cual se debería reflejar en las pruebas de usabilidad.

6.7 CONTENIDO DE PATRÓN DE EJEMPLO

Para ilustrar la manera como desarrolló el contenido de los patrones, se muestra a continuación el contenido para el patrón Builder.

PATRÓN BUILDER

CONTEXTO

Propósito:
Puede ser usado para facilitar la construcción de un objeto complejo a partir de objetos simples. También separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción puede ser usado para crear otra composición de objetos.

Aplicabilidad:
Use Builder cuando

- se quiere realizar la creación de objetos complejos a través de una serie de pasos (un algoritmo).
- se quiere encapsular la creación de objetos complejos.
- el algoritmo que define la construcción de un objeto complejo debe ser independiente de las partes de que se compone dicho objeto.
- el proceso de construcción debe permitir diferentes representaciones para el objeto que es construido.
- se quiere aislar al cliente del conocimiento del proceso de creación de objetos.

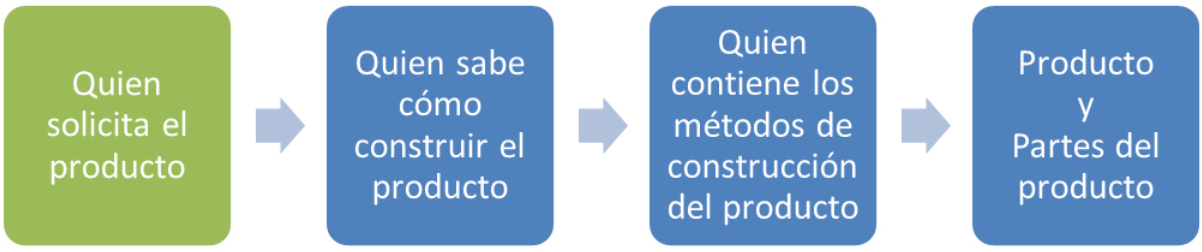
PROBLEMA

El problema que soluciona el patrón Builder (constructor, en español) tiene que ver con la creación de objetos que en ocasiones puede llegar a ser algo complejo; entendiendo por creación como el proceso de construcción, que no solo incluye la instanciación sino también la configuración del objeto (utilizando métodos set, por ejemplo). Una aplicación dada puede requerir que los objetos de un determinado tipo, sigan una serie de pasos en su creación, es decir un algoritmo, y que estén compuestos de otros objetos; de forma que estos objetos compuestos puedan variar mucho unos a otros aun siendo del mismo tipo, creándose de alguna manera personalizados a las necesidades de un cliente. Builder ayuda a abordar de una manera adecuada esta complejidad en la creación de objetos; específicamente cuando se necesita crear objetos complejos, que requieren de una serie de pasos y que se componen de otros objetos.

SOLUCIÓN

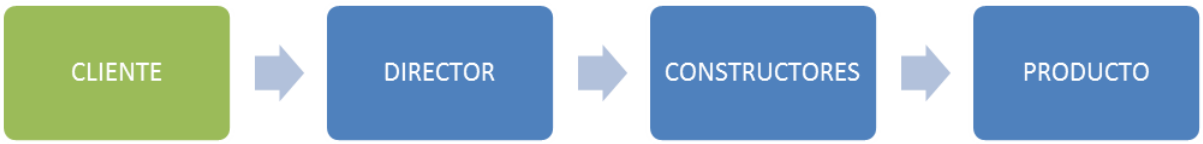
Builder lo primero que hace es separar el código de la aplicación en varias partes: el código cliente quien solicita la creación del objeto (del producto), el código que se encarga del paso a paso de construcción, el código que ejecuta dichos pasos y por último el código de representación del producto en construcción (ver figura 17).

Figura 17. Partes que interactúan en el patrón Builder



Estas partes son respectivamente un cliente dado, y las clases del patrón: Director, los constructores (Constructor y sus subclases ConstructorConcreto) y Producto (las clases que representan el producto y sus partes) (ver figura 18) (ver figura 20).

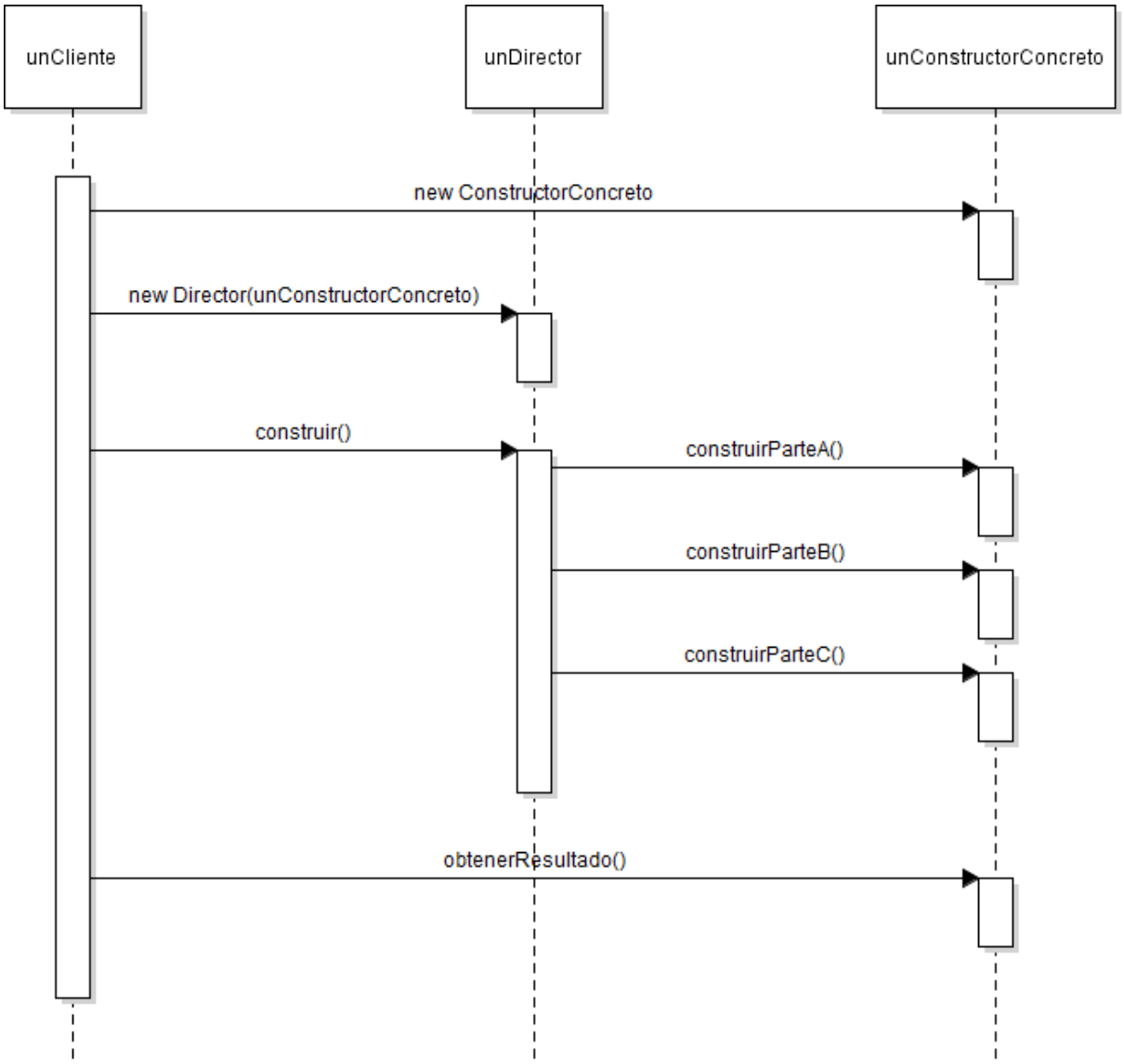
Figura 18. Participantes en el patrón Builder



Y la forma como interactúan estas partes o componentes es la siguiente (ver figura 19):

- El cliente crea el objeto Director y lo configura con el objeto Constructor deseado.
- El director notifica al constructor cada vez que hay que construir una parte de un producto.
- El constructor maneja las peticiones del director y las añade al producto.
- El cliente obtiene el producto del constructor. Este paso puede que no sea necesario o que sea el director y no el constructor quien devuelva el producto al cliente.

Figura 19. Diagrama de secuencia del patrón Builder



El patrón Builder es más complejo que el patrón Factory Method, ya que en este no solo se escoge entre un conjunto de fábricas (como se hace en Factory Method), sino que se sigue un algoritmo y se crean objetos más complejos y diversos, lo cual no es posible con la simple elección de una fábrica.

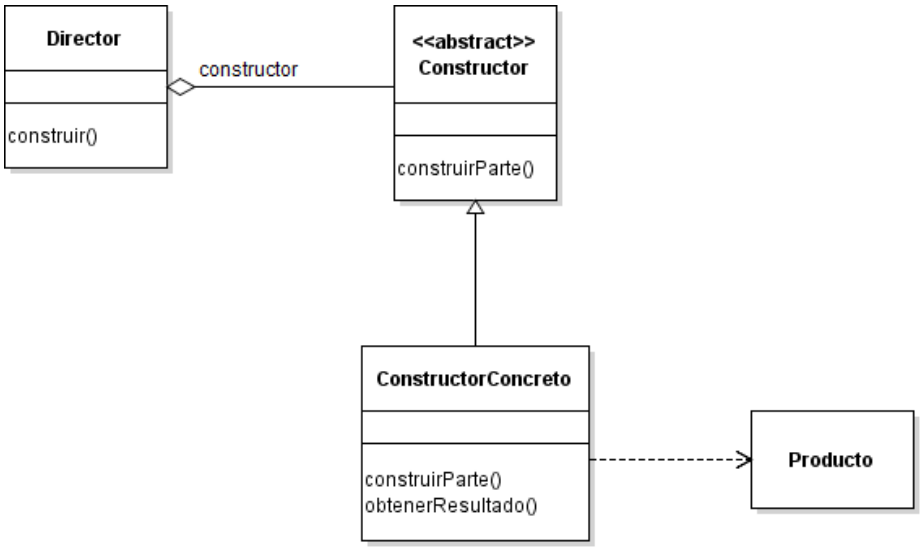
Una de las principales ventajas de Builder es la modularidad, ya que al separar las partes implicadas en la construcción, hace mucho más fácil la mantenibilidad y

extensibilidad de la aplicación a futuro. Por ejemplo, se podrían agregar más constructores simplemente agregando nuevas clases y extendiendo la clase Constructor, sin tener que modificar el código existente. Así mismo, se podría modificar el algoritmo de construcción, yendo directamente a la clase Director y haciendo las modificaciones pertinentes, sin correr el riesgo de afectar otras partes de la aplicación. Y de igual manera podría modificarse la representación del producto y sus partes, agregando por ejemplo más clases o modificando las existentes, sin que esto afecte el proceso de construcción.

Otra ventaja que tiene el patrón, gracias a la modularidad, es que cada componente no tiene conocimiento de la implementación de los demás componentes. Por ejemplo, un cliente solicitará un objeto y no sabrá de qué manera fue construido; tampoco conocerá de la implementación del constructor e incluso del producto que está creando.

Estructura:

Figura 20. Diagrama de clases del patrón Builder



Definición de los elementos:

Director:

- implementa el algoritmo de creación de los Productos.
- construye un objeto Producto usando la interfaz Constructor.

Constructor:

- define la interfaz para crear las partes de un objeto Producto.

ConstructorConcreto:

- implementa la interfaz Constructor para construir y ensamblar las partes del producto.

Producto:

- representa el objeto complejo en construcción.
- junto a Producto estarían las clases que definen las partes que lo componen, incluyendo interfaces.

Consecuencias:

- Encapsula la forma como un objeto complejo es construido.
- Permite que los objetos sean construidos en un proceso de múltiples pasos y variante.
- Construir objetos requiere más conocimiento del dominio del cliente que cuando se crea una Fábrica (cuando se aplica el patrón Factory Method).
- El código de construcción está aislado del código de representación y ambos son fáciles de reemplazar sin afectar el otro.
- Cada constructor específico es independiente de los otros y del resto del programa. Esto hace la adición de otros constructores relativamente simple.
- Gracias a que cada constructor construye el producto final paso por paso dependiendo de los datos, se tiene más control sobre cada producto final que se construye.

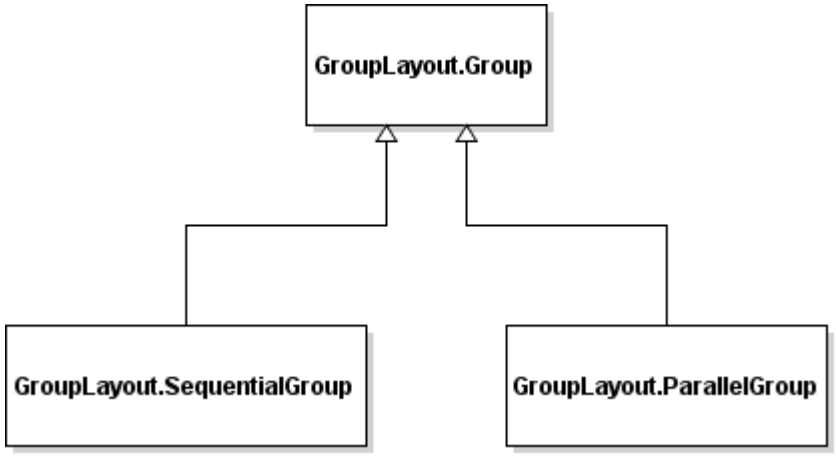
EJEMPLOS

- Ejemplo de la API de Java: javax.swing.GroupLayout.Group#addComponent()

GroupLayout es un LayoutManager que agrupa jerárquicamente componentes para ubicarlos en un Container. GroupLayout está para usarse utilizando constructores (builders), aunque también puede programarse manualmente. La agrupación es hecha por instancias de la clase **Group**. GroupLayout soporta dos

tipos de grupos. Un grupo secuencial ubica sus elementos hijos secuencialmente, uno después del otro. Un grupo paralelo alinea sus elementos hijos en una de cuatro maneras. Estos dos tipos de grupos los definen las clases **SequentialGroup** y **ParallelGroup** respectivamente, las cuales son subclases de Group, y junto a ésta están anidadas dentro de la clase GroupLayout (ver figura 21).

Figura 21. Diagrama de clases del ejemplo de la API del patrón Builder.



La clase Group contiene el método addComponent() que recibe como parámetro un objeto de tipo Component. Sus subclases, SequentialGroup y ParallelGroup, sobrescriben este método.

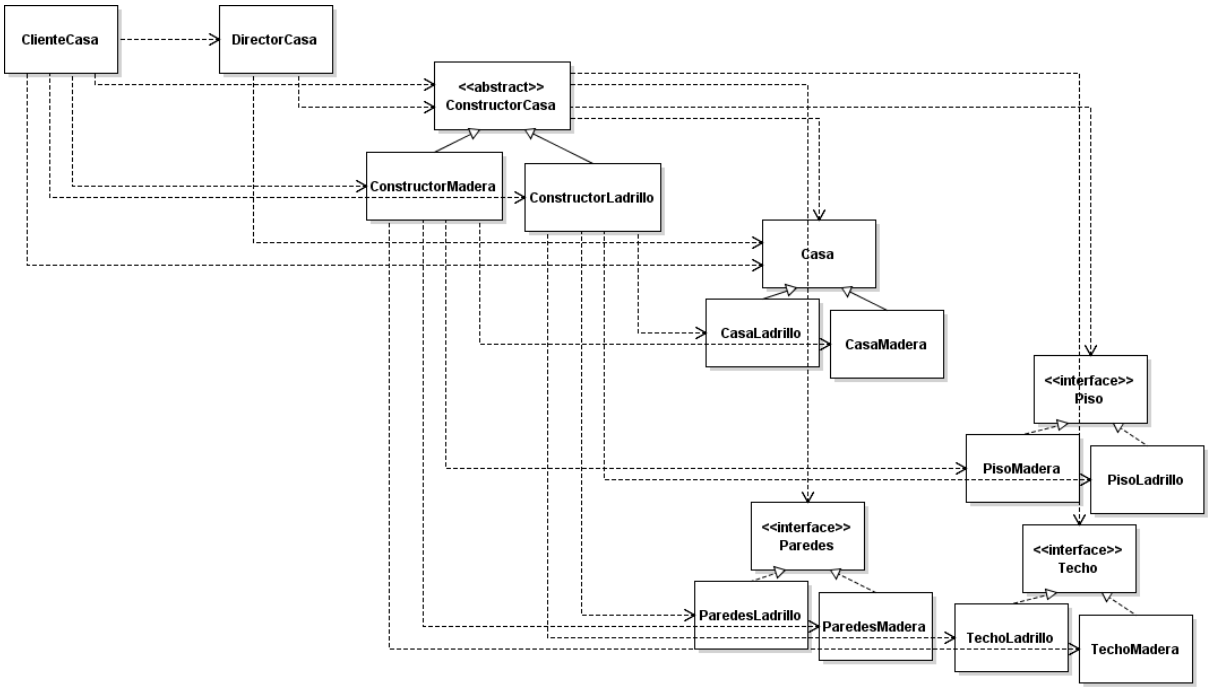
Ni SequentialGroup, ni ParallelGroup tiene constructores públicos, sino que se instancian a través de los métodos createSequentialGroup() y createParallelGroup() de la clase GroupLayout.

Para la estructura del patrón, la clase Group es el Constructor, sus subclases SequentialGroup y ParallelGroup son las clases ConstructorConcreto, y los objetos Component son el Producto.

