Universidad de las Ciencias Informáticas

Facultad 3

Título: Extracción de reglas de asociación utilizando los algoritmos ECLAT y FP-GROWTH.

Trabajo de Diploma para optar por el título en Ciencias Informáticas

Autor: Osvaldo Guerra Cremé

Tutores: MSc. Julio César Díaz Vera

Ing. Guillermo Manuel Negrín Ortiz

Julio 2018

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Osvaldo Guerra Cremé **Guillermo M. Negrín Ortiz** **Julio César Díaz Vera**

_____	_____	_____
Firma del Autor	Firma del Tutor	Firma del Tutor

Datos de Contacto

MSc. Julio Cesar Díaz Vera

Universidad de las Ciencias Informáticas, La Habana, Cuba.

Correo electrónico: jcdiaz@uci.cu

Ing. Guillermo Manuel Negrín Ortiz

Universidad de las Ciencias Informáticas, La Habana, Cuba.

Correo electrónico: gmnegrin@uci.cu

Dedicatoria

Dedico la presente investigación a mi familia, principalmente a mis hermanos y primos menores, a esos a quienes les queda un poco para formarse profesionalmente, también quisiera dedicarle este trabajo a todos los que contribuyeron en poder hacer de este sueño una realidad.

–Osvaldo Guerra Cremé

Agradecimientos

La presente Tesis es un esfuerzo en el cual, directa o indirectamente, participaron varias personas, leyendo, opinando, corrigiendo, teniendo paciencia, dándome ánimo, acompañándome en los momentos de crisis y en los momentos de felicidad. Quiero agradecer a mi familia ante todo por haber estado ahí siempre de forma incondicional, primeramente, a mi mamá que sin su educación y su arduo esfuerzo para que yo fuera siempre hacia adelante no sería la persona que soy hoy en día. Siempre aconsejándome o peleándome, pero siempre a mi lado, cuidando de mí. A mi papá, persona a quien estimo y admiro mucho, ese que me ha inculcado principios y valores muy importantes en mi vida desde pequeño, a mi otro "Orlo", ese padre peleón que tengo indirectamente, ese que ha velado siempre por mí y por mi bienestar brindándome el apoyo indispensable en todo momento; a mi Mayka, quien me ha demostrado que no existen cosas imposibles, quien ha hecho posible en gran parte el que hoy me haga ingeniero. Quiero agradecer profundamente a mis abuelos, empezar por el que ya no está con nosotros mi abuelo Dionisio, a mis dos viejitas mi abuelita Iris y mi abuelita Quiri, siempre preocupadas por mí, siempre malcriándome; agradecer infinitamente, quisiera repetirlo, agradecer infinitamente a mi abuelo Ramón, mi estrella, ese personaje que DESDE SIEMPRE ha estado conmigo, en las buenas, en las malas, en todas; agradecer a mis hermanos Fredito y Oscar, ellos, a los que tengo que darle el ejemplo. Agradecer a mi novia, a mi cosita, a Jeidy, ella que también ha estado en buenos y malos momentos brindándome su amor. Agradecer quiero a mis amigos y compañeros que hicieron de estos años, los mejores de mi vida. A todos los profesores de esta carrera profesional en especial al GUILLE y a JULIO, mis tutores, que desde el día cero siempre ahí fajao's conmigo porque saliera bien este proceso, porque han aportado su granito de arena en mi formación, dedicando todo el tiempo que fuera necesario.

…gracias a todos, MUCHAS GRACIAS por estar ahí para mí, los quiero…

–Osvaldo Guerra Cremé

Resumen

El uso de algoritmos para extraer conocimiento en forma de reglas de asociación es una herramienta potente para la toma de decisiones estratégicas en una organización. Dentro del proceso de extracción de conocimiento de bases de datos, la minería de datos es la fase que más tiempo consume y seleccionar el algoritmo más eficiente es la piedra angular para satisfacer las demandantes necesidades de los decisores. Para abordar esta arista de la problemática se lleva a cabo una comparación entre los elementos del desempeño computacional de los algoritmos Eclat y FP-GROWTH, por medio del desarrollo de una aplicación informática en la que se implementan ambos. Para este desarrollo se presentan los elementos fundamentales de las fases del proceso, así como cada uno de los entregables generados del tránsito por las mismas. Una vez realizada la implementación se procede a realizar la verificación y validación del software y posteriormente se comprueban las propiedades estructurales de conjuntos de elementos frecuentes para facilitar el descubrimiento rápido y determinar cuál tiene mejor desempeño computacionalmente. Para ello se incluyen casos de estudio para presentar los resultados y realizar su discusión en donde se evidencia que es necesario incluir una ponderación a cada indicador para determinar cuál es mejor en dependencia de las condiciones de aplicación.

Palabras clave

Extracción de reglas de asociación, minería de datos, reglas de asociación.

Tabla de contenidos

Contenido

Introducción

La creciente necesidad que tienen los directivos de las empresas de encontrar información oculta y de importancia estratégica dificulta en muchas ocasiones la toma de decisiones. Obtener información puntual sobre sus negocios es una tarea especialmente difícil, puesto que esta información generalmente puede encontrarse dispersa en grandes volúmenes de datos. Como consecuencia de esto la minería de datos se ha convertido en una herramienta de suma importancia en el mundo actual porque tiene de poder descubrir patrones útiles o conocimiento a partir de fuentes de datos. La extracción de reglas de asociación es una de las tareas más utilizadas e importantes en la minería de datos, consiste en encontrar asociaciones interesantes en forma de relaciones de implicación entre los valores de los atributos de los objetos de un conjunto de datos.

En minería de datos y aprendizaje automático, las reglas de asociación se utilizan para descubrir hechos que ocurren en común dentro de un determinado conjunto de datos (T. Menzies, 2003). Se han investigado ampliamente diversos métodos para aprendizaje de reglas de asociación que han resultado ser muy interesantes para descubrir relaciones entre variables en grandes conjuntos de datos. Numerosos estudios abordados en (Alatas, 2008), (LaRosa, 2008), (HAN, 2007) avalan su actualidad e importancia y su aplicación en áreas como comercio, bioinformática, medicina y seguridad de redes entre otras.

La extracción de reglas de asociación emergió en la década de los 90 con aplicación práctica en el análisis de información de ventas en supermercados (AGRAWAL, y otros, 1993). Mediante ella se descubrían las relaciones entre los datos recopilados a gran escala por los sistemas de terminales de punto de venta. Los datos consistían en bases de datos que almacenan las transacciones ocurridas en el mercado donde cada transacción expresa qué productos compró un cliente (AGRAWAL, 1994). Un ejemplo de este tipo de colecciones puede ser que *"Si un cliente compra pan y leche, entonces también compra mantequilla"*, lo que formalmente formaría la asociación siguiente:

$$(pan \wedge leche) \Longrightarrow (mantequilla)$$

El interés de una regla de asociación está dado por su soporte y su confianza (Negrín Ortiz, 2014), entendiéndose por soporte la frecuencia de aparición de la combinación de productos

involucrados en la regla. Por ejemplo, para la regla formada anteriormente se tiene que el valor del soporte es:

$$supp\big((pan \wedge leche) \Longrightarrow (mantequilla)\big) = supp(pan \wedge leche \wedge mantequilla) = \frac{2}{6}$$

Por confianza (Negrín Ortiz, 2014) de una regla se entiende cuánto representa el soporte de la regla, por lo tanto el valor de la confianza es:

$$conf\big((pan \wedge leche) \Longrightarrow (mantequilla)\big) = \frac{supp(pan \wedge leche \wedge mantequilla)}{supp(pan \wedge leche)} = \frac{2}{3}$$

Se considera que una regla es interesante si su soporte y confianza son mayores o iguales que ciertos umbrales mínimos que se especifican (Negrín Ortiz, 2014).

Apriori, Eclat y FP-GROWTH son los algoritmos básicos más conocidos en la extracción de reglas de asociación. A priori, a pesar de su efectividad tiende a tener un desempeño computacional insuficiente. Sin embargo, Eclat y FP-GROWTH se han desarrollado para mejorar el desempeño de este en muchas condiciones, pero no abundan las evidencias concluyentes que sustenten la supremacía de uno de estos algoritmos con respecto al otro (BORGELT, 2003).

Con motivo de esta situación y para guiar el desarrollo de la investigación en la búsqueda de una solución factible se formula el siguiente problema a resolver:

Problema a resolver: ¿Cómo evaluar computacionalmente el desempeño de los algoritmos de extracción de reglas de asociación ECLAT y FP-GROWTH de forma automatizada?

En concordancia con lo anterior se toma como objeto de estudio:

Objeto de estudio: Proceso de desarrollo de software.

En respuesta al problema se define como objetivo general:

Objetivo general: Desarrollar una aplicación que compare el desempeño de modelos de los algoritmos de extracción de reglas de asociación ECLAT y FP-GROWTH.

Con lo anteriormente planteado se deriva el siguiente campo de acción y los objetivos específicos pertinentes:

Campo de acción: Proceso de desarrollo de software de aplicación.

Objetivos específicos:

1. Establecer el marco conceptual para el desarrollo de la investigación.

2. Realizar el diseño e implementación de la aplicación para la comparación.

3. Validar la aplicación mediante casos de estudio.

Como posibles resultados en esta investigación se tiene:

Posibles resultados: Aplicación para comparar el desempeño de los algoritmos de extracción de reglas de asociación ECLAT y FP-GROWTH.

Para dar cumplimiento al objetivo general se desarrollarán las siguientes tareas a cumplir en la investigación:

Tareas a cumplir:

1. Revisión bibliográfica
2. Definición del marco comparativo
3. Modificación de los algoritmos FP- GROWTH y ECLAT para computar las métricas definidas en el marco comparativo
4. Implementación de una aplicación que permita comparar el desempeño de los algoritmos
5. Presentación de los resultados

Descripción de los capítulos

En el Capítulo 1 se presentan los elementos esenciales sobre los que se sustenta la investigación realizada. Inicialmente se exponen conceptos fundamentales tales como: minería de datos, reglas de asociación como resultado de este proceso y el problema que representa su extracción, desempeño computacional y lo que constituye a la hora de la elección de alguno los algoritmos de extracción, posteriormente se aborda acerca del proceso de desarrollo, así como los tipos de software que existen. Finalmente se hace un estudio acerca de distintas metodologías de desarrollo que existen para determinar cuál de estas se ajusta al presente trabajo; teniendo cimentado el proceso de desarrollo de software por el cual se regirá la investigación.

Posteriormente en el Capítulo 2 se aborda acerca de las tareas ingenieriles que se realizaron para poder lograr una comparación entre los algoritmos de extracción de reglas de asociación Eclat y FP-GROWTH; definiendo requisitos funcionales y no funcionales. Luego se realiza un modelado de relación de las clases que dan solución al problema a resolver; se define una línea base tecnológica, se definen las herramientas y lenguajes. Finalmente, con todo esto planteado se lleva a cabo la implementación.

Finalmente, en el Capítulo 3 se trata acerca de la validación de la investigación que se aborda a través de tres pruebas a tres casos de estudio para comprobar cuál de los dos algoritmos tiene mejor desempeño computacional. En último lugar se presentan los resultados de los experimentos obtenidos de la ejecución de los algoritmos, resumiéndolos en tablas y gráficos y se arriba a conclusiones.

Capítulo 1

Fundamentación Teórica

1.1 Introducción

En este capítulo se presentan los conceptos sobre los que se sustenta la investigación realizada. Inicialmente se exponen las definiciones fundamentales de minería de datos, reglas de asociación como resultado de este proceso y el problema que representa su extracción. Luego se trata acerca del desempeño computacional de algoritmos de extracción de reglas de asociación y su comparación. Posteriormente se estudian los conceptos esenciales dentro del proceso de desarrollo de software para poder adaptarlo al desarrollo de una aplicación que evalúe computacionalmente el desempeño de los algoritmos ECLAT y FP-GROWTH.

1.2 Minería de datos

La minería de datos es una fase del *Knowledge Discovery in Databases* (KDD) a la cual anteceden la recopilación e integración y la selección, limpieza y transformación, también conocido como pre-procesamiento (Logreira, 2011). En estas fases se extraen los datos de distintas fuentes para disponerlos de forma unificada y accesible y se tratan para eliminar inconsistencias, datos repetidos, datos anómalos, rellenar valores y suavizar el ruido, (VALLEJOS, 2006). La minería, encargada de extraer conocimiento en forma de patrones es complementada posteriormente con la evaluación, difusión y uso. La etapa de post-procesamiento tiene como objetivo el incremento de comprensibilidad e interés del conocimiento extraído, (Martínez, 2003).

La minería de datos se define como el descubrimiento automático por métodos de *machine-learning* de patrones previamente desconocidos o relaciones en largos y complejos conjuntos de datos: bases de datos, almacenes de datos. Es un proceso que intenta descubrir patrones en conjuntos de grandes volúmenes de datos a través de los métodos de la inteligencia artificial, aprendizaje automático, estadística y sistemas de bases de datos, consiste en extraer información, transformarla y hacerla más comprensible para su uso posterior (Rennolls, 2005).

1.1.2 Reglas de asociación

Las reglas de asociación (AGRAWAL, y otros, 1993) son las relaciones entre variables que se utilizan para descubrir hechos que ocurren en común dentro de una determinada colección de datos, los cuales pueden ser procesados mediante una técnica llamada extracción de reglas de asociación con algoritmos que se analizarán en la presente investigación.

Normalmente se buscan relaciones entre las tuplas o entre los atributos, estas asociaciones se pueden representar de diversas formas. Entre las formas de representación se encuentran: dependencias funcionales, reglas de asociación, dependencias funcionales aproximadas, dependencias funcionales difusas, reglas de asociación difusas, datos inconsistentes, patrones, anomalías, entre otras.

