MÉTODO DE RECUPERACIÓN DE ARQUITECTURAS BASADO EN LA FOCALIZACIÓN EVALUATIVA Y LA VISUALIZACIÓN DE SOFTWARE: UN ESTUDIO DE CASO EN UNA PEQUEÑA ORGANIZACION DE SOFTWARE

Monografía

Yuli Andrea Ordoñez Guzmán
Edwar Alejandro Giraldo Muñoz

Facultad de Ingeniería Electrónica y Telecomunicaciones
Grupo IDIS – Investigación y Desarrollo en Ingeniería de Software
Departamento de Sistemas
Popayán, Octubre de 2016

MÉTODO DE RECUPERACIÓN DE ARQUITECTURAS BASADO EN LA FOCALIZACIÓN EVALUATIVA Y LA VISUALIZACIÓN DE SOFTWARE: UN ESTUDIO DE CASO EN UNA PEQUEÑA ORGANIZACION DE SOFTWARE

Monografía

Yuli Andrea Ordoñez Guzmán
Edwar Alejandro Giraldo Muñoz

Director: Ph.D Julio Ariel Hurtado

Facultad de Ingeniería Electrónica y Telecomunicaciones
Grupo IDIS – Investigación y Desarrollo en Ingeniería de Software
Departamento de Sistemas
Popayán, Octubre de 2016

Agradecimientos, Yuli Andrea Ordoñez Guzmán
Doy infinitas gracias a Dios por permitirme culminar esta última etapa de mi carrera como futura Ingeniera de Sistemas, por su gran amor, ayuda y fortaleza para seguir adelante y ser Él quien guie mi camino para lograr mis sueños.

A mi hijo Alejandro Vega Ordoñez quien ha sido mi motor y mi apoyo incondicional durante todo estos años. A mi familia que sin ellos no hubiera podido culminar.

Al PhD. Julio Ariel Hurtado, por su dedicación, apoyo, paciencia, quien con sus conocimientos y experiencia nos guio y permitió llevar a cabo junto con mi compañero de tesis este trabajo de grado.

A todos mis amigos, compañeros y profesores que han sido partícipes de mi formación tanto personal como académica.

Agradecimientos, Edwar Alejandro Giraldo
Principalmente agradezco a mi familia por su amo y apoyo incondicional durante todo este proceso. A mi tutor PhD Julio Ariel Hurtado por toda sus dedicación y acompañamiento en la elaboración de este trabajo de grado. A mis amigos por ser aquellos que me impulsaron a salir adelante en mis momentos de derrota, finalmente a mis compañeros y profesores del programa por su motivación y aportes para mi desarrollo como profesional.

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ANEXOS (DOCUMENTO DE ANEXOS)

INTRODUCCIÓN

1.1. CONTEXTO

El desarrollo de software es una actividad compleja, intensiva en conocimiento y por tanto altamente dependiente del talento humano [1]. Cada una de las áreas de conocimiento dentro de la ingeniería de software, rescata su importancia para resolver el tema de la crisis crónica del software. Áreas como los requisitos y la validación del software, se orientan hacia la construcción del sistema correcto, el diseño, la programación, el aseguramiento de calidad, se orientan hacia la construcción del sistema en forma correcta y la gestión se orienta hacia la construcción administrada del software bajo restricciones de tiempo y recursos [2]. Dentro de estas áreas, conectando el tema de los requisitos y el diseño de software se encuentra el de la arquitectura de software [3].

Inicialmente el diseño de software fue entendido como la implementación de la funcionalidad del sistema, definida como la capacidad del sistema para hacer el trabajo para el que fue construido [4], dejando por fuera cualidades tales como la interoperabilidad, modificabilidad y la portabilidad que también son propiedades a cubrir a través del desarrollo de un sistema software. Estas propiedades son conocidas como atributos de calidad, los cuales son principalmente determinados por la arquitectura del sistema; mientras que muchas estructuras pueden satisfacer la funcionalidad, pocas estructuras pueden satisfacer la funcionalidad y también los atributos de calidad necesarios en el sistema.

1.2. PROBLEMA

Las arquitecturas de software surgen como un medio para satisfacer tanto las necesidades funcionales, como los atributos de calidad de un sistema [5]. El conocimiento sobre la arquitectura de un sistema ayuda a los desarrolladores a identificar y localizar las partes del sistema que deben ser modificadas durante un ciclo de evolución o un requerimiento de mantenimiento, así como las demás partes del sistema que se verán afectadas por dichos cambios [6]. A medida que un sistema software evoluciona, su arquitectura puede requerir ajustes, de manera que ésta erosiona sino se considera su documentación, su coherencia con su implementación y en especial su rationale[1] [7] haciéndose propensa a errores para las labores de incremento en las funcionalidades y en las actividades de mantenimiento, lo cual contribuye al rápido envejecimiento del software. Por lo anterior, se considera que la arquitectura software es parte fundamental en la evolución de los sistemas software [8][9], por lo que su recuperación es una necesidad de gran importancia cuando esta no se encuentra definida de forma

[1] Rationale- la inclusión explícita de las decisiones tomadas durante el proceso de diseño y las razones por las cuales esas decisiones fueron tomadas.

explícita [10] o hay divergencia entre la arquitectura prescriptiva[2] y la arquitectura descriptiva[3].

Se entiende la recuperación de la arquitectura de software como el proceso de ingeniería inversa, en el que se obtiene la arquitectura explícitamente documentada a partir de la arquitectura implícita o prescriptiva presente en un sistema implementado y para el cual se requiere del conocimiento arquitectónico en escenarios de mantenimiento o de expansión del mercado de un producto. El problema de la falta de arquitectura acorde a los sistemas desarrollados, se acrecienta en casos donde por el mismo negocio del software, inicia sin un plan arquitectónico en dónde además de ser realizados por un grupo variado de desarrolladores, no necesariamente posee un grupo completamente destinado al ejercicio de la arquitectura, llevando a que no siempre se cuente con toda la información acerca de la arquitectura del sistema o definitivamente no se cuente con una arquitectura explícita.

La reutilización es de interés cuando necesitamos construir sistemas que sean grandes, complejos, fiables, menos costosos y que se terminen a tiempo. La reutilización sistemática es una forma de incrementar la productividad en el desarrollo de software y de su calidad [11]. Sin embargo, el reutilizar software involucra un riesgo [12], debido que hacerlo de manera no exitosa (construir para reutilizar y luego no aprovechar los elementos reutilizables significativamente) involucra una pérdida de tiempo y recursos para la organización, por otro lado si la competencia hace un uso exitoso de la reutilización repercute en una posible pérdida de mercado [12]. Además de la reutilización oportunista, está la reutilización sistemática de software, la cual sigue un proceso mediante el cual una organización define un conjunto de procedimientos operativos sistemáticos para especificar, producir, clasificar, recuperar y adaptar los artefactos de software con el propósito de utilizarlos en sus actividades de desarrollo [13]. La reutilización depende no sólo de la búsqueda y la reutilización de componentes, sino también en la forma como esos componentes se combinan [14] y esto depende del conocimiento y diseño de la arquitectura. Específicamente en el marco de las líneas de productos[4], la arquitectura del software es considerada como el activo de software más valioso para la reutilización, debido a que en sí mismo es un activo reutilizable y es a su vez el que define el marco de reutilización para la mayoría de los otros activos reutilizables. Dentro de la construcción de las líneas de productos, una de las estrategias es realizar una minería de activos para ser incluidos en la línea de procesos, y uno de esos activos candidatos de gran prioridad a ser recuperado, es la arquitectura de software [15].

[2] Decisiones tomadas respecto a la arquitectura.
[3] Decisiones explícitamente descritas en el documento de la arquitectura.
[4] Una línea de producto es un conjunto de productos que comparten un grupo de características comunes que satisfacen necesidades específicas de un segmento particular del mercado y que son construidas a partir de un conjunto de activos de software pre-construidos para tal fin.

En particular, la pequeña industria de software puede verse beneficiada por este tipo de soluciones, bien sea para evolucionar sus sistemas ya implementados o para fortalecer su proceso de desarrollo al reutilizar soluciones desde el universo de las soluciones del mercado, dentro de su misma organización o las disponibles a nivel open source, con el fin de generar valor rápidamente a sus clientes. Sin embargo, la comprensión de los sistemas, son las mayores barreras para su extensión y reutilización [16]. Por tanto, la ingeniera inversa está siendo valorada como un proceso importante en el ciclo de vida de los sistemas, pero todavía se presentan muchas problemáticas debido a que es un área relativamente joven.

Debido a la gran variedad y complejidad de sistemas que se pueden recuperar, han surgido una gran variedad de lenguajes, enfoques y soluciones. Así, se busca dar respuesta a la pregunta de ¿cómo guiar a un arquitecto de software en la tarea de recuperar arquitecturas de sistemas? y teniendo en cuenta la pregunta anterior, se plantea resolver si ¿A través de un enfoque evaluativo y mecanismos de recuperación visual podría un arquitecto en forma práctica e intuitiva recuperar el conocimiento arquitectónico de una aplicación ya implementada? por lo anterior se realizaron diferentes investigaciones en documentos, analizando las diferentes técnicas, herramientas, métodos de evaluación de arquitectura donde se logra definir un método de recuperación de arquitecturas basado en la visualización y enfoques de evaluación de arquitecturas como facilitadores de la recuperación del conocimiento implícito, con el cual se procede a realizar estudio de caso exploratorios, obteniendo diversos resultados según el caso y verificando que el método es adaptable en ciertas circunstancias y bajo otras se identifica las necesidades de soporte tecnológico para automatizar ciertos pasos dentro del proceso de recuperación de la arquitectura.

1.3. OBJETIVOS

1.3.1. Objetivo General

Definir un método de recuperación de arquitecturas basado en el análisis visual del software y en el conocimiento implícito de los participantes.

1.3.2. Objetivos Específicos

- Examinar la viabilidad de los métodos de evaluación de arquitecturas como herramienta de análisis de recuperación del conocimiento implícito de la arquitectura mediante la caracterización de métodos y técnicas actuales de la recuperación de arquitecturas software.
- Determinar los elementos del soporte tecnológico del método mediante la revisión y selección de herramientas visuales para la recuperación de arquitecturas.

- Definir un método de recuperación de arquitecturas basado en la visualización y un método de evaluación de arquitecturas como facilitador de la recuperación del conocimiento implícito.
- Evaluar el método propuesto mediante un estudio de caso, donde se evalúe la aplicabilidad funcional, práctica e intuitiva del método.

1.4. METODOLOGÍA Y DESCRIPCIÓN DEL TRABAJO.

Para esta investigación se está siguiendo el proceso definido por Hurtado [17] llamado “Método Científico en Ingeniería de Software” - MCIS compuesto por tres fases (exploración, formulación y ejecución) con sus respectivos hitos (Comprensión del problema, Propuesta de investigación y Reporte de resultados) y en la conducción y reporte de estudios de caso propuesta por Runeson et al. [18].

La metodología utilizada para el desarrollo de este trabajo consistió en partir del conocimiento referente a recuperación de arquitecturas, métodos de evaluación y herramientas para contextualizar y guiar la construcción del método. Dicha revisión estuvo dividida en grandes momentos: caracterización de los métodos, técnicas y herramientas de recuperación de arquitecturas, analizar elementos de soporte tecnológico para la recuperación de arquitectura y creación del método. El método está formado por 5 actividades; presentación, identificación y priorización de escenarios, la recuperación visual de la arquitectura, la evaluación de los escenarios y la recuperación del rationale, discusión y análisis de la arquitectura recuperada y por último la presentación de los diferentes resultados.

Esta monografía está organizada en cinco capítulos, en el capítulo 2 se presenta el marco conceptual del proyecto y los trabajos relacionados; resultado de la profundización de las actividades correspondientes al primer objetivo específico: “Examinar la viabilidad de los métodos de evaluación de arquitecturas como herramienta de análisis de recuperación del conocimiento implícito de la arquitectura mediante la caracterización de métodos y técnicas actuales de la recuperación de arquitecturas software.” Se define y caracteriza las herramientas para la recuperación de Arquitectura profundizando en las actividades correspondientes al tercer objetivo específico: “Determinar los elementos del soporte tecnológico del método mediante la revisión y selección de herramientas visuales para la recuperación de arquitecturas”.

En el capítulo 3, de acuerdo a la información previamente recolectada sobre la recuperación de arquitectura por medio de herramientas, se profundiza en las actividades correspondientes al cuarto objetivo específico: “Definir un método de recuperación de arquitecturas basado en la visualización y un método de evaluación de arquitecturas como facilitador de la recuperación del conocimiento implícito”.