- Ejemplo de construcción de una casa:

La construcción de una casa requiere de una serie de pasos; a groso modo, primero construir el piso, luego construir las paredes y por último construir el techo. Asimismo, un objeto casa estaría compuesto por varios objetos, que para el ejemplo serían precisamente piso, paredes y techo. Pero además existen diferentes tipos de piso, diferentes tipos de paredes y diferentes tipos de techo, así que habría que definir el conjunto de partes específicas que tendría una casa dada. También existen diferentes tipos de casas, como casas de madera por ejemplo, donde tanto el piso, como las paredes y el techo serían de madera; en este caso, al especificar el tipo de casa, no sería necesario especificar de qué partes específicas se compone (ya estaría definido). Teniendo en cuenta todo lo anterior, se sabe que la creación (construcción) de estos objetos casa no es una tarea tan simple y que la aplicación necesita un diseño apropiado.
En este ejemplo se definen dos constructores, ConstructorMadera y ConstructorLadrillo para construir casas de madera y ladrillo respectivamente. DirectorCasa define los pasos a seguir en la construcción. ClienteCasa es quien solicita la construcción de una casa. Y el producto es Casa con sus respectivas partes (Piso, Paredes y Techo).

Figura 22. Diagrama de clases del ejemplo del patrón Builder



6.8 EJEMPLOS DE LOS CONTENIDOS

El contenido en todos los patrones describe en la parte final uno o dos ejemplos prácticos de aplicación del patrón. De dicho(s) ejemplo(s) se muestra el diagrama de clases, una descripción breve y un enlace a través del cual el estudiante puede descargar el código de dicho ejemplo. Todos los ejemplos están en Java y listos para utilizar en el IDE Bluej.

Estos ejemplos sirven como guía y complemento del contenido teórico de los patrones. En algunas ocasiones, desde el contenido teórico, se hace referencia a dichos ejemplos.

7. DESARROLLO DEL GENERADOR DE CÓDIGO

Como se había mencionado anteriormente, se eligió el modelo en cascada como metodología en el desarrollo del generador de código. A continuación, se explica de forma detalla cómo se llevó a cabo cada una de las etapas del modelo, además se presentan los documentos y herramientas de las que se hizo uso.

7.1 COMUNICACIÓN

En esta etapa se discutió cuáles eran los requerimientos que debía cumplir el generador de código para lograr el objetivo de ser una herramienta que da soporté al aprendizaje de los patrones de diseño [25].

7.1.1 Enunciado del problema

- Propósito del sistema:

  Una aplicación web que ofrezca al usuario la posibilidad de interactuar con los elementos de los diagramas de clases de los patrones, en donde pueda modificar dichos elementos y adaptarlos a las necesidades del problema específico que quiera resolver; en general, utilizar un determinado patrón y generar su código en Java.

- Alcance del sistema:

  Los patrones con los que el usuario podrá interactuar son los 13 patrones GoF que se explicaron en el contenido teórico: Factory Method, Singleton, Builder, Composite, Decorator, Adapter, Strategy, State, Observer, Command, Iterator, Template method, Memento.

  El sistema le permite al usuario cambiar los atributos, los métodos y el nombre del elemento que seleccione. Una vez modifique los datos de la forma en que desea, el usuario puede generar y descargar el código fuente correspondiente.

- Objetivos y criterios de éxito:

  - Superar las pruebas de usabilidad.
  - Terminar el generador en los tiempos estipulados en la planeación.

- Cumplir los requerimientos aquí definidos.

7.1.2 Obtención de requerimientos. En el siguiente paso de esta etapa se describe el proceso para la obtención de los requerimientos del generador de código:

- Panorama:

  El sitio web dentro del que se ubica el generador de código, tiene como objetivo facilitar a los estudiantes el aprendizaje de los patrones de diseño; para esto alberga un contenido teórico de explicación de cada patrón, acompañado de ejemplos. De modo que el generador de código sirve para que el estudiante, una vez haya estudiado un patrón, practique aplicando dicho patrón a un problema específico que esté resolviendo. La manera como lo hace es: eligiendo de una lista desplegable el patrón que quiere implementar, luego interactuando con el diagrama de clases de dicho patrón, modificando los nombres de los elementos, así como sus atributos y métodos. Una vez adaptado el diagrama/patrón al problema que quiere resolver el usuario, éste hace clic en el botón Generar, con lo que obtendrá a modo de descarga el código fuente del diagrama como lo definió.

- Requerimientos funcionales

  Las funcionalidades que debía cumplir el generador de código son:

  - Permitirle al usuario generar el código de uno de los patrones de diseño.

  - Permitirle al usuario modificar las características de uno de los elementos que conforman la estructura del patrón de diseño, pero restringiendo que un mal uso de esta funcionalidad estropee la aplicación adecuada del patrón.

  - El generador debe asegurarse que la información ingresada por el usuario cumple con la sintaxis del lenguaje de programación Java, mientras que las palabras reservadas e implementaciones que utilice estén soportadas.

- Requerimientos no funcionales

  - Usabilidad:

    El usuario posee conocimientos avanzados de informática y programación, lo cual permite hacer uso del lenguaje técnico que sea

necesario. Es necesario incluir un tutorial o ayuda para explicar el funcionamiento de la aplicación.

- Confiabilidad:

  La aplicación debe de generar de forma correcta el código que el usuario solicite y en caso de generarse un error informar de lo sucedido al usuario. El generador de código debe estar disponible siempre que el sitio web se encuentre en funcionamiento. En caso de producirse un fallo o no encontrarse disponible el módulo de generación de código debe mostrarse una alerta para informar al usuario.

  En cuanto a la seguridad de los datos, la aplicación no maneja información delicada y un fallo no representa la pérdida de datos indispensables. Sí debe garantizarse que la comunicación con el servidor sea segura y este no se vea expuesto a ataques.

- Rendimiento:

  La aplicación debe responder de forma rápida (menos de 1 segundo) a las interacciones del usuario, exceptuando la generación de código que dependerá del tamaño. No se espera una carga excesiva del sistema en determinados momentos y no hay tareas que puedan perjudicar el rendimiento. Aun así, debe de garantizarse que la aplicación pueda ser utilizada por dos o más usuario sin que esto repercuta en fallas o demoras en el sistema.

- Soportabilidad:

  Se espera en un futuro poder incluir más funcionalidades al software como agregar o quitar elementos y ampliar el número de patrones soportados. Con respecto al mantenimiento del sistema solo es necesario: a corto plazo controlar el funcionamiento del servidor y a largo plazo realizar actualizaciones a medida que las tecnologías cambien y sea indispensable actualizar el sistema.

- Implementación:

  Pueden existir limitaciones con respecto a los recursos disponibles en el servidor para prestar el servicio a una gran cantidad de usuarios y esto repercutir en el rendimiento. Es necesario tener en cuenta esta limitante durante el desarrollo para evitar el uso inadecuado de los recursos por parte de la aplicación. Por la parte

del cliente, es indispensable verificar la compatibilidad con las tecnologías que se estén utilizando en el desarrollo. Para el cliente con un ordenador de gama media o baja no debería representar un problema hacer uso de la aplicación.

- Interfaz:

  La interfaz de la aplicación puede inspirarse o guardar similitudes con las herramientas CASE que se encuentran en el mercado. Es de vital importancia tener en cuenta los estándares existentes, hacer que sea intuitiva y fácil de utilizar; la usabilidad juega un papel fundamental en el sitio web en el cual se albergará el generador.

- Operación:

  El sistema siempre se encuentra en ejecución en el servidor en el que está albergado. Cada vez que el servidor inicie, la aplicación debería estar lista para su uso.

- Empaquetamiento:

  La aplicación se entregará en un archivo comprimido en alguno de los formatos populares (.rar, .zip o 7zip). Para instalar la aplicación será necesario seguir los pasos especificados en el archivo README.txt, que contiene las especificaciones de las modificaciones necesarias para que el generador y el sitio web funcionen correctamente dentro del servidor en el que sean instalados. El documento también especifica las tecnologías necesarias para poder funcionar.

- Legal:

  La aplicación no modificará sistemas externos y por tanto no estará relacionado con un fallo en el equipo del cliente o el servidor. En cuanto a los derechos de la aplicación, pertenecen a la UIS y a los desarrolladores del proyecto. Todo el software que se utilizase en el desarrollo de la aplicación se encuentra bajo alguna licencia open source y por tanto no es necesario pagar regalías o algún tipo de licencia.

- Modelo funcional

  A continuación, se muestra un ejemplo de los escenarios desarrollados

durante la etapa de obtención de requerimientos. Todos los escenarios y casos de uso se encuentran en el anexo A.

- Escenario:

Tabla 1. Ejemplo de escenario

Nombre: Utilización de Singleton para la conexión con una base de datos en una aplicación.
Actor(es): Luis (Estudiante).
Flujo de eventos: 1. Luis ingresa a la página web (http:) y esta se carga en el navegador. 2. Luis ingresa al Generador de código clicándolo en el panel de la izquierda. 3. El sistema muestra la interfaz del Generador de código con la cual Luis puede empezar a interactuar. 4. Luis despliega la lista de patrones en la casilla Patrón. 5. Luis selecciona Singleton en la lista. 6. El sistema carga el diagrama de clases de Singleton mostrándolo en pantalla, del mismo modo carga los botones correspondientes a sus clases a su derecha, en este caso solo un botón correspondiente a la única clase, Singleton. 7. Luis hace clic en el botón Singleton. 8. El sistema carga en el panel de la derecha en las casillas Nombre, Atributos y Métodos, los datos por defecto para la clase Singleton. 9. Luis modifica estas tres casillas, asignando “UsoBaseDeDatos” en la casilla Nombre, “INSTANCIA” y “conexión” en la casilla Atributos, y “UsoBaseDeDatos()”, “getInstancia()”, “insert()”, “select()”, “showTables()” y “cerrarConexion()” en la casilla Métodos. 10. Luis hace clic en el botón Guardar. 11. El sistema actualiza los datos modificados por Luis en el diagrama de clases, mostrando el nuevo nombre de la clase y sus nuevos atributos y métodos. 12. Luis hace clic en el botón Generar código. 13. El sistema envía al servidor los datos del diagrama, el servidor procesa estos datos, genera el código fuente correspondiente y lo envía de vuelta al cliente en un archivo comprimido, y el navegador muestra en pantalla la descarga a realizarse, pidiendo confirmación. 14. Luis confirma la descarga y el archivo comprimido se

descarga en su computador.
15. Luis extrae el contenido del archivo comprimido, obteniendo las clases correspondientes al diagrama que manipuló, en este caso una sola clase (la clase Singleton).
16. Luis hace uso de este código en su programa, implementando los métodos definidos, entre otras cosas.
17. Luis vuelve al navegador y cierra la página.

- Caso de uso:

Tabla 2. Ejemplo de caso de uso

Caso de uso:	
Nombre del caso de uso:	GenerarCódigo
Actor participante:	Iniciado por Usuario
Condición inicial:	1. El usuario carga la página en la que está albergado el generador de código.
Flujo de eventos:	2. El usuario selecciona el patrón de diseño que desea trabajar.
	3. El usuario puede seleccionar uno de los elementos que conforman el diagrama de clases del patrón.
	4. La aplicación resalta el elemento seleccionado y carga sus características en un formulario.
	5. El usuario puede modificar las características del elemento seleccionado.
	6. A terminar la modificación del elemento, el usuario guarda los cambios por medio de un botón.
	7. Después de modificar los elementos que el usuario necesite modificar, puede dar clic en el botón generar código y la aplicación inicia la tarea de generación.
	8. La aplicación retorna en un archivo comprimido el código

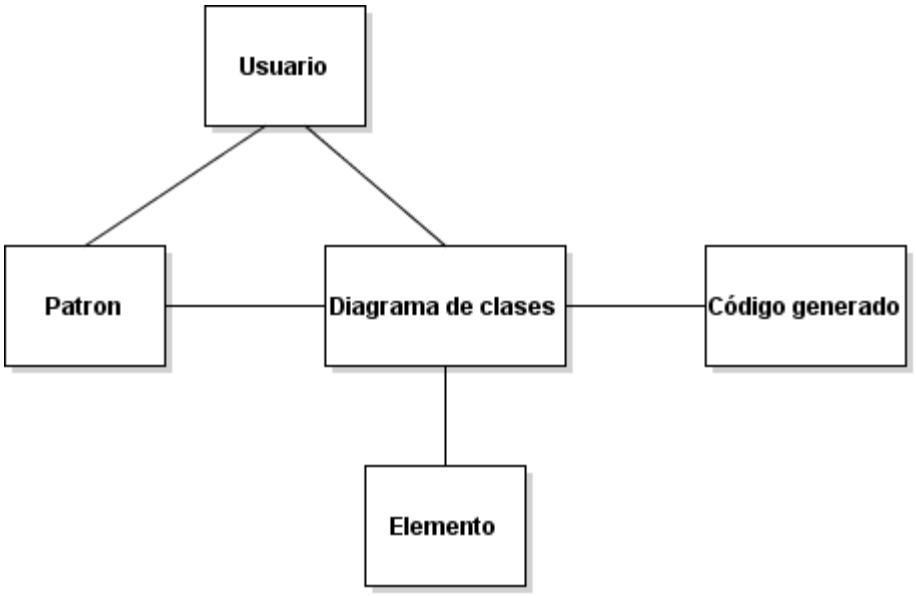
	generado.
Condición de salida	9. El usuario guarda el archivo que contiene el código.

- Modelo de objetos:

  Objetos de análisis:
  - Usuario: Desarrollador que requiere implementar un patrón en algún software en el que está trabajando.
  - Patrón: Solución a un problema de diseño identificado. Puede estar formado por uno o varios elementos.
  - Diagrama de clases: Representación en UML de la solución que plantea el patrón.
  - Elemento: Clase o interfaz que hace parte del diagrama de clases del patrón.
  - Código generado: Archivos con el código fuente Java de los elementos del diagrama.

  En base a este modelo de objetos, se creó el primer diagrama de objetos del software. Este diagrama de objetos fue sometido a mejora en la etapa de diseño, donde se analizó a mayor profundidad cada uno de sus elementos.

Figura 23. Diagrama de objetos producto de la obtención de requerimientos.



7.2 PLANEACIÓN

En la etapa de planeación diseñamos documentos para controlar y llevar registro de las tareas que era necesario hacer a lo largo del desarrollo del generador. Los documentos ayudaron a prevenir que las tareas que dependían de otras que aún no estaban terminadas, causasen retrasos; lo cual es uno de los problemas comunes del modelo de desarrollo en cascada. También se seleccionaron herramientas para dar soporte al versionado y se decidió una forma de manejar las pruebas. Finalmente se hizo un análisis de los posibles riesgos que podrían afectar la planeación durante la construcción del generador.

7.2.1 Documentos para el control y registro de tareas. Para llevar control de las tareas que debían ser desarrolladas, las que estaban siendo desarrolladas y las que estaban terminadas, de forma ordenada y eficiente, se planteó el siguiente formato:

- Tareas en espera:

  En esta parte del documento se almacenaron las tareas que se definieron en la etapa de diseño, especificando: el modulo al que pertenecen (aunque también podían ser tareas de rediseño, investigación o aprendizaje), el tiempo estimado que debería tomar la tarea, la prioridad y un espacio para notas que se consideraran pertinentes. La prioridad indica si la tarea depende de otras o no, entre menor sea el número mayor es su prioridad. Las tareas de prioridad dos en adelante, dependen de otras tareas y estas se pueden encontrar en las notas.

Figura 24. Tareas en espera

TAREAS EN ESPERA

Nombre / descripción	Módulo/tipo	Tiempo estimado	Prioridad	Notas
JS - Terminar de estudiar artículo "JavaScript Prototype in Plain Language"	Aprendizaje	1h30'	1	Terminar artículo
Estudiar JSP 1: Definir temas necesarios para el desarrollo	Aprendizaje	~3h	1	

- Tareas en desarrollo:

  Comparte los mismos campos que la parte anterior del documento, pero incluye la fecha y/o hora de entrega y el encargado de realizar dicha tarea.

Figura 25. Tareas en ejecución

TAREAS EN EJECUCIÓN

Nombre / descripción	Módulo/tipo	Tiempo estimado	Prioridad	Notas	Fecha y hora límite	Responsable
JS - Terminar de estudiar artículo "JavaScript Prototype in Plain Language"	Aprendizaje	1h30'	1	Terminar artículo	26/03/2016	**Camilo**
Estudiar JSP 1: Definir temas necesarios para el desarrollo	Aprendizaje	~3h	1		26/03/2016	**Manuel**

- Tareas terminadas:

  Finalmente, las tareas que ya habían sido desarrolladas se almacenaban en una nueva tabla, en la que se agregaron dos campos para registrar el tiempo que tomó la tarea y la fecha en la que se completó. El sistema de colores (rojo para los tiempos y fechas que no se cumplieron y verde para los que sí) facilita evaluar el cumplimiento en la ejecución de las tareas por parte de los miembros.

Figura 26. Tareas terminadas.

Nombre / descripción	Módulo/tipo	Tiempo estimado	Prioridad	Notas	Fecha y hora límite	Responsable	Tiempo gastado	Fecha y hora entregado
JS - Terminar de estudiar artículo "JavaScript Prototype in Plain Language"	Aprendizaje	1h30'	1	Terminar artículo	26/03/2016	**Camilo**	2h15'	26/03/2016
Estudiar JSP 1: Definir temas necesarios para el desarrollo	Aprendizaje	~3h	1		26/03/2016	**Manuel**	3h30'	27/03/2016

Para que fuera fácil de utilizar por parte de los desarrolladores, el archivo se ubicó en Google Drive[6]. De esta forma el archivo podía ser visto y modificado por todos sin que se generaran conflictos o fuera necesario sincronizar los archivos de cada miembro para ver el estado actual del proyecto.

[6] servicio de alojamiento de archivos creado por Google.

7.2.2 Herramienta de versionado. Los criterios para elegir una herramienta de versionado fueron: que los desarrolladores estuviéramos familiarizados con ésta, que dicha herramienta facilitara unificar, guardar y revertir los cambios hechos por cada miembro del proyecto, y que fuera accesible sin costo alguno. Por estos motivos decidimos hacer uso de Git, ya que es una herramienta open source, varias empresas ofrecen almacenar sus proyectos de forma privada y gratuita, posee un sistema distribuido, es capaz de sincronizar modificaciones hechas por diferentes usuarios sobre un mismo archivo [26], y por último los desarrolladores la habíamos utilizado anteriormente en otros proyectos.