(AGRAWAL, 1994) define como regla de asociación:

"Sea $I = \{i_1, i_2, \ldots, i_n\}$ un conjunto de literales y D un conjunto de transacciones definidas sobre I. Una regla de asociación es una implicación de la forma $X \rightarrow Y$, donde $X \subset I, Y \subset I$ y $X \cap Y = \emptyset$."

1.3 Desempeño computacional en la extracción de reglas de asociación

La extracción de reglas de asociación está relacionada al descubrimiento de patrones en los datos. Es decir, descubre relaciones entre los atributos de una base de datos; produciendo sentencias de tipo: *si-entonces*, relativas a los atributos y sus valores (Witten, 2016).

Está dividida en dos pasos:

1. Descubrir los conjuntos de elementos frecuentes, es decir, los conjuntos de elementos con soporte superior a un umbral prefijado.
2. Utilizar los conjuntos de elementos frecuentes para generar reglas de asociación.

El primero de los pasos es el que consume mayores recursos de tiempo y espacio. La generación de reglas, una vez que se tienen los conjuntos de elementos frecuentes, es simple. Con el fin de obtener estos conjuntos se han propuesto múltiples algoritmos, tomando como punto de partida uno en particular y después intentando mejorar su eficiencia. Ejemplos de

estos son el algoritmo Apriori, AprioriTID y AprioriHybrid, DHP, Eclat, FP-GROWTH (Fernandez, 2005), teniéndose como propósito en esta investigación tratar el desempeño computacional de ECLAT y FP-GROWTH.

Para lograr la extracción de reglas de asociación es de vital importancia saber acerca de las siguientes definiciones (Medina, y otros, 2007) para poder entender correctamente el funcionamiento de los algoritmos de extracción de reglas de asociación que serán explicados en el **Capítulo 3** de esta investigación:

1. Conjunto de ítems o itemset: es aquel cuyos elementos son atributos, variables o campos de cualquier base de datos.
2. Conjunto de ítems (itemset) frecuente o extenso: un conjunto de ítems (itemset) frecuente (frequent itemset) o extenso (large itemset) es aquel que ocurre con un soporte no menor que un mínimo dado (minsup).

Los algoritmos anteriormente mencionados a la hora de su ejecución no necesariamente se desenvuelven de igual forma, aunque estos arrojen los mismos resultados, es por ello que incide mucho el desempeño computacional en estos.

Cuando se aborda el tema de desempeño computacional se debe tener en cuenta que viene regido por diferentes tipos de indicadores tales como: uso de memoria, tiempo de ejecución en forma general (Felipe Alfaro, 2015), uno de los aspectos que se podrían incluir teniendo en cuenta el contexto de esta investigación sería la cantidad de reglas generadas por cada uno de los algoritmos, pero al estar hablando de elementos frecuentes en un conjunto de transacciones, para los mismos valores de soporte, se tendrá el mismo resultado en dependencia del algoritmo que se esté empleando.

Teniendo en cuenta los aspectos sobre los que se va a desarrollar la evaluación de desempeño de los algoritmos ECLAT y FP-GROWTH, es necesario contar con una plataforma sobre la que puedan evaluarse estos indicadores, por lo que es de vital importancia abordar acerca del proceso de desarrollo de software para la elección del tipo de aplicación que se va a utilizar.

1.4 Proceso de desarrollo de software

El término “software” fue usado por primera vez en este sentido por John W. Tukey en 1957. En la ingeniería de software y en las ciencias de la computación, el software es toda la información procesada por los sistemas informáticos: programas y datos (Clasificación de Software, 2011). Existen definiciones más completas como la de Pressman (Pressman) que define software como:” …el producto que construyen los programadores profesionales y al que después le dan mantenimiento durante un largo tiempo. Incluye programas que se ejecutan en una computadora de cualquier tamaño y arquitectura, contenido que se presenta a medida que se ejecutan los programas de cómputo e información descriptiva…”. Puede definirse también como un el conjunto de programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación. Considerando estas definiciones el concepto software va más allá de los programas de computación en sus distintos estados: código fuente, binario o ejecutable; también su documentación, los datos a procesar e incluso la información de usuario forman parte del software, es decir, abarca todo lo intangible, todo lo no físico relacionado (IEEE standard glossary of software engineering terminology, 1990).

Una rama de la ingeniería que se dedica a la construcción de software es la ingeniería de software. La ingeniería de software es una disciplina que integra métodos, herramientas y procedimientos para el desarrollo de software de computadora (Pressman). Asimismo, se trata de la aplicación sistemática, disciplinada y cuantificable para el desarrollo, operación y mantenimiento de un software.

Cuando se va a desarrollar software es necesario llevar a cabo un proceso para la obtención del mismo. El proceso de desarrollo de software es un modelo utilizado para la generación de un producto de software, describiendo enfoques diferentes para una variedad de tareas y actividades a ser ejecutadas a lo largo del proceso (Noriega Martínez, 2017).

Los apartados siguientes muestran los pasos a ejecutar en forma general durante el proceso de desarrollo de software:

1. Investigar los requisitos de los usuarios. Esto se lleva a cabo durante la fase de análisis.
2. Definir las características necesarias para el sistema (especificación).
3. Crear o adaptar una solución adecuada al problema, es decir, la creación del proyecto.
4. Desarrollar la solución propuesta, o sea, la implementación.
5. Garantizar que la solución responde al problema propuesto y que esta funciona correctamente en el contexto a ser ejecutada, para ello se llevan a cabo diferentes pruebas.
6. Modificar las soluciones de trabajo cuando nuevos requisitos son presentados o identificados (Noriega Martínez, 2017).

En la presente investigación se identifican desde el primero hasta el quinto pasos, puesto que para realizar una comparación entre los algoritmos ECLAT Y FP-GROWTH es necesario implementarlos para así evaluar el desempeño de estos. Luego, se comprueba si la implementación funciona correctamente y si quedan satisfechas las métricas por las que se hará la comparación definidas en el próximo capítulo.

El proceso de desarrollo de software varía en dependencia de lo que se quiere lograr con este. La solución en la presente investigación se va a realizar a través de la implementación de un software que permita evaluar el desempeño de los algoritmos ECLAT y FP-GROWTH. Teniendo en cuenta lo planteado anteriormente, se necesita saber qué tipo de software se va a desarrollar para luego definir el proceso que se va a utilizar.

1.4.1 Tipos de software

Según anuncia (Darshan, 2015) dentro de los diferentes tipos de softwares están:

1. Software de programación.
2. Software de sistema.
3. Software de aplicación

Los **software de programación** son el conjunto de herramientas que permiten al programador desarrollar programas informáticos, usando diferentes alternativas y lenguajes de programación, de una manera práctica. Un lenguaje de programación es un idioma artificial diseñado para expresar instrucciones que pueden ser llevadas a cabo por máquinas como las computadoras. Este tipo de sistema puede usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Apoyado por un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos definidos en el lenguaje ofrecen desde hace ya algunos años elementos de análisis de cada una de las fases de la compilación (léxica, sintáctica y lógica), además de la identificación y corrección de errores en el código, permiten escribir, probar, depurar, compilar y mantener el código fuente de un programa informático (Darshan, 2015).

Los **software de sistemas** (Darshan, 2015) son un conjunto de programas que administran recursos de un sistema informático, realizando una variedad de funciones. El software del sistema se puede clasificar ampliamente en tres tipos según anuncia (Darshan, 2015):

Programas de control del sistema: controlan la ejecución de los programas, administrar los recursos de almacenamiento y procesamiento de la computadora y realizar otra función de gestión y supervisión. El más importante de estos programas es el sistema operativo. Otro ejemplo son los sistemas de gestión de bases de datos (DBMS) y monitores de comunicación.

Programas de soporte del sistema brindan funciones de servicio de rutina para los otros programas de computadora y usuarios de computadoras.

Programas de desarrollo de sistema ayudan en la creación de aplicaciones programas. Por ejemplo, traductores de idiomas y generadores de aplicaciones.

El **software del sistema** se puede ver como un software que vincula lógicamente los componentes de una computadora para que funcione como una sola unidad y proporciona la infraestructura sobre la cual pueden funcionar los programas. Es responsable de controlar el hardware de la computadora y otros recursos, y permite que el software de aplicación interactúe con las computadoras para realizar su tarea. Algunos software de sistemas específicos son ensambladores, enlazadores, cargadores, macro procesadores, editores de

texto, compiladores, sistema operativo, sistema de depuración, sistema de control de código fuente (Darshan, 2015).

Los **software de aplicación** son prácticamente incontables. Existen programas que nos permiten resolver necesidades de organizaciones, reproducir música, resolver cálculos matemáticos, jugar, ver videos, dibujar y comunicarnos con otras computadoras a través de la línea telefónica o redes inalámbricas. Existen distintos programas para proyectar un edificio, para hacer videos musicales, para crear efectos especiales en una película, etc. Las posibilidades crecen día a día, y éstas dependen de los programas y del tipo de hardware que se utilice (Darshan, 2015). Se tiene como software de aplicación también como programas aislados que resuelven una necesidad específica de negocios, conocido dentro de esta clasificación como software de gestión. Las aplicaciones en esta área procesan datos comerciales o técnicos en una forma que facilita las operaciones de negocios o la toma de decisiones administrativas o técnicas. Además de las aplicaciones convencionales de procesamientos de datos, el software de aplicación se usa para controlar las funciones de negocios en tiempo real (Pressman).

De los tipos de softwares antes mencionados, el software de aplicación es el que mejor se adapta a la necesidad de la presente investigación, puesto es el que facilita las prestaciones específicas para realizar la comparación entre los algoritmos ECLAT y FP-GROWTH y porque también es un software de gestión.

Antes de continuar es necesario definir el propósito u objetivo que regirá el desarrollo del software que dará solución al problema.

1.5 Requisitos de alto nivel

Un requisito de alto nivel (POHL, 2016) está asociado a un objetivo, meta o propósito que un sistema debe satisfacer. Los requisitos de alto nivel son un elemento fundamental para la confección de un proyecto, son este tipo de requisitos los que se utilizan para establecer el primer entendimiento entre la persona que desea el sistema y el equipo encargado de su desarrollo. Y es finalmente asociado a estos requisitos que se evalúa si el sistema cumplimentó el objetivo para el que fue desarrollado.

Teniendo definido el concepto de requisito de alto nivel, se tiene como requisito de alto nivel en esta investigación:

- Evaluar el desempeño computacional de los algoritmos de extracción de reglas de asociación ECLAT y FP-GROWTH.

Con los aspectos mencionados anteriormente no es suficiente para la elaboración de un software, puesto que no se cuenta con una metodología por la cual regir el proceso de desarrollo del mismo.

1.6 Metodología del proceso de desarrollo de software

Las metodologías de desarrollo de software son marcos o modelos de trabajos que se utilizan para construir, planificar y controlar el proceso de desarrollo de sistemas. Hoy en día existen infinidades de metodologías para desarrollar software. Entre ellas encontramos las Metodologías Tradicionales, las Metodologías Iterativas/Evolutivas, las Metodologías basadas en Tecnología Web, y las Metodologías Ágiles (Zamora, 2014) .

Las Tendencias modernas en el desarrollo de software apuntan hacia el uso de metodologías más flexibles con un enfoque simple, donde el cliente está presente en todo el proceso de avance, estas son las metodologías ágiles mencionadas anteriormente, siendo estas las que mejor se adaptan en la presente investigación debido a que no se desarrollará un software de gran magnitud.

1.6.1 Metodología XP

Una de las principales es la metodología de Programación Extrema, a continuación XP, por sus siglas en inglés, la cual garantiza contar con una herramienta accesible al usuario, sencilla y a la misma vez dinámica. XP constituye un modelo de trabajo compartido, donde existe la conexión entre el cliente y el desarrollador, lo que permite la construcción de un sistema de acuerdo a los requerimientos establecidos por el cliente al principio de llevar a cabo el proyecto (Darío, 2016).

Para alcanzar el objetivo de software como solución ágil, la metodología XP se estructura en tres capas que agrupan las doce prácticas básicas de XP según (Pacienzia, 2015):

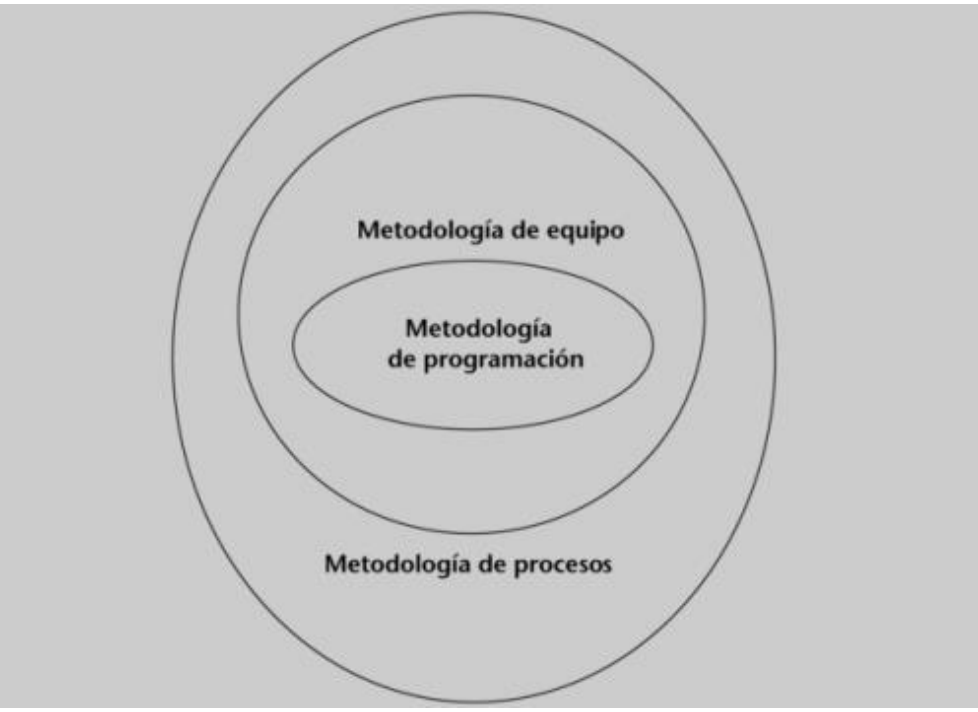


Figura 1: Capas de la programación extrema

1. Metodología de programación: diseño sencillo, pruebas, refactorización y codificación con estándares.
2. Metodología de equipo: propiedad colectiva del código, integración continua, entregas semanales e integridad con el cliente.
3. Metodología de procesos: entregas frecuentes y planificación.