En el capítulo 4, se evalúa el método propuesto en estudios de caso que consisten en: documentar la arquitectura de ArgoUml a un nivel de diseño, Aplicación del

método de evaluación ATAM-AR, en un sistema E-LearningSync, evaluación del método EV-AR (método de recuperación arquitecturas a partir del conocimiento tácito de la organización, capturado a través de los diferentes stakeholders involucrados en la obtención de la arquitectura y los artefactos que de una u otra forma la describen: modelos, requerimientos, código y pruebas) en una pequeña organización de software". En este capítulo se incluyen los estudios exploratorios que permitieron la construcción empírica del método.

Finalmente, en el capítulo 5, se presentan las conclusiones, limitaciones y trabajo futuro de esta investigación.

MARCO CONCEPTUAL Y ESTADO DEL ARTE

2.1. EL CONCEPTO DE ARQUITECTURA DE SOFTWARE

La arquitectura de software es un concepto que ha incrementado su importancia como disciplina en las últimas décadas, debido a la complejidad resultante de comprensión de los sistemas de software durante su desarrollo y evolución. Particularmente por la necesidad de comunicación, puesto que dichos sistemas son especificados, diseñados, implementados, probados y gestionados por varios individuos con diferentes necesidades y habilidades de información y conocimiento. La Arquitectura de Software se refiere a “las estructuras de un sistema, compuestas de componentes con sus propiedades externamente visibles hacia los otros componentes y las relaciones que existen entre estos” [4].

La arquitectura de software es reconocida como un elemento crítico en el éxito del desarrollo y en la evolución de sistemas intensivos en software, a pesar del rol tan importante que tienen las arquitecturas de software muchos sistemas no tienen una representación confiable. Uno de los aspectos que motivan el estudio en este campo están relacionados con el cubrir aspectos del conocimiento y la comunicación humana debido a que, su diseño es una actividad intensiva en conocimiento, por ejemplo inspecciones de diseño, comunicación a alto nivel de abstracción entre los miembros del equipo de desarrollo, reutilización de componentes y comparación de diseños alternativos a un alto nivel de abstracción [19]. Un sistema de software tiene un gran número de interesados, y cada uno de ellos tiene preocupaciones acerca de diferentes aspectos, incluida la aplicación de software, calidad, e incluso valor económico. Por lo tanto, los arquitectos y los diseñadores necesitan combinar diferentes tipos de puntos de vista para abordar las preocupaciones de los distintos grupos de interés y apoyar efectivamente la evolución del sistema [20]. Pero todas estas preocupaciones no pueden ser presentadas en un solo modelo, por lo que es necesario presentar la arquitectura en diferentes vistas para poder presentar la información relevante para cada interesado [21].

2.2. IMPORTANCIA DE LA ARQUITECTURA

Si la arquitectura de un sistema es deficiente en su concepto o diseño, se tendrán grandes dificultades ya que no alcanzaría el total de requerimientos establecidos generando re-trabajo o fracaso en la ejecución del software. El desarrollo de una arquitectura es de vital importancia y debe realizarse a partir de un documento de especificación de requerimientos donde se deben plasmar requerimientos funcionales y no funcionales. La característica fundamental de una arquitectura son los requerimientos no funcionales, específicamente los atributos de calidad, es decir

el desempeño, confiabilidad, seguridad, facilidad de modificación, facilidad de uso, robustez, portabilidad, escalabilidad, reutilización y disponibilidad [4].

2.3. IMPORTANCIA DE LA ARQUITECTURA

La arquitectura se puede determinar a partir de estilos arquitectónicos, que pueden ser utilizados y adaptados para la organización de la arquitectura, la cual describe un sistema software especificado por medio de vistas arquitectónicas, que a su vez están expresadas mediante algún ADL [22]. Un estilo arquitectónico define la estrategia arquitectónica de una familia de sistemas, en términos de patrones estructurales, de control y de comunicación. La arquitectura de un sistema de software puede basarse en uno o en varios estilos arquitectónicos [23].

Un estilo arquitectónico describe un conjunto de componentes con sus responsabilidades, un conjunto de conectores entre componentes su comunicación, coordinación y cooperación, restricciones que definen cómo se integran los componentes para formar el sistema, modelos que permiten comprender las propiedades de un sistema general en función de las propiedades conocidas de las partes que lo integran [24], existen diferente estilos arquitectónicos como : estilos de flujo de datos (Tubería y filtros), estilos en centrados en datos (Arquitecturas de Pizarra o Repositorio), estilos de llamada y retorno (Model-View-Controller (MVC)), estilo modular de capas, arquitecturas orientadas a objetos, arquitecturas basadas en componentes, estilos de código móvil (arquitectura de máquinas virtuales), estilos heterogéneos (sistemas de control de procesos, arquitecturas basadas en atributos), estilos Peer-to-Peer (red de ordenadores en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí).

La arquitectura por capas (muy utilizado en aplicaciones empresariales), los sistemas dirigidos por eventos donde cada subsistema genera eventos según su condición y define que eventos de otros subsistemas desea atender y el manejador de eventos se encarga de enviar los eventos generados a aquellos sistemas interesados en recibirlos, RPC/RMI, Arquitecturas de objetos distribuidos CORBA el nodo cliente utiliza (de forma transparente) los objetos que existe y están corriendo en los demás nodos, Arquitecturas Peer to Peer un par puede jugar un rol de cliente y/o de servidor, dependiendo de las necesidades del momento, arquitectura basada en Plugins API (Application programming interface) Interfaz bien definida, una estrategia basada en plugins hace que un sistema pueda ser fácilmente extendido y complementado por terceras partes. Finalmente están las arquitecturas de referencia, las cuales normalmente se obtienen por medio del estudio de una clase de aplicación (de un dominio en particular) y representan una arquitectura ideal que incluye todas las características que cierto tipo (clase) del sistema las podría incorporar [25].

2.4. DECISIONES ARQUITECTURALES: ATRIBUTOS, TÁCTICAS, ESTRATEGIAS Y PATRONES ARQUITECTÓNICOS

Existen dos tipos de requerimientos en la creación de un producto software; los requerimientos funcionales que ayudan a especificar las actividades del negocio del cliente y la funcionalidad que la aplicación permitirá realizar al usuario final. El requerimiento no funcional hace referencia a las características, objetivos de desempeño y atributos que el cliente proporciona, no siempre de manera explícita.

Los requerimiento de software son catalogados en reglas de negocio, atributos de calidad, interfaces externas y restricciones [26]. Los atributos de Calidad y las restricciones intervienen directamente en el diseño de la arquitectura de software y son representados de diferentes formas Tabla 1.

Tabla 1. Atributos de Calidad

IEEE Std 1061	ISO Std 9126	MITRE Guía para Control de Calidad de Software Total	
Eficiencia	Funcionalidad	Eficiencia	Integridad
Funcionalidad	Confiabilidad	Confiabilidad	Supervivenciabilidad
Mantenibilidad	Usabilidad	Usabilidad	Correctitud
Portabilidad	Eficiencia	Mantenibilidad	Verificabilidad
Confiabilidad	Mantenibilidad	Expansibilidad	Flexibilidad
Usabilidad	Portabilidad	Interoperabilidad	Portabilidad
		Reusabilidad	

Los atributos de calidad ayudan a medir y controlar la calidad de un sistema y se pueden ver como cualidades que son clasificadas por disponibilidad, modificabilidad, desempeño, seguridad, facilidad de pruebas y usabilidad. Los atributos de calidad pueden ser manejados por los interesados en el proyecto, estos atributos de calidad son escenarios según la SEI (Software Engineering Institute). Las restricciones son las limitaciones del diseño e implementación que deben ser consideradas por el desarrollador.

Los arquitectos usan los patrones arquitecturales que satisfagan los atributos controladores, bajo los escenarios expuestos y siguiendo las recomendaciones dadas por las tácticas los aplican al resolver el problema. Los patrones son conocidos como soluciones a los problemas en común, mientras que las tácticas se centran en cómo satisfacer atributos específicos de calidad. Los arquitectos necesitan entender cómo las tácticas arquitectónicas y patrones se relacionan y cómo utilizarlos de forma eficaz, se exploran las relaciones entre los patrones arquitectónicos y las tácticas a través de la lente de un atributo de calidad – por ejemplo si es el caso de la modificabilidad, se determina que el patrón arquitectural proporciona un mecanismo para cambiar la estructura y el comportamiento de un sistema de software de forma dinámica y es compatible con la modificación de los aspectos fundamentales, tales como tipo de estructura y mecanismos para la

llamada a una función [27]. En la Tabla 2, se puede observar la relación de los patrones arquitecturales con respecto a las tácticas para la modificabilidad.

Tabla 2. Patrones Arquitecturales con respecto a tácticas de modificabilidad.

Patrones Arquitecturales	Tácticas para la modificabilidad									
	Aumentar cohesión		Reducir el acoplamiento					Aplazar tiempo de unión		
	Aumentar la cohesión	Servicios comunes abstractos	Uso de la encapsulación	Utilizar una capa	Patrones comunes restringidos	Uso de un intermediario	Aumentar el nivel abstracto	Registro de tiempo de ejecución en uso	Use sttart-up time binding	Use runtime binding
Layers	X	X	X		X	X	X			
Pipes and Filtres	X		X		X	X			X	X
Blackboard	X	X			X	X	X	X		
Broker	X	X	X		X	X	X	X		X
Model View Controller	X		X			X				
Presentation abstraction control	X		X			X	X			
Microkernel	X	X	X		X	X				
Reflection	X		X							

2.5. LENGUAJES DE REPRESENTACIÓN DE ARQUITECTURAS

Se han presentado múltiples notaciones formales y lenguajes para representar arquitectura [28] normalmente conocidos como ADLs (Lenguajes de Descripción de Arquitecturas): UML 2.0, un estándar para el modelado que puede ser utilizado para la describir arquitectura, aunque sus abstracciones en ocasiones no son el mejor camino para representar una arquitectura; SysML, aunque se trata de un lenguaje de modelado de sistemas (hardware & software), provee abstracciones necesarias para especificar las vistas diferentes de la arquitectura; AADL un lenguaje de especificación de arquitecturas que provee de una sintaxis textual y un DSL(Lenguaje especifico de dominio) para la descripción de arquitecturas, permitiendo especificar varias vistas de la arquitectura para así como modelar y analizar otros aspectos relacionados con la calidad de arquitectura, como el análisis de fallos o el análisis de tiempo de respuesta, y ACME, que permite especificar de manera sintáctica y gráfica los componentes de una arquitectura de software [4].

2.6. MÉTODOS DE ARQUITECTURA

En el proceso de desarrollo arquitectónico, existen diversos métodos, tanto de análisis, diseño, implementación, así como de evaluación y recuperación arquitectónica, donde cada uno ellos se basan en concepciones que pueden ser

similares, complementarias o alternativos. Estos métodos ayudan a la identificación de los procedimientos y actividades que guían al difícil proceso de generar una arquitectura en función de un conjunto de requisitos iníciales.

Un método debe ofrecer una técnica para capturar los requisitos no funcionales que se componen de objetivos de calidad a alcanzar por la arquitectura, el método debe contener actividades sólidas para predecir los atributos de calidad con respecto a la arquitectura y así poder realizar un evaluación arquitectónica, es importante contar con un conjunto de requisitos de calidad y con una especificación de la arquitectura en la que se estructuren de forma clara decisiones de diseño.

Los métodos de evaluación tienen un objetivo en común, estos pueden variar en cuanto a objetivos específicos que persiguen. Según Dobrica y Niemela [29] existen dos grupos de técnicas de evaluación: las técnicas de Interrogación y las de medición. Las técnicas de interrogación se caracterizan por generan preguntas cualitativas que están destinadas a un arquitecto mientras que las técnicas de medición se basan en la ejecución de mediciones cuantitativas y se aplican para obtener atributos de calidad [29].

2.7. EVALUACIÓN DE ARQUITECTURAS

El estudio de la evaluación de la arquitectura se origina cuando ya se ha especificado o implementado la arquitectura. Los usuarios de los modelos del ciclo de vida iterativos o incrementales pueden evaluar las decisiones arquitectónicas realizadas durante la última iteración. Sin embargo, uno de los aspectos atractivos de la evaluación de la arquitectura es que puede aplicarse en etapas tempranas [30].