7.2.3 Manejo de las pruebas durante el desarrollo. Con el objetivo de facilitar el manejo de las pruebas y no condensarlas en un solo punto del proceso de desarrollo, se decidió hacer uso de pruebas unitarias soportadas por algún framework que simplificara su implementación, ejecución y mantenimiento. Las pruebas unitarias permitirían verificar el correcto funcionamiento de un elemento del software luego de terminada su implementación. Si posteriormente se realizaban cambios en el elemento, la ejecución de las pruebas permitiría verificar de manera fácil si se habían introducido errores.

7.2.4 Riesgos. Para el manejo de los riesgos se decidió seguir el proceso planteado por Sommerville [27]:

- Identificación de los riesgos:

  Los riesgos identificados, en orden de importancia, fueron los siguientes:

  1. El riesgo de que el desconocimiento de algunas tecnologías necesarias para el desarrollo, demandara una gran cantidad de tiempo para su aprendizaje y pudiera causar retrasos en el proyecto.

  2. El riesgo de no contar con los recursos necesarios para la construcción y despliegue del sistema; recursos como hosting y dominio para el sitio, una herramienta para pruebas unitarias y un repositorio privado para hospedar el sitio y manejar el versionado, entre otros.

3. El riesgo de pasar por alto tareas importantes durante la planeación y el modelado de la aplicación, que luego causaran atrasos durante la etapa de construcción.

- Planeación de riesgo:

  Con el objetivo de evitar que los riesgos anteriormente listados se convirtieran en problemas que pudieran impedir cumplir con los objetivos del desarrollo del generador, se plantearon algunas acciones a llevar a cabo:

  - En caso de que fuera necesario que los desarrolladores aprendieran algunas de las tecnologías que se iban a utilizar, este aprendizaje se incluiría dentro de las tareas a realizar y se le haría el debido seguimiento dentro del documento correspondiente. Esto permitiría mantener un control constante sobre el tiempo que sería necesario dedicar a estudiar alguna herramienta o lenguaje.

  - Para prevenir que los recursos se convirtieran en un problema se decidió iniciar la búsqueda de opciones para obtener los recursos que fuesen necesarios para la construcción y despliegue del generador de código. Luego de que se definieran las opciones, se seleccionarían las que convinieran más al proyecto.

  - Para lidiar con el riesgo de pasar tareas importantes por alto en la etapa de modelado sería necesario realizar esta etapa con detenimiento, procurando abarcar todos los aspectos, teniendo en cuenta que cualquier omisión en esta etapa podría repercutir en problemas en las etapas posteriores.

- Monitoreo del riesgo:

  Para los riesgos anteriormente listados, se incluyó dentro de la planeación del riesgo una forma de monitorizarlos. Por otra parte, con respecto a la aparición de nuevos riesgos, se debería realizar un análisis cada semana para determinar si existen nuevos riesgos que necesitaran ser incluirlos en el listado y decidir un plan para manejarlos según su prioridad.

7.3 MODELADO

En esta etapa se retomaron los diagramas y casos de uso de la etapa de comunicación y a partir de estos se inicia el análisis y diseño.

7.3.1 Identificación de los objetivos de diseño. A continuación, se enuncia los objetivos principales que debe cumplir el diseño del generador de código:

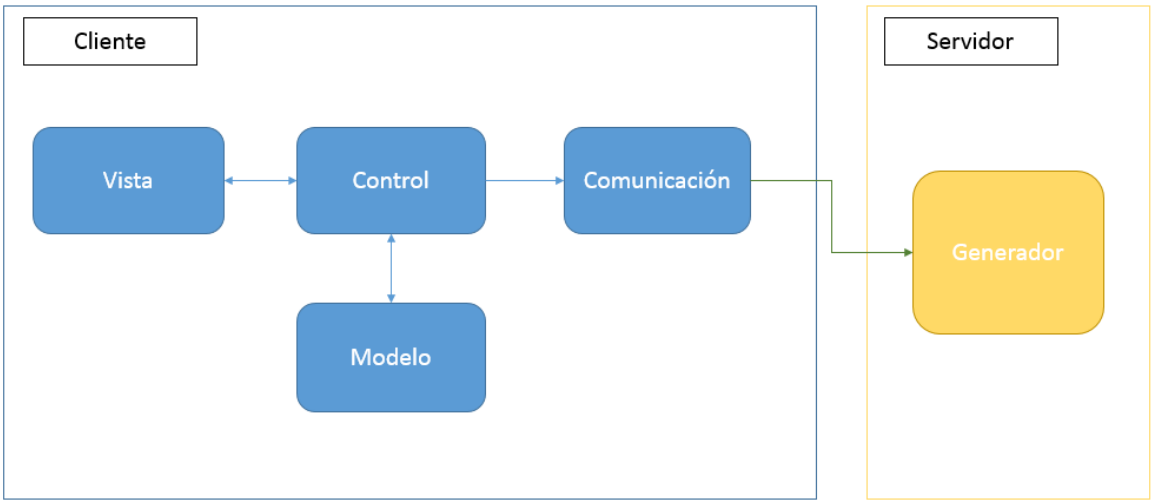
- La interfaz debe ser intuitiva y fácil de utilizar.
- El software debe responder en tiempos aceptables a las acciones del usuario.
- El software debe facilitar los cambios y extensiones en su funcionalidad que puedan ser necesarios en un futuro.
- El software debe ser capaz de ser utilizado en la mayoría de los navegadores actuales sin inconvenientes.
- El software debe ser robusto y poder reaccionar adecuadamente a cualquier acción del usuario.
- Cualquier falla que se presente debe de ser detectada por el software y notificar de lo sucedido.
- El software debe ser desarrollado teniendo en cuenta las medidas de seguridad necesarias para no comprometer la seguridad del servidor.
- El código debe estar comentariado adecuadamente y ceñirse a los estándares, tanto de Java como propios, para nombrar variables, clases y métodos.
- Los casos de uso deben verse fácilmente reflejados en el código de la aplicación.
- Al usuario debe tomarle menos tiempo generar el código del patrón que crear y escribirlo él mismo.

7.3.2 Subsistemas del software. Según el diagrama de objetos obtenido al final de la etapa de comunicación pudimos definir los siguientes subsistemas en el generador de código:

- Modelo
- Vista
- Controlador
- Generador
- Comunicación

Todo el sistema se ejecuta en el cliente, a excepción de Generador que se ejecuta en el servidor. Comunicación se encarga de comunicar a Control con el generador. Control se encarga de la comunicación entre todos los componentes. A continuación, la arquitectura del sistema:

Figura 27. Arquitectura del sistema



7.3.3 Diagramas de clases. El siguiente paso fue agregar las clases que eran necesarias dentro de cada subsistema. En la siguiente parte de la etapa de modelado fue necesario realizar un análisis de los casos de uso, a partir del cual, fue creado el siguiente diagrama de secuencia con las acciones que el usuario podía realizar en el generador de código:

Figura 28. Diagrama de secuencia obtenido a partir de los casos de uso – parte 1

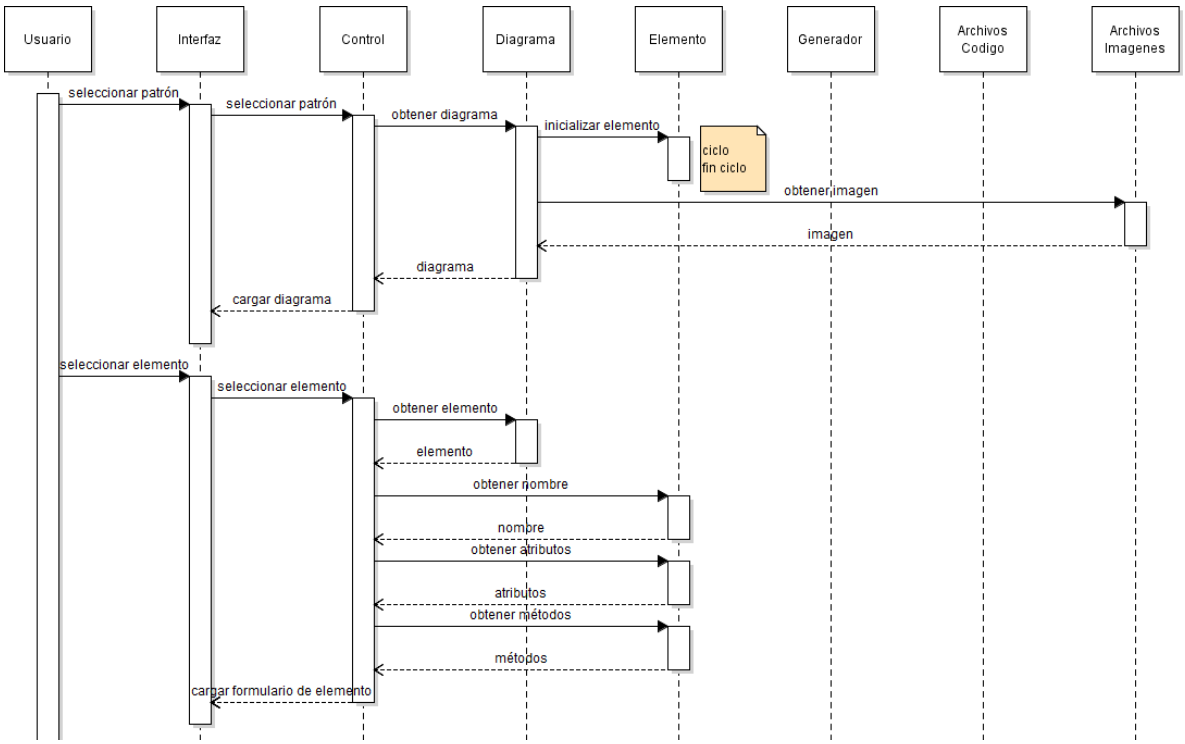


Figura 29. Diagrama de secuencia obtenido a partir de los casos de uso – parte 2

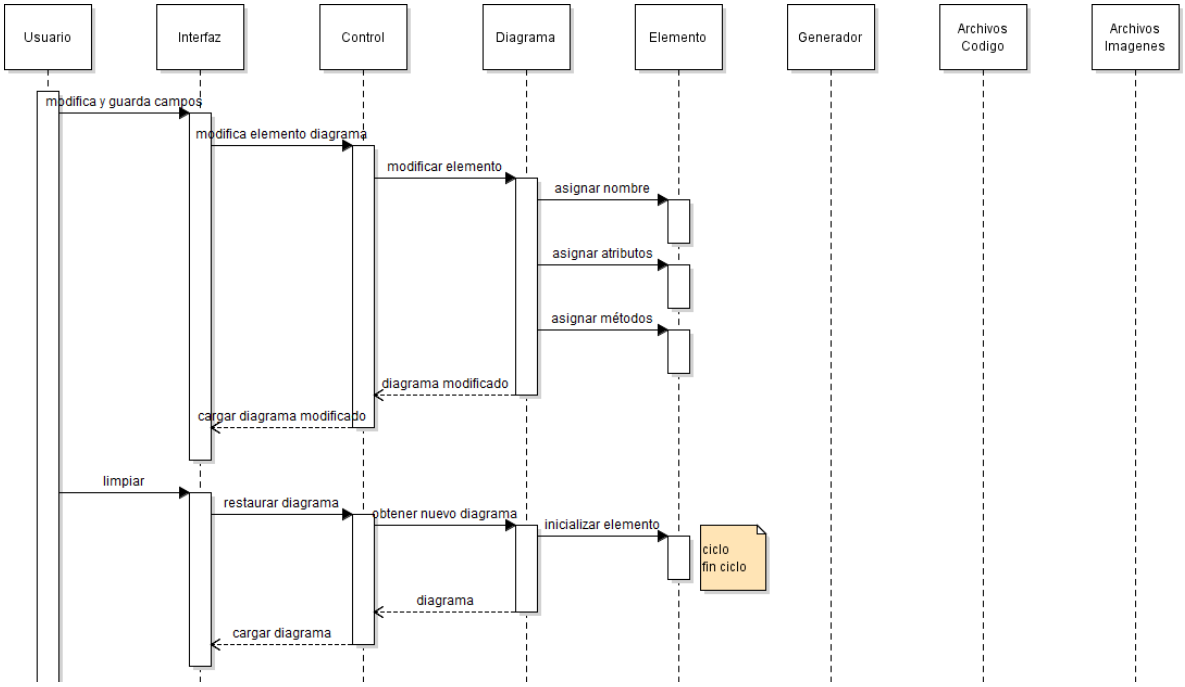
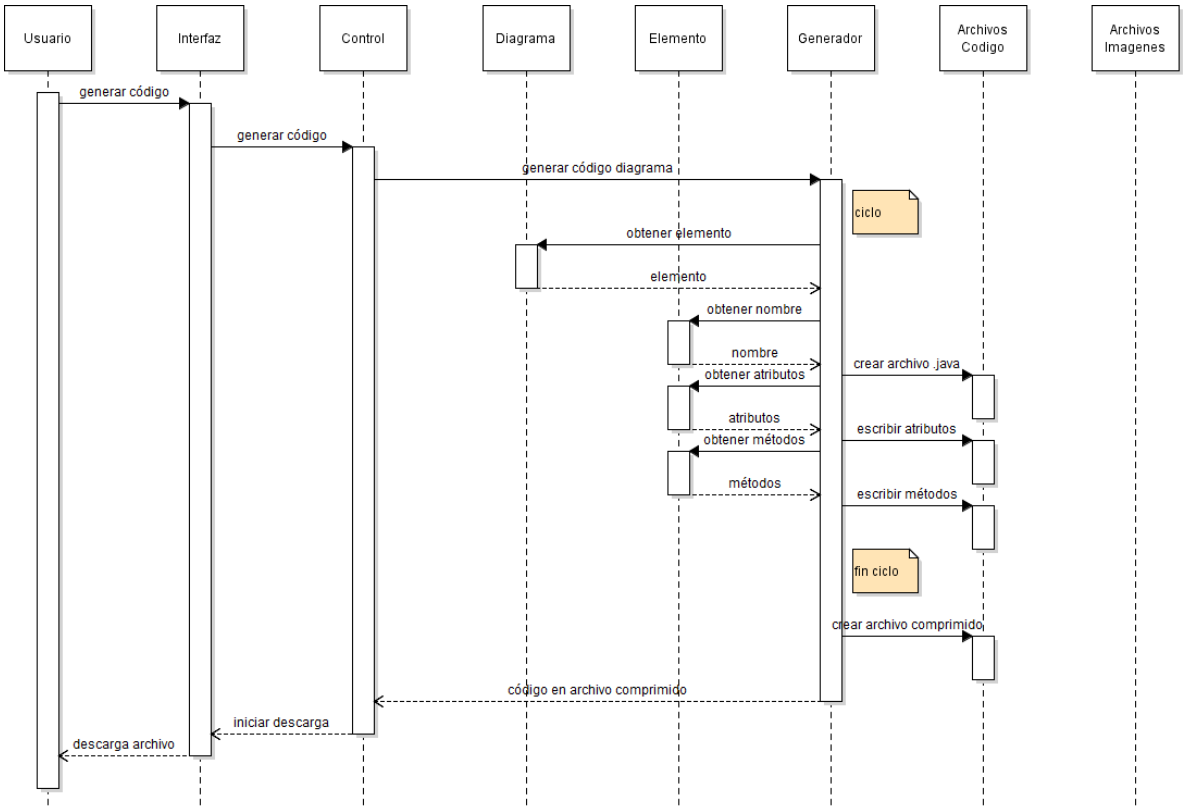
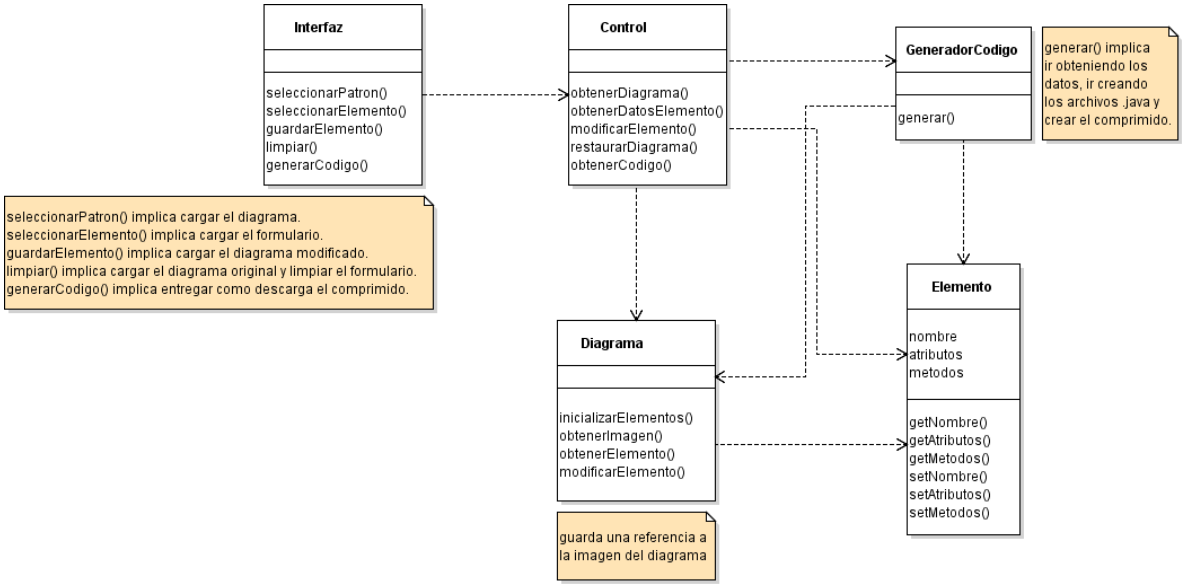