1.6.1.1 Fases de la metodología XP

Por otra parte, el proceso de desarrollo de software (Pressman) de la metodología de Programación Extrema XP, está guiado por 4 fases:

- Planeación

La Metodología XP plantea la planificación como un diálogo continuo entre las partes involucradas en el proyecto, incluyendo al cliente, a los programadores y a los coordinadores. El proyecto comienza recopilando las historias de usuarios, las que se asemejan a los tradicionales casos de uso. Una vez obtenidas estas historias de usuarios, los programadores evalúan rápidamente el tiempo de desarrollo de cada una. El objetivo de esta fase es fijar la prioridad de cada una de las historias de usuario y se establece cuál va a ser el contenido de la primera entrega. Los programadores estiman cuánto esfuerzo requiere cada historia de

usuario y se establece el cronograma. La duración del calendario para la primera entrega al cliente no suele superar los dos meses (Darío, 2016).

Según (Darío, 2016), los conceptos básicos de la planificación son:

Las historias de usuarios: descritas por el cliente, en su propio lenguaje, como descripciones cortas de lo que el sistema debe realizar, en el desarrollo ágil, las historias de usuarios es el sustituto liviano a lo que tradicionalmente se ha usado para especificar requisitos.

- ✓ El Plan de Entregas: establece que las historias de usuarios serán agrupadas para conformar una entrega y el orden de las mismas. Este cronograma será el resultado de una reunión entre todos los actores del proyecto.
- ✓ Plan de Iteraciones: las historias de usuarios seleccionadas para cada entrega son desarrolladas y probadas en un ciclo de iteración, de acuerdo al orden preestablecido.

- Diseño

La Metodología XP hace especial énfasis en los diseños simples y claros (Darío, 2016). Los conceptos más importantes de diseño en esta metodología son los siguientes:

- ✓ Simplicidad: un diseño simple se implementa más rápidamente que uno complejo. Por ello XP propone implementar el diseño más simple posible que funcione.
- ✓ Soluciones “Spike”: cuando aparecen problemas técnicos, o cuando es difícil de estimar el tiempo para implementar una historia de usuario, pueden utilizarse pequeños programas de prueba (llamados “Spike”), para explorar diferentes soluciones.
- ✓ Recodificación: consiste en escribir nuevamente parte del código de un programa, sin cambiar su funcionalidad, a los efectos de crearlo más simple, conciso y entendible. Las metodologías de XP sugieren re codificar cada vez que sea necesario.

- Codificación

Uno de los requerimientos de XP es tener al cliente disponible durante todo el proyecto. No solamente como apoyo a los desarrolladores, sino formando parte del grupo. El involucramiento del cliente es fundamental para que pueda desarrollarse un proyecto con la metodología XP. Al comienzo del proyecto, este debe proporcionar las historias de usuarios, pero dado que estas historias son expresamente cortas y de “alto nivel”, no contienen los

detalles necesarios para realizar el desarrollo del código. Estos detalles deben ser proporcionados por el cliente, y discutidos con los desarrolladores, durante la etapa de desarrollo. XP promueve la programación basada en estándares, de manera que sea fácilmente entendible por todo el equipo, y que facilite la recodificación. También brinda una programación dirigida por pruebas ("Test-Driven Programming"); en las metodologías tradicionales, la fase de pruebas, es usualmente realizada sobre el final del proyecto, o el final del desarrollo de cada módulo. La metodología XP propone un modelo inverso, primero se escriben las pruebas que el sistema debe pasar, luego el desarrollo debe ser el mínimo necesario para pasar las pruebas previamente definidas. Las pruebas a los que se refiere esta práctica, son las pruebas unitarias, realizados por los desarrolladores. La definición de estas pruebas al comienzo, condiciona o dirige el desarrollo (Darío, 2016).

- Pruebas

Es en esta última fase donde se verifica si el software cumple con las necesidades por las que fue desarrollado (Darío, 2016), se evalúan algunos de estos aspectos:

- ✓ Pruebas unitarias: todos los módulos deben de pasar las pruebas unitarias antes de ser liberados o publicados; por otra parte, como se mencionó anteriormente, las pruebas deben ser definidas antes de realizar el código ("Test-Driven Programmming"). Que todo código liberado pase correctamente las pruebas unitarias, es lo que habilita que funcione la propiedad colectiva del código.
- ✓ Detección y corrección de errores: cuando se encuentra un error ("Bug"), éste debe ser corregido inmediatamente, y se deben tener precauciones para que errores similares no vuelvan a ocurrir. Asimismo, se generan nuevas pruebas para verificar que el error haya sido resuelto.
- ✓ Pruebas de aceptación: son creadas en base a las historias de usuarios, en cada ciclo de la iteración del desarrollo. El cliente debe especificar uno o diversos escenarios para comprobar que una historia de usuario ha sido correctamente implementada. Asimismo, en caso de que fallen varias pruebas, deben indicar el orden de prioridad de resolución. Una historia de usuario no se puede considerar terminada hasta que pase correctamente todas las pruebas de aceptación.

1.6.1.2 Artefactos de la metodología XP

De acuerdo a Pressman (Pressman), dentro de cada fase de la metodología XP se generan diferentes tipos de artefactos. La tabla que se muestra a continuación define por cada fase de la metodología los artefactos que se generan de las mismas:

Fase de la metodología	**Artefactos que se generan**
Planeación	Historias de usuario, plan de entregas
Diseño	Tarjetas clase-responsabilidad creador (CRC)
Codificación	Estándares de codificación, código
Pruebas	Pruebas unitarias

Tabla 1: Fases y artefactos que se generan en la metodología de desarrollo XP

De la tabla anterior se tienen elementos que no se han definido concretamente tales como historias de usuarios, tarjetas CRC, estándares de codificación y código.

1.6.1.3 Estándares de codificación

Los estándares de codificación o convenciones de código (Hommel, 1999) son importantes para los programadores por un gran número de razones, las convenciones de código mejoran la lectura del software, permitiendo entender un código nuevo mucho más rápido y más a fondo, en caso de que lo lea otro programador, también para que el software desarrollado esté bien estructurado y pueda ser presentado como un buen producto para futuros mantenimientos o modificaciones. Existen numerosos aspectos para mantener un estándar de codificación correcto, tales como:

1. Comentarios de comienzo de cada método o clase, para así especificar cuál es el propósito de esta clase o método.

```
70
71      /* Nombre de la clase u objeto
72      *
73      * Informacion de la version
74      *
75      * Fecha
76      *
77      * Propósito de la clase u objeto
78      * Copyright
79      */
80
81
```

Figura 2: Ejemplo de comentario de método en el código

2. Longitud de la línea: evitar las líneas de más de 80 caracteres, ya que no son manejadas bien por muchas terminales y herramientas.
3. Rompiendo líneas: cuando una expresión no entre en una línea, romperla de acuerdo con estos principios:

- Cambio de línea después de una coma.
- Cambio de línea antes de un operador.
- Alinear la nueva línea con el comienzo de la expresión al mismo nivel de la línea anterior.

Existe otro estándar de codificación muy usado en Java que es el CamelCase (BINKLEY, 2009), CamelCase puede dividirse en dos tipos:

Upper Camel Case: Cuando la primera letra de cada una de la palabra es mayúscula. También es denominado Pascal Case; un ejemplo es: EjemploDeNomenclatura.

Lower Camel Case: Igual que la anterior o que con la excepción de que la primera letra es en minúscula; un ejemplo es: ejemploDeNomenclatura.

En la presente investigación se utilizará el Lower Camel Case, como preferencia del programador de la aplicación que resolverá el problema principal.

1.6.2 Metodología Scrum

Otra de las metodologías existentes es Scrum (Molina Romero, y otros, 2015) que, es una forma ágil de gestionar un proyecto. Realmente se puede definir como un conjunto de buenas prácticas para trabajar colaborativamente en equipos altamente productivos. Se basa en

equipos multifuncionales y auto-organizados donde no existe un líder global que decide qué persona, qué tarea o cómo se resolverá un problema.

Scrum (Molina Romero, y otros, 2015) está estructurada por varios aspectos tales como:

- Equipo Scrum: los equipos Scrum entregan productos de forma iterativa e incremental, maximizando las oportunidades de obtener retroalimentación. Las entregas incrementales de un producto terminado aseguran que siempre estará disponible una versión potencialmente útil y funcional de este, consiste en:
    1. Dueño de producto: es quien define los objetivos, planifica el proyecto, crea y mantiene la lista de requisitos priorizados que son necesarios para cumplir los objetivos, establece calendario de entregas;
    2. Equipo de desarrollo: es un equipo de desarrollo auto-organizado y multifuncional que, como equipo, cuenta con todas las habilidades necesarias para crear un incremento del producto.
    3. El facilitador: es el responsable de que el quipo trabaje ajustándose a las reglas de Scrum, es un líder que está al servicio del equipo y ayuda a resolver problemas o impedimentos que el equipo tiene para que se logre el objetivo de cada iteración y poder finalizar el proyecto con éxito.
- El sprint: es considerado como el corazón de Scrum. Es un bloque de tiempo de un mes o menos de duración durante el cual se crea in incremento del producto terminado.
- Reunión de planificación de Sprint: es donde se planifica el trabajo a realizar durante el sprint.
- Scrum diario: es una reunión diaria de 15 minutos que tiene el objetivo de facilitar la transferencia de la información, la colaboración entre los miembros.
- Revisión de sprint: se trata de una reunión informal donde el equipo presenta al cliente los requisitos completados en el Sprint, es aquí donde el cliente puede ver de manera objetiva como han sido desarrollados los requisitos que proporcionó y ver si cumplen sus expectativas.

1.6.3 Metodología RUP

También dentro de las metodologías prescriptivas está RUP (Molina Romero, y otros, 2015), que asegura la producción de software de alta calidad dentro de unos tiempos y presupuestos predecibles. RUP promueve la productividad del trabajo en equipo proporcionando a cada miembro del equipo un fácil acceso a una base de conocimiento con una serie de directrices, plantillas y herramientas para actividades de desarrollo críticas. Es una metodología extensa y con cierto grado de complejidad.

RUP es una metodología de desarrollo de software formal (Cicilia, y otros, 2009), orientadas a objetos, con un ciclo de vida espiral. Este proceso de desarrollo de software utiliza el lenguaje unificado de modelado UML, y constituye una de las mejores y más utilizadas para el análisis, implementación y documentación de sistemas orientados a objetos. Pertenece a un conjunto de metodologías adaptables al contexto y necesidades de cada organización. Las principales características de RUP abordadas por (Cicilia, y otros, 2009) son:

- Dirigido por casos de uso: los casos de uso son una técnica que se utiliza para la captura de requisitos por parte de los clientes/usuarios, se define un caso de uso como un fragmento de funcionalidad del sistema que proporciona al usuario un valor añadido, los casos de uso representan los requisitos funcionales del sistema
- Centrado en arquitectura: La arquitectura es la organización o estructura de todas las partes más relevantes del sistema, la arquitectura juega un papel muy importante en el desarrollo de software ya que nos permite tener una visión común entre todos los involucrados en el proceso. La Arquitectura en RUP ocupa un papel muy importante, el establecimiento temprano de una buena arquitectura que haga frente a cualquier cambio posterior durante la construcción y el mantenimiento.
- Iterativo e incremental: RUP apuesta por procesos interactivos e incrementales en donde el trabajo se divide en partes más pequeñas o mini proyectos permitiendo el equilibrio entre casos de uso y arquitectura.

En la actualidad el 39% de los proyectos de desarrollo de software logran completarse correctamente (Molina Romero, y otros, 2015), una de las causas de fracaso en estos proyectos es la falta de una metodología efectiva en los mismos, lo que provoca un mal

seguimiento en los avances, mala gestión del crecimiento del software y facilita la aparición de *bugs* y de fallos, al no seguir una metodología clara, los responsables no son capaces de realizar una estimación efectiva de la duración y riesgos del proyecto ya que en la mayoría de los caso no se lleva a cabo planificación y documentación alguna.

De las metodologías mencionadas anteriormente ninguna se adapta concreta y totalmente a las necesidades de esta investigación puesto que, XP plantea la programación por pares y la incorporación del cliente al equipo de desarrollo, aspectos que no se cumplen porque en este caso el equipo de desarrollo cuenta solamente con un programador. RUP está diseñada para equipos de desarrollo grandes donde existe colaboración y retroalimentación entre estos, aspectos que nuevamente no se adaptan a esta investigación. Scrum aborda también sobre la existencia de más de una persona en el equipo de desarrollo existiendo en este caso un cliente que es quien define los aspectos esenciales del proceso de desarrollo de software. Como consecuencia de esto se selecciona de estas metodologías aspectos y artefactos como el modelado de clases, levantamiento de requisitos, que favorezcan esta investigación no quedando definida una metodología de desarrollo de software específica, pero sí un proceso de desarrollo de software que posibilite dar solución al problema en esta investigación.