Existen dos grupos de roles que deben participar en la evaluación de una arquitectura como los son el equipo de evaluación, que son aquellas personas que realizarán el análisis y la evaluación y los interesados (stakeholders), que tienen un interés personal en la arquitectura y en el sistema que se construye a partir de ella. La evaluación de arquitecturas es una actividad que puede ser útil por diversas razones, por ejemplo para tomar mejores decisiones, es decir frente a una arquitectura que me está dando problemas, la evaluación me puede ayudar a decidir si conviene invertir en mejorarla o solucionar sus problemas o si es mejor cambiarla y empezar de nuevo. Una evaluación de arquitectura genera un informe, la forma y el contenido de las cuales varían según el método utilizado. Algunos métodos que ayudan en la evaluación de la arquitectura son: SAAM (Software Architecture Analysis Method), fue el primer método de evaluación basado en escenarios que surgió, el foco de este método es la modificabilidad, SAAM utiliza el agrupamiento de escenarios como criterio para evaluar la arquitectura. SAAM provoca la participación de los grupos de interés para identificar explícitamente la calidad y metas a establecer para la arquitectura. SAAM se concentra en atributos tales como

la modificabilidad, la variabilidad (adecuado para la línea de productos) y el logro de la funcionalidad [31].

ATAM (Architecture Trade-off Analysis Method) es un método de evaluación que explora varias cualidades y su consecución equilibrada [30] . Este método evalúa con más profundidad, en relación con otros métodos, cuestiones referentes a la arquitectura, como los atributos de calidad y está conformado por un conjunto de pasos importantes: presentación de ATAM, presentación de los objetivos de negocio, presentación de la arquitectura, investigación y análisis, identificación de las aproximaciones arquitecturales, generación del árbol de atributos de utilidad, análisis de las aproximaciones arquitecturales, pruebas testing, lluvia de ideas y priorización de escenarios y presentación de resultados [32]. En la Figura 1. Se puede observar el flujo conceptual del método ATAM.

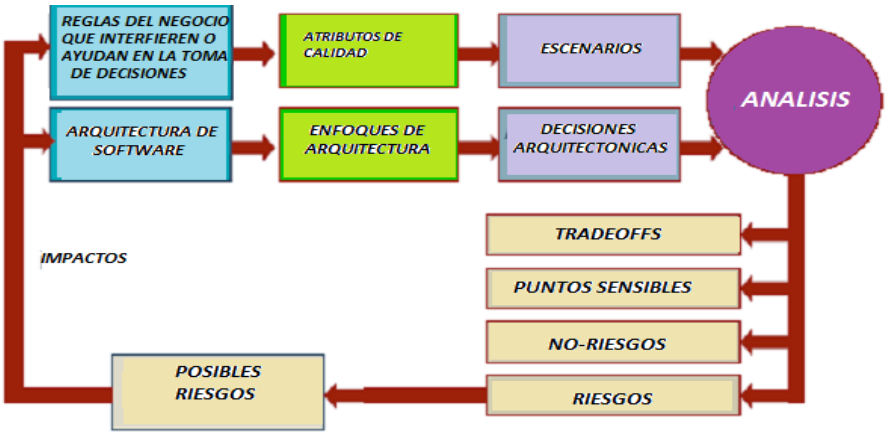


Figura 1. Flujo Conceptual del método ATAM

ARI (The Active Reviews for Intermediate Designs) es un método para evaluar sub-diseños de arquitecturas parciales en sus etapas tempranas o fases conceptuales. La evaluación se puede llegar a cabo aún en ausencia de documentación, al igual que ATAM Y SAAM, las personas involucradas en su evaluación, son el grupo de evaluación y los demás stakeholders, el método maneja escenarios donde todos los participantes interactúan generando versiones y realizando votaciones que permiten elegir los escenarios que participarán en dicha evaluación facilitando el proceso de arquitectura y la integridad de la documentación [33], Figura 2.

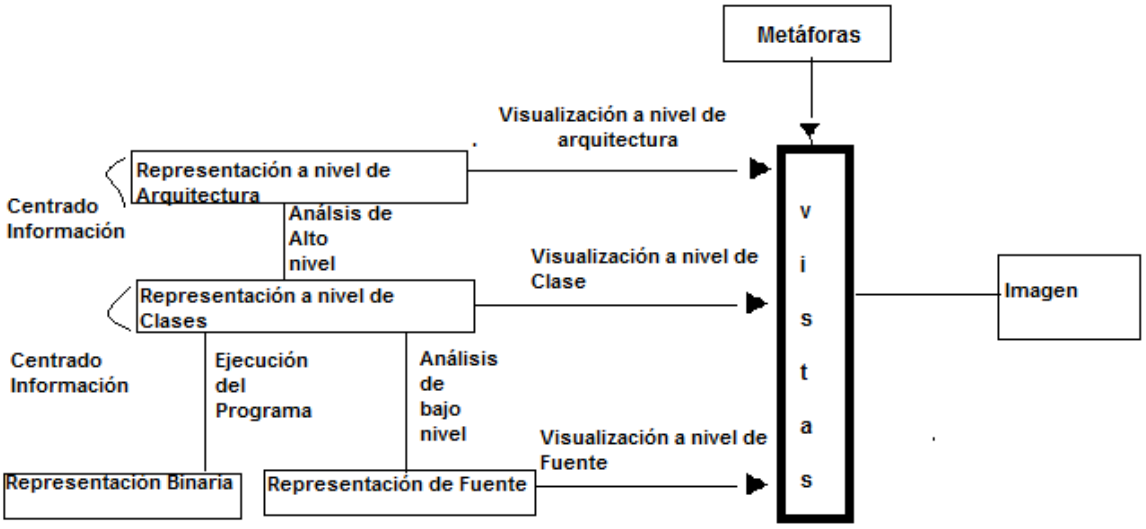


Figura 2. Proceso de recuperación Unificado.

2.8. RECUPERACIÓN DE ARQUITECTURAS

La ingeniería inversa es el análisis de un sistema para identificar sus componentes reales y las dependencias que existen entre ellos para extraer y crear abstracciones de dicho sistema e información de su diseño [34]. Existen dos tipos de ingeniería inversa de acuerdo a dos perspectivas arquitecturales: estática y dinámica, la ingeniería inversa de tipo estática se utiliza, ante todo, cuando se generan diagramas que describen su estructura, como el diagrama de clases y el de paquetes, mientras que la ingeniería inversa que analiza el sistema en ejecución se utiliza, primordialmente, cuando se requieren modelos que revelen las características dinámicas del sistema, como los diagramas de objetos y de interacción [35]. Dado que el aporte de esta tesis es precisamente un método de recuperación de arquitecturas, los avances a la fecha en esta actividad arquitectónica se presentarán más adelante en detalle como trabajos relacionados.

2.9. VISUALIZACIÓN DE SOFTWARE

La visualización de software ayuda a encontrar información de forma eficiente e intuitiva a la percepción humana, la visualización de software está enfocada en la representación visual que proporcionan los sistemas de software y se basa en su estructura o comportamiento y ayuda a mantener, reutilizar, y aplicar reingeniería. Existen muchas herramientas y técnicas de visualización de software, particularmente existen herramientas para la extracción y el análisis de las

arquitecturas de software, como Rigi[5] que ayuda proporcionar no sólo la visualización, sino también sirven de ayuda para que el usuario simplifique y navegue a través de la representación del sistema visual. Sin embargo, cada herramienta o sistema tiene sus limitaciones y restricciones en cuanto a las fases de recuperación que cubre, su apoyo a las aplicaciones desarrolladas en diferentes lenguajes de programación y su flexibilidad en el apoyo de análisis personalizado [19].

2.10. TRABAJOS RELACIONADOS

2.10.1. Técnicas de Recuperación de Arquitecturas

El dominio del conocimiento significativo para descubrir los módulos arquitectónicos de un sistema es un proceso arduo [36]. La investigación se ha centrado mucho en la forma de automatizar el descubrimiento e identificación de módulos de código. Recientemente, el uso de técnicas de agrupación propias de otros campos ha hecho camino en el campo de la investigación dentro de la ingeniería de software. Una de las técnicas estáticas que han sido ampliamente estudiadas con el fin de recuperar una estructura de alto nivel de un sistema de software, es la técnica de clustering, que consisten en descubrir una estructura dentro de un conjunto de datos D = {x1, . . . , xn}, dividiéndolo en subconjuntos que muestren una cierta ”coherencia[6]” de software aplicada al código, pero que en la práctica resulta poco aprovechada [37][38]. En el trabajo de Cai et. al [39] se presentan diferentes técnicas de agrupamiento para particionar automáticamente un sistema de software en sub-sistemas significativos.

Aunque estas técnicas han demostrado su eficacia, se observa que una característica clave en la mayoría de los sistemas de software no ha sido plenamente explotado: los sistemas mejor diseñados deben seguir fuertes reglas de diseño arquitectónico que dividen el sistema global en módulos. Estas reglas de diseño a menudo se manifiestan como construcciones de programas especiales, como estructuras de datos compartidas o interfaces abstractas, que no deben pertenecer a cualquiera de los módulos subordinados. Aportan una nueva perspectiva de la recuperación de la arquitectura basada en esta lógica, que permite la combinación de agrupamiento basado en el diseño de reglas con otras técnicas de agrupamiento, así como permitir la división de un gran sistema en subsistemas. Las técnicas de recuperación de la arquitectura son aplicadas a aplicaciones legadas escritas en lenguajes orientados a objetos [39].

Rambabu D. [40], dice que existe un gran número de relaciones entre propiedades de arquitectura y que por tanto se requiere una enorme base de conocimiento para

[5] Herramienta de Visualización.
[6] “Coherencia”: muestras dentro de un mismo grupo o clúster son más “parecidas” entre sí que a las muestras de otros clusters.

saber acerca de la las mejores prácticas y las relaciones existentes entre ellos. En este trabajo, se presenta una ontología para las propiedades de arquitectura donde se describe cómo ésta ontología se puede utilizar en diversas aplicaciones, como búsqueda basada en la semántica con fines académicos y la creación de nuevos sistemas. Figura 3.

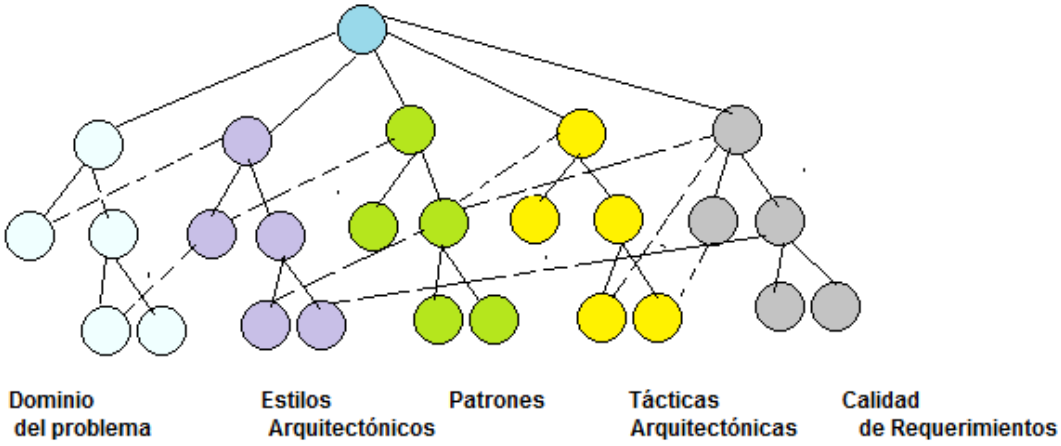


Figura 3. Ontología para diversas aplicaciones

La visión general conceptual de la ontología que se sugiere, se concentra en propiedades arquitectónicas; como problemas de dominio, estilos arquitectónicos, modelos, tácticas arquitecturales, requisitos de calidad y las relaciones que existen entre ellos. Se representan estos metadatos en un archivo XML y la búsqueda se realiza en estos metadatos como una búsqueda de etiquetas XML.

Existen varias relaciones entre los patrones y estilos, que a su vez se relacionan con el dominio del problema del sistema ayudando de diversas formas al arquitecto en la búsqueda y recuperación de documentos en una arquitectura semántica.