Figura 30. Diagrama de secuencia obtenido a partir de los casos de uso – parte 3



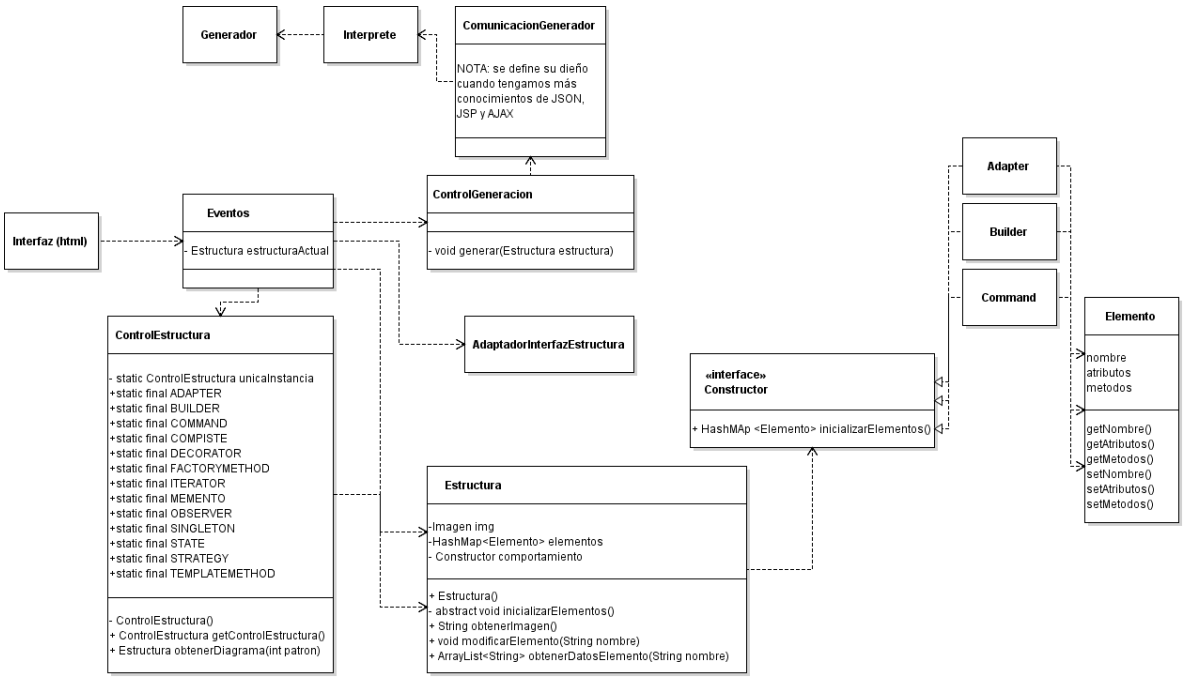
A partir del diagrama de secuencia fue posible identificar algunos objetos y ubicarlos dentro de los componentes establecidos en la arquitectura del sistema. Este proceso dio como resultado el primer diagrama de clases del generador de código:

Figura 31. Primer diagrama de clases



Aplicando los principios SOLID y patrones de diseño con el objetivo de reducir el acoplamiento y la complejidad en el anterior diagrama se obtiene una versión mejorada (ver Figura 32). En esta nueva versión, aparecen clases que no pertenecen al dominio de la aplicación y otras clases que son subclases de las existentes en el diagrama original. Además, algunas clases que realizaban demasiadas labores fueron divididas en dos o más clases que solo tienen una labor. Todo esto, permitió crear un diagrama de clases más completo y claro para iniciar el desarrollo del generador de código, aun así; los módulos de comunicación y generación no pudieron ser diseñados a un mayor detalle debido a falta de conocimiento sobre la forma en la que el generador debía ser implementado y las dificultades que supondría la comunicación. Se determinó que más adelante cuando se tuviera claridad, estos modulo serian diseñados con mayor profundidad.

Figura 32. Diagrama de clases mejorado



7.3.4 Prototipo no funcional. Para mejorar la concepción que se tenía del generador durante la etapa de diseño también se decidió que era necesaria la creación de un prototipo no funcional de la interfaz gráfica del generador. Para su desarrollo se hizo uso del software Axure, dando como resultado el siguiente prototipo:

Figura 33. Prototipo no funcional de la interfaz del generador

Basados en este prototipo se desarrolló la interfaz gráfica de usuario en la etapa de construcción. Este prototipo además ayudó a detectar otras tareas que fue necesario realizar durante la etapa de construcción, como la implementación del botón limpiar para restablecer la interfaz a su estado original y la creación de forma dinámica de los botones para cada uno de los elementos del diagrama.

7.4 CONSTRUCCIÓN

Etapa del desarrollo en la cual se llevó acabo la codificación, además de algunas tareas de rediseño que estaban pendientes de la etapa anterior y otras que se consideraron necesarias.

7.4.1 Tecnologías y lenguajes de programación utilizados. Para llevar a cabo la codificación del generador de código se seleccionaron los lenguajes de programación que tentativamente se enunciaron en el plan de proyecto. Para crear

la parte de la aplicación que funcionaria en el lado del cliente se hizo uso de JavaScript, HTML5 y CSS3, los cuales son lenguajes populares actualmente en el desarrollo de sitios web[7]. Por otra parte, para el generador en el lado del servidor se decidió hacer uso de JSP y servlets, por la gran librería que posee java y el hecho de que los desarrolladores tenemos bastante experiencia desarrollando con este lenguaje.

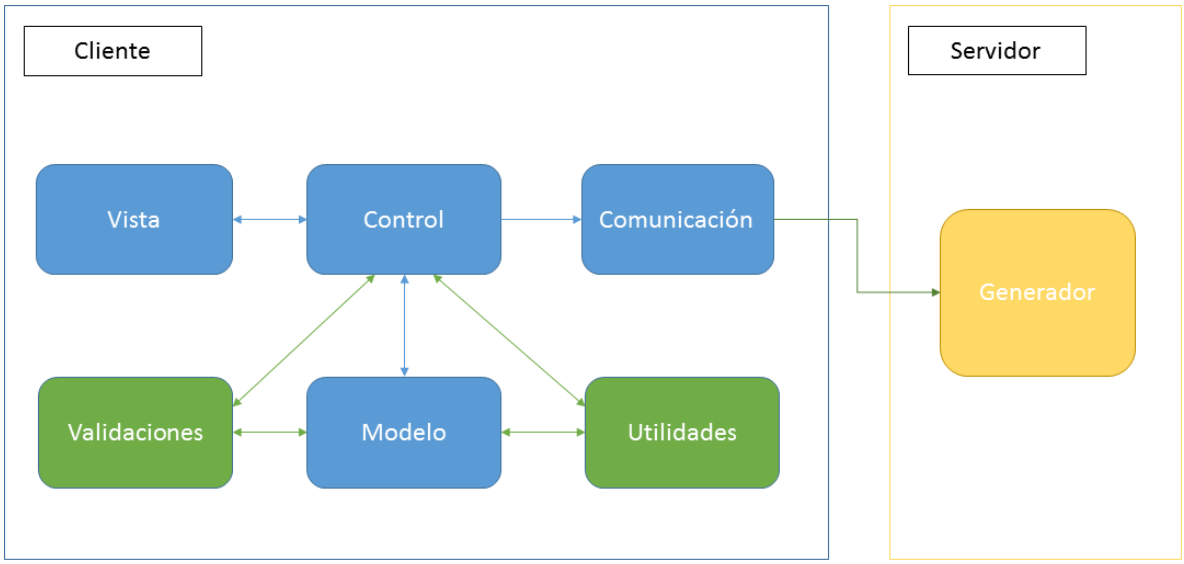
Para llevar a cabo el versionado ya se había seleccionado Git como herramienta, solo quedaba seleccionar el servicio que se utilizaría para almacenar el repositorio. En ese momento se manejaron dos opciones GitHub y GitLab, y al final se optó por este último principalmente debido a que ofrecía repositorios privados de forma gratuita y GitHub no.

Para el desarrollo de las pruebas unitarias, luego de una investigación de las diferentes opciones que había en el mercado, se seleccionó Jasmine, framework el cual está destinado a realizar pruebas unitarias en JavaScript. Se seleccionó Jasmine debido a la corta curva de aprendizaje necesaria para poder empezar a crear y utilizar pruebas, además de una interfaz agradable para para ejecutar y ver los resultados de las pruebas [14].

7.4.2 Refactorizaciones realizadas al diseño del generador de código. La codificación del generador se desarrolló en su mayor parte de acuerdo al plan creado en la etapa anterior, exceptuando la necesidad de agregar dos nuevos módulos a la arquitectura del software.

[7] Lenguajes de programación más utilizados en github: https://github.com/blog/2047-language-trends-on-github

Figura 34. Arquitectura final del sistema



Fue necesario separar en un módulo a parte las validaciones de sintaxis del código que ingresa el usuario y las validaciones de la estructura del patrón. La validación de sintaxis contendría los prototipos encargados de validar la sintaxis de los cambios que hiciera el usuario sobre el código por defecto del patrón. Las normas que se validaban y la forma en la que se validaron son las siguientes:

Normas y formatos para la declaración de métodos y atributos:

- Normas:

    - Los caracteres especiales dentro de los formatos estarán rodeados por comillas simples (‘’). Ejemplos: espacio ‘espacio’ o coma ‘,’.

    - Los caracteres que sean difíciles de identificar serán reemplazados por su nombre.

- Formatos:

    Los formatos son los siguientes para métodos y atributos:

- Atributos: <modificador de privacidad>’espacio’<tipo de dato>’espacio’<Identificador>’;’.

- Constantes: <modificador de privacidad>’espacio’static’espacio’final<tipo de dato>’espacio’<Identificador>’’;’.

- Métodos:

    - <modificador de privacidad>’espacio’<tipo de dato de retorno>’espacio’<identificador>()’;’.

    - <modificador de privacidad>’espacio’<modificador>’espacio’<tipo de dato de retorno>’espacio’<identificador>()’;’.

- Si a la instanciación de algún método o clase no se le incluye modificador de acceso el generador tomara por defecto private.

- Identificadores:

    - Los identificadores deben cumplir las normas estipuladas por Sun Microsystems para clases y métodos:

        - Los métodos, atributos y objetos deben iniciar con minúscula.

        - Las clases deben iniciar con mayúscula.

        - Si un identificador está compuesto por varias palabras, a partir de la segunda deben iniciar con mayúscula y no tener espacios entre ellas. Pero validar esto representa un reto que no podemos asumir en este desarrollo.

    - Los identificadores solo pueden contener los caracteres permitidos e iniciar con una letra o con los caracteres ‘_’ y ‘$’.

A partir del segundo carácter se puede utilizar cualquier combinación de caracteres y son válidos: números, letras, caracteres de conexión y caracteres de moneda.

Validaciones de sintaxis:

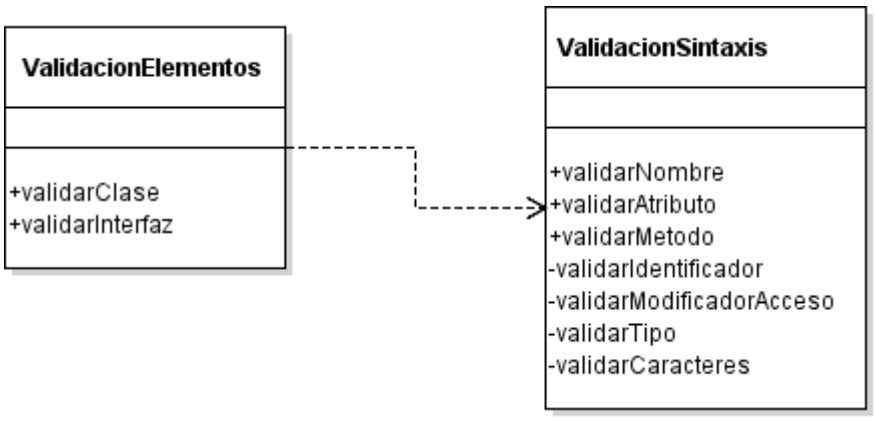
1. Validar el uso de caracteres permitidos.
2. Validar los modificadores de acceso de métodos y atributos.
3. Validar tipo retorno y tipo dato de métodos y atributos.
4. Validar que los identificadores inicien con '_' '$' o letra.

Validaciones de estructura:

1. Validar que los elementos no repitan identificadores.
2. Validar que los elementos cumplan las normas de la estructura.
3. Validaciones interfaces: Validar que los métodos terminen en ');'.
4. Validaciones Clases:
    a. Validar que existan los atributos de la estructura del patrón dentro de los elementos.
    b. Validar que se implementen los métodos de las interfaces.

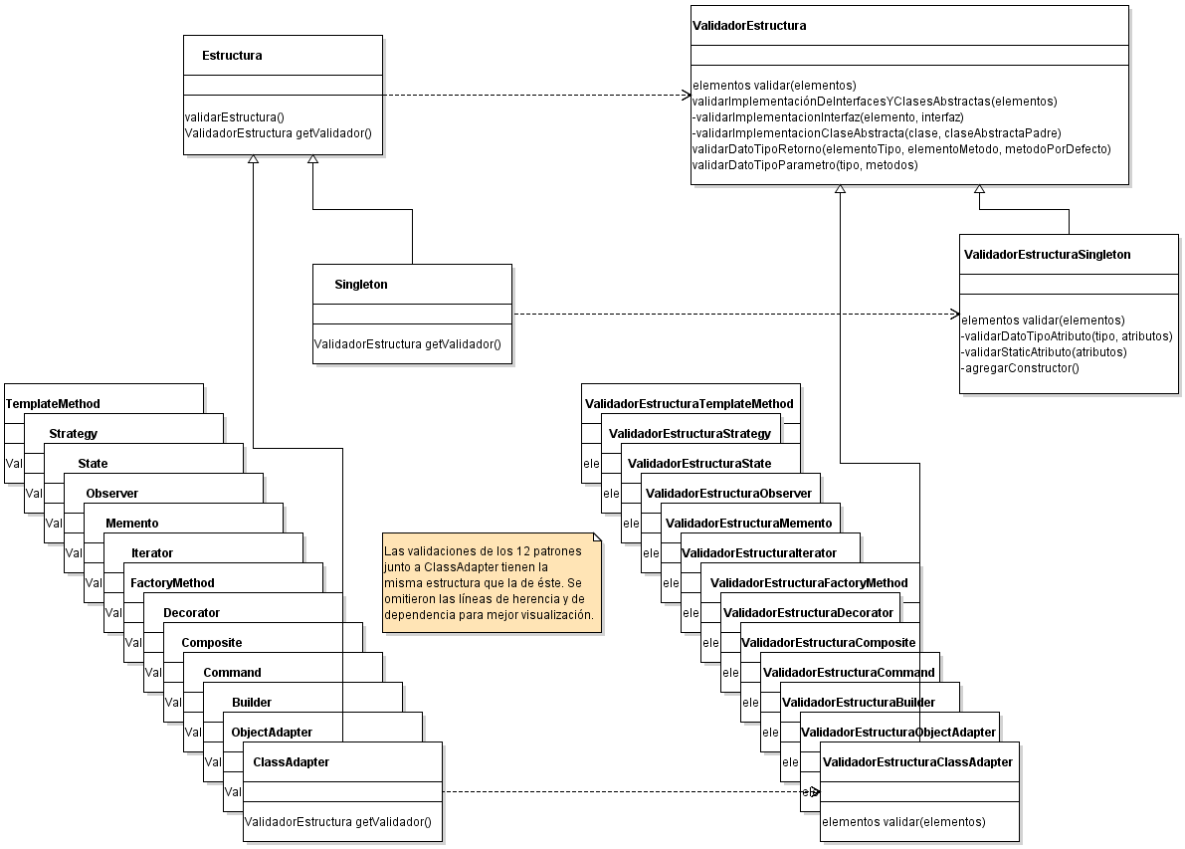
El diagrama del módulo encargado de realizar estas validaciones se diseñó e implemento como se puede ver en el siguiente diagrama de clases.

Figura 35. Diagrama de prototipos del módulo de validaciones de sintaxis



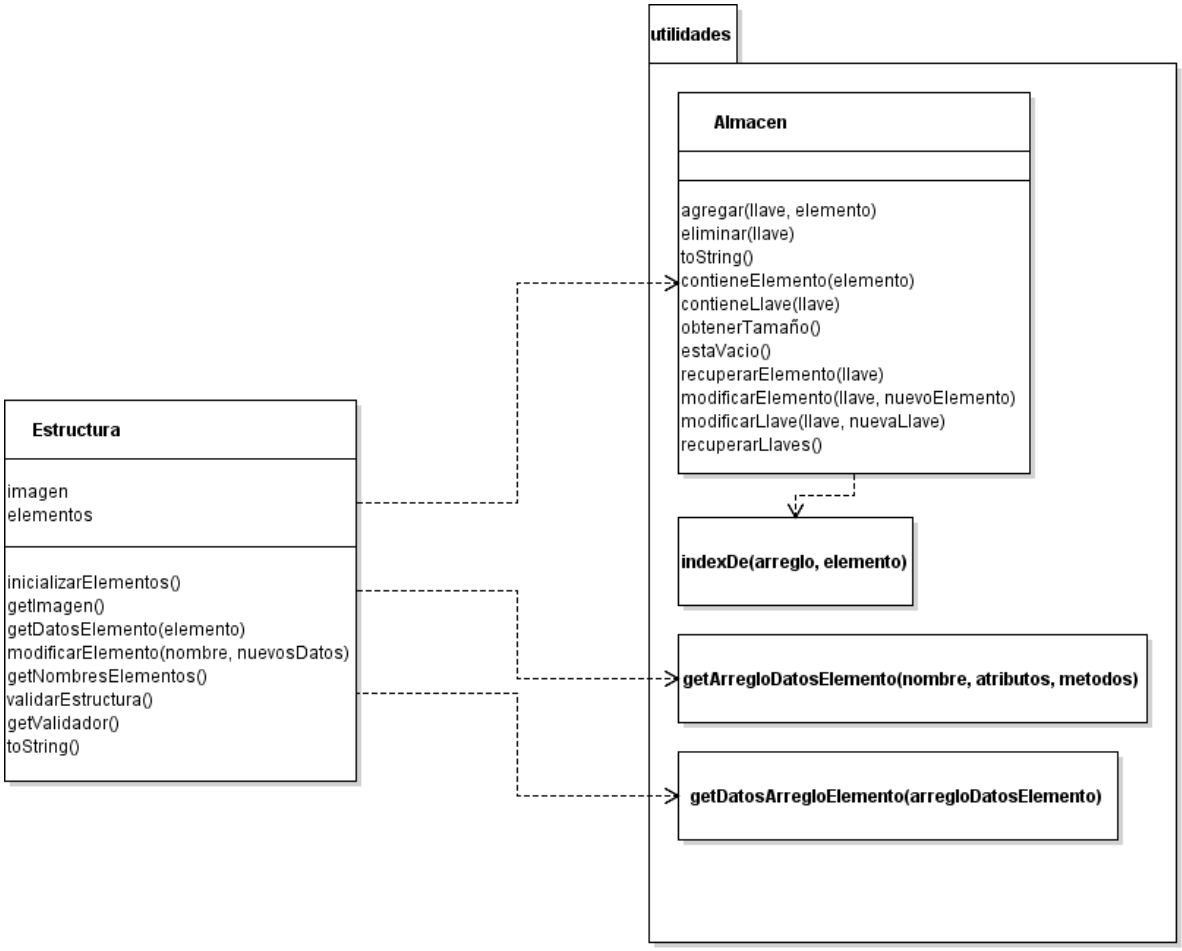
Para la validación de la estructura del patrón, se debía garantizar que se implementaron correctamente las interfaces, las variables y métodos tuvieran el tipo de dato correcto y finalmente que los métodos abstractos se implementen en las subclases. Para cumplir con este trabajo se diseñaron e implementaron los siguientes prototipos.