1.6.4 Proceso de desarrollo de software

Estando definidos el tipo de software y requisitos de alto nivel, el proceso de desarrollo de software en esta investigación queda reflejado en los siguientes pasos:

1. Derivar requisitos funcionales y no funcionales a partir de los requisitos de alto nivel.
2. Modelado de clases de la aplicación.
3. Definición de una línea base tecnológica.
4. Implementación del modelo de clases.

Conclusiones parciales

En este capítulo se definió el marco conceptual de referencia para soportar los fundamentos teóricos de la investigación que permitió definir conceptos esenciales, en este caso están: desempeño computacional y lo que representa a la hora de la elección de uno de los algoritmos de extracción, tipos de software donde quedó reflejado que el tipo de software donde se enmarca la solución es el de aplicación, requisitos de alto nivel, proceso de desarrollo de

software quedando en este definido que no se utilizará una metodología en específico, sino que se utilizará algunos aspectos de estas mencionados anteriormente.

Capítulo 2

En el capítulo anterior se definió y enunció la definición de requisito de alto nivel por el cual se regirá la presente investigación, puesto que es unos de los pilares fundamentales en la elaboración y desarrollo del software que dará solución al problema a resolver. Como consecuencia de esto en este capítulo se generan los entregables presentados en el capítulo anterior a partir de los requisitos funcionales y no funcionales.

2.1 Requisitos funcionales

Como se anuncia en (Plazas, 2014), los requisitos funcionales describen las funciones que el software debe ejecutar, a veces se conocen como capacidades. Los requisitos funcionales se los puede dividir en:

- ✓ De usuario: Los requerimientos de usuario; son declaraciones, en lenguaje natural y en diagramas, de los servicios que se espera que el sistema provea y de las restricciones bajos las cuales se debe operar.
- ✓ De sistema: Establecen con detalle los servicios y restricciones del sistema

El proceso de captura de requisitos es una etapa de suma importancia dentro del proceso de desarrollo de software. Éste se encarga de analizar las necesidades del cliente, describirlas en detalle para conformar el sistema de la manera más precisa, cumpliendo siempre con las especificaciones deseadas. En el presente trabajo, se identificaron los siguientes requisitos funcionales a través de entrevistas realizadas al cliente:

1. El sistema debe ser capaz de aplicar la misma entrada a los algoritmos Eclat y FP-GROWTH.
2. El sistema debe ser capaz de computar las reglas de asociación utilizando el algoritmo Eclat.
3. El sistema debe ser capaz de computar las reglas de asociación utilizando el algoritmo FP-GROWTH.
4. El sistema debe ser capaz de medir el tiempo de ejecución del algoritmo Eclat.
5. El sistema debe ser capaz de medir el tiempo de ejecución del algoritmo FP-GROWTH.
6. EL sistema debe ser capaz de determinar uso de memoria del algoritmo Eclat.
7. EL sistema debe ser capaz de determinar uso de memoria del algoritmo FP-GROWTH.

8. EL sistema debe ser capaz de determinar la cantidad de elementos frecuentes del algoritmo Eclat.
9. EL sistema debe ser capaz de determinar la cantidad de elementos frecuentes del algoritmo FP-GROWTH.

2.2 Requisitos no funcionales

Estos requisitos incluyen áreas tales como el rendimiento, el diseño y las limitaciones de la aplicación; son los que actúan para limitar la solución, se los conoce como restricciones o requisitos de calidad. Además, los requisitos no funcionales pueden estar relacionados con propiedades emergentes del sistema, describen restricciones externas del sistema; definen las cualidades globales que el sistema ha de exhibir y son más críticos que los requisitos funcionales. Los requisitos no funcionales son propiedades que el producto debe tener lo que no puede ser evidente al usuario, incluyendo atributos de calidad (Plazas, 2014).

Los requisitos no funcionales (Hernández Martínez, 2015), pueden clasificarse en dos categorías fundamentales:

- Cualidades de ejecución, como la seguridad y facilidad de uso, que son observables en tiempo de ejecución.
- Cualidades de evolución, como la capacidad de prueba, mantenimiento, ampliación y escalabilidad, que se enfoca en la estructura estática del sistema.

Para el siguiente trabajo, atendiendo a lo anterior, se especifican los siguientes requisitos no funcionales:

Número	Categoría	Descripción
1	Hardware	Los requisitos mínimos para el sistema de alojamiento serán los siguientes: 4GB de memoria, microprocesador con 3 GHz.
2	Software	El sistema debe contar con la máquina virtual de java (Java 8) instalada

Tabla 2: Requisitos no funcionales

2.3 Modelado de clases de la aplicación

El modelado de clases en la presente investigación se presenta exponiendo dos ejemplos de las clases principales de la forma siguiente:

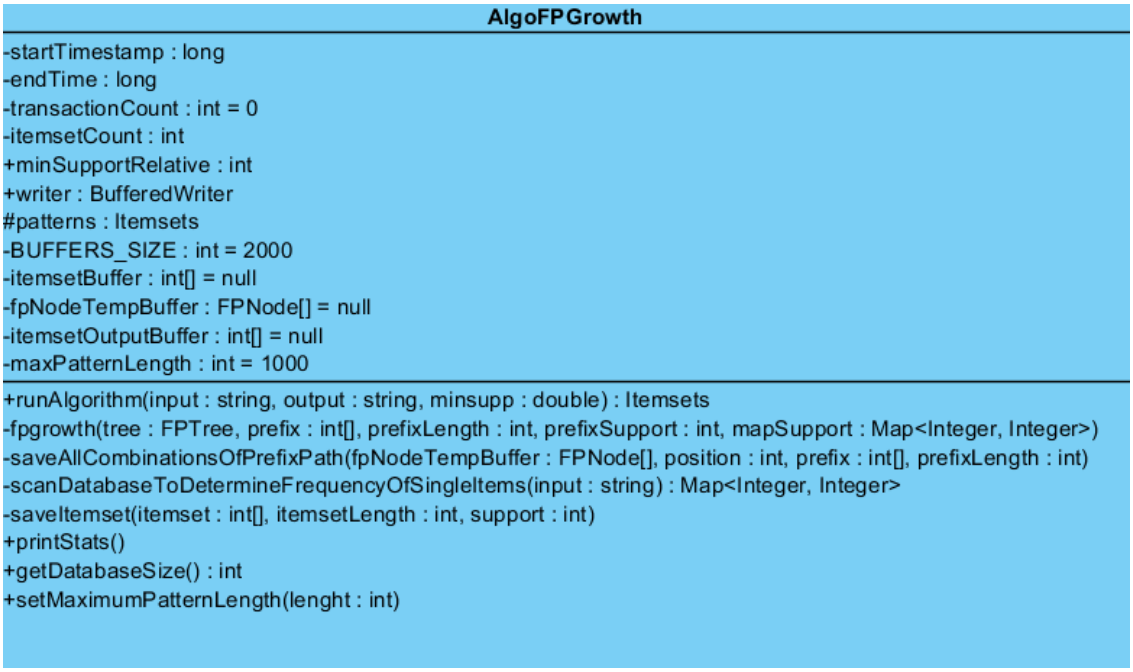


Figura 3: Modelado de clase algoritmo FP-GROWTH

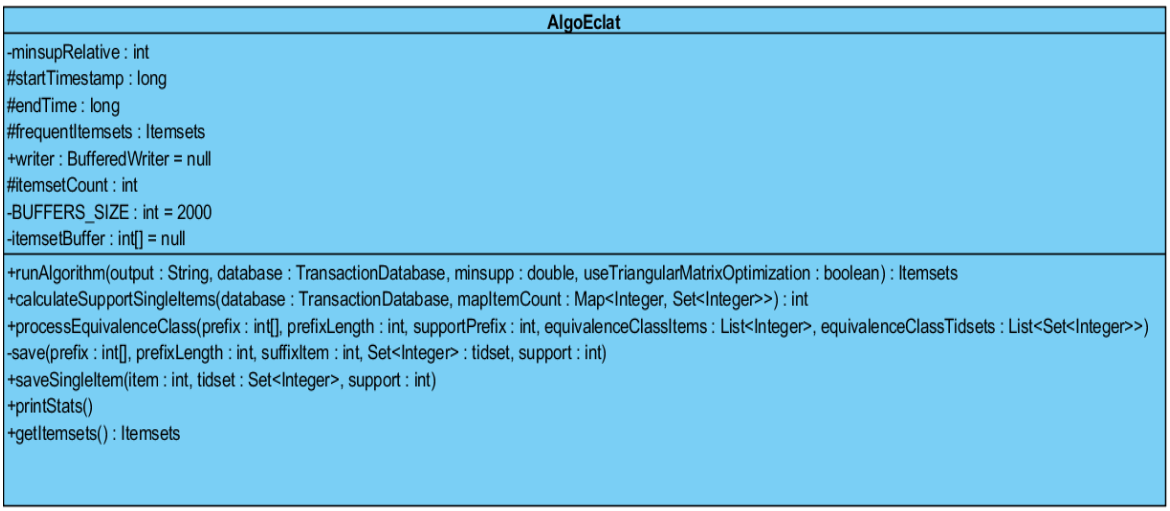


Figura 4: Modelado de clase algoritmo Eclat

A continuación, se presentará el modelo de relación de clases, donde se evidencia las conexiones entre las clases a implementar en la investigación:

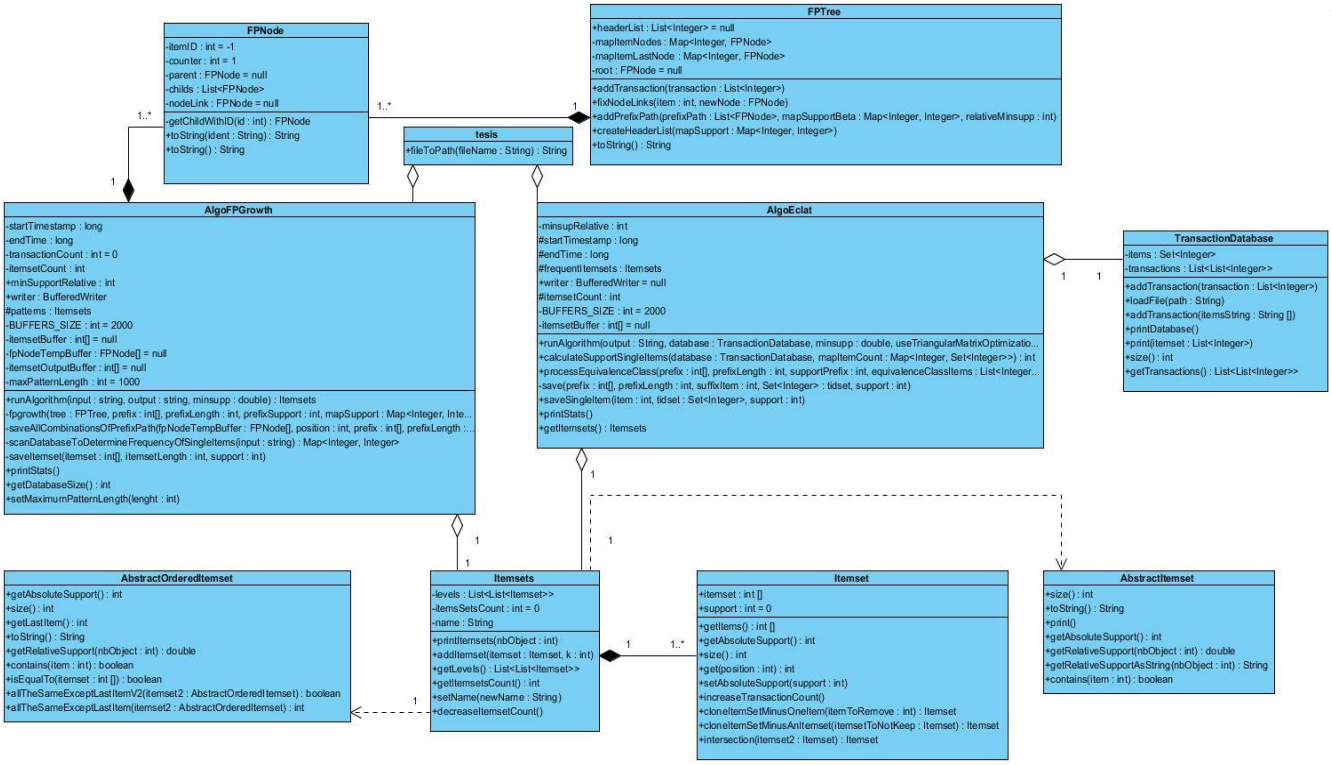


Figura 5: Modelo de relación de clases de la aplicación

2.4 Línea base tecnológica

En la presente investigación se propone definir una línea base tecnológica para lograr un correcto proceso de desarrollo de software. Como se mencionó anteriormente con las actividades se generan respectivamente entregables mostrados en la tabla siguiente:

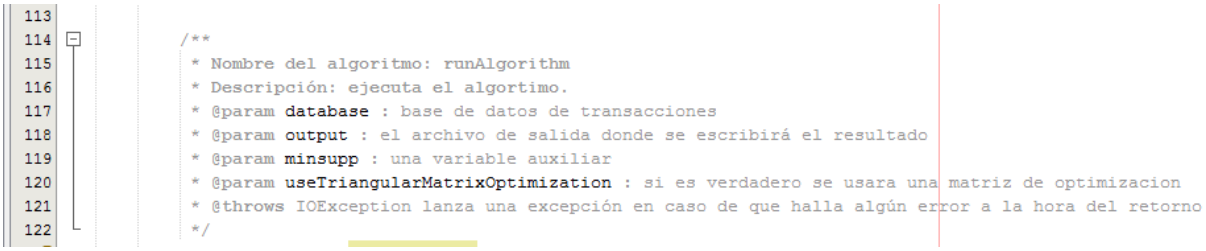
Actividades	**Artefacto que se genera**
Estándares de codificación	Código
Lenguaje de programación	Definición del lenguaje de programación
IDE de desarrollo	Definición del entorno de desarrollo