2.10.2. Metodologías y Procesos de Recuperación de Arquitecturas

Panas, et al. [41], definen el proceso unificado para ingeniería inversa, incluyendo análisis de los programas, el filtrado, la compresión de información sobre el programa y el programa visualizado. La Ingeniería inversa consta de tres partes: el análisis, el enfoque y la visualización de un sistema. Para evitar que el programa de resultados del análisis sea incomprensible, la información se concentra, es decir, se filtra. El programa de visualización es responsable de la representación gráfica de la información recuperada del programa. Además, se hace una mejor comprensión mediante visualización interactiva. La tarea principal en arquitectura de software es la comprensión, la identificación de los componentes de un paquete gráfico. El

análisis de los programas debe ser capaz de identificar los componentes y las comunicaciones. La principal ventaja de este proceso radica en la disociación de análisis y visualización, prestar apoyo a determinadas herramientas y procesos para cada uno de ellos.

Monroy, et al. [42], proponen una herramienta para caracterizar un método de ingeniería inversa que fue construido con la revisión técnica en el marco de la metodología DESMET. El instrumento fue aplicado a diez métodos de ingeniería inversa para validar su utilidad. Esto permitió concluir que la metodología facilita la selección del método más apropiado para un caso determinado bajo un escenario de aplicación de ingeniería inversa. Los diez métodos utilizados fueron: Design Recovery, Code Browsing, Code Visualization, Annotation Processing, Traceability, Slicing, Impact Analysis, Concept/Feature Location, Clone Detection y Clustering, cada uno de estos métodos fue analizado teniendo en cuenta la entrada, el procedimiento de ingeniería inversa y la salida.

Constantinou et al. [43], Proponen una metodología en la cual dividen la arquitectura de software en 4 capas, la interfaz de usuario, controladores, lógica de negocio e Infraestructura, esta metodología sigue 5 pasos importantes como dividir las capas por medio de un gráfico, combinar estas capas extraídas, realizar el cálculo de métricas para las clases y mover iterativamente los límites de las capas hasta que la correlación de métricas no crezca.

Zhu et. al [44], proponen un método de agrupamiento gráfico basado en el modelo de paseo aleatorio corto, el cual permite coordinar el agrupamiento basándose en la similitud y la densidad de los elementos del sistema. A este gráfico le incluyen dos tipos de bordes para presentar la conectividad y la similitud de los elementos del sistema. ArchMine, un enfoque de recuperación de la arquitectura basada en el análisis dinámico y de minería de datos, tiene por objeto ayudar en la comprensión del programa y la reutilización del software mediante la detección de clases cohesivas que implementan un conjunto de funcionalidades relacionadas, es decir, elementos arquitectónicos [45].

Según M. Pinzger et al. [46], un escenario clásico de la recuperación de arquitecturas es la construcción de arquitecturas de referencia, enfoque que utiliza sistemas existentes relacionados y tiene en cuenta los conocimientos adquiridos fuera de los sistemas individuales y experiencias realizadas con soluciones ya probadas. En este escenario se tiene en cuenta un proceso de construcción con cuatro pasos principales donde se determinan puntos de vista y conceptos arquitectónicos, se hace la recuperación de la Arquitectura, el análisis de la arquitectura recuperada y el diseño de una arquitectura de referencia. El proceso de la construcción de arquitectura se puede observar en la figura 4.

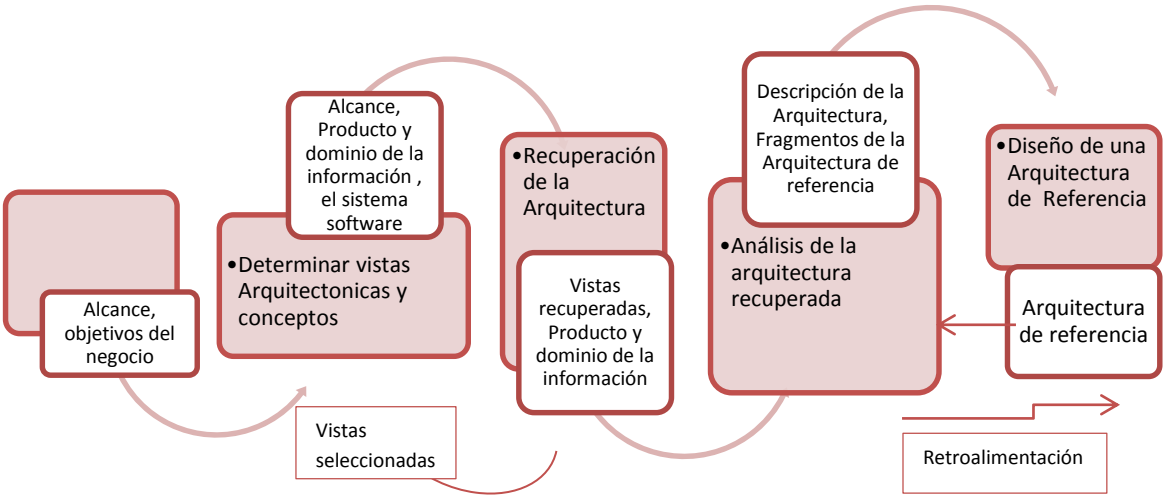


Figura 4. Proceso de la Construcción de Arquitectura

Para la realización de la arquitectura de referencia se basan en el método PuLSE-DSSA [46] (A Method for the Development of Software Reference Architectures). PuLSE-DSSA es un método para la definición sistemática y el desarrollo iterativo de arquitecturas de referencia para el software en el contexto de líneas de productos[7]. Éste método proporciona tanto un marco para el desarrollo sistemático de una referencia arquitectura, y una manera de analizar la calidad del software de arquitecturas con respecto a las propiedades específicas.

2.10.3. Herramientas de Recuperación de Arquitecturas

La recuperación de la arquitectura se apoya en un conjunto de herramientas integradas en la reutilización de desarrollo software basándose en el análisis dinámico. Con el fin de alcanzar mejor los objetivos propuestos, ArchMine [45] está integrado con otros dos enfoques de minería de datos, como ArqCheck [47], un enfoque de evaluación arquitectónica basada en la inspección, evaluación de la arquitectura y reutilización de Software ArchToDSSA [48], un dominio variabilidad detección. El objetivo es contribuir para la reutilización de software mediante la agrupación clases funcionalmente coherentes que representan conceptos del dominio y posiblemente los artefactos.

Softwarenaunt [7] es una herramienta que provee recuperación de arquitectura a través de exploración interactiva y visualización, tiene una arquitectura general de

[7] De acuerdo al SEI (Software Engineer Institute), una línea de productos de software se refiere a un conjunto de sistemas de software que comparten características y que son desarrollados a partir de un conjunto común de bienes núcleo.

recuperación de arquitectura de software donde se tienen tres pasos: la extracción de datos, la abstracción de la información y la visualización. Esta herramienta tiene muy en cuenta la descomposición jerárquica de un sistema, cuando hay muchas versiones ésta herramienta crea una historia modelando cada versión y permite visualizar la arquitectura. La herramienta permite la visualización de un grafo con diferentes grados de detalle visual. Aunque softwarenaunt hace la recuperación de la arquitectura de manera eficiente y permite al usuario navegar a través de esta, la herramienta presenta una fuerte dependencia de otra herramienta externa, la cual traduce el código del sistema a un modelo que pertenece a la familia de meta-modelos FAMIX [49], el cual fue creado para apoyar el intercambio de información entre las herramientas de análisis de software mediante la captura de las características comunes de los diferentes lenguajes de programación orientados a objetos.

IPlasma [50] es un entorno integrado para el análisis de calidad de los sistemas de software orientado a objetos que incluye soporte para todas las fases de análisis necesarios: desde la extracción de modelos (incluyendo análisis escalable para C ++ y Java) hasta el análisis basado en métricas de alto nivel, o la detección de la duplicación de código. iPlasma tiene tres grandes ventajas: la extensibilidad de análisis soportado, la integración con otras herramientas de análisis y capacidad de ampliación [50].

En la Figura 5, se puede observar la arquitectura de iPlasma, la cual cuenta con unos extractores de modelos, estos son los encargados de tomar el código fuente y construir el modelo MEMORIA de dicho código, para que en la capa de análisis que es donde se encuentran las herramientas encargadas de tomar el modelo y realizar el respectivo análisis para que en la capa de presentación se muestren los resultados de los análisis.

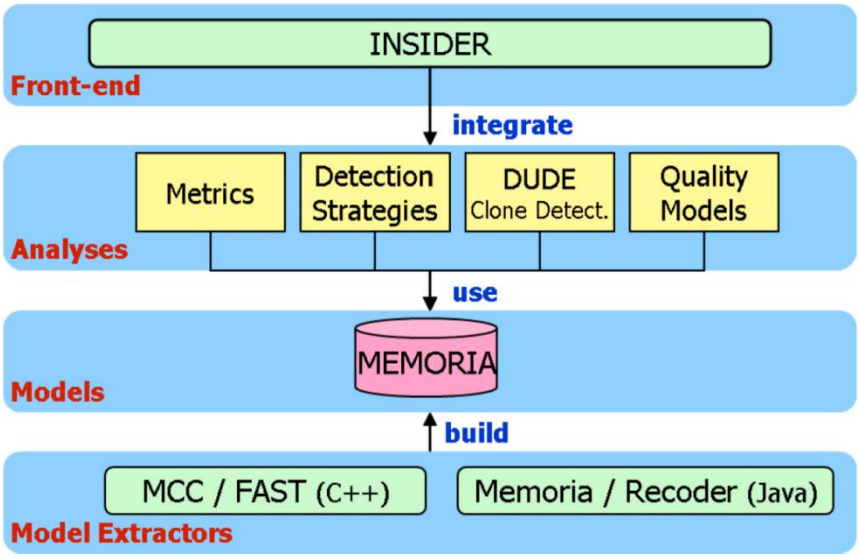


Figura 5. Arquitectura iPlasma

Para que iPlasma pueda hacer el análisis de un sistema software y pueda presentar resultados que sean relevantes para un grupo de interesados, este requiere que el código fuente pase por unos extractores, estos tienen como finalidad extraer del código fuente los datos de diseño más relevantes y traducirlos a un modelo llamado MEMORIA, ver figura 6, este es un meta-modelo que puede representar sistemas Java y C ++ de una manera uniforme, mediante la captura de la información de diseño (por ejemplo, clases, métodos). Una de las funciones clave de MEMORIA es proporcionar un modelo consistente, incluso en presencia de código incompleto o bibliotecas que faltan, para permitir el análisis de grandes sistemas y facilitar la navegación dentro de un sistema [51].

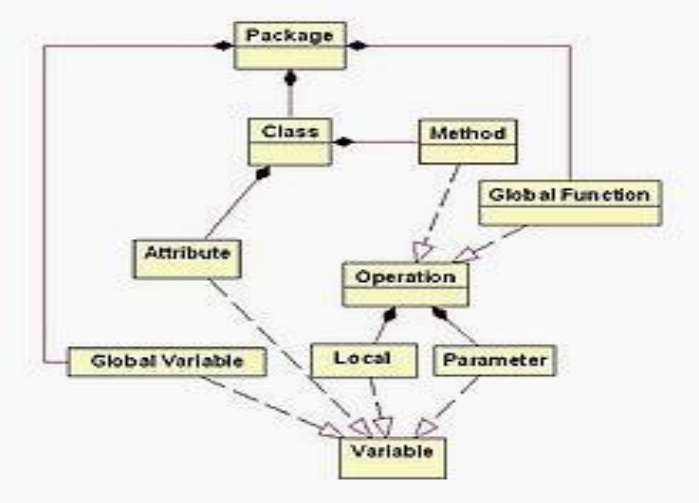


Figura 6. Metamodelo Memoria

Dentro de los resultados que produce iPlasma está la pirámide de visión general, ver figura 7, esta pirámide se utiliza para visualizar un sistema de software completo de una manera muy compacta. para ello se recoge un conjunto de métricas de las categorías de herencia, de acoplamiento y tamaño y complejidad, y las pone en relación [52].

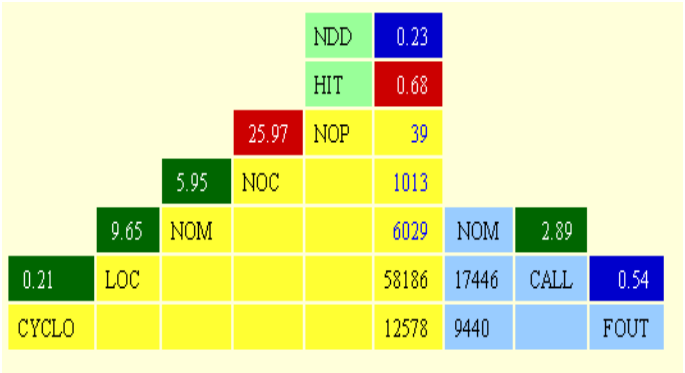


Figura 7. Pirámide de visión general

Esta pirámide muestra un grupo de métricas, ver Tabla 3, las cuales están compuestas por su valor y un color, el valor indica el valor de esta métrica y el color corresponde a un indicador de que tanto encaja esta métrica con respecto a la industria, en la Tabla 4, se muestran los rangos usados por la herramienta.