Figura 36. Diagrama de prototipos del módulo de validaciones de estructura



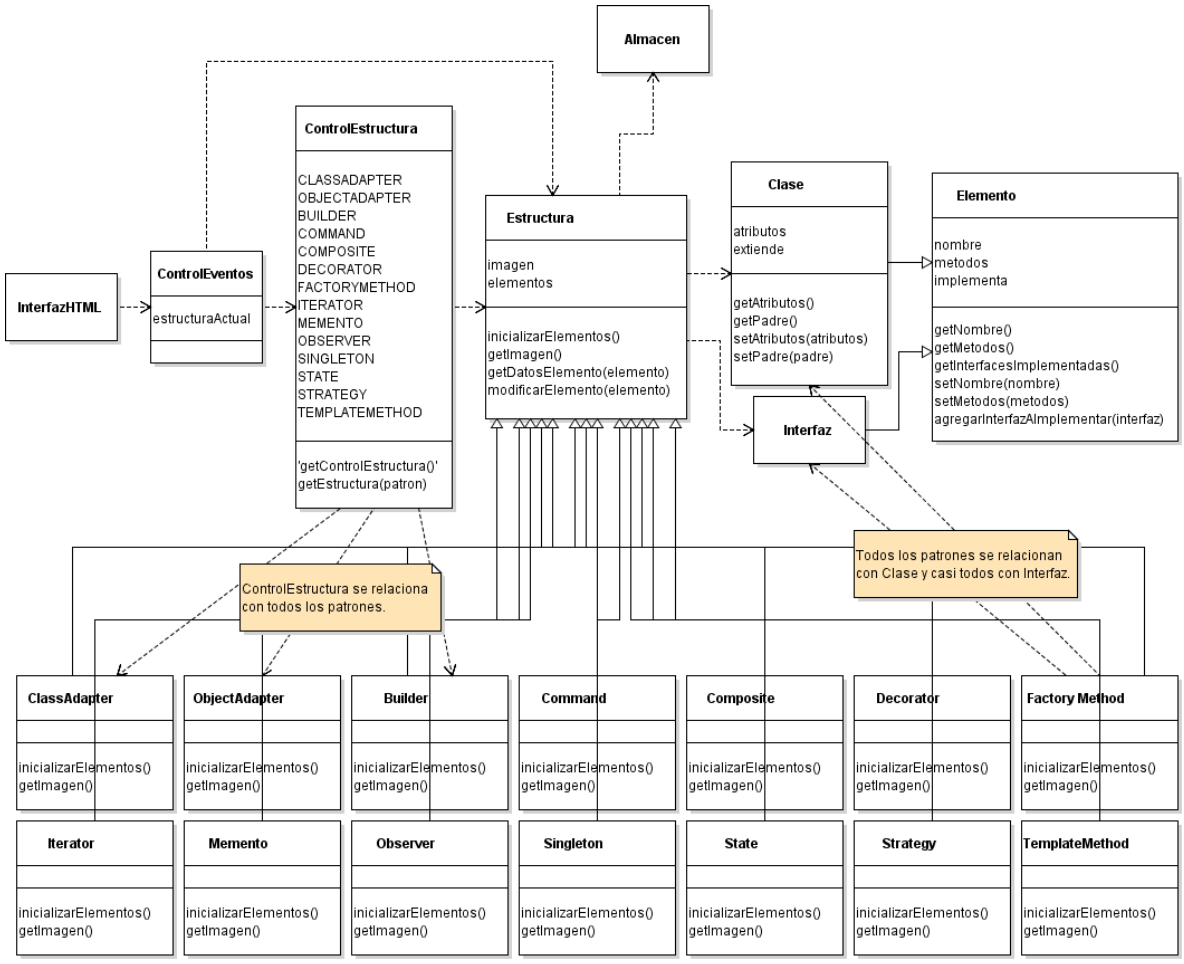
El segundo módulo que se agregó, fue el de utilidades, que contiene prototipos que son necesarios en varias partes de la aplicación, pero no pertenecen a ninguno de los módulos existentes. Por esta razón fueron almacenados y organizados dentro de este módulo.

Figura 37. Diagrama de prototipos del módulo utilidades



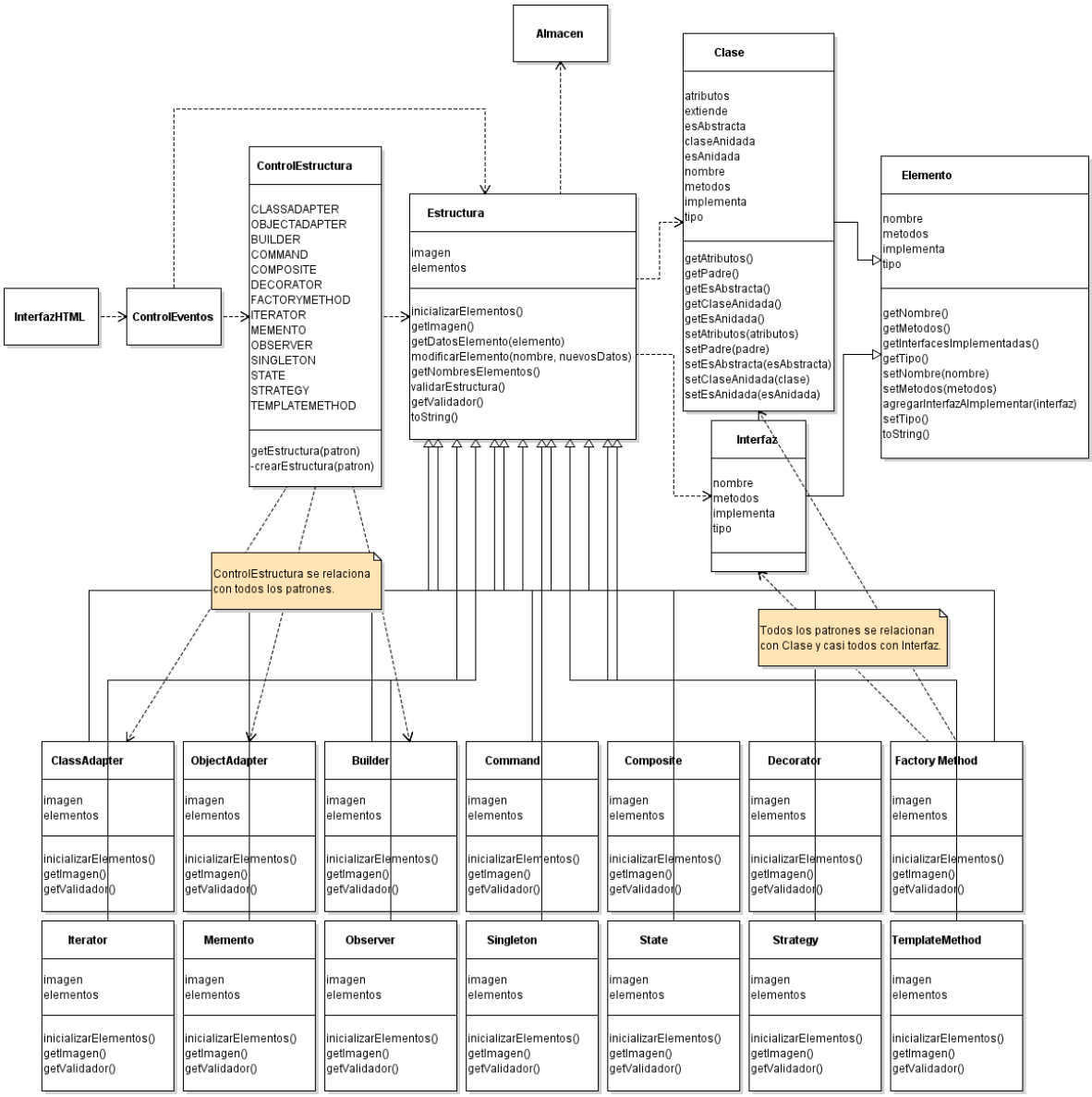
Otros rediseños que se hicieron en la etapa de construcción y que se hicieron evidente al momento de empezar a codificar el generador, fue la necesidad de realizar modificaciones en el diagrama del modelo que se diseñó al final de la etapa de modelado. Elemento no podía servir para representar las clases e interfaces del lenguaje Java a la vez, por esta razón se decidió crear dos prototipos hijos de Elemento, Clase e Interfaz, que contendrían el código específico del tipo de elemento que representan. Elemento ahora contendría el código común a sus dos prototipos hijos.

Figura 38. Diagrama de prototipos del módulo del modelo



Luego se agregaron los métodos necesarios a los nuevos prototipos hijo de Elemento y al prototipo Estructura para comunicarse con el nuevo módulo de validación de estructura. Cada patrón conoce su propio ValidadorEstructura y los llama cuando se le solicita que valide la estructura que actualmente contiene. Además, se hicieron refactorizaciones a algunos métodos dentro de las interfaces de varios prototipos, para mejorar su legibilidad y uso.

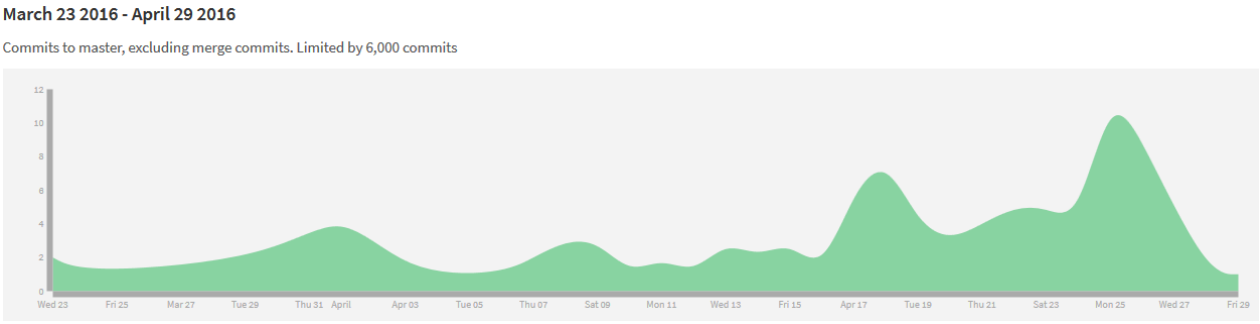
Figura 39. Diagrama de prototipos del módulo modelo con nuevos métodos



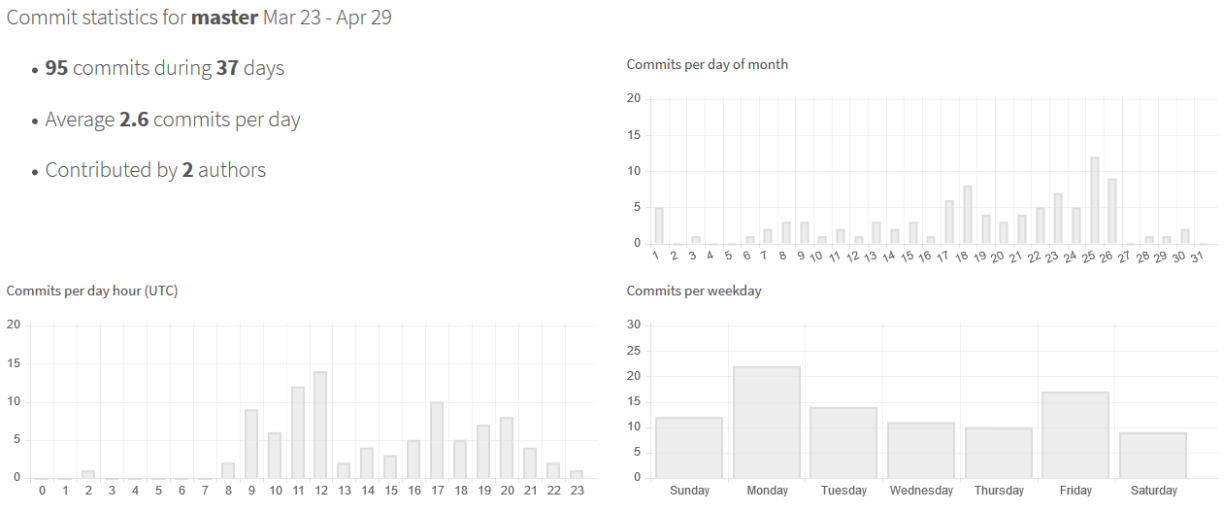
Este último rediseño concluye los cambios que fue necesario hacer durante la construcción del generador al diseño original. La inclusión de los módulos que no se detectaron y diseñaron durante la etapa anterior representó un incremento considerable en el trabajo que al comienzo se esperaba.

7.4.3 Codificación del generador. La codificación se llevó a cabo sin contratiempos, aparte de las refactorizaciones en el diseño que fueron nombradas anteriormente. Del repositorio en GitLab podemos recuperar la información del avance del trabajo a lo largo de tiempo, el número de commits y el porcentaje que cada representa en el total del generador de código. A continuación, se muestran los gráficos tomados de la página de GitLab y de Git.

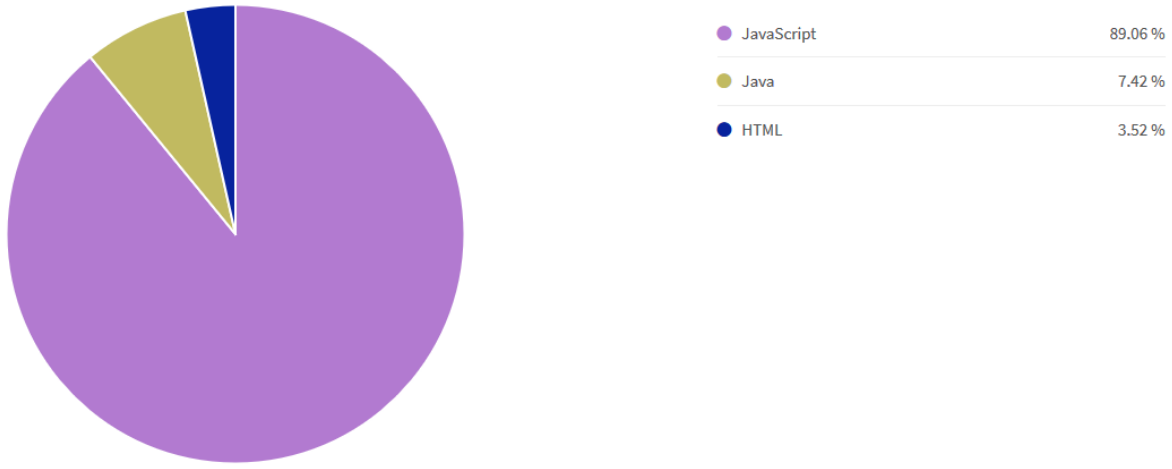
Grafica 1. Cantidad de commits por día a lo largo del desarrollo



Grafica 2. Cantidad de commits por día del mes, día de la semana y hora del día a lo largo del desarrollo

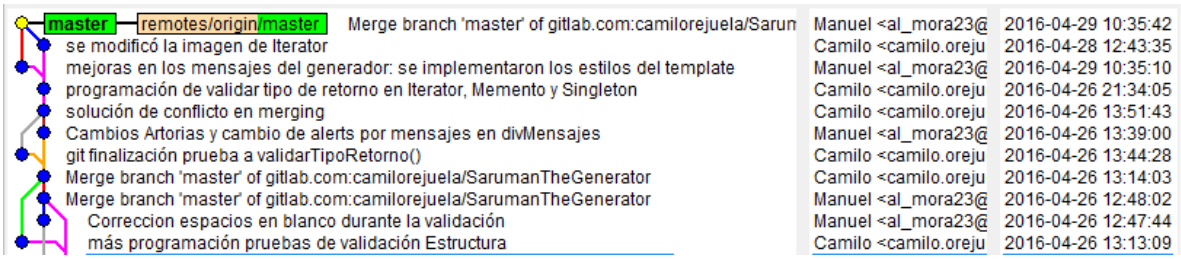


Grafica 3. Porcentaje de cada lenguaje sobre el total del software



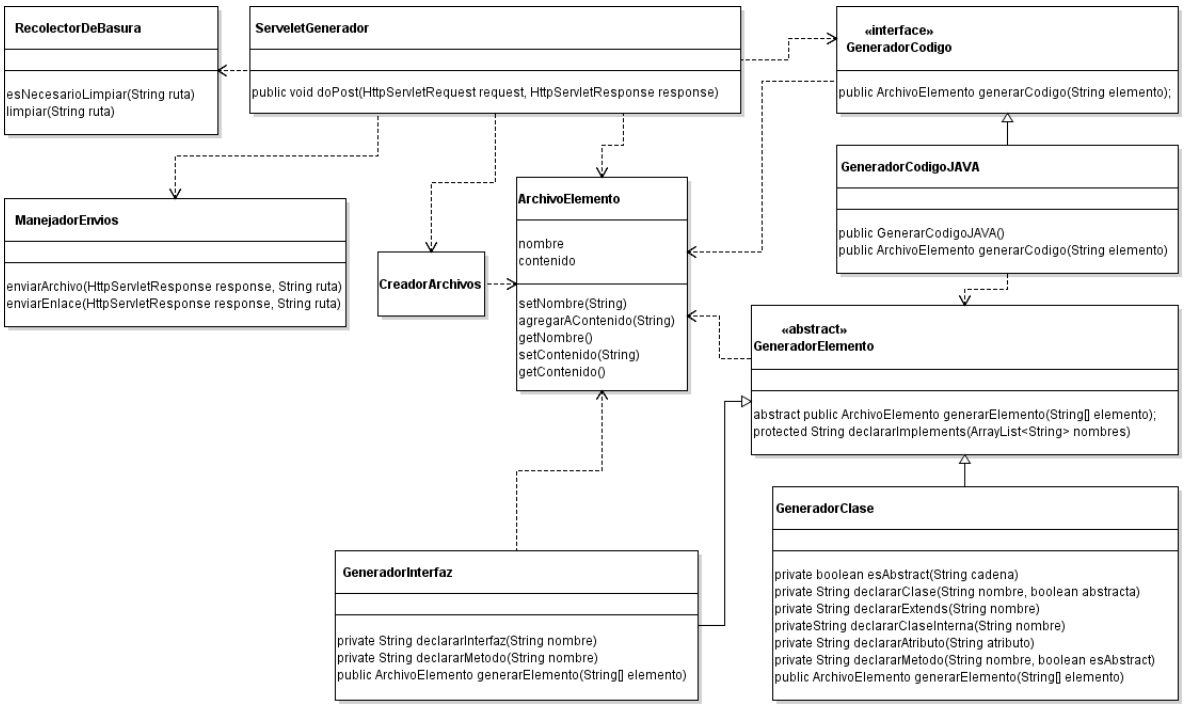
En la siguiente imagen, se muestra una parte de los commits hechos al final del desarrollo, donde se ve en la descripción de cada uno a que tarea pertenece, cual desarrollador lo creo y la fecha y hora en la que se realizó.

