Ciencia e Ingeniería
Revista Interdisciplinar de Estudios en Ciencias Básicas e Ingenierías.
Año 2021, Julio- Diciembre, Vol. (8) N° (2), e5716304, ISSN 2389-9484.
Universidad de La Guajira, Facultades de Ciencias Básicas y Aplicadas e Ingeniería.
La Guajira-Colombia.
Revista en Línea http://revistas.uniguajira.edu.co/index.php/cei
Este documento fue depositado en Zenodo. DOI: https://www.doi.org/10.5281/zenodo.5716304

LINEAMIENTOS PARA EL DISEÑO DE APLICACIONES WEB SOPORTADOS EN PATRONES GRASP

GUIDELINES FOR THE DESIGN OF WEB APPLICATIONS SUPPORTED IN GRASP PATTERNS

Gilberto Andrés Vargas Ortega

Fundación Universitaria San Mateo, Facultad de Ingeniería y afines
Bogotá, Colombia, E-mail direccion.sistemas@sanamteo.edu.co

Recibido: Diciembre 11 de 2020 Aceptado: Abril 23 de 2021

RESUMEN

El constante avance de las tecnologías de la información permite que las empresas puedan emplear nuevas formas para la producción de software con calidad. Sin embargo, un problema latente es la falta de personal capacitado que cumpla los requerimientos necesarios para los procesos de desarrollo, debido a que los profesionales recién egresados en la mayoría de los casos, no han adquirido las competencias necesarias en el manejo de herramientas de última tecnología y buenas prácticas. Un proceso complejo de desarrollo software requiere de personal no solo capacitado, sino con un alto nivel de creatividad que le permita aprender tecnologías nuevas para aplicarlas directamente en los proyectos. La capacitación y formación depende en parte de la institución de educación superior en donde el individuo forma sus competencias, y el nivel de creatividad lo adquiere participando activamente en procesos de desarrollo en donde tenga la oportunidad de interactuar con casos reales. Por lo anterior se plantea un proyecto que permita: Realizar un estudio de los patrones de diseño sugeridos por GRASP para su divulgación y empleo en la programación orientado a objetos, a través de instrumentos documentales que acompaña un prototipo software. Por tal motivo el estudiante de pregrado podrá contar con una herramienta documental acerca del uso de patrones de diseño de GRAPS, con el cual se rompe con el paradigma de diagramas de flujo de información y se crea uno nuevo, que permite al diseñador, concentrarse en las funciones específicas más que la interacción entre las mismas, lo que incide en el manejo de conceptos fundamentales de análisis, diseño y programación bajo el paradigma de orientación a objetos, (en procesos de ingeniería de software). Permitiéndole ser profesionales desarrolladores de aplicaciones flexibles, con diseños y códigos reutilizables, estableciendo de esta forma un vocabulario común y creando sistemas independientes del lenguaje en el que se vaya a desarrollar. De esta forma los estudiantes de ingeniería de sistemas aprenderán a desarrollar software con buenas prácticas, es decir con calidad

Palabras clave: Patrones de Grasp, Diseño de Grasp, ingeniería de Software

ABSTRACT

The constant advancement of information technology allows companies to use new ways to produce quality software. However, a latent problem is the lack of trained personnel who meet the necessary requirements for development processes, because recently graduated professionals, in most cases, have not acquired the necessary skills in handling the latest tools. technology and good practices. A complex software development process requires not only trained personnel, but also a high level of creativity that allows them to learn new technologies to apply them directly to projects. Training and education depends in part on the higher education institution where the individual forms their skills, and the level of creativity is acquired by actively participating in development processes where they have the opportunity to interact with real cases. Therefore, a project is proposed that allows: Carrying out a study of the design patterns suggested by GRASP for their dissemination and use in object-oriented programming, through

documentary instruments that accompany a software prototype. For this reason, the undergraduate student will be able to count on a documentary tool about the use of GRAPS design patterns, with which the paradigm of information flow diagrams is broken and a new one is created, which allows the designer to concentrate on the specific functions rather than the interaction between them, which affects the management of fundamental concepts of analysis, design and programming under the object-oriented paradigm (in software engineering processes). Allowing you to be professional developers of flexible applications, with reusable designs and codes, thus establishing a common vocabulary and creating systems independent of the language in which it is to be developed. In this way, systems engineering students will learn to develop software with good practices, that is, with quality

Keywords: Grasp Patterns, GRASP Design, Software Engineering

I. INTRODUCCIÓN

En el proceso de construcción de software los desarrolladores se enfrentan constantemente a problemas, provocados por el permanente aumento en la complejidad de las aplicaciones que deben desarrollar, esto provoca que provoca que las técnicas y estructuras que resultan adecuadas en un momento, con el paso del tiempo se vuelvan inadecuadas. La programación orientada a objetos es un paradigma que surge para apoyar la reutilización de software, POO presenta grandes ventajas sobre el paradigma de programación tradicional y se constituye como una manera especial de programar que intenta expresar un software lo más parecido al mundo real. Es una forma de pensar diferente, sin embargo, algunos nuevos desarrolladores no encuentran una metodología clara y eficiente que permita su uso.