Tabla 3. Métricas de la pirámide

Sigla	Descripción
NDD	Numero de descendientes directos
HIT	Altura árbol de herencia
NOP	Numero de paquetes
NOC	Numero de clases
NOM	Numero de métodos
LOC	Líneas de código
CYCLO	Complejidad ciclomatica
CALL	Llamados por método

Tabla 4. Rango de métricas

Métrica	Bajo	Medio	Alto
CYCLO/LOC	0.16	0.20	0.24
LOC/NOM	7	10	13
NOM/NOC	4	7	10
NOC/NOP	6	17	26
CALLS/NOM	2.01	2.62	3.2
FANOUT/CALLS	0.56	0.62	0.68
ANDC	0.25	0.41	0.57
AHH	0.09	0.21	0.32

Moose es una plataforma de código abierto para la expresión de los análisis de los sistemas de software y de los datos en general, su objetivo principal es ayudar a los ingenieros a comprender grandes cantidades de datos, y los sistemas de software en particular, desde un punto de vista conceptual, Moose se organiza de la siguiente manera, ver figura 8.

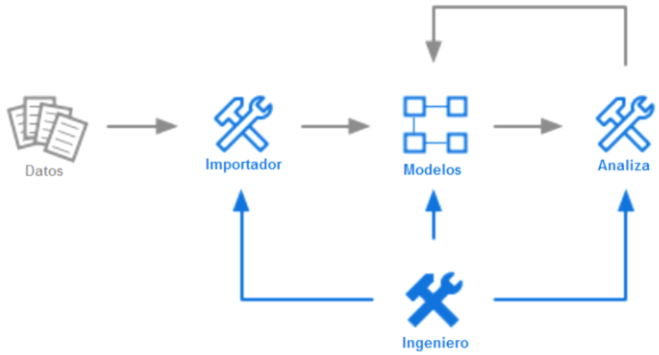


Figura 8. Flujo de trabajo general de Moose

Moose se dirige a varias categorías de personas:

- Investigadores en el área de ingeniería de análisis de software, minería de datos e ingeniería inversa.
- Ingenieros y arquitectos que quieren entender los sistemas y datos.
- Constructores/Desarrolladores de herramientas.

El insumo o entrada para el motor de Moose son datos. Por datos se entiende que son todo los tipos de estructuras que contienen objetos, propiedades y relaciones. Por ejemplo, un sistema de software escrito en Java, o bien, un conjunto de archivos de configuración escritos en XML, entre otros. Estos datos se cargan en Moose través de importadores. En Moose, se puede importar datos de diversas fuentes y en diferentes formatos. Por ejemplo, Moose puede manejar la importación de la estructura de los sistemas de software escritos en varios lenguajes de programación ya sea a través de importadores internos (por ejemplo, Smalltalk14, XML, MSE), o a través de las externas (por ejemplo, Java, JEE, C++).

Una vez importado, los datos se almacenan en los modelos con los cuales se pueden comenzar a realizar distintos tipos de análisis por parte de los ingenieros. Moose permite manipular grandes cantidades de datos, pero presenta un problema al momento de comprender y evaluar su estado, ya que los datos no tienen una forma física o visual entendible. La falta de forma gráfica de los datos hace inútil la habilidad humana para interpretar los modelos por medio de estímulos visuales. Mondrian, es un motor de scripting basado en Smalltalk que permite diseñar visualizaciones personalizadas de datos en una manera rápida dentro de la plataforma Moose [52].

Para realizar el análisis en este proyecto se hace uso de la familia de meta-modelos FAMIX, la cual familia de meta-modelos que sirve para la representación de modelos relacionados con las diversas facetas de los sistemas de software. Estos meta modelo permiten el análisis y proporcionan una API rica que se puede utilizar para consulta y navegación, ver Figura 9.

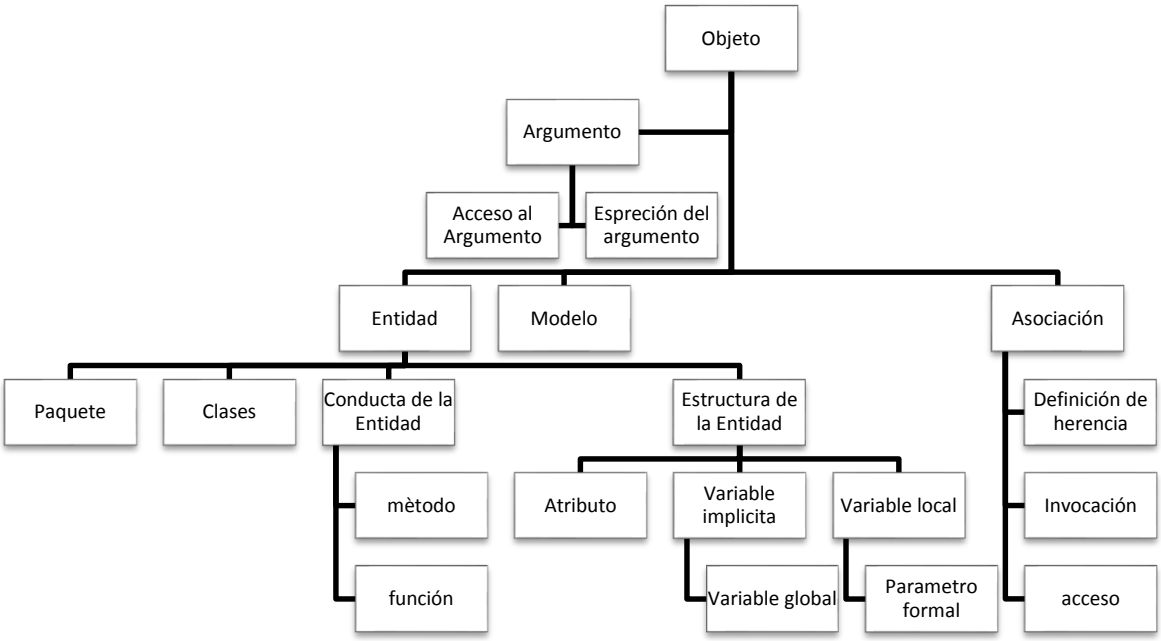


Figura 9. Familia de meta-modelos FAMIX

Una vez se ha hecho la extracción del código fuente, Moose provee una serie de funcionalidades, dentro de las cuales están [53]:

- Vista de Dependencia de espacio de nombres y paquetes

Es una vista polimétrica en el que cada nodo es un espacio de nombres o paquete y cada arco representa una dependencia. La gráfica se presenta en capas de tal manera que cada nodo se coloca debajo del nodo cliente, ver Figura 10. Por lo tanto, los espacios de nombres o paquetes de la parte inferior es el que se utiliza por muchos clientes, pero no es usa por ninguno y los de la parte superior son utilizados por ningún otro espacio de nombres o paquete.

Los nodos también presentan las siguientes métricas:

- La anchura viene dada por el número de clases (NOC) en el espacio de nombres o paquete
- La altura es proporcionado por el número de métodos (NOM) en el espacio de nombres o paquete.
- Complejidad del sistema

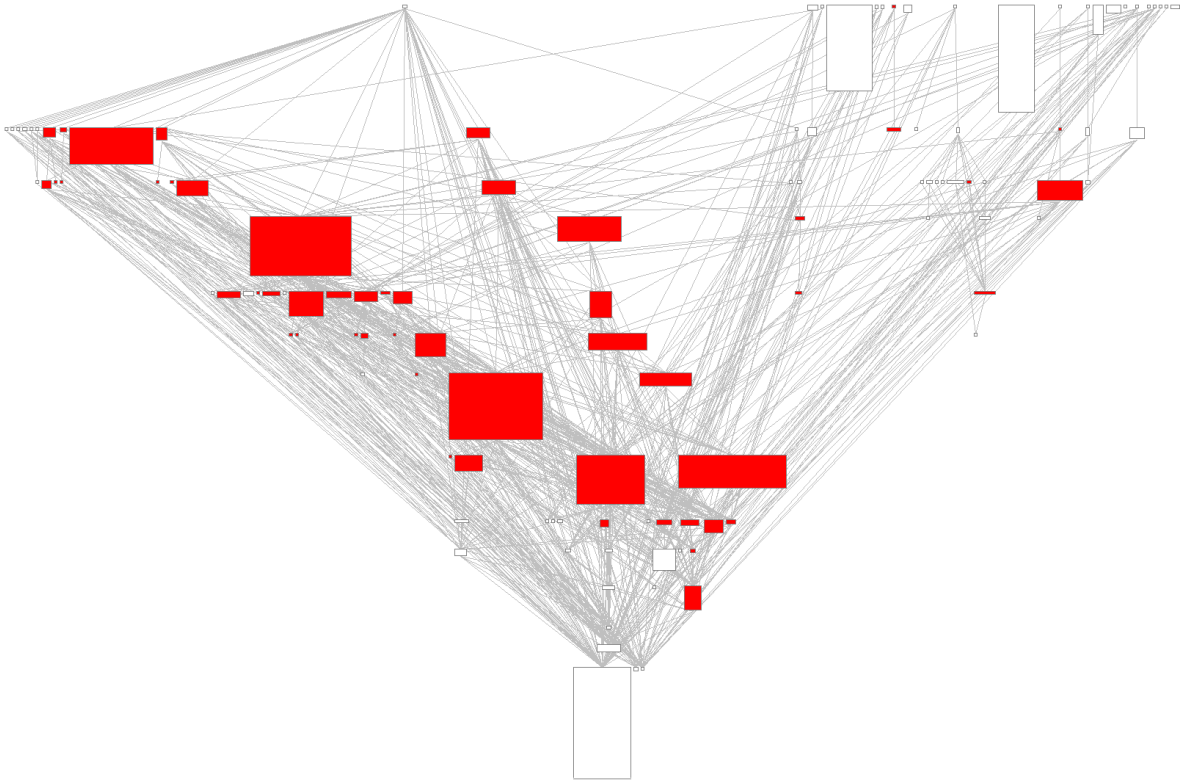


Figura 10. Vista de dependencias de espacio de nombres y paquetes

Es una vista que muestra un gráfico en el cual los nodos representan las clases y los nodos representan relaciones de herencia entre clases. Los nodos se enriquecen visualmente con tres métricas:

- El número de métodos (NOM) se asigna en la altura.
- El número de atributos (NOA) se asigna de la anchura.
- El número de líneas de código (LOC) se asigna en el color.

Moose ofrece una versión ligeramente mejorada. En primer lugar, para maximizar el espacio en pantalla, en lugar de mostrar todas las jerarquías junto a la otra horizontal, que se muestran en varias filas. En segundo lugar, las clases que no heredan de cualquier otra clase se colocan en un "clases Solitarios" grupo de la derecha.

Softwarenaut es una herramienta que provee recuperación de arquitectura a través de exploración interactiva y visualización, tiene una arquitectura general de recuperación de arquitectura de software donde se tienen tres pasos: como la extracción de datos luego la abstracción de la información, esta herramienta tiene muy en cuenta la descomposición jerárquica de un sistemas, cuando hay muchas

versiones ésta herramienta crea una historia modelando cada versión y permite visualizar la arquitectura.

La figura 11 muestra una vista de Softwarenaut, esta vista ofrece una visión de alto nivel de los módulos del sistema y las relaciones fundamentales entre ellos. Los módulos se representan como diagramas de árbol, y el tamaño de cada módulo es proporcional a su tamaño real en líneas de código; la anchura de cada relación es proporcional al número de invocaciones entre los módulos correspondientes. Las vistas producidas son interactivas, lo que permite inspeccionar los detalles o hacer zoom en los módulos y las relaciones individuales. Las visitas también pueden ser guardadas y compartidas [36].