Figura 40. Algunos commits almacenados en el repositorio de Git



El diagrama de clases final del servlet encargado de completar el código, crear los archivos, comprimirlos y enviarlos al usuario es el mostrado en la siguiente figura.

Figura 41. Diagrama de clases del generador de código



7.4.4 Creación de pruebas unitarias. Como se decidió al comienzo de la etapa de creación, se hizo uso de Jasmine para la creación y ejecución de las pruebas unitarias necesarias en la aplicación. En la siguiente figura se puede ver las pruebas que se implementadas para el generador de código.

Figura 42. Algunas de las pruebas implementadas y ejecutadas en el framework Jasmine

Las pruebas unitarias demostraron ser de gran utilidad cuando fue necesario identificar y corregir errores en el desarrollo, también fueron un gran apoyo

durante la depuración de errores que se encontraron en las pruebas del funcionamiento del generador de código.

7.5 DESPLIEGUE

Para realizar el despliegue se contó con el apoyo del grupo de investigación GID-CONUSS que nos brindó una máquina virtual y el DSI nos brindó acceso a esta desde internet y un subdominio (http://patdis.uis.edu.co), como se mencionó en la creación del contenido y el sitio web.
Para poner en funcionamiento el generador se instaló Apache Tomcat y configurarlo adecuadamente para poder ejecutar el servlet que contiene la parte del generador del lado del servidor. Luego se subieron a la maquina por medio de SFTP los archivos del generador, se recompilo el servlet y se reinició el server para lo reconociera.

8. PRUEBA PARA MEDIR LA USABILIDAD

La realización de una prueba para medir la usabilidad, sigue una serie de pasos desde su planeación hasta su ejecución y posterior análisis. A continuación el desarrollo de estos pasos.

- Planear el test para medir la usabilidad:

  En conjunto con el director de proyecto se determinó que los criterios que se medirían en las pruebas serían los siguientes:

  - Que el sitio sea fácil de usar/aprender: el usuario debe poder navegar a través de las diferentes secciones de la página, encontrar fácilmente algún contenido que quiera buscar, o alguna sección específica dentro de un contenido; asimismo debe poder usar el generador de código con facilidad, intuyendo su funcionamiento y haciendo uso de él consiguiendo realizar alguna tarea.

  - Que el usuario cometa pocos errores: la forma como está organizado el sitio y el generador de código en específico, deben procurar que el usuario no cometa errores en su uso o cometa pocos, y estos errores no deben representar un problema más allá que detener al usuario por un momento. Un ejemplo de error es que el usuario en el generador de código, al modificar un elemento de un patrón, ingrese un método o un atributo que no sigue el formato esperado.

  - Que usar el sitio sea una experiencia agradable: el usuario debe sentirse cómodo con la organización de la información en el sitio y con cómo luce visualmente.

También se determinó que se realizarían las pruebas con un muestreo no probabilístico por conveniencia [28], y que se utilizaría un cuestionario de 5 preguntas, las cuales calificarían con un puntaje de 1 a 5. Los usuarios tendrían que probar la página y responder el cuestionario en un tiempo de entre 15 y 30 minutos, según consideren.

- Definir objetivos:

El objetivo de las pruebas es detectar problemas y posibles mejoras al sistema, así como verificar qué aspectos del sistema funcionan bien y cumplen con su objetivo.

Esta evaluación al sistema permitirá definir las mejoras que debe tener la siguiente versión.

- Decidir quienes deberían participar:

  Por los conocimientos que necesita tener el estudiante para poder hacer uso adecuado del sistema y basándonos en los contenidos de las asignaturas del plan de estudios de Ingeniería de Sistemas de la UIS [29], definimos que los participantes serán estudiantes de esta carrera que hayan aprobado Ingeniería de Software I.

- Reclutar participantes:

  Se contactaron estudiantes de diferentes niveles que cumplieran con el requisito anteriormente especificado. Estos fueron contactados por medios electrónicos y su participación fue voluntaria. Se les explicó el objetivo de la prueba, así como las características y objetivo del sistema.

- Seleccionar y organizar tareas a evaluar:

  De los criterios que medirá la prueba, el de mayor interés para nosotros es si usar el sitio fue una experiencia agradable y en torno al cual giraran los otros criterios. Por esta razón, las tareas que el usuario debió realizar durante la prueba y las forma de evaluarlas se enfocaron en plasmar como se sintió el usuario durante el uso del sitio web. Teniendo esto en cuenta, nos interesó más definir tareas en las que el usuario tuviera libertad de elegir como desarrollarla en vez de limitarlo una tarea detallada, después de todo no estábamos interesados en el tiempo en el que la desarrollo, sino en cómo se sintió al desarrollarla.

Siguiendo con el enfoque anteriormente descrito, era necesario que: el usuario navegara por el sitio web, leyera por lo menos uno de los contenidos e interactuara con la forma en que estaban organizados y escritos, y finalmente hiciera uso del generador. Con estas tareas cubrimos todas las funcionalidades y características del sitio en las que estábamos interesados que el usuario entrara en contacto y experimentara.

Para conducir la prueba se le entrego a los participantes un documento que especifica las tareas que debían realizar y la forma en la cual debían plasmar sus impresiones al realizarlas (ver Anexo B). El participante luego de desarrollar las tareas debía responder a una serie de sentencias, en las cuales se afirmaba de forma positiva o negativa acerca de alguna de las características del sitio. A continuación, se muestran las sentencias incluidas en el documento:

1. Fue muy fácil navegar a través del sitio y entender la organización del contenido.
2. El contenido parece ser muy confuso y difícil de entender.
3. El generador de código es muy fácil de utilizar.
4. Cometí muchos errores al utilizar el generador de código.
5. Cuando cometí errores, los mensajes del generador de código fueron muy claros en indicármelos.
6. Usar PatDis fue una experiencia muy frustrante.
7. Es muy agradable trabajar con PatDis.
8. Creo que PatDis es una herramienta realmente útil y de calidad.

Las sentencias 1, 2 y 3, evalúan qué tan fácil de usar/aprender es el sistema. Las sentencias 4 y 5 evalúan que el participante cometa pocos errores y pueda ubicarlos fácilmente. Y las 6, 7 y 8 evalúan que el sitio (la experiencia de utilizarlo) sea subjetivamente agradable. Asimismo, las 1 y 2 se refieren únicamente al contenido teórico de la página. Las 3, 4 y 5 se refieren al generador de código. Y las 6, 7 y 8 se refieren al sitio en general, que incluye las dos cosas. Para facilitarle al participante expresar sus opiniones acerca de las sentencias hicimos uso de la escala de Likert [30] como las opciones de respuesta que se podían escoger:

(1) Totalmente en desacuerdo

(2) En desacuerdo

(3) Ni de acuerdo ni en desacuerdo

(4) De acuerdo

(5) Totalmente de acuerdo

Finalmente, se incluyó un espacio para que el participante pudiera expresar sus opiniones acerca del sitio y su experiencia con él.

- Conducir la prueba

  Para la ejecución de la prueba, se envió al participante por internet el formato anteriormente mencionado y se le dio libertad en el tiempo que podía invertir en la prueba y el lugar donde desarrollarla. Luego de desarrollada la prueba, el participante enviaba de regreso el formato completado.

- Tabular y analizar los datos

  Guiándonos por la fuente que tomamos como guía para el estudio de la usabilidad [17] y del hecho de que decidimos hacer uso de un muestreo no probabilístico por conveniencia, obtuvimos una muestra de 12 participantes de una población de 145 estudiantes[8], con una desviación estándar del 46% y una confianza del 80%[9].

  - Resultados:
    Los datos obtenidos a partir de la prueba fueron los siguientes:

[8] Cantidad de estudiantes que se encuentran en los niveles 8, 9 y 10 en el primer semestre del año 2016. Dato proporcionado por el grupo de desarrollo Calumet.

[9] Nielsen en su libro Usability Engineering [13] indica que el 80% de confianza es suficiente para propósitos prácticos de desarrollo.

Tabla 3. Resultados de la prueba de usabilidad

Sentencia		Participantes de la prueba											
		Sujeto 1	Sujeto 2	Sujeto 3	Sujeto 4	Sujeto 5	Sujeto 6	Sujeto 7	Sujeto 8	Sujeto 9	Sujeto 10	Sujeto 11	Sujeto 12
	1.	5	3	5	5	5	4	4	4	4	5	5	5
	2.	1	2	2	2	2	2	3	1	2	1	3	1
	3.	5	4	3	5	4	3	2	3	5	3	5	3
	4.	2	2	4	3	2	3	3	3	2	2	4	3
	5.	4	2	3	3	5	3	1	3	2	3	4	2
	6.	1	1	2	1	1	1	1	2	3	2	2	3
	7.	5	4	3	4	4	4	5	4	3	4	4	3
	8.	5	4	4	5	4	4	5	4	4	4	4	4

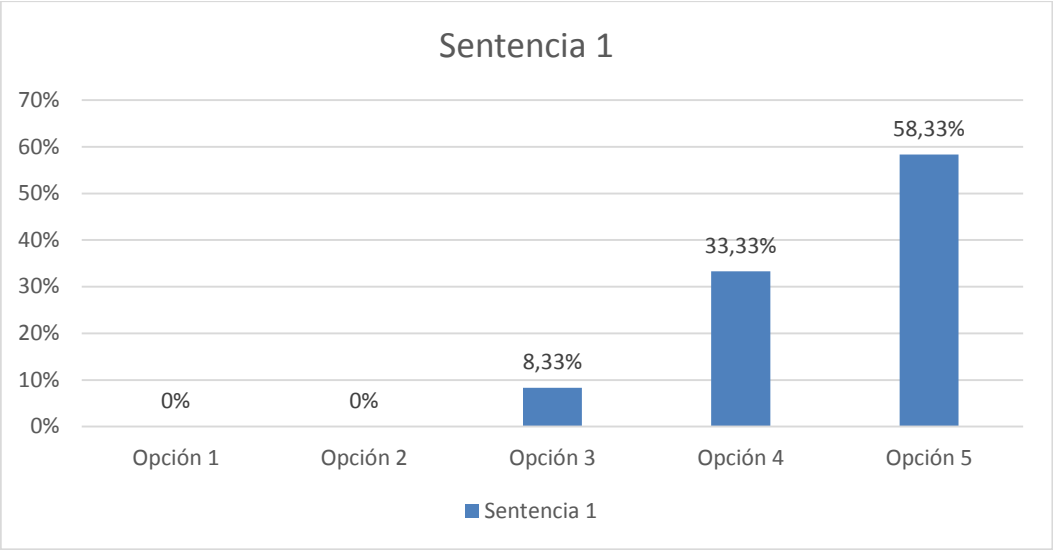
- Análisis de los resultados:

  A partir de los resultados obtenidos de aplicar la prueba a la muestra, se lleva a cabo el análisis de cada una de las preguntas de la prueba.

  - Sentencia 1:

    Fue muy fácil navegar a través del sitio y entender la organización del contenido.

Grafica 4. Resultado para la sentencia 1

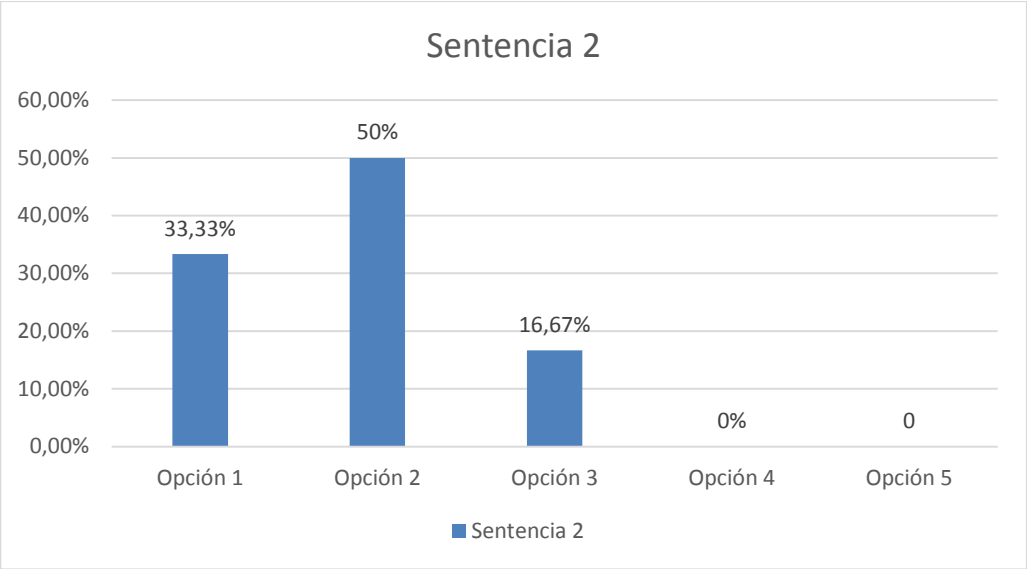


El 58.33% de los participantes están muy de acuerdo con que es fácil navegar a través del sitio y entender la organización del contenido; esto puede deberse a la forma organizada y clara en que se presenta el contenido del sitio.

- Sentencia 2:

  El contenido parece ser muy confuso y difícil de entender.

Grafico 5. Resultado para la sentencia 2

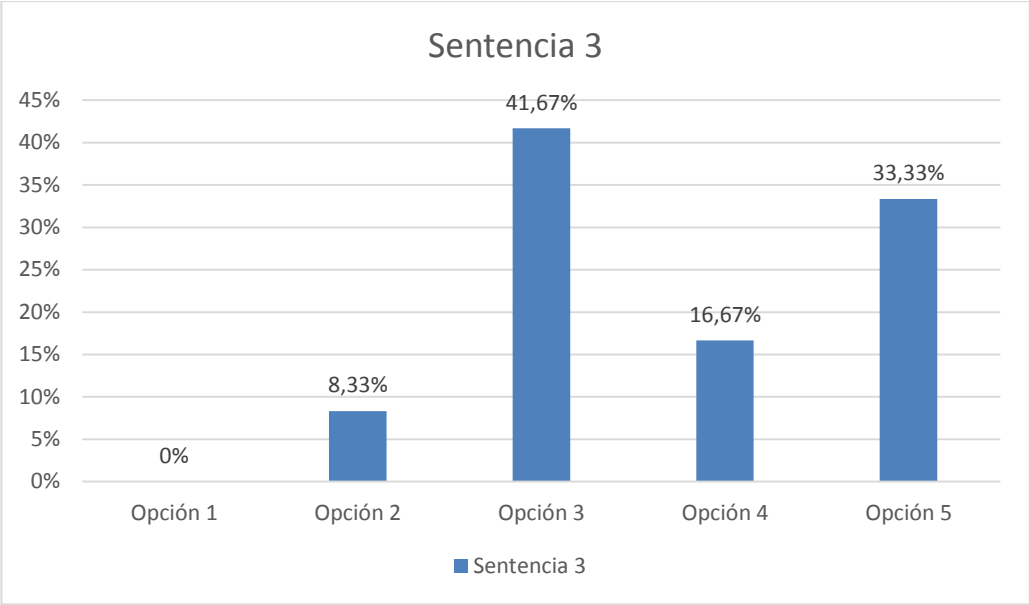


El 50% están en desacuerdo con la afirmación de que el contenido parece ser muy confuso y difícil de entender. El resultado es bueno, sin embargo, el hecho de que la mayoría no esté muy en desacuerdo, puede deberse a los términos técnicos utilizados dentro de los documentos y a que algunos diagramas puedan ser difíciles de entender.

- Sentencia 3:

  El generador de código es muy fácil de utilizar.