En el proceso de desarrollo de software OO se presentan situaciones repetitivas a las cuales ya se ha encontrado una solución probada y comprobada a nivel funcional, estos son los patrones, sin embargo, en el sector académico, así como muchas veces no se manejan los conceptos de POO, es probable que no se conozca el concepto de Patrones, sin embargo, debido a sus grandes beneficios se considera de vital importancia el manejo y uso de estos conceptos. El uso de los patrones GRASP - General Responsibility Asigment Software Patterns (patrones generales de software para asignar responsabilidades) ayuda al entendimiento y la aplicación de los conceptos sobre la POO y a diseñar software con mayor calidad aplicando buenas prácticas que provienen de la experiencia de años de desarrollo.

El estudiante de pregrado al contar con una herramienta documental brindada por el programa de ingeniería de sistemas se beneficiará en los siguientes aspectos:

- ✓ Asimilar los conceptos fundamentales de análisis y diseño orientado a objetos.
- ✓ Introducirlo en los conceptos básicos del proceso de ingeniería de software, y sus etapas (obtención de requerimientos, análisis, diseño, implementación, pruebas, puesta en marcha).
- ✓ Despertar la habilidad para implementar software basado en el paradigma de orientación a objetos,
- ✓ Modelar gráficamente la solución de problemas con un enfoque orientado a objetos, usando un lenguaje de modelado, en cualquier herramienta visual.

Disponer de instrumentos físicos para afianzar los conocimientos recibidos en clase, permitirá a los estudiantes del programa de ingeniería de sistemas llegar a ser profesionales desarrolladores de aplicaciones con buenas practicas, con calidad, flexibles, con diseños y códigos reutilizables, estableciendo de esta forma un vocabulario común y creando sistemas independientes del lenguaje en el que se vaya a desarrollar.

Aunque el estudio de los patrones GRASP involucra temas básicos sobre diagramación en UML y programación orientada a objetos estos conceptos no se profundizan, se referencian e invitan a la consulta y aprendizaje de estos conocimientos que son esenciales en el Diseño de software y programación orientada a objetos. Por lo tanto el presente artículo se presenta como una guía para el uso de los patrones GRASP (ver figura 1), así como los pasos a seguir para lograr un óptimo desarrollo en un proyecto software en sus etapas de análisis, diseño y programación.