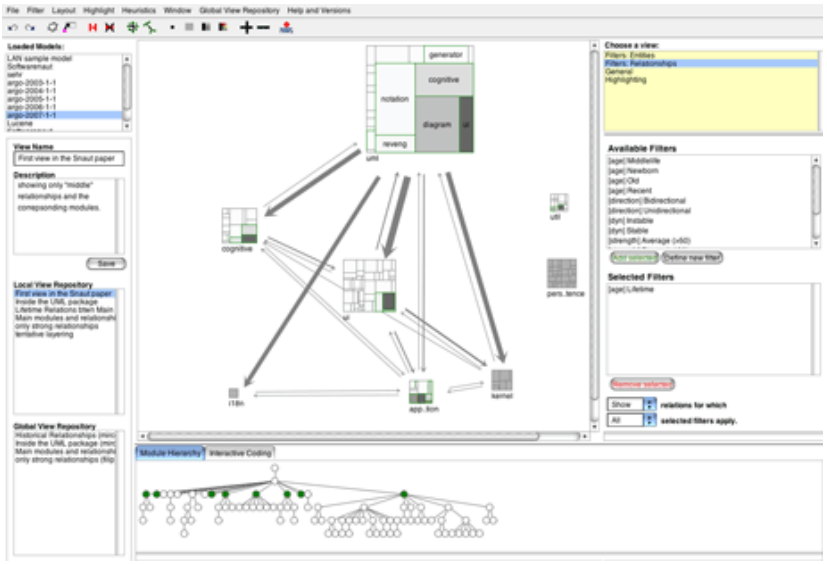


Figura 11. Vista principal Softwarenaut

2.10.4. Recuperación de Arquitecturas basadas en la Visualización

Muchas herramientas de ingeniería inversa son utilizadas para hacer una exploración top-down y así proporcionar un proceso visual e interactivo para la recuperación de la arquitectura, este proceso de exploración ayuda a mostrar diversos puntos de vista que en muchas ocasiones no detallan información relevante de la arquitectura [54].

Para ello se presenta una aproximación visual de forma interactiva para la recuperación de la arquitectura, basada en la información de patrones de paquetes. Los patrones se definen basándose en la estructura del paquete interno y en las relaciones entre paquetes, detectando los diferentes patrones y usando estos patrones para guiar al usuario en la exploración del sistema. Este enfoque fue validado en seis proyectos de código abierto observando que la actividad de recuperación de la arquitectura de un sistema software es compleja y lleva mucho

tiempo, por esto es necesario apoyarse en herramientas como Softwarenaut, Nenad y Vladimir [55], proponen un ligero enfoque de recuperación arquitectónica, llamado Foco, que tiene tres aspectos importantes.

En primer lugar, utiliza un sistema de requisitos de evolución incremental que permite recuperar sólo el fragmento de la arquitectura del sistema afectado por la evolución, ayudando a que el enfoque tenga su atención en partes afectadas o impactadas en el sistema, en segundo lugar, además de componentes de software, que son los habituales de los enfoques de recuperación, también recupera partes claves en la arquitectura de software, conectores y estilos arquitectónicos, este enfoque se basa en la recuperación y evolución de las arquitecturas de aplicaciones OO indocumentadas, permitiendo que mejore la mantenimiento y la evolución del sistema. En esta misma línea está Moose [53], una plataforma de código abierto para la expresión de los análisis de los sistemas de software y de los datos en general, su objetivo principal es ayudar a los ingenieros a comprender grandes cantidades de datos y los sistemas de software en particular desde un punto de vista conceptual.

Según G. Y. Guo et al. [56], la recuperación de la arquitectura de software se puede dividir en dos fases: identificación y extracción de los artefactos de código fuente, incluyendo los elementos arquitectónicos, y análisis de los artefactos de origen extraídos para derivar una vista de la arquitectura implementada.

Existen muchas herramientas de extracción como lo son LSME, ManSART y Magix las cuales son útiles para la extracción y el análisis de las arquitecturas de software, al igual que Rigi, CIA y SAAMTool, proporcionan no sólo la visualización sino también mecanismos de manipulación para ayudar al usuario a simplificar y navegar a través de la representación del sistema visual. El banco de trabajo Dali [56] es una infraestructura para la integración de una gran variedad de arquitecturas, el cual permite la extracción, manipulación, análisis y presentación de herramientas.

Las herramientas de visualización se pueden implementar en Dali para presentar el modelo de la fuente y el resultado de análisis de la arquitectura. Dali soporta diversos tipos de manipulación externa y herramientas de análisis, como Grok, IAPR y RMTool Para ayudar recuperación de la arquitectura de software de los sistemas diseñados y desarrollados con los patrones, en este trabajo también utilizan el Método (ARM) Método de Reconstrucción de Arquitectura que se basa en el análisis semi-automático para la reconstrucción de arquitecturas basado en el reconocimiento de patrones arquitectónicos. ARM se compone de cuatro fases que son: el desarrollo de un plan de reconocimiento de patrones concretos, la extracción de un modelo de origen, el detectar y evaluar patrones y la reconstrucción y análisis de la arquitectura Figura 12.

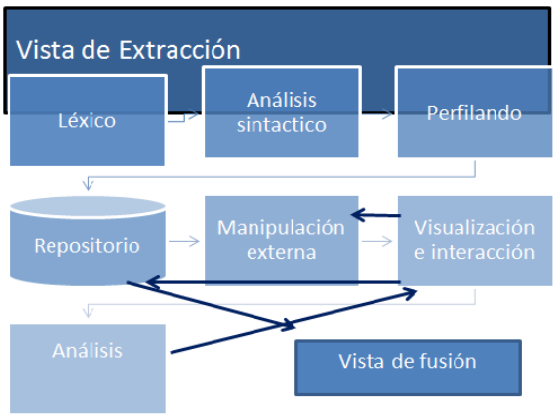


Figura 12. El Dali Workbench

2.10.5. Recuperación de Arquitecturas y Evaluación de Arquitecturas.

Vasconcelos y Werner [57], presentan un enfoque de recuperación y evaluación de arquitectura para la reutilización soportado por la herramienta ArchMine la cual permite extraer el conocimiento basado en la inspección de sistemas existentes. Dicho conocimiento es expresado en un modelo arquitectónico conceptual con información de trazabilidad hacia las funcionalidades implementadas, ayudando a generar artefactos reutilizables. Christoph et al. [58], definen un método llamado MAP (minería de arquitecturas para líneas de producto), tiene un enfoque que se basa en el mapeo de estilos arquitectónicos y atributos que se relacionan con la minería de la arquitectura, donde se combinan la reconstrucción de la arquitectura y el análisis de técnicas de líneas de productos. Si la vista arquitectónica después de haber realizado la evolución no es factible, se procede a hacer un análisis con ATAM. También se muestra un estudio de caso donde se hace el análisis usando el método MAP en varios sistemas en la industria automotriz, dando lugar a mejoras en la extracción donde se identifican elementos y relaciones que ayudan a mejorar el MAP.

R. R. Lutz and G. C. Gannod [59], describen las experiencias con la especificación de arquitectura y análisis arquitectónico por una herramienta asistida de alto rendimiento de la línea de productos de software. El enfoque utilizado, define una arquitectura de línea de productos bueno en términos de calidad de los atributos requeridos en fase de desarrollo en particular. El modelo arquitectónico se analiza respecto a varios criterios como métodos manuales y herramientas soportadas. El enfoque descrito ofrece un análisis estructurado de una arquitectura de referencia de una línea de productos utilizando la recuperación de la arquitectura y las especificaciones, la evaluación de la arquitectura, y la comprobación de modelo de comportamiento para determinar el nivel de robustez y tolerancia a fallos a nivel de arquitectura que son necesarios en todos los sistemas de la línea de productos. Los

resultados de una aplicación a una línea de productos de software de telescopios espaciales se utilizan para explicar el enfoque y describir las lecciones aprendidas.

Describen los tres principales pasos en el proceso de análisis arquitectónico donde se tomó como línea base un modelo ADL que se utilizó para evaluar la adecuación y el nivel de soporte proporcionado por la arquitectura utilizado, para especificar los diferentes requerimientos de los sistemas individuales en línea de productos, se analizaron diferentes métodos como SAAM y ATAM ya que el enfoque es centrado en los atributos de calidad que son específicos en las arquitecturas línea de productos, utilizando ACME con el apoyo de una herramienta de especificación de arquitectura, ACMEStudio, para la construcción y la manipulación gráfica de arquitecturas de software y el análisis automatizado fue principalmente para analizar los diferentes atributos de calidad, como la robustez y tolerancia a fallos, que fueron vistos como comunes en todos los miembros de la línea de productos [59].

A continuación la Tabla 5, sintetiza los principales factores comparativos considerados respecto a los trabajos relacionados con la recuperación de arquitecturas de software.

De los trabajos relacionados encontrados a través de artículos puede concluirse que existen muchos trabajos dedicados a la recuperación de arquitecturas de sistemas software, donde hay una visión clara entre usar o no a los interesados de recuperar la arquitectura de un sistema, también podemos concluir que estos pretenden producir resultados visuales para los usuarios, nuestro trabajo no solo pretende la inclusión de los interesados en la recuperación de una arquitectura, también queremos recuperar las decisiones arquitectónicas que se tuvieron en cuenta en la construcción de esta arquitectura para lo cual usamos mecanismos visuales para que las persona involucradas en el proceso puedan analizar e inferir en las decisiones arquitecturales que se tomaron que nunca fueron documentadas. Nuestra propuesta está enfocada en la creación de un método explícito y se presentan casos de aplicación con variados escenarios que permiten realizar la recuperación del rationale y la visualización con las herramientas de recuperación.

Tabla 5. Principales factores comparativos considerados respecto a los trabajos relacionados con la recuperación de arquitecturas de software

Nombre	Tipo	Propuesta	Requiere Interesados	Artefactos de Entrada	Artefactos de Salida
ArchDRH	Herramienta	Proponen un algoritmo en el cual se identifican los módulo de funciones, se identifican lo componentes compartidos y por último se identifican lo módulos conceptuales.	No	Código Fuente	Modelo Conceptual
Using fold-in and fold-out in the architecture recovery of software systems	Técnica	En este trabajo el autor propone un enfoque el cual consiste en 2 fases, en la primera se realiza el cálculo de la disimilitud de cada una de las entidades del software y en la segunda fase e realiza el agrupamiento de las entidades dependiendo del valor de disimilitud.	No	Código Fuente	Matriz de semejanza de entidades
Towards the unified recovery architecture for reverse engineering	Proceso	En este trabajo se define un proceso para ingeniería inversa, donde se realiza el análisis de los programas, e filtra y comprende la información y por último se realiza la visualización del programa.	Si	Código Fuente	Vistas Arquitectónicas
Towards Open Source Software System Architecture Recovery Using Design Metrics	Enfoque	Los autores proponen una metodología para dividir la arquitectura de un software en 4 capas.	Si	Código Fuente	Descripción estática de la arquitectura
Recovery of Software Architecture Using Partitioning Approach by Fiedler Vector and Clustering	Técnica	En este trabajo se propone un método en el que se particiona la arquitectura en módulos y luego estos son agrupados usando medidas de similitud o disimilitud.	Si	Código Fuente	Grafico
Software Architecture Recovery Through Similarity-Based Graph Clustering	Técnica	Proponen un método el cual permite agrupar los elementos del sistema dependiendo de la similitud de estos.	No	Código Fuente	Gráfico de Agrupamiento
ArchMine	Herramienta	En este trabajo se propone un enfoque para recuperar la arquitectura de un software mediante la comprensión de este y la detección de clases cohesivas.	No	Código Fuente	Modelo Arquitectónico