Grafico 6. Resultado para la sentencia 3

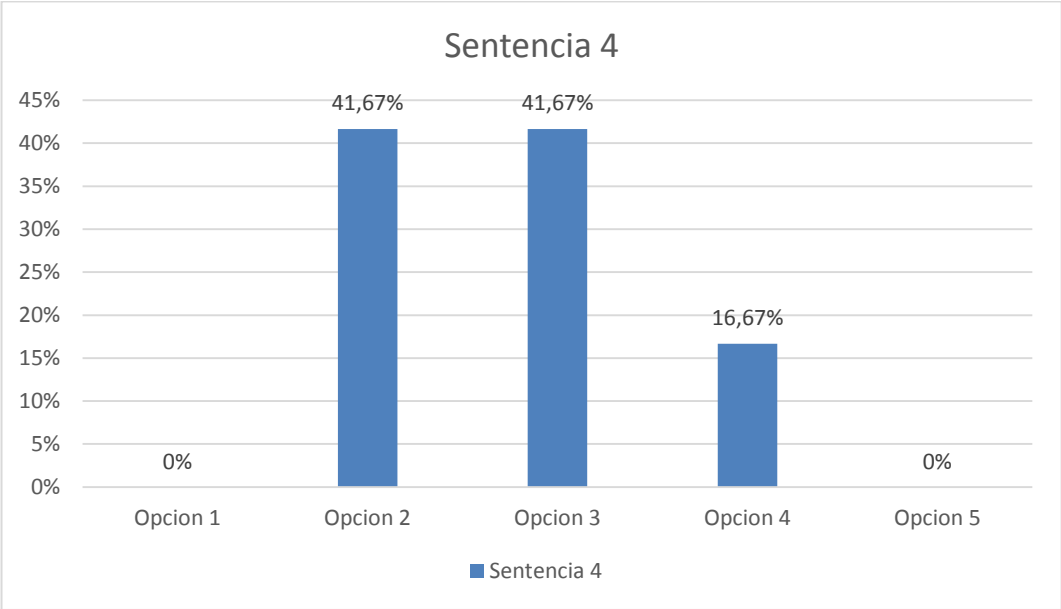


El 41,67% no está ni de acuerdo ni en desacuerdo con la afirmación de que el generador de código es muy fácil de utilizar, aun así, el 50% está de acuerdo o muy de acuerdo con la afirmación. La razón de esto puede estar en que a pesar de que el diseño no es complicado, aun no es lo suficientemente intuitivo para que la mayoría de usuario lo puedan utilizar fácilmente.

- Sentencia 4:

  Cometí muchos errores al utilizar el generador de código.

Grafico 7. Resultado para la sentencia 4

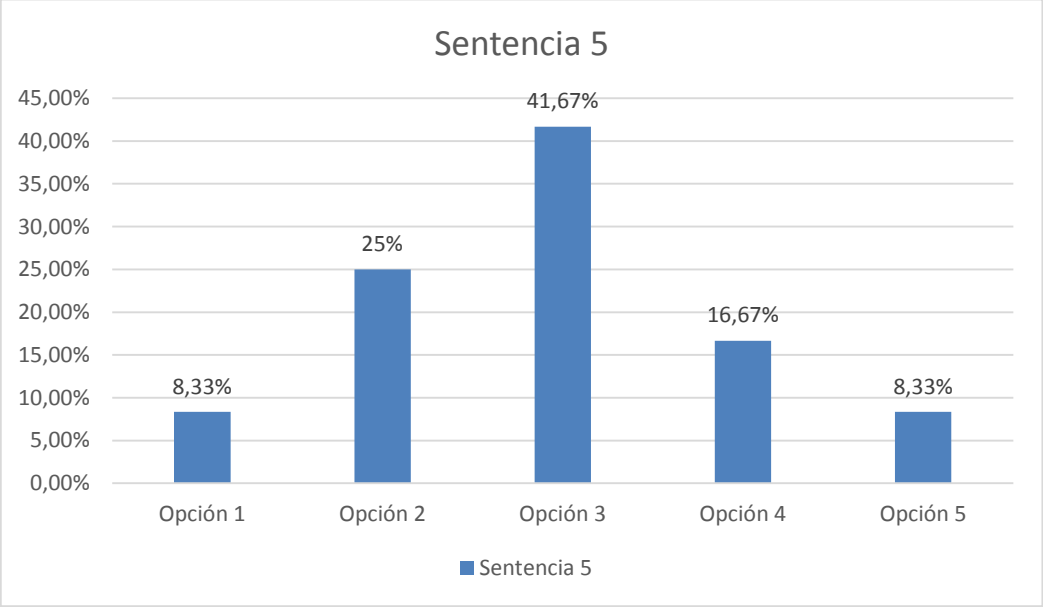


El 41,67% está en desacuerdo con la afirmación “cometí muchos errores al utilizar el generador de código” y solo el 16.67% está de acuerdo con la afirmación, la razón de esto puede deberse a algunas características no son lo suficientemente intuitivas para que el usuario las utilice correctamente.

- Sentencia 5:

  Cuando cometí errores, los mensajes del generador de código fueron muy claros en indicármelos.

Grafico 8. Resultado para la sentencia 5

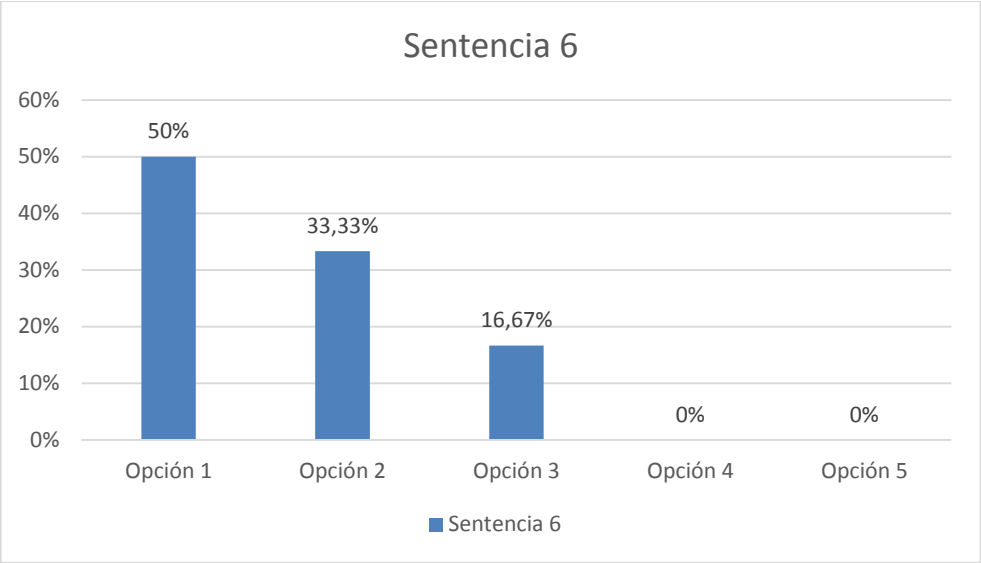


El 41,67% de los participantes ni está de acuerdo, ni en desacuerdo con la afirmación “cuando cometí errores, los mensajes del generador de código fueron muy claros en indicármelos”. Esto puede deberse a que los mensajes informan cuando un error ha sucedido, pero no son lo suficientemente descriptivos para que el usuario pueda entender la causa del error.

- Sentencia 6:

  Usar PatDis fue una experiencia muy frustrante.

Grafico 9. Resultado para la sentencia 6

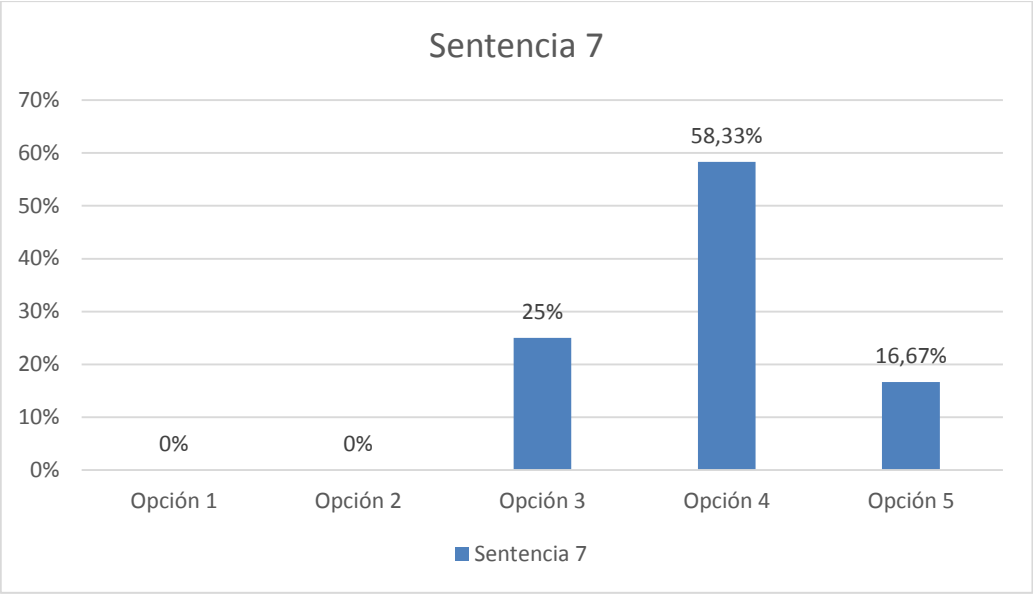


El 50% de los usuarios se encuentra muy en desacuerdo con la afirmación “usar PatDis fue una experiencia muy frustrante”, la razón de esto puede estar en el buen diseño de los contenidos del sitio.

- Sentencia 7:

  Es muy agradable trabajar con PatDis.

Grafico 10. Resultado para la sentencia 7

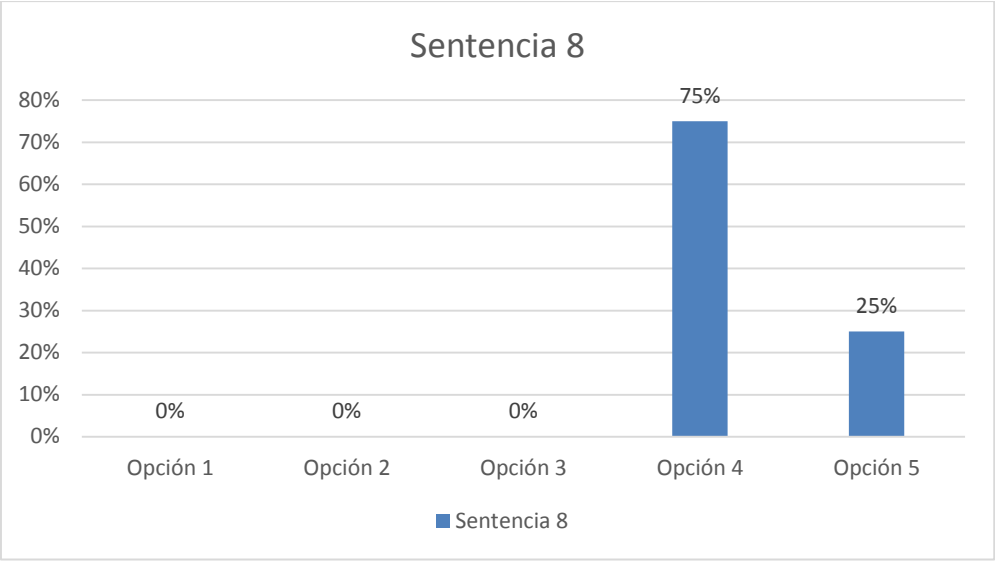


El 58% de los usuarios está de acuerdo con la afirmación “Es muy agradable trabajar con PatDis”, pero aun así solo el 16,67% está muy de acuerdo. Esto puede deberse a dificultades que tuvieron unos los usuarios entendiendo el funcionamiento del generador de código o a algunos diagramas dentro del contenido que no eran muy claros.

- Sentencia 8:

  Creo que PatDis es una herramienta realmente útil y de calidad.

Grafico 11. Resultado para la sentencia 8



El 100% de los usuarios está de acuerdo o muy de acuerdo con la afirmación “creo que PatDis es una herramienta realmente útil y de calidad”, esto puede deberse a que los usuarios encontraron el sitio web agradable y bien diseñado, que los contenidos se esfuerzan en ser claros y sencillos, y a que ven gran utilidad en el generador de código.

- Recomendar cambios

  A partir de los resultados de la prueba se pueden recomendar los siguientes cambios:

  - Dentro de las opiniones que escribieron los participantes de la prueba, la mayoría describía que algunas características del generador de código no son intuitivas y que este necesita una ayuda que guie al usuario en su uso.

  - Los participantes también indicaron que algunos diagramas de los contenidos se veían muy pequeños y eso los hacia difíciles de leer.

  - El contenido del sitio contiene algunos errores de digitación y ortográficos deben ser corregidos.

- Se encontró un bug en el mensaje que muestra el enlace de descarga del código java generado. El bug solo se presentaba cuando se generaba varias veces código y por parte de varios usuarios.

- Los mensajes que muestra el generador de código deben de ser más descriptivos, brindando al usuario una idea más clara del motivo del error.

- Es necesario evaluar la organización de la interfaz del generador de código, también es necesario indicar al usuario cuál elemento está editando actualmente y una fuente o color de letra que diferencie los textos propios de la interfaz del texto que puede editar el usuario.

9. CONCLUSIONES

1) Se logró el objetivo de desarrollar un catálogo de patrones GoF, utilizando para los ejemplos el lenguaje de programación Java, y sirviendo de herramienta que le sirva de apoyo a los estudiantes en su aprendizaje.

2) Se seleccionaron y desarrollaron los patrones: Builder, Factory Method, Singleton, Composite, Decorator, Adapter, Command, Iterator, Memento,Observer, State, Strategy y Template Method.

3) Se realizaron diagramas de clases y de secuencias, todos en UML y utilizando una única herramienta para mantener un único formato de presentación; esta herramienta fue Violet. Los diagramas de secuencia solo se realizaron en los patrones en los que se consideró necesario.

4) Se realizó de forma exitosa pruebas para medir la usabilidad que permitieron identificar algunas consideraciones y debilidades del sistema, así como cualidades y fortalezas a destacar.

5) Se desarrolló un sistema pensado para mantenerse y perdurar a futuro. Éste pretende ser una herramienta útil en el mediano plazo y que permita mantenerse y extenderse para ser útil en el largo plazo.

6) El desarrollo de este proyecto fue un proceso largo y fructífero que implicó trabajar diferentes aspectos, los cuales se pueden considerar como sub-proyectos independientes, de diferentes naturalezas, que al unirlos conforman un solo producto, al alcance de estudiantes e interesados en aprender acerca de los patrones de diseño. Estos sub-proyectos fueron los siguientes:

   - Se creó un contenido teórico para los 13 patrones de diseño abarcados, además de textos introductorios a estos, como la historia de los patrones y los principios SOLID. Todos estos textos, si los uniéramos, abarcarían una extensión aproximada de 120 páginas, incluyendo en ellos además diagramas UML e imágenes explicativas. Todo este contenido sirve como base para el aprendizaje de estos patrones y está

a disposición de la escuela para que haga uso de él, y si lo considera pertinente pueda realizar ajustes al mismo.

- Se realizó una recopilación de aplicaciones de ejemplo sencillas para todos los patrones, las cuales se ofrecen a los usuarios listas para ser descargadas y ejecutadas, todas en el idioma español y adaptadas a contextos con los que estuvieran familiarizados los estudiantes. Este conjunto de ejemplos acompañan y apoyan el contenido teórico.

- Se desarrolló un generador de código que permite al estudiante interactuar con los diagramas de los patrones y llevar a la práctica su uso, de modo que no se quedara solo con la teoría y los ejemplos, sino que diera un paso más allá y continuara en su aprendizaje involucrándose de una forma más activa.

  Además de lo anterior, es importante destacar que las cualidades del generador de código van más allá de su funcionalidad, la cual puede ser evidenciada por un usuario al hacer uso de éste por unos minutos. Una de las cualidades más importantes del generador de código es la ingeniería de software que hay detrás de él. Hubiese sido falto de coherencia si luego de resaltar la importancia de un buen diseño de software y de los mismos patrones de diseño, se hubiese pecado de un diseño que no cumpliera estos principios ni aplicara estos patrones; en vez de eso, se construyó un diseño que busca al máximo un bajo acoplamiento, aprovecha las bondades de la abstracción, goza de un código limpio y bien documentado, y aplica al menos dos de los patrones de diseño, como son Strategy en el módulo modelo y Factory Method en el módulo de validaciones. Otro aspecto importante es la aplicación de pruebas unitarias que además de facilitar el desarrollo de este software, facilitará el desarrollo de versiones posteriores del mismo. En definitiva, es una herramienta robusta y potente que puede ser explotada y extendida, lo cual es uno de los objetivos más difíciles en el desarrollo de software y el cual se pudo conseguir.

- Se desarrolló un sitio web con un buen diseño, agradable a la vista y el cual agrupa dichos contenido teórico, ejemplos y generador de código, con una organización adecuada que permitiera al estudiante navegar a través de estos y disfrutar de la experiencia completa.

10. RECOMENDACIONES

Se recomienda extender el número de patrones contenidos dentro del sitio, no solo para incluir los patrones faltantes de la banda de los cuatro, sino para incluir otros patrones que puedan ser útiles en el diseño y desarrollo de software.

Incluir más contenido relacionado a los patrones de diseño, que pueda facilitar el aprendizaje de los mismos. De momento el sitio solo trata temas como los principios SOLID y las relaciones entre patrones, pero podrían incluirse temas como el uso de patrones de diseño para la refactorización de software, o su uso de la mano con patrones arquitectónicos como MVC entre otros.

La versión actual del generador puede ser mejorada incluyendo más funcionalidades que hagan más flexible la forma en la cual el usuario puede modificar el código generado. Una funcionalidad importante que por su complejidad no se incluyó en esta versión, es la necesidad de que los cambios hechos en el patrón por el usuario se reflejen en el diagrama de clases que se muestra. Otro ejemplo de las funcionalidades que pueden ser incluidas está la posibilidad de agregar y eliminar elementos del diagrama de clases del patrón, según este lo permita.

Recomendamos seguir dando soporte al sitio web, para mantenerlo en línea y que sea de utilidad a la comunidad educativa.