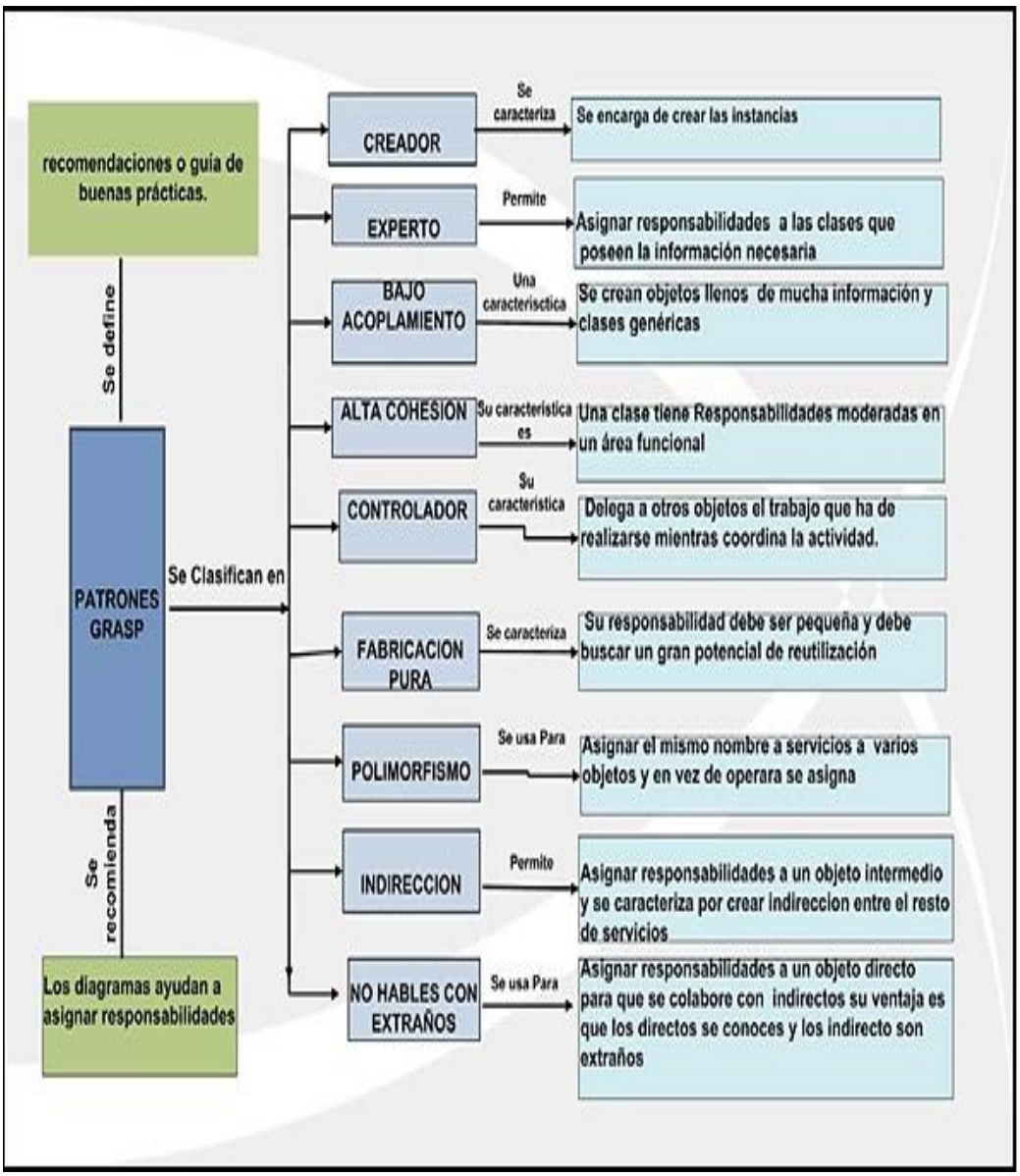

Fig.1 Mapa conceptual diseños de GRASP, Autores

Los patrones GRASP, basándose en la asignación de responsabilidades que es uno de los objetivos primordiales, lleva al usuario a cumplimiento de este en el momento de la creación de los diagramas UML y de colaboración, siendo estos la base de la creación del diseño de software, por lo tanto, es importante para GRASP que el diseñador tenga una claridad y dominio sobre el manejo y aplicación de los diagramas. Ahora bien, el uso de los patrones de diseño GRASP como apoyo a las metodologías de enseñanza en las asignaturas de diseño y programación de sistemas, de la carrera de ingeniería de sistemas se convierte en la luz del tren cuando el empleo de nuevas herramientas tecnológicas es uno de los objetivos para la generación de profesionales de alto perfil por parte de las instituciones educativas ya que permite al estudiante ir desarrollando habilidades en el manejo de los instrumentos necesarios para el buen rendimiento a la hora de la estructuración de proyectos.

1. Importancia de Asignación de Responsabilidades

Uno de los aspectos principales que maneja GRASP es la asignación de responsabilidades, durante la elaboración del diseño de una aplicación se asignan una serie de responsabilidades a los objetos a crear, basándose en dos condiciones muy importantes como son el hacer y el conocer.

- ✓ El hacer se refiere, a la capacidad que se puede tener para crear algo, controlar una serie de actividades o ser quien inicia actividades en otros.
- ✓ El conocer se refiere, al conocimiento total que se posee sobre la información contenida, los objetos relacionados o asociados, la información de estos y los datos a procesar y productos a fabricar.
- ✓ Orientado a la vida real es designar a la persona más idónea o la que más posea información para realizar una actividad específica.

Un ejemplo puede ser el ocurrido en una empresa donde todos los empleados poseen un computador para realizar sus tareas diarias, existe un departamento de sistemas, y se requiere formatear uno de los equipos de cómputo, quien sería el más indicado para esto, el ingeniero de sistemas que dirige el departamento y que tiene los conocimientos para poder realizarlo o uno de los técnicos encargados del área de mantenimiento de computadores (que no posee el perfil profesional del ingeniero). Lo más adecuado sería asignar esta tarea al técnico ya que él tiene la experiencia y trucos necesarios para la realización de esta actividad, pues es muy normal que el haga esto en sus labores diarias lo cual le genera más experiencia, en esa área, que el ingeniero.

En el diseño de sistemas orientados a objetos es similar, reemplazamos a las personas por las clases y las actividades por los objetos a crear, es decir, asignar responsabilidades es definir cuáles son los métodos e instanciaciones exactas a definir en las clases dependiendo de los datos que estas posean o se les hayan fijado.

2. Los patrones GRASP

Antes de abarcar la explicación de los patrones es preciso enfatizar en el uso de los diagramas de uso, de colaboración, de secuencia, etc., ya que son la base para la asignación de las responsabilidades a objetos y para la implementación de estas guías. A continuación, se pretende describir de una manera muy clara y sencilla nueve (9) patrones GRASP relacionados (tabla 1). GRASP es un catálogo que agrupa los siguientes patrones:

Tabla 1. Patrones GRASP

NUMERO	*PATRON*
1	Experto
2	Creador
3	Controlador
4	Bajo acoplamiento
5	Alta cohesión
6	Polimorfismo
7	Fabricación pura
8	Indirección

Fuente: Craig Larman

2.1 Experto

- ✓ Problema: Cuales son las clases y que nivel de responsabilidad se les puede llegar a asignar dentro del sistema.
- ✓ Solución: Asignar responsabilidades a las clases cuyos objetos poseen la información necesaria para cumplirlas. Para poder cumplir con esto es posible que requiera información distribuida en otros objetos, es decir, otros expertos. Para asignar las responsabilidades es indispensable apoyarse en los diagramas de colaboración (UML), quienes son la base para la toma de decisiones en la elaboración del diseño orientado a objetos bajo los Patrones GRASP.

2.2. Creador

- ✓ Problema: A cuáles clases asignar la responsabilidad de crear las instancias u objetos de otra clase.
- ✓ Solución: Utilizando los diagramas como el de colaboración se puede tener una relación de las actividades asignadas a las clases para que en conjunto con las siguientes condiciones se pueda definir claramente cuál o cuáles son los patrones creadores, permitiendo así tener una clara secuencia de las actividades que realiza la aplicación y el control en la generación de código excesivo que solo creara problemas en el momento del mantenimiento.

2.3. Controlador

- ✓ Problema: Dentro de las operaciones del sistema se generan eventos que son producto de acciones derivadas de otras operaciones, y estos están encargados de realizar el enlace con cada uno de los procesos, por lo tanto se debe definir quien se encargara, y en qué momento, atender dichos eventos.
- ✓ Solución: Con este patrón se intenta asignar responsabilidades, en el manejo de eventos, a las clases que tienen la capacidad de resolverlas manteniendo una alta cohesión con estas.