Tabla 3: Actividades y artefactos de la línea base tecnológica

2.5 Estándares de codificación

A medida que se va desarrollando la aplicación se van cumpliendo los estándares de codificación definidos usando el estándar de codificación CamelCase en el capítulo anterior:

- Comentarios de comienzo de cada método o clase, para así especificar cuál es el propósito de esta clase o método.



```
113
114         /**
115          * Nombre del algoritmo: runAlgorithm
116          * Descripción: ejecuta el algortimo.
117          * @param database : base de datos de transacciones
118          * @param output : el archivo de salida donde se escribirá el resultado
119          * @param minsupp : una variable auxiliar
120          * @param useTriangularMatrixOptimization : si es verdadero se usara una matriz de optimizacion
121          * @throws IOException lanza una excepción en caso de que halla algún error a la hora del retorno
122          */
```

Figura 6: Estándar de codificación: Descripción de clase

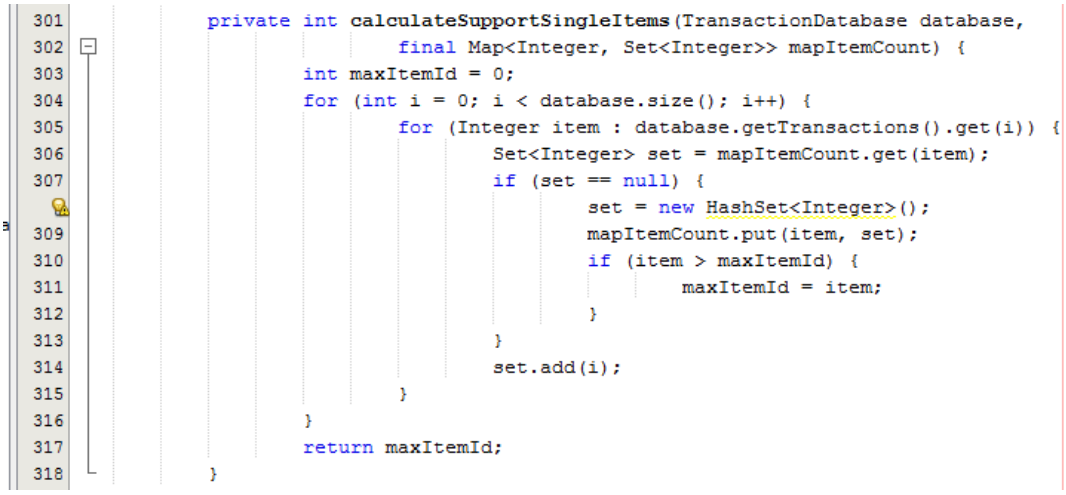
Longitud de la línea: evitar las líneas de más de 80 caracteres, ya que no son manejadas bien por muchas terminales y herramientas: todo el código desarrollado ha cumplido con esta característica.

- Rompiendo líneas: cuando una expresión no entre en una línea, romperla de acuerdo con estos principios:

➢ Cambio de línea después de una coma.

➢ Cambio de línea antes de un operador.

➢ Alinear la nueva línea con el comienzo de la expresión al mismo nivel de la línea anterior.



```
301         private int calculateSupportSingleItems(TransactionDatabase database,
302                 final Map<Integer, Set<Integer>> mapItemCount) {
303             int maxItemId = 0;
304             for (int i = 0; i < database.size(); i++) {
305                 for (Integer item : database.getTransactions().get(i)) {
306                     Set<Integer> set = mapItemCount.get(item);
307                     if (set == null) {
309                         set = new HashSet<Integer>();
310                         mapItemCount.put(item, set);
311                         if (item > maxItemId) {
312                             maxItemId = item;
313                         }
314                     }
315                     set.add(i);
316                 }
317             }
318             return maxItemId;
        }
```

Figura 7: Ejemplo del estándar “rompiendo líneas” en el código

Lower Camel Case

A continuación, se muestra cómo se evidencia en el código el uso del estándar de codificación Camel Case, en este caso el uso de Lower Camel Case.

```
32      private int minsupRelative;
        private double supp;
34      protected TransactionDatabase database;
35      protected long startTimestamp;
36      protected long endTime;
37      protected Itemsets frequentItemsets;
38      BufferedWriter writer = null;
39      protected int itemsetCount;
40      private int[] itemsetBuffer = null;
41      boolean showTransactionIdentifiers = false;
```

Figura 8: Ejemplo del estándar de codificación Camel Case

2.6 Lenguaje de programación

Un lenguaje de programación, es aquel elemento dentro de la informática que permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que se ponen a disposición del programador para que este pueda comunicarse con los dispositivos hardware y software existentes. Representan en forma simbólica y en manera de un texto los códigos que podrán ser leídos por una persona, son independientes de las computadoras a utilizar (Fernández, 2002).

Java es un lenguaje de programación y multiplataforma, lo que quiere decir que se ejecuta en la mayoría de los sistemas operativos, inclusive en sistemas operativos de móviles. Es un software de distribución libre, no es necesario pagar una licencia para poder desarrollar en él. Así mismo es un lenguaje muy completo y poderoso, se pueden realizar muchas tareas con el mismo, pues posee bibliotecas y utilidades muy completas que facilitan la programación. Entre sus principales características se incluyen: simple, orientado a objetos, multihilo, dinámico, arquitectura neutral, interpretado, portable, alto rendimiento, robusto, seguro (Jorge, 2010).

Para el desarrollo de la aplicación se eligió el lenguaje de programación Java, ya que es una tecnología libre, por lo que no es necesario pagar licencia para poder utilizarlo, posee versatilidad al momento de escribir código, sencillez en la sintaxis, e inclusive su seguridad.

2.7 IDE de desarrollo

NetBeans es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo y ofrece ventajas para la corrección de errores. Todo esto justifica su utilización. NetBeans IDE es un producto libre y gratuito sin restricciones de uso, por las razones antes mencionadas ambos se utilizan en esta investigación (Trujillo Gálvez, 2016).

Se decide utilizar NetBeans 8.2 debido a que ofrece opciones de desarrollo importantes, así como completamiento de código. Es una herramienta fácil de instalar y consume pocos recursos. El editor de código fuente que presenta es muy ágil y a la vez robusto, características que hacen que sea una excelente herramienta para el desarrollo de soluciones informáticas.

2.8 Implementación del modelo de clases

Para poder lograr una correcta implementación del modelo de clases es necesario profundizar acerca de conceptos básicos de los algoritmos que darán solución a la presente investigación, mencionando primeramente el algoritmo para extracción de reglas de asociación Apriori por su importancia en el desarrollo de otros algoritmos y modificaciones de estos.

2.8.1 Algoritmo Apriori

La implementación Apriori se basa en una representación en árbol de prefijos de los contadores necesarios y utiliza un esquema doblemente recursivo para contar las transacciones, permite encontrar de forma eficiente *conjuntos de ítems frecuentes*, los cuales sirven de base para generar reglas de asociación. Procede identificando los ítems individuales frecuentes en la base y extendiéndolos a conjuntos de mayor tamaño siempre y cuando esos conjuntos de datos aparezcan suficientemente seguidos en dicha base de datos. Este algoritmo se ha aplicado grandemente en el análisis de transacciones comerciales y en problemas de predicción (Wu, 2008). La estrategia que este algoritmo sigue es del tipo descendente a lo ancho, siendo su concepción general la base para muchos de los algoritmos desarrollados posteriormente. De forma general, la mayoría de los algoritmos tipo Apriori primero construyen un conjunto de itemsets candidatos basados en alguna heurística y, posteriormente, determinan el subconjunto que realmente contiene los itemsets frecuentes (Medina, y otros, 2007).

En la siguiente figura se muestra el pseudocódigo del algoritmo a priori donde el procedimiento ” apriori_gen(L_{k-1})” se corresponde al conjunto de los k-itemsets candidatos.

```
Algoritmo: Apriori
 1)  L_1 = {large 1-itemsets};
 2)  for ( k = 2; L_k-1 ≠ ∅; k++ ) do begin
 3)    C_k = apriori_gen(L_k-1); // New candidates
 4)    forall transactions t ∈ D do begin
 5)      C_t = subset(C_k, t);      // Candidates contained in t
 6)      forall candidates c ∈ C_t do
 7)        c.Support ++ ;
 8)    end
 9)    L_k = {c ∈ C_k | c.Support ≥ minsup};
10)  end
11)  Answer = ⋃_k L_k;
```

Figura 9: Pseudocódigo del algoritmo Apriori

2.8.2 Algoritmo Eclat

La implementación de Eclat (*Equivalence Class Transformation*) usa matrices de bits dispersas para representar listas de transacciones y filtrar conjuntos de elementos cerrados y máximos, básicamente, hay dos formas en que una matriz de bits se puede representar: ya sea como una matriz de bits, con un bit de memoria para cada elemento, o usar una lista identificadores de transacciones para cada ítem, la selección de una de estas depende de la densidad del conjunto de datos (Zaki, 2000). El algoritmo Eclat desciende recursivamente por cada clase de equivalencia encontrada (Zaki, 1997). A continuación, se muestra el pseudocódigo del algoritmo Eclat.

```
Algoritmo: Eclat
1)  L_1 = { large 1-itemsets };
2)  F = L_1;
3)  Bottom-up(L_1, F);
4)  Answer = F;
```

Figura 10: Pseudocódigo del algoritmo Eclat

```
Procedimiento: Bottom-up
Entrada: F_k - Conjunto de k-itemsets de una
               clase de equivalencia
Entrada/Salida: F - Frequent l-itemsets, l > k
1)   forall itemset I_i ∈ F_k do begin
2)     F_{k+1} = ∅;    // F_{k+1} will be E(I_i)
3)     forall itemset I_j ∈ F_k and i < j do begin
4)       N = Extend(I_i, I_j);
5)       if N.Support ≥ minsup then
6)         F_{k+1} = F_{k+1} ∪ {N};
7)     end
8)     if F_{k+1} ≠ ∅ then begin
9)       Bottom-up(F_{k+1}, F);
10)      F = F ∪ F_{k+1};
11)    end
12)  end
```

Figura 11: Pseudocódigo del procedimiento Bottom-up

El procedimiento Bottom-up es recursivo, teniendo como entrada cada clase de equivalencia que se va generando y como salida los itemsets frecuentes encontrados.

2.8.3 Algoritmo FP-GROWTH

En (Martín, 2012) se aborda que FP-GROWTH está basado en una representación de árbol de prefijos de una base de datos de transacciones llamado FP-Tree Frequent Pattern Tree. La idea básica del algoritmo FP-GROWTH (Frequent Pattern Growth) puede ser descrita como un esquema de eliminación recursiva: en un primer paso de pre procesamiento se borran todos los ítems de las transacciones que no son frecuentes individualmente o no aparecen en el mínimo soporte de transacciones, luego se seleccionan todas las transacciones que contienen al menos un ítem frecuente, se realiza esto de manera recursiva hasta obtener una base de datos reducida. Al volver, se borran los ítems procesados de la base datos de transacciones en la memoria y se empieza otra vez, y así con el siguiente ítem frecuente. Los ítems en cada transacción son almacenados y luego se ordena descendentemente su frecuencia en la base de datos.

Un Frequent Pattern Tree (FP-Tree) es una estructura de datos de tipo árbol, que consiste en un nodo raíz que tiene como sus hijos a sub-árboles de prefijos de ítems. Cada nodo del sub-árbol de prefijo de ítem contiene tres campos: el nombre del ítem al cual el nodo representa,

un contador que registra el número de transacciones que satisfacen la rama del árbol que va de la raíz hasta este nodo, y un puntero al siguiente nodo del FP-Tree que contenga el mismo nombre de ítem o un puntero vacío si no existe tal nodo. A su vez, el FP-Tree posee una estructura de datos auxiliar denominada tabla de encabezados. Cada entrada en esta tabla posee dos campos. El primero es el nombre del ítem y el segundo es un puntero al primer nodo del FP-Tree que posee el mismo nombre de ítem (CASTILLO, y otros, 2015).

Después de que se han borrado todos los ítems infrecuentes de la base de datos de transacciones, se pasa al árbol FP. En un árbol FP cada camino representa el grupo de transacciones que comparten el mismo prefijo, cada nodo corresponde a un ítem y a su vez los nodos en el árbol son reordenados de forma tal que los de mayor soporte tengan más probabilidad de compartir nodos que otros con menos soporte (Medina, y otros, 2007). Todos los nodos que referencian al mismo ítem se ponen juntos en una lista, de modo que todas las transacciones que contienen un ítem específico pueden encontrarse fácilmente y contarse en la lista. Esta lista puede ser indexada a través de la cabeza, lo cual también expone el número total de ocurrencias del ítem en la base de datos.