CITAS BIBLIOGRAFICAS

[1] MARTIN, Robert. Agile Principles, Patterns, and Practices in C#. Westford: Prentice Hall, 2006.

[2] RALPH, P. y WAND, Y. A proposal for a formal definition of the design concept. En Design Requirements Engineering: A Ten-Year Perspective (LNBIP 14). Cleveland: Springer-Verlag, Junio 2007.

[3] BOOCH, Grady, RUMBAUGH, James y JACOBSON, Ivar. The Unified Modeling Language User Guide (2nd Edition). Boston: Addison-Wesley, 2005.

[4] BOOCH, Grady; et al. Object-Oriented Analysis and Design with Applications (3rd Edition). Boston: Addison Wesley, 2007.

[5] MARTIN, Robert. Clean Code: A handbook of agile software craftsmanship. Stoughton: Prentice Hall, 2009.

[6] GAMMA, Erich, HELM, Richard, JOHNSON, Ralph, VLISSIDES, John. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1994.

[7] LUJÁN, Sergio. Programación en Internet: Clientes Web. San Vicente: Editorial Club Universitario, 2001.

[8] MOZILLA DEVELOPER NETWORK Y COLABORADORES INDIVIDUALES. CSS. Consultado el 20 de agosto del 2014. Disponible en: [https://developer.mozilla.org/en-US/docs/Web/CSS]

[9] MOZILLA DEVELOPER NETWORK Y COLABORADORES INDIVIDUALES. JavaScript. Consultado el 28 de abril de 2016. Disponible en: [https://developer.mozilla.org/es/docs/Web/JavaScript]

[10] ORACLE. JavaServer Pages Technology. Consultado el 28 de abril de 2016. Disponible en: [http://www.oracle.com/technetwork/java/javaee/jsp/index.html]

[11] THE PHP GROUP. PHP Manual, Preface. Consultado el 28 de abril de 2016. Disponible en: [http://php.net/manual/en/preface.php]

[12] HETZEL, William C. The Complete Guide to Software Testing (2nd Edition). A Wiley-QED Publication, 1993.

[13] MICROSOFT. Unit Testing. Consultado el 20 de abril de 2016. Disponible en: [https://msdn.microsoft.com/en-us/library/aa292197%28v=vs.71%29.aspx]

[14] PIVOTAL LABS. Jasmine, A JavaScript testing framework. Consultado el 20 de marzo del 2016. Disponible en: [http://jasmine.github.io/]

[15] THE JQUERY FOUNDATION. QUnit: A JavaScript UnitTesting Framework. Consultado el 4 de abril de 2016. Disponible en: [https://qunitjs.com/]

[16] CHACON, Scott y STRAUB, Ben. Pro Git Second Edition. Consultado el 25 de febrero del 2016. Disponible en: [https://git-scm.com/book/es/v1]

[17] NIELSEN, Jakob. Usability Engineering. Oxford: Academic Press, 1994.

[18] PRESSMAN, Roger. Ingeniería de software: un enfoque práctico, sexta edición. New York: Mc Graw Hill, 2005.

[19] FREEMAN, Eric y FREEMAN, Elisabeth. Head First Design Patterns. Sebastopol: O'Reilly Media, 2004.

[20] HORSTMANN, Cay. Object-Oriented Design & Patterns (Second Edition). Hoboken: John Wiley & Sons, Inc., 2005.

[21] CHRISTIANSSON, Benneth, FORSS, Mattias, HAGEN, Ivar, HANSSON, Kent, JONASSON, Johan, JONASSON, Mattias, LOTT, Fredrik, OLSSON, Sara, ROSEVALL, Thomas. GoF Design Patterns - with examples using Java and UML2. Publicado por: varias plataformas de libros electrónicos como Scribd, spotbit.com y freetechbooks.com. 2008.

[22] COOPER, James W. The Design Patterns Java Companion. Boston: Addison-Wesley, 1998.

[23] METSKER, Steven J. Design Patterns Java Workbook. Boston: Addison-Wesley, 2002.

[24] KUCHANA, Partha. Software Architecture Design Patterns in Java. Boca Raton: Auerbach Publications, 2004.

[25] BRUEGGE, Bernd y DUTOIT, Allen. Ingeniería de software orientado a objetos. Naucalpan de Juárez: Pearson Education de México, 2002.

[26] GIT. Git Fast version control. Consultado el 30 de marzo del 2016. Disponible en: [https://git-scm.com/]

[27] SOMMERVILLE, Ian. Software Engineering, ninth edition. Boston: Pearson Education, 2011.

[28] ABASCAL, Elena y GRANDE, Ildefonso. Análisis de encuestas. Madrid: ESIC Editorial, 2005.

[29] UNIVERSIDAD INDUSTIAL DE SANTANDER. Plan de Estudios del Programa Académico de Ingeniería de Sistemas. Consultado el 03 de mayo de 2016. Disponible en [http://www.uis.edu.co/webUIS/es/academia/facultades/fisicoMecanicas/escuelas/ingenieriaSistemas/programasAcademicos/ingenieriaSistemas/planEstudios.html]

[30] LIKERT, Rensis. A Technique for the Measurement of Attitudes. En: Archives of Psychology. Junio, 1932. vol. 22, no. 140.

BIBLIOGRAFIA

BOOCH, Grady, RUMBAUGH, James y JACOBSON, Ivar. The Unified Modeling Language User Guide (2nd Edition). Boston: Addison-Wesley, 2005.

BRUEGGE, Bernd y DUTOIT, Allen. Ingeniería de software orientado a objetos. Naucalpan de Juárez: Pearson Education de México, 2002.

CHRISTIANSSON, Benneth, FORSS, Mattias, HAGEN, Ivar, HANSSON, Kent, JONASSON, Johan, JONASSON, Mattias, LOTT, Fredrik, OLSSON, Sara, ROSEVALL, Thomas. GoF Design Patterns - with examples using Java and UML2. Publicado por: varias plataformas de libros electrónicos como Scribd, spotbit.com y freetechbooks.com. 2008.

COOPER, James W. The Design Patterns Java Companion. Boston: Addison-Wesley, 1998.

FREEMAN, Eric y FREEMAN, Elisabeth. Head First Design Patterns. Sebastopol: O'Reilly Media, 2004.

HORSTMANN, Cay. Object-Oriented Design & Patterns (Second Edition). Hoboken: John Wiley & Sons, Inc., 2005.

KUCHANA, Partha. Software Architecture Design Patterns in Java. Boca Raton: Auerbach Publications, 2004.

MARTIN, Robert. Agile Principles: Patterns, and Practices in C#. Westford: Prentice Hall, 2006.

MARTIN, Robert. Clean Code: A handbook of agile software craftsmanship. Stoughton: Prentice Hall, 2009.

METSKER, Steven J. Design Patterns Java Workbook. Boston: Addison-Wesley, 2002.

NIELSEN, Jakob. Usability Engineering. Oxford: Academic Press, 1994.

PRESSMAN, Roger. Ingeniería de software: un enfoque práctico, sexta edición. New York: Mc Graw Hill, 2005.

ANEXOS

Anexo A

Escenarios y casos de uso

Escenarios:

Nombre: Utilización de Singleton para la conexión con una base de datos en una aplicación.
Actor(es): Luis (Estudiante).
Flujo de eventos: 1. Luis ingresa a la página web (https:…) y esta se carga en el navegador. 2. Luis ingresa al Generador de código clicándolo en el panel de la izquierda. 3. El sistema muestra la interfaz del Generador de código con la cual Luis puede empezar a interactuar. 4. Luis despliega la lista de patrones en la casilla Patrón. 5. Luis selecciona Singleton en la lista. 6. El sistema carga el diagrama de clases de Singleton mostrándolo en pantalla, del mismo modo carga los botones correspondientes a sus clases a su derecha, en este caso solo un botón correspondiente a la única clase, Singleton. 7. Luis hace clic en el botón Singleton. 8. El sistema carga en el panel de la derecha en las casillas Nombre, Atributos y Métodos, los datos por defecto para la clase Singleton. 9. Luis modifica estas tres casillas, asignando “UsoBaseDeDatos” en la casilla Nombre, “INSTANCIA” y “conexión” en la casilla Atributos, y “UsoBaseDeDatos()”, “getInstancia()”, “insert()”, “select()”, “showTables()” y “cerrarConexion()” en la casilla Métodos. 10. Luis hace clic en el botón Guardar. 11. El sistema actualiza los datos modificados por Luis en el diagrama de clases, mostrando el nuevo nombre de la clase y sus nuevos atributos y métodos. 12. Luis hace clic en el botón Generar código. 13. El sistema envía al servidor los datos del diagrama, el servidor procesa estos datos, genera el código fuente correspondiente y lo envía de vuelta al cliente en un archivo comprimido, y el navegador muestra en pantalla la descarga a realizarse, pidiendo confirmación. 14. Luis confirma la descarga y el archivo comprimido se descarga en su computador. 15. Luis extrae el contenido del archivo comprimido, obteniendo las clases

correspondientes al diagrama que manipuló, en este caso una sola clase (la clase Singleton).

16. Luis hace uso de este código en su programa, implementando los métodos definidos, entre otras cosas.

17. Luis vuelve al navegador y cierra la página.

Nombre: Implementación del patrón Composite en una aplicación para el menú de un restaurante.

Actor(es): Luis (Estudiante).

Flujo de eventos:

1. Luis luego de estudiar el contenido del patrón Composite en la página, se dirige al Generador de código clicándolo en el panel de la izquierda.
2. El sistema muestra la interfaz del Generador de código con la cual Luis puede empezar a interactuar.
3. Luis despliega la lista de patrones en la casilla Patrón.
4. Luis selecciona Composite en la lista.
5. El sistema carga el diagrama de clases de Composite mostrándolo en pantalla, del mismo modo carga los botones correspondientes a sus clases a su derecha, en este caso Componente, Hoja, Compuesto y Cliente.
6. Luis hace clic en el botón Componente.
7. El sistema carga en las casillas Nombre, Atributos y Métodos (del panel de la derecha), los datos por defecto para la clase Componente.
8. Luis modifica estas tres casillas, asignando “ComponenteMenu” en la casilla Nombre, y los datos correspondientes en Atributos y Métodos.
9. Luis hace clic en el botón Guardar.
10. El sistema actualiza los datos modificados por Luis en el diagrama de clases, mostrando el nuevo nombre de la clase modificada (ComponenteMenu, antes llamada Componente) y sus nuevos atributos y métodos.
11. Del mismo modo como modificó esta clase, Luis modifica una por una las clases Hoja, Compuesto y Cliente, nombrándolas ItemMenu, Menu y Mesero, respectivamente, modificando también sus atributos y métodos, y utilizando el botón Guardar al modificar cada una.
12. Luis hace clic en el botón Generar código.
13. El navegador le muestra una descarga a realizarse, pidiendo confirmación.
14. Luis confirma la descarga y un archivo comprimido se descarga en su computador.
15. Luis extrae el contenido del archivo comprimido, obteniendo las clases correspondientes al diagrama que manipuló.
16. Luis hace uso de este código, implementando los métodos definidos, entre otras cosas.

17. Luis vuelve al navegador y cierra la página.

Nombre: Implementación del patrón Strategy.

Actor(es): Luis (Estudiante)

Flujo de eventos:

1. Luis ingresa a la página y se dirige al Generador de código clicándolo en el panel de la izquierda.
2. El sistema muestra la interfaz del Generador de código con la cual Luis puede empezar a interactuar.
3. Luis despliega la lista de patrones en la casilla Patrón y selecciona Strategy.
4. El sistema carga el diagrama de clases de Strategy mostrándolo en pantalla, del mismo modo carga los botones correspondientes a sus clases a su derecha, en este caso Estrategia, EstrategiaConcreta1, EstrategiaConcreta2, EstrategiaConcreta3 y Contexto.
5. Luis modifica los datos de las cinco clases utilizando sus botones correspondientes, las casillas donde se cargan los datos y el botón Guardar.
6. El sistema actualiza los datos modificados por Luis en el diagrama de clases.
7. Luis hace clic en el botón Generar código.
8. El navegador le muestra una descarga a realizarse, pidiendo confirmación.
9. Luis confirma la descarga y un archivo comprimido se descarga en su computador.
10. Luis extrae y revisa las clases contenidas en el archivo comprimido
11. Luis decide que quiere modificar EstrategiaConcreta2 y EstrategiaConcreta3, así que vuelve al navegador, hace las modificaciones pertinentes y da clic nuevamente en Generar código, descargando nuevamente el código correspondiente.
12. Luis hace uso de este nuevo código, implementando los métodos definidos en todas las clases, entre otras cosas.
13. Luis vuelve al navegador y cierra la página.

Nombre: Interacción con los patrones Factory Method y Builder

Actor(es): Luis (Estudiante)

Flujo de eventos:

1. Luis ingresa a la página y se dirige al Generador de código clicándolo en el panel de la izquierda.
2. Luis despliega la lista de patrones en la casilla Patrón y selecciona

Factory Method.

3. El sistema carga el diagrama de clases de Factory Method mostrándolo en pantalla, del mismo modo carga los botones correspondientes a sus clases a su derecha.
4. Luis modifica los datos de algunas de sus clases utilizando sus botones correspondientes.
5. El sistema actualiza va actualizando los datos modificados por Luis en el diagrama de clases cada vez que hace clic en Guardar.
6. Luis ahora decide interactuar con el diagrama del patrón Builder, así que lo selecciona en la lista desplegable.
7. El sistema ahora carga el diagrama del patrón Builder y los botones correspondientes a sus clases.
8. Luis interactúa con el diagrama, haciendo algunas modificaciones a sus clases.
9. Finalmente decide que usará el patrón Factory Method, así que regresa este seleccionándolo en la lista desplegable.
10. Nuevamente modifica las clases del diagrama adaptándolas a lo que necesita.
11. Luis hace clic en el botón Generar código.
12. Luis descarga el código y hace uso de este.
13. Luis cierra la página.

Casos de uso:

Caso de uso:

Nombre del caso de uso:	GenerarCódigo
Actor participante:	Iniciado por Usuario
Condición inicial:	1. El usuario carga la página en la que está albergado el generador de código.
Flujo de eventos:	2. El usuario selecciona el patrón de diseño que desea trabajar. 3. El usuario puede seleccionar una de los elementos que conforman el diagrama de clases del patrón. 4. La aplicación resalta el elemento seleccionado y carga sus características en un formulario. 5. El usuario puede modificar las características del elemento seleccionado. 6. A terminar la modificación del elemento, el usuario guarda los cambios por medio de un botón.

	7. Después de modificar los elementos que el usuario necesite modificar, puede dar clic en el botón generar código y la aplicación inicia la tarea de generación. 8. La aplicación retorna en un archivo comprimido el código generado.
Condición de salida	9. El usuario guarda el archivo que contiene el código.

Caso de uso refinado:

Nombre del caso de uso:	GenerarCódigo
Actor participante:	Iniciado por Usuario
Condición inicial:	1. El usuario carga la página en la que está albergado el generador de código.
Flujo de eventos:	2. El usuario selecciona el patrón de diseño que desea trabajar. Puede elegir de un total de 13 patrones de diseño. 3. El usuario puede seleccionar una de los elementos que conforman el diagrama de clases del patrón por medio de un botón que lo representa. Los elementos pueden ser clases o interfaces y el diagrama sigue el estándar UML 2.X. 4. La aplicación resalta el elemento seleccionado y carga sus características en un formulario. Las clases tienen un nombre, atributos y métodos. Por su parte las interfaces tienen un nombre y métodos, pero no poseen atributos. 5. El usuario puede modificar las características del elemento seleccionado por medio de cada uno de los campos en los cuales se cargó cada una de sus características actuales. 6. A terminar la modificación del elemento, el usuario guarda los cambios por medio de un botón y la interfaz se actualizará para mostrarlos. 7. Después de modificar los elementos que el usuario necesite modificar, puede dar clic en el botón generar código y la aplicación inicia la tarea de generación. La aplicación informa que el proceso ha iniciado y en pocos segundos iniciara la descarga. 8. La aplicación retorna en un archivo comprimido el código generado.
Condición de salida	9. El usuario guarda el archivo que contiene el código.

Anexo B

Formato prueba de usabilidad

PRUEBA DE USABILIDAD - PATDIS

Descripción de PatDis: sitio web orientado a apoyar el proceso de aprendizaje de los patrones de diseño. Contiene un material teórico para el estudio de los patrones y un generador de código a través del cual el estudiante puede interactuar con las estructuras de los patrones y aplicarlos a sus propios proyectos.

Participantes de la prueba: estudiantes de Ingeniería de Sistemas de la UIS que hayan aprobado la materia Ingeniería de Software I.

INDICACIONES

Ingrese a http://patdis.uis.edu.co y siga las siguientes instrucciones:

- Navegue y explore toda la página (tiempo estimado: entre 5 y 10 minutos).
- Explore un poco la forma como están escritos los contenidos (tiempo estimado: entre 5 y 10 minutos).
- Interactúe con el generador de código, probando y utilizando toda su funcionalidad (tiempo estimado: entre 5 y 20 minutos).
- Responda el test a continuación (tiempo estimado: entre 5 y 10 minutos).

TEST

Por favor indique en una escala de 1 a 5, el grado de acuerdo o desacuerdo con las sentencias de la siguiente tabla a cerca de PatDis.

(1) Totalmente en desacuerdo

(2) En desacuerdo

(3) Ni de acuerdo ni en desacuerdo

(4) De acuerdo

(5) Totalmente de acuerdo

Sentencias	Grado de acuerdo o desacuerdo
Fue muy fácil navegar a través del sitio y entender la organización del contenido	
El contenido parece ser muy confuso y difícil de entender	
El generador de código es muy fácil de utilizar	
Cometí muchos errores al utilizar el generador de código	
Cuando cometí errores, los mensajes del generador de código fueron muy claros en indicármelos	
Usar PatDis fue una experiencia muy frustrante	
Es muy agradable trabajar con PatDis	
Creo que PatDis es una herramienta realmente útil y de calidad	

Por favor agregue a continuación cualquier sugerencia, apreciación o comentario que desee:

_____
_____
_____
_____
_____