2.4. Bajo acoplamiento

- ✓ Problema: Como evitar la gran dependencia generada entre las clases que forman parte del sistema sin que esto conlleve a traumatismo y la generación de código excesivo para compensar en algún momento la ruptura en la relación de clases muy dependientes.
- ✓ Solución: Se debe tener poca dependencia entre las clases, el tener una alta dependencia entre clases creará una gran cantidad de software, el objetivo de este patrón es invitar a la creación de clases con la suficiente información para que su relación con otras se reduzca y puedan en algún momento subsistir si una de las clase a las cuales se relaciona deja de existir.

2.5. Alta Cohesión

- ✓ Problema: Como crear clases que contengan la suficiente información, sin que se llegue al exceso de contenido, para que puedan funcionar de una manera óptima y así evitar la alta dependencia con otras que le suministren datos necesarios para su existencia.
- ✓ Solución: Una clase con alta cohesión posee una importante funcionalidad y poco trabajo, colabora con otros objetos para compartir tareas muy grandes. Su alto grado de funcionalidad permite simplificar el mantenimiento y los mejoramientos.

2.6. Polimorfismo

- ✓ Problema: Como aplicar el uso de métodos con nombres y funcionalidades específicas en diferentes clases.
- ✓ Solución: Con la aplicación del patrón polimorfismo, es posible la creación de métodos genéricos que reciban y/o procesen información y ser usados en múltiples clases sin que esto sea un inconveniente ni se convierta en una redundancia de información.

2.7. Fabricación Pura

- ✓ Problema: Como crear objetos que no alteren o entren en conflicto con el bajo acoplamiento y la alta cohesión ya establecida en el diseño. Estos objetos no han sido concebidos desde un comienzo solo serán usados en tareas específicas.
- ✓ Solución: Crear clases que permitan la instanciación de objetos sin interferir en el bajo acoplamiento y la alta cohesión del diseño. Con la fabricación pura se permite el uso del patrón FACTORY el cual nos permite la instanciación de objetos en tiempo de programación, así como el uso de estos por diferentes objetos.

2.8. Indirección

- ✓ Problema: Como solucionar el tener que relacionar dos clases sin que exista una comunicación directa entre ellas sin que se violen las reglas de bajo acoplamiento y alta cohesión.
- ✓ Solución. El uso del patrón indirección nos orienta a intervenir en los conflictos creados en la solución de inconvenientes que se puedan presentar entre dos clases utilizando una tercera que se encargue de las actividades que ninguna de las dos pueda resolver por la naturaleza de su creación. Esta nueva se encargará de la relación con otras sin que exista un acoplamiento directo.

II. METODOLOGIA

3. Tipo de Investigación

Este estudio se describe como una investigación de enfoque cuantitativo de diseño no experimental de tipo exploratorio y descriptivo.

4. Población

Se tomó como población a evaluar 40 estudiantes de ingeniería de sistemas de la UCC de los semestres 7, 8 y 9 que han cursado las materias calidad software, gerencia de sistemas y compiladores. Para el cálculo de la población a evaluar se basó en la siguiente formula (ver figura 2), la cual a través de la asignación de un nivel de confianza del 10%, nos da a conocer cuál debe ser la cantidad de personas a evaluar. Con esta fórmula es posible calcular el tamaño de la muestra para la estimación de un porcentaje de variables dicotómicas.

$$n = \frac{0.25N}{\left(\frac{\alpha}{z}\right)^2 (N-1) + 0.25}$$

Fig. 2. Fórmula para el cálculo de la población Fuente: U.D. Bioestadística. Facultad de Medicina. Universidad de Málaga.

Donde N es el tamaño de la población, alfa es el valor del error tipo 1 z es el valor del número de unidades de desviación estándar para una prueba de dos colas con una zona de rechazo igual alfa. 0.25 es el valor de p2 que produce el máximo valor de error estándar, esto es p = 0.5 n es el tamaño de la muestra.

Cuando tenemos una variable dicotómica (o de Bernoulli) a menudo interesa saber en qué proporción de casos, p, ocurre el éxito en la realización de un experimento. También nos puede interesar el comparar la diferencia existente entre las proporciones en distintas poblaciones. También es de interés calcular para un nivel de significación dado, el tamaño muestral necesario para calcular un intervalo de confianza de cuyo radio sea menor que cierta cantidad (Correa y Parra, 2012).

El Tamaño muestral podrá cambiar de acuerdo a dos variables muy significativas: la población y el porcentaje de confianza que se puede esperar del experimento. A mayor población será menor el tamaño de esta si se espera un nivel bajo de confianza, y a menor población será mayor el tamaño si se espera u nivel alto de confianza.

5. Instrumento de Evaluación de Conocimientos

Se construyó a partir de un banco de preguntas tipo ECAES que se utilizan en las evaluaciones de los estudiantes para la preparación de las pruebas de estado y evaluaciones finales.

6. Factibilidad

La factibilidad de implementación del proyecto desarrollado en este documento se considera alta ya que para su uso se necesitan tres elementos totalmente existentes en la universidad y que no generaran inconvenientes ni trastornos en las actividades normales del claustro.