Softwarenaut	Herramienta	En esta propuesta se realiza la recuperación de la arquitectura de un sistema mediante la extracción de los datos del sistema, la abstracción de la información y su visualización.	No	Código Fuente	Visualización de la arquitectura de un sistema
iPlasma	Herramienta	Esta propuesta es un entorno para el análisis de la calidad de un sistema software, el cual permite obtener métricas de alto nivel y la detección de duplicación de código.	No	Código Fuente	Grafos y Tablas
Package Patterns for Visual Architecture Recovery	Enfoque	En este trabajo se propone una aproximación visual que sea interactiva para recuperar la estructura de paquetes de un sistema, este enfoque esta implementado en la herramienta Softwarenaut.	Si	Código Fuente	Visualización de un árbol de jerarquías de los paquetes de un sistema
Focus	Enfoque	Los autores proponen un enfoque ligero que permite la recuperación de fragmentos de la arquitectura de un sistema.	Si	Código Fuente	Diagrama de Arquitectura evolucionada
Moose\Agile Visualization	Plataforma\Enfoque	Esta plataforma permite el análisis de datos en general, estos datos pueden ser de procesos o de sistemas software, lo que permite la comprensión de grandes cantidades de datos.	No	Código Fuente	Grafos
Architecture Recovery and Evaluation Aiming at Program Understanding and Reuse	Enfoque	Los autores presentan u enfoque de recuperación y evaluación de arquitecturas el cual es soportado por ArchMine.	Si	Escenarios de casos de uso, código	Modelo
MAP	Método	En este trabajo proponen un método basado en el mapeo de estilos arquitectónicos y atributos que se relacionan con la minería de la arquitectura.	Si	Conocimiento acerca de la línea de productos de la organización	Evaluación de las similitudes y variabilidades
EV-AR	Método	Basado en una guía para arquitecto de software que permite recuperar arquitecturas de sistemas por medio de un enfoque evaluativo y mecanismos de recuperación visual donde permite que un arquitecto en forma práctica e intuitiva recupere el conocimiento arquitectónico de una aplicación ya implementada.	SI	Artefactos, código fuente	Rationale y vistas arquitectónicas

MÉTODO DE RECUPERACIÓN DE ARQUITECTURAS BASADO EN LA FOCALIZACIÓN EVALUATIVA Y LA VISUALIZACIÓN DE SOFTWARE EV-AR

3.1. INTRODUCCION

En esta sección, se detalla el método EV-AR que corresponde al modelo empírico propuesto en esta tesis para conducir de una forma práctica y liviana la recuperación de arquitecturas de software,

EVAR es un método de recuperación de arquitecturas a partir del conocimiento tácito de la organización, capturado a través de la interacción con los diferentes stakeholders involucrados en el desarrollo de la arquitectura y los artefactos que la describen en forma explícita o implícita, tales como modelos, requerimientos, código y pruebas. La información recuperada hace referencia a vistas, escenarios, drivers, tácticas y decisiones, es decir: además de recuperar vistas que representan la arquitectura se recupera el “rationale” que fue aplicado para obtener la arquitectura bajo recuperación.

3.2. ENFOQUE

El enfoque combina aspectos propios de los métodos de evaluación arquitectónica, con los enfoques de recuperación basada en la visualización de software, con la intención de no sólo recuperar vistas del sistema sino el conocimiento asociado a las decisiones tomadas para obtener la arquitectura hasta el momento de la recuperación. En este método se hace uso de escenarios para identificar comportamientos esperados en el sistema software, que han ayudado en el análisis, diseño y evaluación [60][61], arquitecturas de software, El uso de escenarios ha sido considerado porque de acuerdo a Krutchen [62], el uso de una vista de escenarios permite definir, validar y hacer consistentes las demás vistas. En uno de los estudios empíricos preliminares del método propuesto se usaron los escenarios como ayuda a las tareas de recuperación de software resultando de gran utilidad [60]. En este caso se hizo uso del método de evaluación ATAM para recuperar arquitecturas de software. El uso de las visualizaciones permite generar distintos puntos de vista sobre un escenario en específico, además de ser un mecanismo de ayuda para que los integrantes del proceso, especialmente los que hicieron parte del proceso de desarrollo, analicen el sistema y que de manera más natural provean información sobre su funcionamiento y de los componentes que lo conforman, por ultimo

identificar las diferentes decisiones arquitecturales que se tomaron para la construcción su construcción.

3.3. MÉTODO

EV-AR es un método para ayudar a las personas interesadas en la recuperación de la arquitectura de un sistema software, este involucra una serie de pasos donde los involucrados en el proceso de recuperación van a definir un objetivo para el cual necesitan recuperar la arquitectura de un sistema, y a partir de este objetivo se van a identificar las partes importantes del sistema que se desean analizar para obtener como resultado la descripción de la arquitectura de estas. Para realizar el análisis el método propone el uso de las vistas arquitecturales del sistema, las cuales pueden ser recuperadas usando herramientas de visualización de arquitecturas. El método EV-AR, como se describe en la Figura 13, está conformado por 7 tareas.

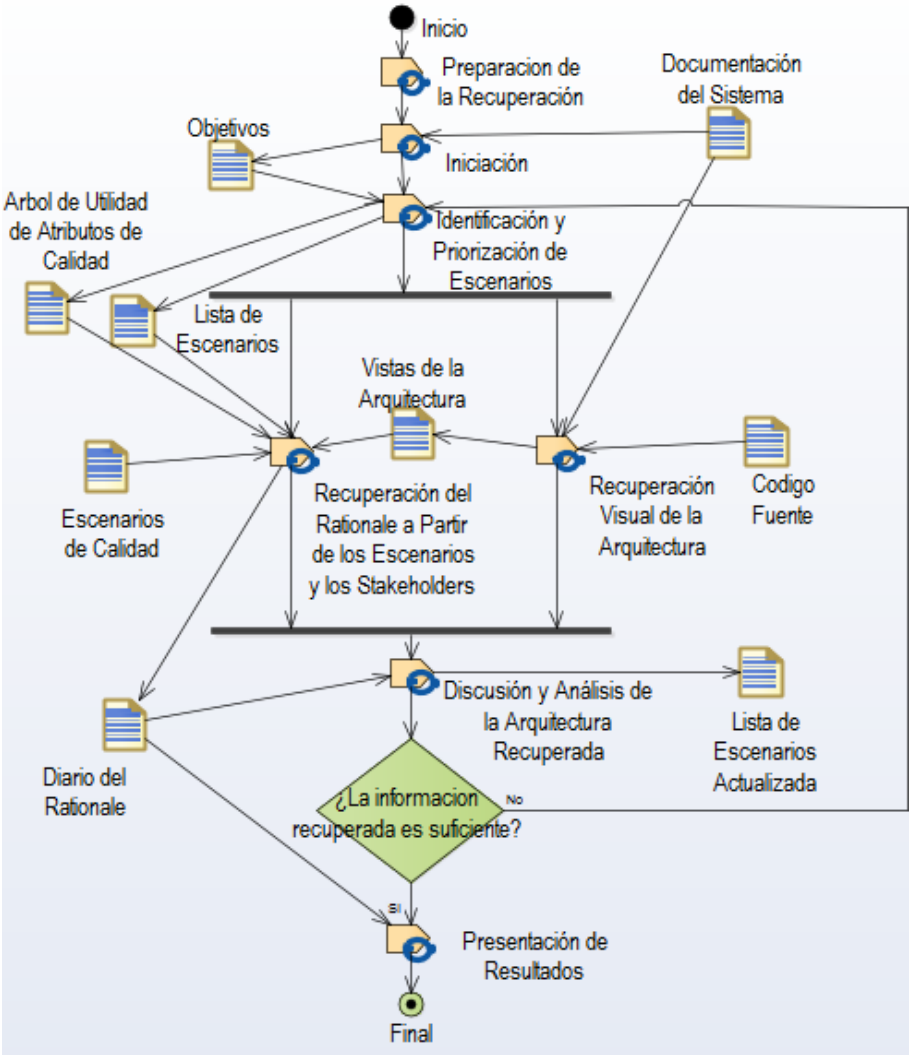


Figura 13. Método EV-AR

3.3.1. Preparación de la Recuperación

El objetivo de esta tarea es preparar todos los elementos necesarios para el método, los cuales son: los involucrados, los objetivos de la recuperación, las fuentes de información y las herramientas de visualización. El tener listos estos elementos nos permiten realizar una planificación detallada, teniendo en cuenta la disponibilidad de las personas que participan en la recuperación. Esta tarea consiste de 5 pasos:

- Paso 1. Identificar las necesidades de la recuperación, para esto se debe indagar sobre las necesidades que se tienen para tener que recuperar la arquitectura de un sistema.

- Paso 2. Identificar los interesados, los cuales son las personas que van a estar encargadas de trabajar con el sistema ya sea para una modificación, un soporte o una corrección de errores.

- Paso 3. Identificar el equipo de desarrollo. Para conformar este equipo se deben identificar las personas que han estado en el desarrollo del sistema a recuperar, además de incluir aquellas personas que influyeron en la toma de decisiones sobre el desarrollo de este.

- Paso 4. Preparación de la recuperación. En este paso se identifican las herramientas de recuperación visual que se van a utilizar, para esto se debe realizar una búsqueda de las herramientas disponibles y se realiza una caracterización de cada una estas, con esas características el arquitecto selecciona la herramienta adecuada para el sistema que está bajo recuperación, para esto se aplican tres 3 filtros a la lista de herramientas (técnica construida en este método para soportar este paso):

    - El primer filtro es el correcto funcionamiento de la herramienta, dentro de la gran lista de herramientas que se encuentran, gran parte no funciona desde la descarga directa o es difícil hacerla funcionar, hay otras que dependen de otras herramientas o plataformas y lenguajes específicos para que funcionen, por ese motivo hay que establecer claramente si las herramientas se pueden usar y además que los resultados que arrojan sean comprensibles y relevantes al objetivo de la recuperación.

    - El segundo filtro es que la herramienta sea adecuada para el sistema al cual se desea recuperar su arquitectura, cada herramienta está dirigida a la recuperación visual de sistemas con características particulares, puede ser el código bajo un lenguaje específico e incluso a una versión en específico del lenguaje de programación en el que fue escrito el sistema o el tipo de aplicación.

- El tercer filtro es el tipo de resultados que se desean obtener en la recuperación visual, ya que incluso hay herramientas que solo permiten la exploración del sistema y no se pueden obtener resultados fácilmente.

Si no encontramos una herramienta adecuada para la recuperación del sistema, contamos con un último recurso para hacer esto, la recuperación manual del sistema, pero esto tiene como consecuencia que el proceso de recuperación puede llegar a consumir muchos recursos bien sea humanos como de tiempo.

- Paso 5. Realizar cronograma de actividades, el objetivo de este paso es la construcción de un cronograma de las tareas del proceso de recuperación teniendo en cuenta la disponibilidad de los participantes de este, cada una de las actividades debe detallar cual es la tarea a realizar, el tiempo esperado que dura, el responsable y los asistentes a esta, es importante que cada uno de los participantes conozcan este cronograma para que eviten faltar a alguna de las tareas que tienen en el proceso.

3.3.2. Iniciación del Método

La iniciación tiene como objetivo el contextualizar a todos los participantes en el proceso de recuperar la arquitectura de un sistema, conocer cómo funciona el sistema al cual se le va hacer la recuperación de su arquitectura y cuáles son las ideas arquitectónicas que se conocen sobre este. Esta tarea consiste de tres pasos:

- Paso 1 Realizar la presentación del método EV-AR para que los participantes de esta actividad tengan claros los pasos del método y cuál es el objetivo que se tiene en cada paso, en este espacio se resuelven todas las preguntas que se tengan sobre el método.

- Paso 2. Presentación del sistema y las ideas arquitectónicas presentes, donde se realiza la presentación del sistema en funcionamiento.

- Paso 3. Consiste en definir los objetivos de la recuperación donde todos los interesados del sistema exponen sus motivaciones para que se realice la recuperación de la arquitectura del sistema.

En la Tabla 6 se relacionan los artefactos que son requeridos en la tarea y los cuales son producto final de la ejecución de esta.

Tabla 6. Iniciación del método de entrada y salida

Artefactos de Entrada	Artefactos de Salida
Documentación del sistema	Objetivos de la recuperación

3.3.3. Identificación y Priorización de Escenarios

Aquí se pretende realizar la identificación de los escenarios que ayudarán al arquitecto en la recuperación de la arquitectura de un sistema, esto consiste en dos pasos: la identificación de escenarios y la priorización de escenarios.

- Paso 1. Identificación de escenarios: se identifican los escenarios, que van a ser los objetivos o inquietudes que el arquitecto tiene sobre el sistema, el hecho de usar los escenarios para la recuperación de la arquitectura permite al arquitecto concentrarse en un único aspecto del sistema, Kazman et al [63], identifican tres tipos de escenarios, los escenarios de uso, donde se describe como un sistema cumple una funcionalidad, escenarios de crecimiento donde se describe los cambios que ha sufrido el sistema, y escenarios exploratorios donde se expone el sistema a sus límites. El uso de estos tres tipos de escenarios se realiza de acuerdo a las necesidades de recuperación de la arquitectura. Los escenarios de uso le permiten al arquitecto de software entender cómo las partes del sistema interactúan para cumplir una funcionalidad específica, los escenarios de crecimiento nos permiten ver cómo el sistema está construido y de qué forma ha sido evolucionado o modificado, y los escenarios exploratorios permiten entender el sistema para modificarlo e incluso para ver qué tan apropiado puede ser este sistema en caso de querer reutilizarlo.