A continuación, se muestra el pseudocódigo del algoritmo FP-GROWTH (Kuldeep Malik, 2011) donde se presenta el funcionamiento de este:

```
Input: constructed FP-tree
Output: complete set of frequent patterns
Method: Call FP-growth (FP-tree, null).
procedure FP-growth (Tree, α)
{
        1)      if Tree contains a single path P then
        2)      for each combination do generate pattern β ∪ α with support = minimum support
                of nodes in β.
        3)      Else For each header ai in the header of Tree do {
        4)      Generate pattern β = ai∪α with support = ai.support;
        5)      Construct β.s conditional pattern base and then β.s conditional FP-tree Tree β
        6)      If Tree β = null
        7)      Then call FP-growth (Tree β, β)}
}
```

Figura 12: Pseudocódigo del algoritmo FP-GROWTH

A continuación antes de realizar la validación de la investigación se presentará en la siguiente tabla una comparación de los algoritmos encontrada en (K.Vani, 2015) donde se muestran los factores por los cuales se realiza la comparación y las características respecto a estos en los algoritmos Eclat y FP-GROWTH:

Factor	**Algoritmo Eclat**	**Algoritmo FP-GROWTH**
Estructura de datos	Basado en arreglos	Basado en árboles
Técnica que utiliza	Usa intersección de listas de id de transacciones para generar los itemsets candidatos	Construye un árbol de patrones frecuentes y que satisfacen soporte mínimo
Uso de memoria	Poca memoria	Debido a su estructura compacta y sin generación de candidatos requiere menos memoria
Números de escaneos	Continuamente escanea y actualiza la base de datos	Escanea la base de datos dos veces
Tiempo de ejecución	bajo	Bajo
Base de datos	Adecuado para medio y denso conjuntos de datos pero no adecuados para pequeños conjuntos de datos	Adecuado para grandes y medianos conjuntos de datos
Precisión	Preciso	Preciso
Aplicaciones	Mejor para itemsets libres	Gran conjunto de elementos

Tabla 4: Comparación entre algoritmo Eclat y FP-Growth

Conclusiones parciales

En este capítulo se llevó a cabo el proceso del desarrollo de software, evidenciando cada uno de los entregables generados por fase del desarrollo tales como: requisitos funcionales y no funcionales, los diagramas de clases de los algoritmos de extracción de reglas de asociación Eclat y FP-GROWTH, se definieron las actividades de la línea base tecnológica, así como el lenguaje de programación, IDE de desarrollo y la generación del código de las clases modeladas; aspectos que habían sido definidos en el capítulo anterior, posibilitando un correcto desarrollo de la aplicación que dará solución al problema a resolver en esta investigación.

Capítulo 3

Validación de la solución

En este capítulo se lleva a cabo la validación de la investigación a través de casos de estudio por los cuales finalmente se compararán los resultados arrojados de los algoritmos de extracción de reglas de asociación Eclat y FP-GROWTH usando distintos valores de soporte.

A cada caso de estudio está asociado un dataset de transacciones, estos dataset fueron seleccionados de la UCI (University of California Irvine) (Dheeru, 2017), puesto que esta universidad tiene numerosos volúmenes de datos para ser trabajados en diferentes investigaciones.

Muchos de los algoritmos de extracción de reglas de asociación necesitan un espacio de atributos discretos, lo que hace imprescindible la aplicación de algún método de discretización que disminuya la cardinalidad de los valores que los atributos continuos de los dataset a utilizar puedan tomar.

3.1 Análisis de los casos de estudio

Con motivo de realizar la validación de la propuesta de solución se seleccionaron tres datasets de la Universidad Irvin de California, datos públicos en internet. Se describen las especificaciones de estos datasets a continuación:

3.1.1 Dataset Balance Scale

Este dataset (Dheeru, 2017), fue generado para modelar resultados experimentales de psicología. Cada ejemplo se clasifica teniendo la punta de la balanza hacia la izquierda, a la derecha o balanceada. La manera correcta de encontrar la clase es la mayor entre (distancia_izquierda*peso_izquierdo) y (distancia_derecha*peso_derecho). Si son iguales están balanceados. El dataset está conformado por 625 transacciones, cuatro atributos que corresponden a:

Nombre de la clase: 3 (L, B, R)

Peso izquierdo: 5 (1, 2, 3, 4, 5)

Distancia izquierda: 5 (1, 2, 3, 4, 5)

Peso derecho: 5 (1, 2, 3, 4, 5)

Distancia derecha: 5 (1, 2, 3, 4, 5)

3.1.2 Dataset Zoo

Este dataset (Dheeru, 2017), es una base de datos simple que contiene 17 atributos con valores booleanos y un total de 101 instancias. El atributo "tipo" es el atributo de clase. A continuación, se muestra un desglose la descripción de los atributos:

Nombre del atributo	**Descripción del atributo**
Nombre del animal	Único para cada instancia
Pelo	Booleano
Plumas	Booleano
Huevos	Booleano
Leche	Booleano
aerotransportado	Booleano
Acuático	Booleano
Depredador	Booleano
Dentado	Booleano
Backbone	Booleano
Respira	Booleano
Venenoso	Booleano
Aletas	Booleano
Patas	{0,2,4,5,6,8}
Cola	Booleano
Doméstico	Booleano

Catsize	Booleano
Tipo	$x \epsilon \mathbb{R}; 1 \leq x \leq 7$

Tabla 5: Descripción de atributos del dataset Zoo

3.1.3 Dataset Car

La base de datos de evaluación de automóviles (Dheeru, 2017), se derivó de un modelo de decisión jerárquica simple desarrollado originalmente para la demostración realizada en (M. Bohanec, 1990). Este dataset puede ser particularmente útil para probar métodos de inducción constructiva y descubrimiento de estructura. Contiene un total de 1728 transacciones y 6 atributos, a continuación, se presenta la información de los atributos en la siguiente tabla donde se representa el nombre de los atributos y sus posibles valores:

Nombre de atributo	Valores
Compra	Muy alta, alta, media, baja
Mantenimiento	Muy alto, alto, medio, bajo
Puertas	2, 3, 4, 5
Personas	2, 4
Arranque	Alto, medio, bajo
Seguridad	Baja, media, alta

Tabla 6: Descripción de atributos del dataset Car

3.2 Transformación de los casos de estudio

Como se mencionó anteriormente, constituye un detractor ejecutar los algoritmos de extracción de reglas de asociación con entradas donde existan valores de clases con caracteres alfanuméricos, es por ello que es necesario realizar una transformación a los dataset como parte del proceso de extracción de conocimiento que prepare los datos para ser procesados. Cada uno de ellos fue convertido a transacciones y se eliminaron las transacciones con algún valor ausente. Los valores numéricos fueron discretizados a partir de la aplicación de un método de discretización utilizando 4 particiones de igual frecuencia (Data Mining Discretization Methods and Performances, 2014). Esta transformación constituye básicamente

en asignar un índice a cada una de las combinaciones posibles de clases para cada atributo quedando de la siguiente manera para el dataset Balance Scale:

Atributo	**Valores**	**Índice correspondiente**
Nombre de la clase	L,B,R	1,2,3
Peso izquierdo	1,2,3,4,5	4,5,6,7,8
Distancia izquierda	1,2,3,4,5	9,10,11,12,13
Peso derecho	1,2,3,4,5	14,15,16,17,18
Distancia derecha	1,2,3,4,5	19,20,21,22,23

Tabla 7: Transformación del dataset Balance Scale

3.3 Análisis y presentación de los resultados

Luego de haber terminado la fase de preparación a cada uno de los datasets, los datos de entrada para los algoritmos de extracción de reglas de asociación Eclat y FP-GROWTH están listos para ser procesados, la ejecución de estos algoritmos se realizó en un ordenador con las características siguientes:

Característica	**Valor**
Memoria RAM	12 GB RAM DDR4 1600GHz
Procesador	Intel(R) Core(Tm) i7-6700HQ CPU @ 2.60GHz

Tabla 8: Transformación del dataset Balance Scale

Se decide definir tres escenarios de prueba donde cada uno corresponde a la ejecución de los algoritmos sobre cada dataset con distintos valores de soporte para analizar su comportamiento reflejado en los valores de los indicadores: tiempo de ejecución, uso de memoria.

El valor de los soportes se decide que sean en un intervalo de 0.01 a 0.05 y no otro mayor puesto que los datos en los dataset se encuentran muy dispersos, los datos de ejecución se muestran respectivamente en las siguientes tablas, donde se muestran los indicadores de comparación y los valores que se arrojan con los algoritmos:

Indicadores	**Algoritmo Eclat**	**Algoritmo FP-GROWTH**
Uso de memoria	5.767890930175781 mb	7.6879119873046875 mb
Tiempo de ejecución	23 ms	21 ms
Valor de soporte	0.01	0.01

Tabla 9: Ejecución de los algoritmos con valor de soporte 0.01 en dataset Balance Scale

Indicadores	**Algoritmo Eclat**	**Algoritmo FP-GROWTH**
Uso de memoria	5.23786041259765 mb	6.727912902832031 mb
Tiempo de ejecución	21 ms	18 ms
Valor de soporte	0.03	0.03

Tabla 10: Ejecución de los algoritmos con valor de soporte 0.03 en dataset Balance Scale

Indicadores	**Algoritmo Eclat**	**Algoritmo FP-GROWTH**
Uso de memoria	3.8478851318359375 mb	4.807884216308594 mb
Tiempo de ejecución	17 ms	16 ms
Valor de soporte	0.05	0.05

Tabla 11: Ejecución de los algoritmos para valor de soporte 0.05 en dataset Balance Scale

Indicadores	**Algoritmo Eclat**	**Algoritmo FP-GROWTH**
Uso de memoria	2.8100582885742188 mb	3.8401565551757812 mb
Tiempo de ejecución	13 ms	9 ms
Valor de soporte	0.01	0.01

Tabla 12: Ejecución de los algoritmos con valor de soporte 0.01 en dataset Zoo

Indicadores	Algoritmo Eclat	Algoritmo FP-GROWTH
Uso de memoria	2.8001116943359375 mb	3.8401107788085938 mb
Tiempo de ejecución	12 ms	8 ms
Valor de soporte	0.03	0.03

Tabla 13: Ejecución de los algoritmos con valor de soporte 0.03 en dataset Zoo

Indicadores	Algoritmo Eclat	Algoritmo FP-GROWTH
Uso de memoria	2.7855165181362125 mb	2.8801116943359375 mb
Tiempo de ejecución	8 ms	6 ms
Valor de soporte	0.05	0.05

Tabla 14: Ejecución de los algoritmos para valor de soporte 0.05 en dataset Zoo

Indicadores	Algoritmo Eclat	Algoritmo FP-GROWTH
Uso de memoria	7.807861328125 mb	9. 72796630859375 mb
Tiempo de ejecución	24 ms	22 ms
Valor de soporte	0.01	0.01

Tabla 15: Ejecución de los algoritmos con valor de soporte 0.01 en dataset Car

Indicadores	Algoritmo Eclat	Algoritmo FP-GROWTH
Uso de memoria	7.25658712614997 mb	8.15685475374881 mb
Tiempo de ejecución	22 ms	19 ms
Valor de soporte	0.03	0.03

Tabla 16: Ejecución de los algoritmos con valor de soporte 0.03 en dataset Car

Indicadores	Algoritmo Eclat	Algoritmo FP-GROWTH
Uso de memoria	5. 5137881848375393 mb	6.515689201177252 mb
Tiempo de ejecución	20 ms	15 ms
Valor de soporte	0.05	0.05

Tabla 17: Ejecución de los algoritmos para valor de soporte 0.05 en dataset Car

A continuación, se muestran en gráficos de línea cómo se comportan los algoritmos en cuanto a uso de memoria y tiempo de ejecución con los distintos datasets, donde en el gráfico de uso de memoria, en el eje X se representan los valores de soporte y en el Y, se representa el uso de memoria para cada valor de soporte;

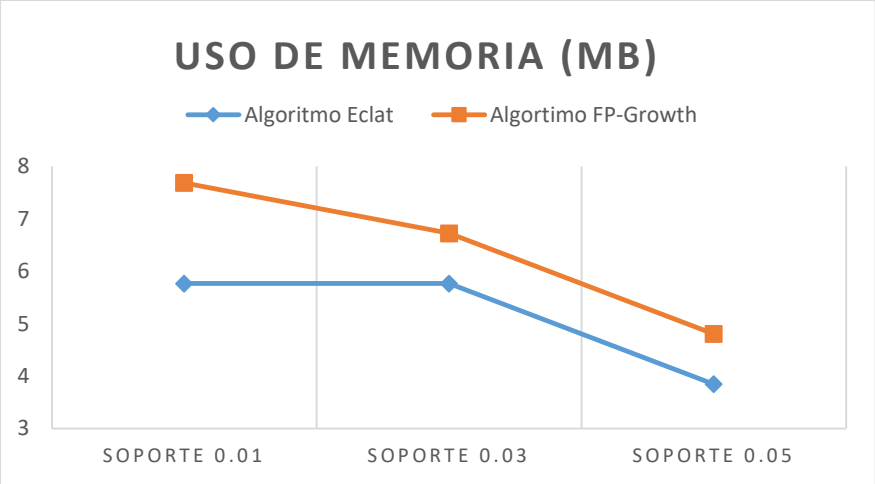


Gráfico 1: Uso de memoria algoritmos Eclat y FP-GROWTH para dataset Balance Scale