61. Factor Tecnológico

Se pueden desarrollar e implementar las labores de programación en cualquier lenguaje orientado a objetos, y que en el mercado se encuentran muchos gratuitos, además la universidad posee ya esta clase de herramientas.

6.2. Factor Humano

Como el objetivo es la inclusión de esta temática en los programas de estudio de la carrera de ingeniería de sistemas, los docentes encargados de estas serían los que analizarían los temas y los pondrían en práctica durante el desarrollo de sus clases.

7. Aspectos de Ingeniería

En este capítulo mediante un ejemplo se mostrará el uso de los patrones: controlador, alta cohesión y bajo acoplamiento. Se escogió como caso modelo el conjunto de actividades básicas que realiza un usuario en un cajero automático. En la tabla 2 se observará los requisitos que se utilizaron para el desarrollo del aplicativo del cajero automático.

Tabla 2. Requisitos del Aplicativo.

Requisito	Tipo de Requisito
Primero	Retiro de dinero, ofrece un menú de seis opciones con valores predeterminados incluyendo la digitación de uno deseado.
Segundo	Ofrece al usuario la posibilidad de observar el estado de cuenta actual
Tercero	Permite cambiar la clave de acceso al aplicativo

Fuente: Autor

Los patrones de GRASP son aquellos que asignan responsabilidades al momento del diseño y desarrollo de un programa orientado objetos, es decir se consideran buenas practicas o practicas básicas en el diseño y desarrollo de software; cabe resaltar que estos no son una influencia en la programación orientada a objetos, los patrones de GRASP se consideran programación orientada a objetos. Esto quiere decir que no compiten o son rivales de los patrones diseños, todo lo contrario, son una guía que permite encontrar mejores métodos en el uso de los patrones de diseño.

El patrón controlador de GRASP es el responsable de asignar el control del flujo de eventos del sistema a clases específicas, esto facilita la centralización de actividades como la validación y la seguridad. En la tabla 3, se explica los tipos de patrón controlador que existen en la actualidad.

Tabla 3. Tipo Patrón Controladores.

Tipo de Patrón Controlador	Descripción
El Controlador De Procesos	Es el encargado de realizar una determinada función y de evidenciar mediante un mensaje el resultado de la misma.
El Controlador DeVista	Es un estilo de arquitectura de software que separa los datos de una aplicación la interfaz de usuario y la lógica de control en tres componentes distintos, esto quieredecir que sin importar cuantas veces se ingresa a Hotmail, la interfaz es la misma, la diferencia es cuandoel usuario entra al aplicativo con su nombre y contraseña.
El Controlador "Controlador"	Este patrón es una clase que realiza llamadas a unas funciones asignando responsabilidades.

Fuente: Autor

8. Patrón Bajo Acoplamiento

El objetivo principal de este patrón es desplegar pocas responsabilidades dentro de una misma clase, es recomendable el uso de varias clases en el diseño del aplicativo para un mejor funcionamiento y evitar la baja cohesión del sistema (ver figura 3).

MODELO: CASO DE ESTUDIO "CAJERO AUTOMATICO"

9. Etapas del Modelo

El aplicativo cajero automático se realiza para presentar el uso de los patrones controlador, alta cohesión y bajo acoplamiento, desde el enfoque de la ingeniería de software, por tanto en el desarrollo del mismo se presentará las etapas de análisis, diseño, construcción y pruebas.

10. Etapa de Análisis

Contempla el análisis de requerimientos que para el ejemplo de cajero automático es el cumplimento de las funciones, retiro de dinero, estado de cuenta y cambio de clave. Como la intención es presentar el uso de los patrones controlador, alta cohesión y bajo acoplamiento, se toma un aplicativo que permita presentar fácilmente la forma de implementar los patrones en una determinada arquitectura. En la figura 3 se muestra las funciones que realiza un usuario en el cajero automático.

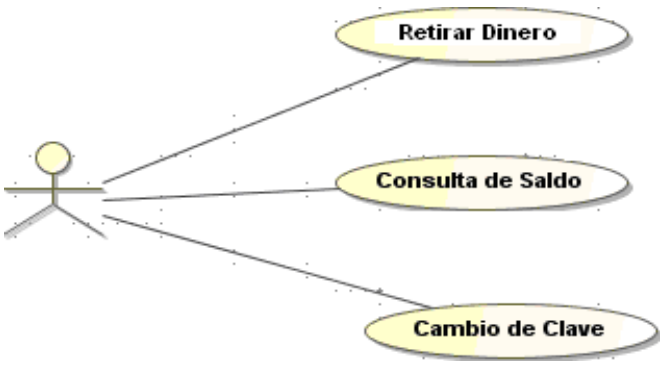

Fig. 3. Caso de Uso de un Cajero Automático

Para presentar el paso a paso de una determina función se hace un diagrama de secuencia, donde se muestra las actividades y el orden de las mismas.

Retirar Dinero. Permite al usuario realizar retiros o transacciones de dinero de la cuenta del usuario (ver figura 4).

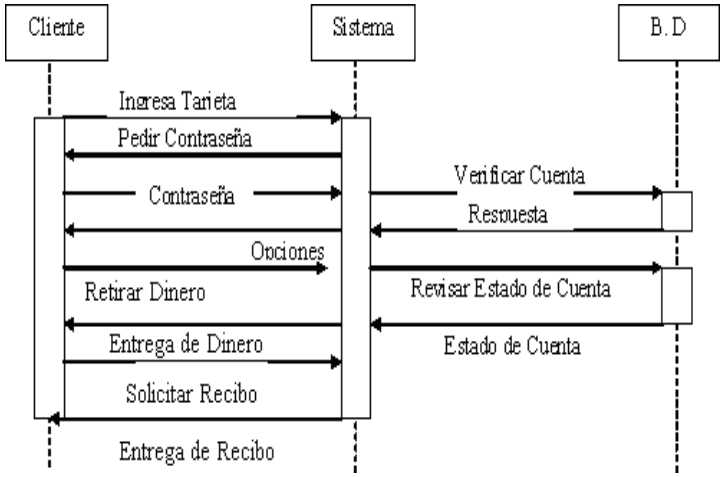

Fig. 4.. Diagrama de Secuencia para Retirar Dinero

La tabla 4 presenta la secuencia lógica del diagrama de secuencias que se diseñó para el caso de uso "Retiro de Dinero"

Tabla 4. Secuencia Lógica Del Diagrama De Secuencia del Caso de Uso "Retiro de Dinero".

Pasos enel Diagrama de Frecuencia	**Descripción**
Paso 1	El Actor desea retirar dinero, el cliente ingresa la tarjeta al cajero automático.
Paso 2	El sistema solicita al actor la contraseña.
Paso 3	El actor ingresa la contraseña.
Paso 4	El sistema confirma con la base de datos que la contraseñasea la correcta según la cuenta del actor y entrega respuesta ingresando a la siguiente interfaz.
Paso 5	El actor toma la opción de retirar dinero.
Paso 6	El sistema verifica con la base de datos el estado de cuenta del actor, si esta contiene dinero, selecciona el valor a retirar y si la cantidad a retirar es correcta, entrega respuesta del estado actual de cuenta al sistema.
Paso 7	El sistema entrega el dinero requerido por el actor

Fuente: Autor

A continuación, la tabla 5 presenta en orden cada uno de los pasos que se llevan a cabo en el caso de uso "Consultar Estado de Cuenta" (ver figura 5).

Tabla 5. Secuencia Lógica Del Diagrama De Secuencia Del Caso de Uso "Consultar el estado de Cuenta".

Pasos en el Diagrama de Frecuencia	Descripción
Paso 1	El Actor desea consultar el estado de cuenta, el cliente ingresa la tarjeta al cajero automático.
Paso 2	El sistema solicita al actor la contraseña.
Paso 3	El actor ingresa la contraseña.
Paso 4	El sistema confirma con la base de datos que la contraseñasea la correcta según la cuenta del actor y entrega respuesta ingresando a la siguiente interfaz.

Fuente: Autor

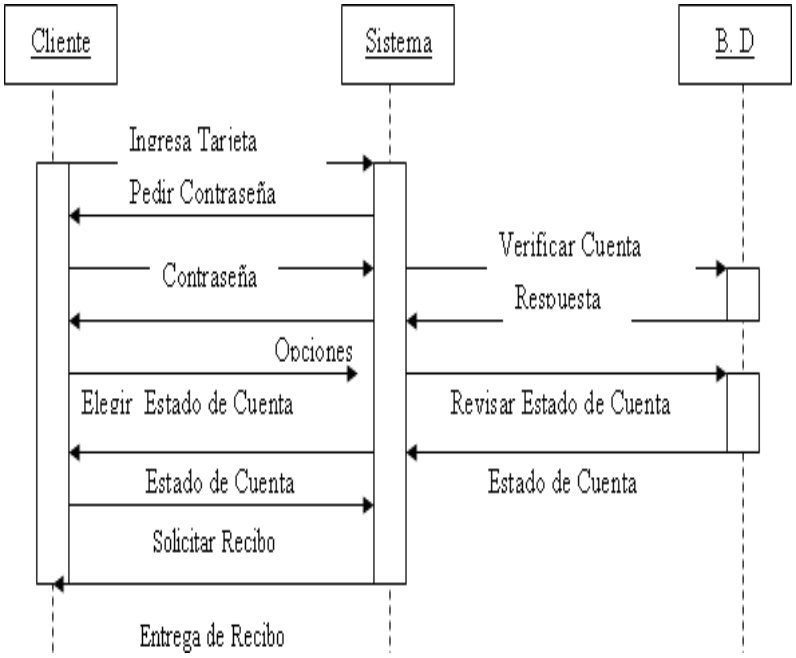

Fig.5. Diagrama de Secuencia del Caso de Uso Consultar el estado de Cuenta.

En la tabla 6 se describe en pasos la funcionalidad o secuencia lógica de realizar cambio de clave de un cajero automático.