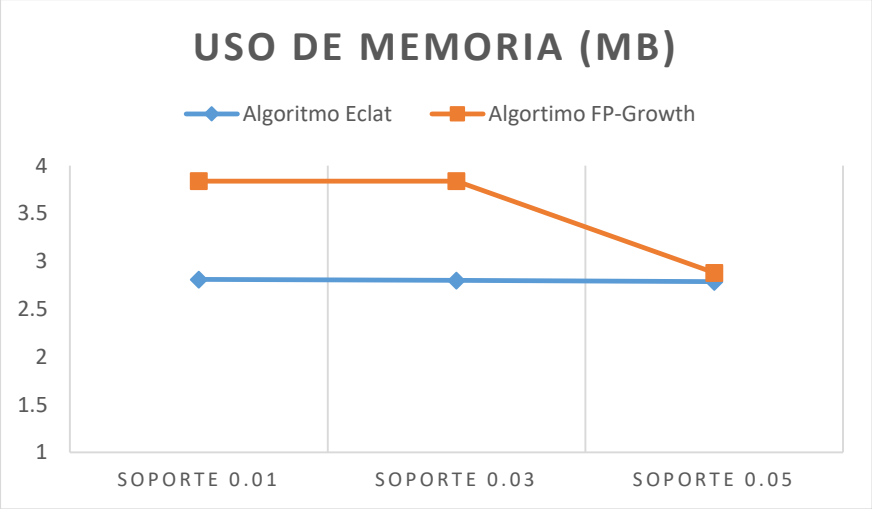


Gráfico 2: Uso de memoria algoritmos Eclat y FP-GROWTH para dataset Zoo

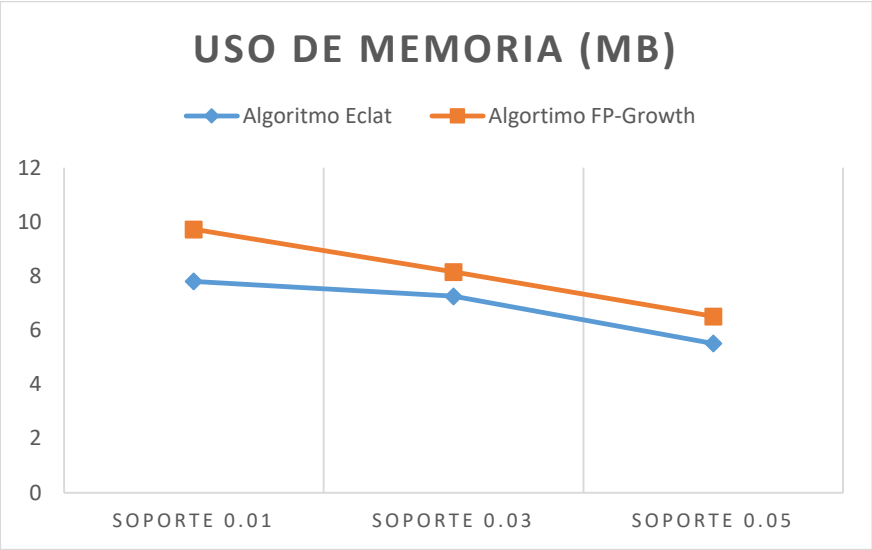


Gráfico 3: Uso de memoria algoritmos Eclat y FP-GROWTH para dataset Car

El análisis de los gráficos 1, 2 y 3 conlleva a que el uso de memoria del algoritmo de extracción de reglas de asociación Eclat se comporta con mejores resultados independientemente del valor de soporte que se utilice.

A continuación, se presentará el gráfico de tiempo de ejecución donde representa en el eje X el valor de los soportes, y en el Y los valores de tiempo de ejecución de los algoritmos:

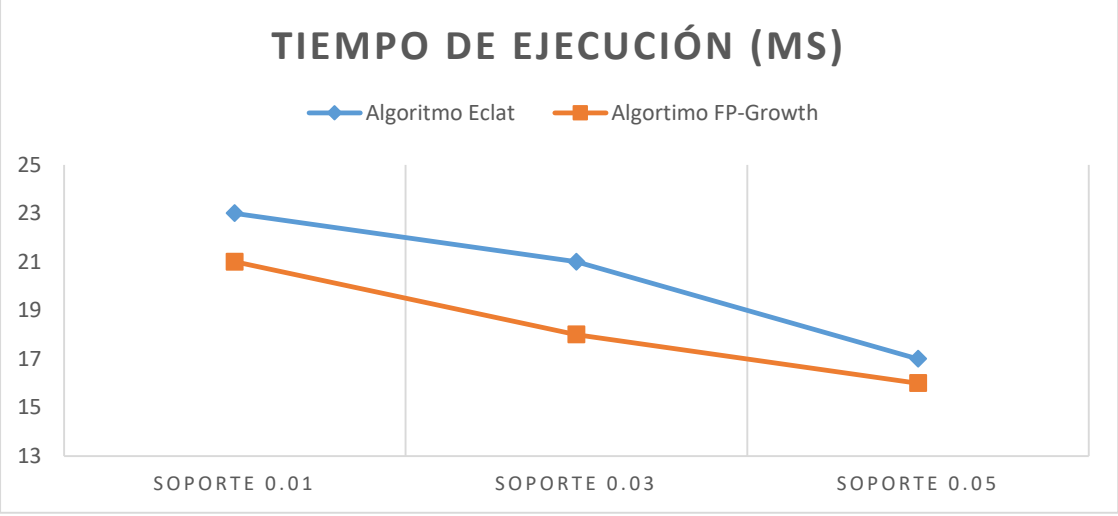


Gráfico 4: Tiempo de ejecución algoritmos Eclat y FP-GROWTH para dataset Balance Scale

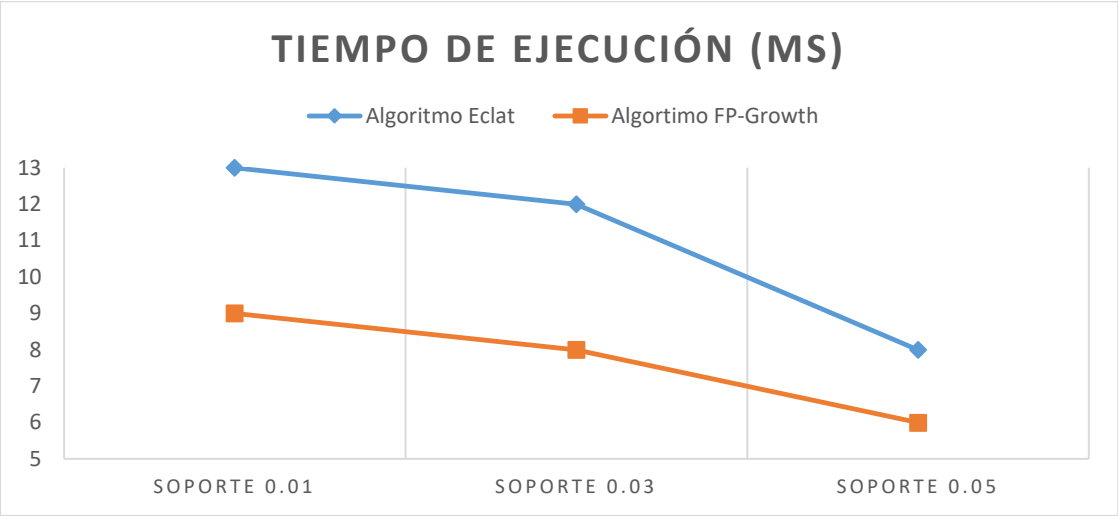


Gráfico 5: Tiempo de ejecución algoritmos Eclat y FP-GROWTH para dataset Zoo

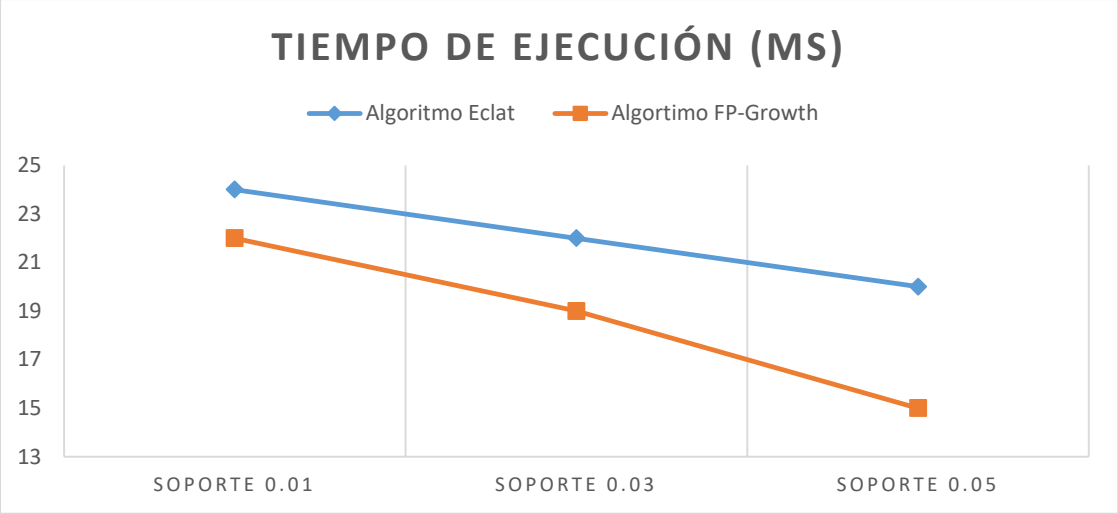


Gráfico 6: Tiempo de ejecución algoritmos Eclat y FP-GROWTH para dataset Car

Se puede evidenciar a través del Gráfico 4, 5 y 6 que, a pesar de haber usado distintos valores de soporte, en cuanto a tiempo de ejecución el algoritmo FP-GROWTH tiene mejor desempeño en todas las circunstancias.

Los resultados arrojados luego de la ejecución de los algoritmos con los casos de estudio, permiten constatar a través de los gráficos 1, 2, 3, 4, 5, 6 que no se puede llegar a una conclusión concreta a la hora de elegir cuál de los dos tiene mejor desempeño. Por tanto, se propone realizar una suma ponderada (TORRES, y otros, 2009) definiendo en esta, qué indicador en particular tiene mayor peso sobre el otro. En caso de que se desee procesar un dataset con un gran número de transacciones, lo conveniente es que tenga más peso el tiempo

de ejecución. Teniendo en cuenta esto y habiéndose analizado los datasets, los cuales no tienen suficientes transacciones como para darle mucho valor al tiempo de ejecución la suma ponderada quedaría de la siguiente manera:

$$SumaPonderada = \alpha * Te + \beta * Um$$

Donde Te es el tiempo de ejecución, Um es uso de memoria, α y β son los valores de los pesos que tienen los indicadores de tiempo de ejecución y uso de memoria, donde $\alpha + \beta \leq 1$

Luego de tener definida la suma ponderada se podrá determinar en dependencia de qué desea el usuario cuál de los dos algoritmos tiene mejor desempeño computacional. A continuación, se procederá a evaluar las ejecuciones pertinentes de los algoritmos para analizar cómo se comportan luego de lo anteriormente planteado con una ecuación de suma ponderada:

$$SumaPonderada = 0.3 * Te + 0.7 * Um$$

Suma Ponderada			
Algoritmo	Valor de soporte 0.01	Valor de soporte 0.03	Valor de soporte 0.05
Eclat	10.9375237	9.96650229	7.79351959
FP-GROWTH	11.6815384	10.109539	8.16551895

Tabla 18: Valores de la suma ponderada de los algoritmos Eclat y FP-GROWTH

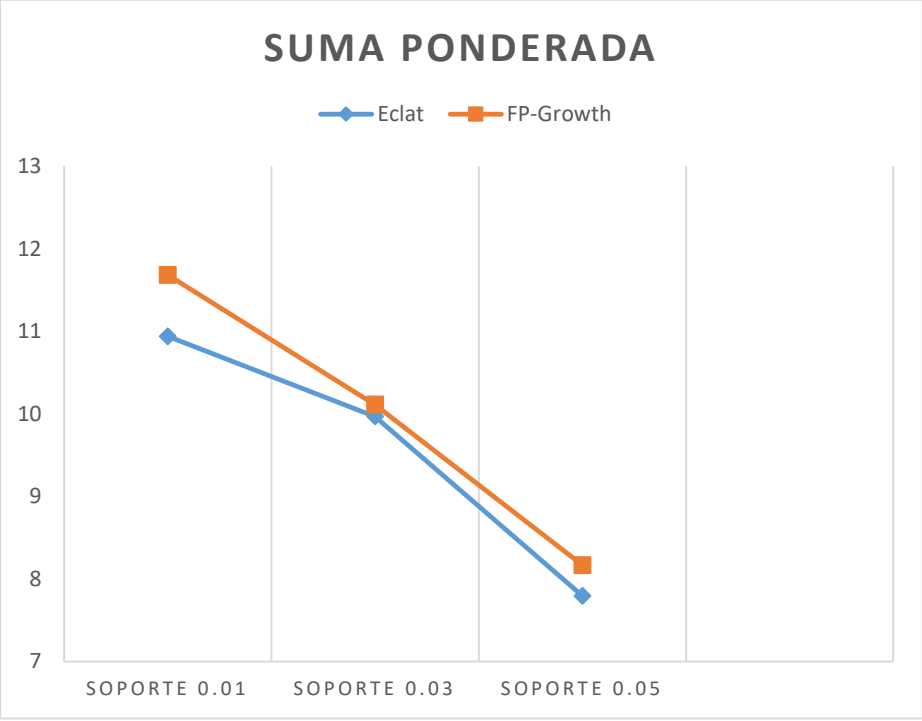


Gráfico 7: Valores de la suma ponderada de los algoritmos Eclat y FP-GROWTH

A través del gráfico anterior se puede evidenciar que, haciendo el uso de la suma ponderada, elemento que fue definido anteriormente, el algoritmo de extracción de reglas de asociación Eclat tiene mejor desempeño computacional.

Conclusiones parciales

En este capítulo se evaluaron a través de casos de estudio la ejecución de los algoritmos de Eclat y FP-GROWTH con distintos valores de confianza, para determinar que, en especial ninguno de los dos se comporta de mejor forma que el otro, teniendo que emplear el uso de una suma ponderada en la que se determina un peso para cada indicador, en dependencia de cuál es más importante para el usuario para determinar cuál de los dos algoritmos tiene mejor desempeño, quedando como algoritmo con mejor desenvolvimiento Eclat.