Tabla 6. Secuencia Lógica Del Diagrama De Secuencia del Caso de Uso Cambio de Clave

Pasos en el Diagrama de Frecuencia	Descripción
Paso 1	El Actor desea cambiar la clave de la tarjeta, el cliente ingresa la tarjeta al cajero automático.
Paso 2	El sistema solicita al actor la contraseña.
Paso 3	El actor ingresa la contraseña.
Paso 4	El sistema confirma con la base de datos que la contraseña sea la correcta según la cuenta del actor yentrega respuesta ingresando a la siguiente interfaz.
Paso 5	El actor toma la opción de elegir nueva clave.
Paso 6	El sistema verifica el requerimiento y entrega respuesta solicitando una nueva clave.
Paso 7	El actor ingresa la nueva clave.
Paso 8	La nueva clave es guardada en la base de datos.
Paso 9	Se entrega respuesta por medio de un mensaje informando que la nueva clave fue guardada

Fuente: Autor

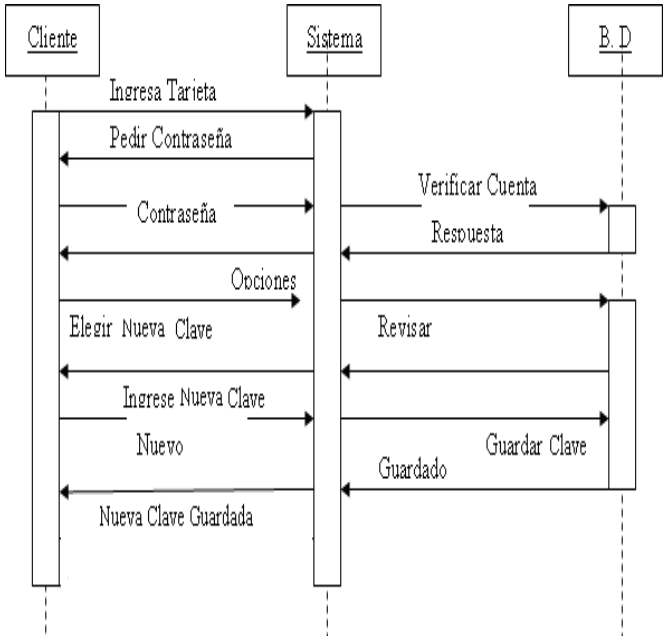

Fig.6. Diagrama de Secuencia de Caso de Uso "Cambio Clave".

A continuación, se describe el diagrama de clases con sus respectivos nombres, variables, métodos y la función que realiza cada una (ver figura 7).

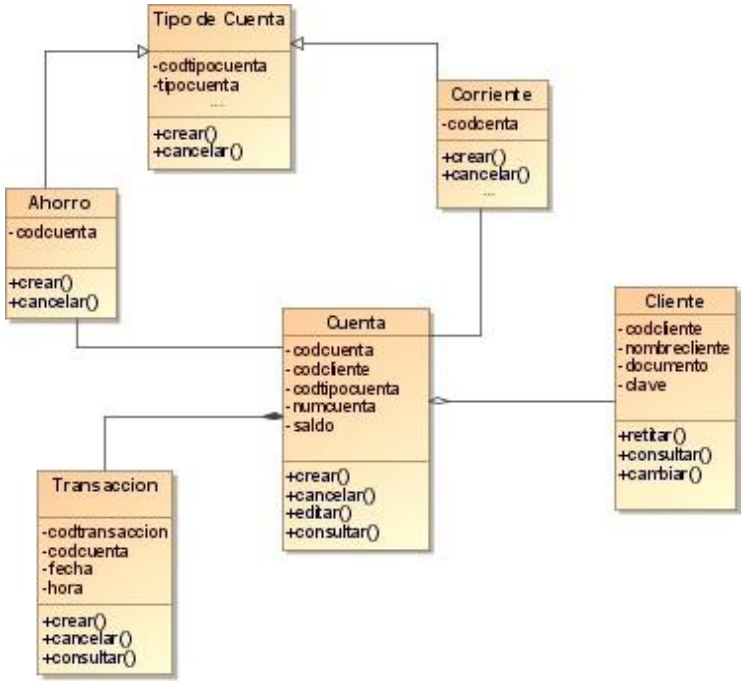

Fig.7. Diagrama de Clases Diseño Cajero Automático

- ✓ Cliente: La clase cliente tiene la función de crear usuarios, igualmente le permite interactuar con el cajero por medio de la cuenta ya que están conectadas a través de una agregación, donde le permite realizar retiros, consultas y cambios de claves.
- ✓ Cuenta: La clase cuenta es la más importante dentro del diagrama de clases, debido a que esta entrega información que desea el cliente, retiros o consultas y es la que maneja la información monetaria del cliente, igualmente se encuentra conectada con una agregación con la clase cliente y una composición con la clase transacción.
- ✓ Tipo de Cuenta: La clase tipo de cuenta define la cuenta que manejara el usuario si es tipo corriente o tipo ahorro, igualmente, esta clase interactúa de forma generalizada con las clases ahorro y corriente.
- ✓ Transacción: La Transacción entrega la información solicitada por el cliente, un retiro, una consulta, un movimiento o cambio de clave, igualmente interactúa con la clase cuenta con una composición ya que los datos que estregué esta clase depende de la clase cuenta.
- ✓ Ahorros: La clase ahorro es una clase que está generalizada con la clase tipo de cuenta, es decir que la información que se almacena en esta clase depende en su totalidad por la clase tipo de cuenta.
- ✓ Corriente: La clase ahorro es una clase que está generalizada con la clase tipo de cuenta, es decir que la información que se almacena en esta clase depende en su totalidad por la clase tipo de cuenta

Se presenta el diseño de la base de datos de la interfaz y de cada uno de los procesos (ver figura 8).

Base de Datos. La siguiente figura presenta el modelo entidad relación.