Conclusiones

En esta investigación se desarrolló una aplicación para poder lograr realizar una comparación del desempeño computacional de los algoritmos de extracción de reglas de asociación Eclat y FP-GROWTH. Para ello se han cumplido los aspectos siguientes:

1. Se definió el marco conceptual de referencia para soportar los fundamentos teóricos de la investigación que permitió definir conceptos esenciales, en este caso están: desempeño computacional y lo que representa a la hora de la elección de uno de los algoritmos de extracción, tipos de software donde quedó reflejado que el software de aplicación es el software que mejor se adapta a el presente trabajo, requisitos de alto nivel, proceso de desarrollo de software quedando en este definido que no se utilizará una metodología en específico, sino que se utilizará algunos aspectos de estas mencionados anteriormente.
2. Se llevó a cabo el proceso del desarrollo de software, evidenciando cada uno de los entregables generados por fase del desarrollo tales como: requisitos funcionales y no funcionales, los diagramas de clases de los algoritmos Eclat y FP-GROWTH, se definieron las actividades de la línea base tecnológica, así como el lenguaje de programación, IDE de desarrollo y la generación del código de las clases modeladas, posibilitando un correcto desarrollo de la aplicación que dio solución al problema a resolver en esta investigación.
3. Se evaluaron a través de casos de estudio la ejecución de los algoritmos de Eclat y FP-GROWTH con distintos valores de confianza, para determinar que, en especial ninguno de los dos se comporta de mejor forma que el otro, teniendo que emplear el uso de una suma ponderada en la que se determina un peso para cada indicador, en dependencia de cuál es más importante para el usuario para determinar cuál de los dos algoritmos tiene mejor desempeño, quedando como algoritmo con mejor desenvolvimiento Eclat.

Bibliografía

1. AGRAWAL, Rakesh. 1994. *Fast algorithms for mining association rules.* 1994.
2. AGRAWAL, Rakesh, IMIELIŃSKI, Tomasz and SWAMI, Arun. 1993. *Mining association rules between sets of items in large databases.* 1993.
3. Alatas, B., E. Akin, and A. Karci. 2008. *MODENAR: Multi-objective differential evolution algorithm formining numeric association rules.* 2008.
4. BORGELT, Christian. 2003. *Efficient implementations of apriori and eclat.* 2003.
5. Cicilia, Belloso and Ivonne, Claudia. 2009. *Monografia sobre la metodología de desarrollo de software RUP.* 2009.
6. *Clasificación de Software.* BRAVO CASTRO, Yohon Jairo. 2011. 2011.
7. Darío, Rubén. 2016. *Metodología ágil de desarrollo de software programación extrema.* 2016.
8. Darshan. 2015. Darshan. *Darshan.* [Online] 11 25, 2015. [Cited: 1 14, 2017.] http://www.darshan.ac.in/Upload/DIET/Documents/CE/Darshan%20-%20Sem5%20-%202150708%20-%20SP_25112015_054658AM.pdf.
9. *Data Mining Discretization Methods and Performances.* Marzuki, Zaharin. 2014. 2014.
10. Dheeru, Dua and Karra Taniskidou, Efi. 2017. UCI Machine learning repository. [Online] University of California, Irvine, School of Information and Computer Sciences, 2017. [Cited: 3 15, 2018.] http://archive.ics.uci.edu/ml.
11. Felipe Alfaro, Jimmy Solano. 2015. Singularities. [Online] agosto 2015. [Cited: diciembre 15, 2017.] https://www.singularities.com/blog/2015/08/apriori-vs-fpgrowth-for-frequent-item-set-mining.
12. Fernandez, Carlos Molina. 2005. *Imprecisión e incertidumbre en el modelo multidimensional: aplicación a la minería de datos.* 2005.
13. Fernández, Gerardo. 2002. *Introducción a Extreme Programming.* 2002.
14. Gonzáles, Carlos. 2003. Metodología XP. Uruguay : s.n., 2003.

15. HAN, Jiawei. 2007. *Frequent pattern mining: current status and future directions. Data Mining and.* 2007.
16. Hernández Martínez, Maiko. 2015. *Herramienta para la representación de un modelo orientado a grafos de las relaciones existentes en un proyecto de software.* 2015.
17. Hommel, Scoot. 1999. Convenciones de código Para el lenguaje de programacion JAVA. 1999.
18. *IEEE standard glossary of software engineering terminology.* RADATZ, Jane, GERACI, Anne and KATKI, Freny. 1990. 1990.
19. Jorge, V. 2010. *Código de programación.* 2010.
20. LaRosa, C., L. Xiong, and K. Mandelberg. 2008. *Frequent pattern mining for kernel trace data.* Ceara, Brazil : s.n., 2008.
21. Logreira, Carlos. 2011. *Minería de datos y su incidencia en la toma de decisiones empresariales en el contexto de CRM.* 2011.
22. M. Bohanec, V Rajkovic. 1990. *Expert system for decision making.* 1990.
23. Martín, D. Álvaro Pita. 2012. *Una metaheurística para la extracción de reglas de.* 2012.
24. Martínez, Apolinar Velarde. 2003. *Minería de Datos. Una Introducción.* 2003.
25. Medina, José E, Hernández palancar, José and Pérez, Airel. 2007. *Generación de conjuntos de ítems y reglas de asociación.* Ciudad de la Habana, Cuba : s.n., 2007.
26. Molina Romero, Javier and Quishpi Betún, Luis. 2015. *Desarrollo de herramienta de gestion de proyectos RUP usando metodología Scrum+XP.* Madrid, España : s.n., 2015.
27. Negrín Ortiz, Guillermo. 2014. *Ontología Del modelo multidimensional y las asociaciones entre sus elementos.* 2014.
28. Noriega Martínez, Raúl. 2017. *El proceso de desarrollo de Software.* 2017.
29. Pacienzia, Esteban Gabriel. 2015. *Metodologías de desarrollo de software.* 2015.
30. Plazas, Laura. 2014. *Estudio de la dimensión empresarial y gremial de la situación actual y prospectiva de la ingeniería de sistemas.* 2014.

31. POHL, Klaus. 2016. *Requirements engineering fundamentals: a study guide for the certified professional for requirements engineering exam-foundation level-IREB compliant.* 2016.

32. Pressman, Roger. *Ingeniería del Software, un enfoque práctico.*

33. Rennolls, Keith. 2005. *An intelligent framework (O-SS-E) for data mining, knowledge discovery and business intelligence.* 2005.

34. T. Menzies, Y. Hu. 2003. *Data Mining For Busy People.* 2003.

35. TORRES, Patricia, CRUZ, Camilo Hernán and PATIÑO, Paola Janeth. 2009. *Índices de calidad de agua en fuentes superficiales utilizadas en la producción de agua para consumo humano: Una revisión crítica.* 2009.

36. Trujillo Gálvez, Belkis. 2016. *Software for health situation analysis in Stomatology.* 2016.

37. VALLEJOS, S. J. 2006. *Minería de Datos.* 2006.

38. Witten, Ian H., Frank, Eibe , Hall, Mark A. 2016. *Practical machine learning tools and techniques.* 2016.

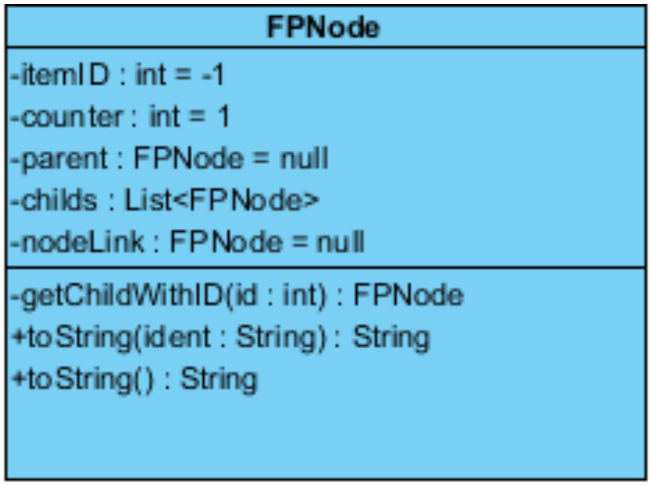
39. Wu, Xindong. 2008. *Top 10 algorithms in data mining. Knowledge and information systems.* 2008.

40. Zaki. 1997. *Localized Algorithm for Parallel Association.* 1997.

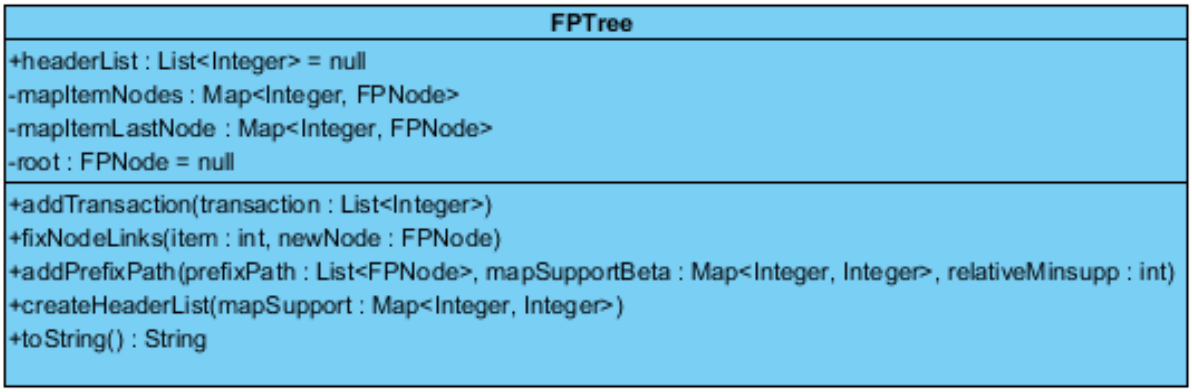
41. Zaki, Mohammed J. 2000. *Scalable Algorithms for Association Mining.* 2000.

42. Zamora, Ezequiel. 2014. *Metodología de desarrollo del software.* 2014.

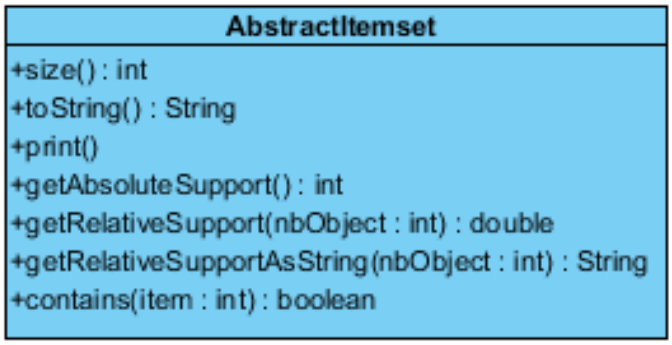
Anexos



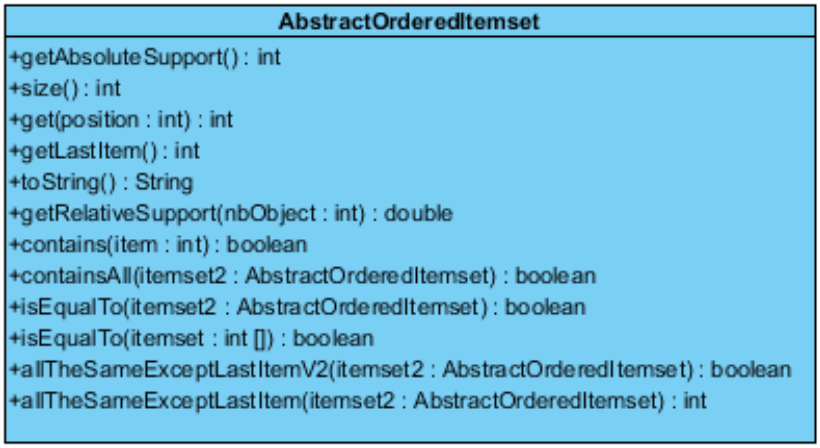
Anexo 1: Modelado de clase de la clase FPNode



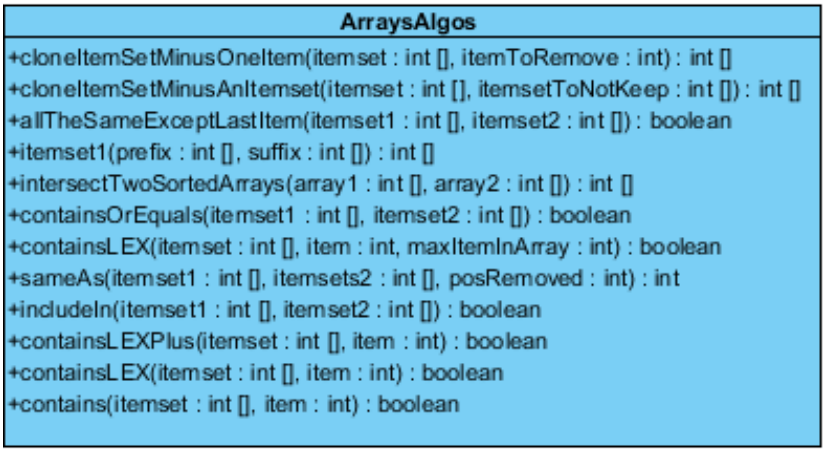
Anexo 2: Modelado de clase de la clase FPNode



Anexo 3: Modelado de clase de la clase AbstracItemset



Anexo 4: Modelado de clase de la clase AbstractOrderedItemset



Anexo 5: Modelado de clase de la clase ArraysAlgos