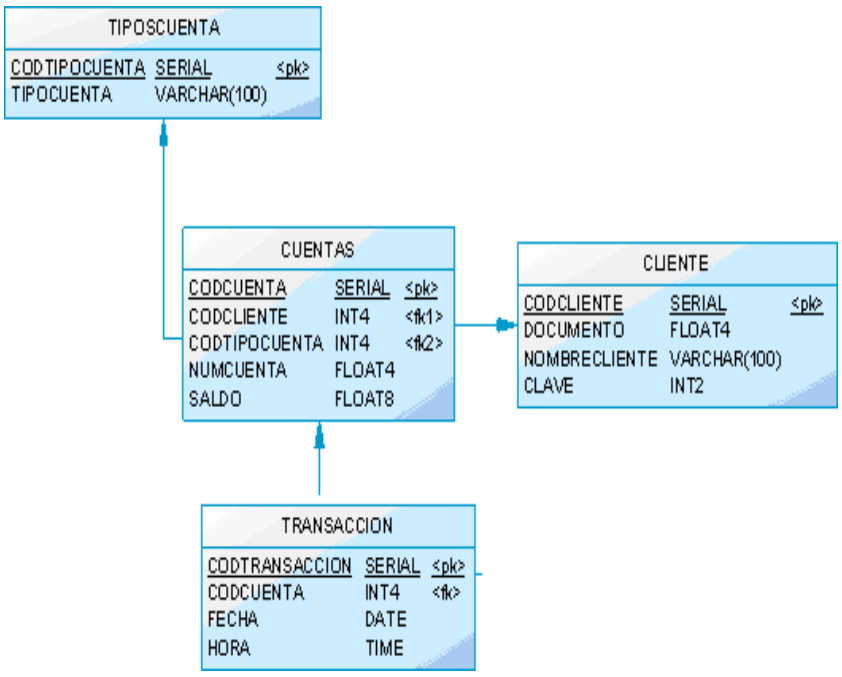

Fig.8. Modelo Entidad Relación.

De esta forma se realiza un diseño aplicando los patrones de GRASP, ahora lo más importante es que cuando se programe o desarrolle este diseño es importante tener en cuenta a cual clase se aplicara y codificara el patrón de control, ya que este patrón es el encargado de mantener todo en orden y que funcione el sistema según las necesidades que se requirieran, este caso cuando un cliente necesite retirar dinero el patrón de GRASP de control de la orden al aplicativo de realizar esta acción y no ingrese a realizar otra acción como cambio de clave por tal motivo es importante tener un esquema o diseño de proceso de interfaces y que sean manejadas por el patrón de control.

De esta forma aplicando los patrones de GRASP es más sencillo realizar diseño y desarrollo de software, es momento de actualizar a nuestros estudiantes y nuestros programadores en una era donde la tecnología y las aplicaciones cada día son indispensables (López y Cárdenas, 2019; Mora, 2019).

Se recomienda a los programas de ingeniería de sistemas que incorporen en el plan de estudios un curso que aborde el tema del uso de los patrones de diseño de GRAPS, y para su profundización ofertar diplomados o cursos intensivos para así lograr desarrollar en sus estudiantes y egresados un perfil de mayor calidad que proporcione las respuestas que el medio empresarial necesita

III. CONCLUSIONES

La población de programadores en Colombia no maneja una buena fundamentación teórica acerca de los Modelo Vista Controlador. De igual forma no existe una buena fundamentación teórica sobre arquitectura de software. Es muy mínimo el conocimiento sobre conceptos de programación orientada a objetos.

El uso de patrones de diseño de GRAPS rompe con el paradigma de diagramas de flujo de información y crea uno nuevo, que permite al diseñador, concentrarse en las funciones específicas más que la interacción entre las mismas.

El uso del patrón controlador es una buena práctica de desarrollo de la programación orientada a objetos ya que sugiere que dentro de una misma clase exista un conjunto de funciones con responsabilidades diferentes en pro de una alta cohesión y el bajo acoplamiento

No es fácil abstraer desde la teoría la funcionalidad de los patrones de diseño de GRAPS (Controlador, alta cohesión y bajo acoplamiento), lo que hace que la curva de aprendizaje sea representativa, por ello se recomienda que en la medida que se avance en el estudio del patrón, se aplique.

IV. AGRADECIMIENTOS

A la Fundación Universitaria San Mateo, por brindarnos la oportunidad de realizar el artículo de los patrones de GRASP y ofrecernos el apoyo necesario para lograr el cumplimiento de este objetivo.

Al personal de la facultad de Ingeniería y afines por su respaldo y colaboración, por su calidad humana el cual nos permitió de un ambiente de trabajo y estudio agradable.

V. LITERATURA CITADA

Correa, M. & Parra, B. (2012). *Modelo y guía para la implementación de gobierno de TI en entidades en entidades bancarias de Colombia*. Proyecto de grado. Universidad ICESI. https://repository.icesi.edu.co/biblioteca_digital/bitstream/10906/70666/5/modelo_gobierno_bancarias.pdf

Larman, C. (2011). *Uml y Patrones Introducción al análisis y diseño orientado a objetos*.

López Garzón, W., & Cárdenas López, J. (2019). Tecnología internet of things (IoT) y el big data. *Mare Ingenii*, *1*(1), 73-79.

Mora, D. (2019). Discapacidad y Software en Colombia. *Mare Ingenii*, *1*(2), 23-28. https://doi.org/10.52948/mare.v1i2.188.

Padilla, M. C. (2016). *Formulación y evaluación de proyectos*. Ecoe Ediciones.