- Paso 2. Priorizar los escenarios: teniendo en cuenta su importancia y la dificultad de implementación, los escenarios son priorizados. Esta priorización se presenta en un árbol de utilidad de atributos de calidad, para la construcción de este se debe identificar la raíz del árbol, la cual va ser el sistema a recuperar, una vez se tiene la raíz se debe recorrer la lista de escenarios el cual es resultado del paso anterior, cada uno de los escenarios debe ser asociado a un atributo de calidad, una vez tenemos el escenario asociado a un atributo se presentan dos casos: El primero es que el atributo calidad no se encuentre en el árbol que estamos construyendo, en esta situación este nuevo atributo pasa a ser una nueva rama del árbol y el escenario será una hoja de esta rama. En el segundo caso el atributo ya se encuentra en el árbol por lo que debemos identificar la rama perteneciente a este atributo de calidad y asociarle una nueva hoja la cual es el escenario que se está priorizando. Cada uno de los participantes califica el escenario con respecto a dos criterios: Dificultad de Implementación e importancia. El criterio dificultad de implementación refiere al nivel de complejidad que pudo tener el desarrollo del escenario, y el segundo criterio refiere a la importancia

que tiene escenario para el cumplimiento de los objetivos que se tienen en el proceso de recuperación.

Para la calificación de cada criterio se indica un valor de la escala de High, Medium, Low (H, M, L), se establece el uso de esta escala debido que para las personas encargadas de realizar esta calificación es difícil calificar en una escala más precisa, por ejemplo una escala de 0 a 10, en cambio es más natural si se usa la escala High, Medium, Low.

En la Tabla 7 se muestra los artefactos que son necesarios para ejecutar esta tarea y cuales son aquellos que se generan durante la realización de esta.

Tabla 7. Identificación y Priorización de escenario, entrada y salida

Artefactos de Entrada	Artefactos de Salida
Objetivos de la recuperación	Lista de Escenarios
	Árbol de Utilidad de Atributos de Calidad

3.3.4. Recuperación Visual de la Arquitectura

En esta tarea se pretende recuperar visualmente la arquitectura del sistema a partir de las diferentes fuentes de información que se tengan del sistema a recuperar, esta recuperación se debe realizar enfocándose en los escenarios identificados y que fueron priorizados, cada una de estas vistas generadas son evaluadas con respecto a la consistencia de la arquitectura preliminar con la que se cuenta. Esta tarea consiste de 4 pasos:

- Paso 1. Cargar las fuentes*:* la carga de fuentes se realiza dependiendo de la herramienta que se va utilizar, debido a esto debemos preparar estas fuentes para su uso, partiendo de la eliminación de la información que no es requerida para la recuperación visual, es importante que estas fuentes no llenen de información innecesaria el mecanismo que se va usar para la extracción visual, ya que puede aplicar carga y demorar el tiempo que se utiliza para esta tarea.

- Paso *2.* Extraer vistas arquitectónicas a partir de la herramienta de visualización: se realiza la extracción visual de la arquitectura del sistema usando el mecanismo definido para la recuperación de arquitecturas, es necesario contar con los escenarios priorizados que se obtuvieron en el paso anterior para que el equipo encargado de realizar esta tarea tenga claro que tipos de vistas arquitectónicas debe recuperar y a cuales componentes del sistema de realizar la tarea. La idea es seleccionar las vistas relevantes a los escenarios priorizados.

- Paso 3*.* Extraer las vistas arquitecturales a partir de otras fuentes: Aquí se realiza una exploración de la documentación que se tiene sobre el sistema

para identificar vistas que ya se tengan de la arquitectura y que sean importantes para los escenarios que se están trabajando.

- Paso 4. Verificar consistencia e integridad conceptual de las vistas arquitecturales: En este paso se evalúan las vistas generadas con las vistas que se encontraron en el paso anterior, cada vista se debe asociar a un escenario, en caso de identificar varias vistas para un escenario se deben evaluar e identificar las diferencias que hayan entre las vistas y generar vistas con información más completa sobre cada escenario

En la Tabla 8 se listan los artefactos que son necesarios para realizar esta tarea y cuales son aquellos artefactos resultantes de la ejecución de esta.

Tabla 8. Recuperación Visual, entrada y salida

Artefactos de Entrada	Artefactos de Salida
Código Fuente	Vistas de la Arquitectura
Documentación del Sistema	

3.3.5. Recuperación del Rationale a partir de los escenarios y los stakeholders

En esta tarea el arquitecto revisa cada uno de los escenarios ordenados por prioridad, toma cada escenario y lo asocia a la vista o vistas arquitecturales relacionadas con el escenario. Para cada vista se deben identificar los componentes que hay y analizarlos, este análisis depende del tipo de escenario.

- En el caso de ser una escenario de uso, se debe identificar la responsabilidad de cada componente en el cumplimiento del escenario al cual se ha asociado, con que otros componentes interactúa y extraer información que se pueda tener sobre el componente, por ejemplo: métricas y comentarios sobre los componentes, que las personas involucradas en el desarrollo del sistema puedan hacer.

- Cuando el escenario es de crecimiento, hay un requerimiento especial y es que los desarrolladores que estuvieron involucrados en el desarrollo del sistema o que han participado en la evolución del sistema estén presentes, estos participantes pueden dar información importante sobre cómo fue evolucionado el sistema, cuáles fueron las otras alternativas que se tuvieron en cuenta y por qué se tomó la decisión de realizar la modificación. En caso de no contar con este requerimiento se definen escenarios posibles de cambio potencialmente viables en la evolución del sistema.

- Por último si el escenario es de tipo exploratorio además de la identificación de los componentes, también se podría identificar como está construido el

sistema para soportar el escenario que se está planteando. El objetivo principal de esta evaluación es que las personas que hicieron o hacen parte del desarrollo del sistema al cual se le está recuperando la arquitectura suministren información adicional sobre como fue el desarrollo de los componentes con los cuales se evalúa el escenario, por qué se construyó de esa forma y si es posible que realicen comentarios sobre las razones que tuvieron para no hacer el desarrollo de otra forma. En este paso se construye el diario del rationale para cada escenario donde se incluyen las vistas obtenidas y las decisiones arquitecturales que se identifican sobre el escenario.

En la Tabla 9 se relacionan los artefactos que son necesarios para la ejecución de la tarea, y los artefactos que son generados durante la ejecución de esta.

Tabla 9. Recuperación del Rationale a partir de los escenarios y los stakeholders, entrada y salida.

Artefactos de Entrada	Artefactos de Salida
Lista de Escenarios	Diario del Rationale
Árbol de Utilidad de Atributos de Calidad	
Vistas de la Arquitectura	
Escenarios de Calidad	

3.3.6. Discusión y análisis de la Arquitectura Recuperada

El propósito de esta tarea es discutir los resultados obtenidos y que las personas involucradas en la recuperación de la arquitectura del sistema, puedan determinar si la información de la arquitectura recuperada es suficiente. Aquí se revisan cada uno de los escenarios y se determina si es claro o no; un escenario se considera que esta claro cuando todos los involucrados del proceso determinan que la información que se obtuvo sobre este, es entendible y muestra la información necesaria para solucionar las dudas que se presentaban sobre este escenario, es necesario que todos estén conformes con el resultado que se obtuvo de no ser así este escenario debe ser sometido a un nuevo ciclo en el proceso. En el caso que se considere que los resultados no son suficientes en amplitud o profundidad, se generarán nuevos escenarios y se vuelve a la tarea Identificación y Priorización de Escenarios para volver a analizar los escenarios que fueron declarados como no claros e incluir los nuevos escenarios.

En la Tabla 10 se listan los artefactos que son necesarios para la ejecución de la tarea, y los que son generados durante la ejecución de esta. En este paso el artefacto “Lista de Escenarios” es opcional ya que este solo es generado cuando se determina que los resultados obtenidos en el proceso de recuperación nos son suficientes para los interesados.

Tabla 10. Discusión y Análisis de la Arquitectura Recuperada, entra y salida

Artefactos de Entrada	Artefactos de Salida
Diario del Rationale	Lista de Escenarios

3.3.7. Presentación de Resultados

En esta actividad el equipo de recuperación realiza una presentación de los resultados. Esta debe ser una presentación formal, donde se describa lo que se hizo, las herramientas usadas para la recuperación y los resultados obtenidos. En esta presentación se pueden involucrar nuevos interesados en el sistema, con el objetivo de generar nuevas necesidades, para poder ejecutar el proceso de recuperación con nuevos objetivos y obtener nuevos resultados.

En la Tabla 11 se listan los artefactos que son necesarios para la ejecución de la tarea.

Tabla 11.Presentación de Resultados, entrada y salida.

Artefactos de Entrada	Artefactos de Salida
Diario del Rationale	

3.4. ARTEFACTOS

El objetivo de los artefactos de EV-AR es ayudar a los participantes del método a concentrarse en los resultados concretos de la recuperación, además de ayudar a dar orden al momento de analizar el sistema y de consignar los resultados que se puedan obtener durante la ejecución del mismo.

A continuación se muestran los detalles de estos:

- *Código Fuente:* El código de la aplicación, este código debe estar completo, y debe ser depurado para eliminar los archivos que no son necesarios para la recuperación.
- *Documentación del Sistema:* Se refiere a la documentación existente sobre el sistema, esta puede ser: los casos de uso, casos de prueba, manuales, tutoriales y otros documentos que describan el funcionamiento del sistema, o cómo fue construido este, importante si se cuentan con documentos de arquitectura del sistema.
- *Lista de Escenarios:* Este artefacto comprende una lista de escenarios que describan el comportamiento que debe tener el sistema frente a determinadas situaciones.
- *Escenarios de Calidad:* Corresponde a una descripción detallada de los escenarios que se tienen en el artefacto Lista de Escenarios, esta descripción detallada debe ser realizada por los integrantes del proceso de recuperación, se recomienda documentar los escenarios como se hace en el método QAW.

- *Árbol de utilidad de atributos de calidad:* Este permite realizar una relación de los escenarios con los atributos de calidad además que se realiza un priorización de estos con respecto a dos criterios, la importancia del escenario en el sistema y la dificultad de su implementación, para la calificación cada uno de los integrantes del equipo debe dar una calificación de cada escenario y se debe llegar a un consenso en la calificación final. En la Tabla 12 se presenta un ejemplo de este artefacto.

Tabla 12. Árbol de Atributos de Calidad.

	Atributo de Calidad	**Escenario**	**Prioridad**
Sistema	Escalabilidad	Almacenamiento	(H, M)
		Crecimiento de usuarios	(H, L)
	Integrabilidad	Integración de nuevas plataformas	(H, H)
	Modificabilidad	Evolución del Servidor de aplicaciones	(H,H)
		Tiempos de respuesta	(H, L)
	Desempeño	Fallos en el proceso de resolución de conflictos	(H, H)
	Confiabilidad	Fallos en el servidor de aplicaciones	(H, L)
		Fallos en el servidor de Almacenamiento	(H, L)

- *Vistas de la arquitectura:* Este es la recopilación de las vistas obtenidas usando una herramienta de recuperación de arquitecturas o mediante la recuperación manual, cada una de las vistas debe ser generada con la intención que ayuden en la descripción de un escenario previamente identificado.
- *Diario del Rationale:* Este consiste en la recuperación de cada escenario con una descripción detallada donde se especifica el estímulo que genera el escenario, la respuesta que debe tener el sistema, además de las diferentes decisiones arquitecturales de se identifican para que el sistema cumpla con el escenario, a cada una de estas decisiones se debe identificar los puntos sensibles, los puntos de negociación y los riesgos, y por último, se incluye el rationale y una vista donde se identifican los componentes del sistema que atacan el escenario planteado. (Ver ANEXO A. Apartado 1) se presenta la plantilla de este artefacto.

3.5. ROLES

EV-AR cuenta con cuatro roles los cuales agrupan a los integrantes del proceso de recuperación dependiendo de la información que tienen sobre el sistema. A continuación se describen los roles que actúan durante la ejecución del método.

- *Equipo de Recuperación:* Los integrantes de este equipo tienen la responsabilidad de ejecutar y controlar el método, ellos van a tener la función de moderador en todas las actividades que tiene el método. Los integrantes del equipo deben tener completo conocimiento sobre EV-AR y tener conocimientos sobre arquitecturas de software.
- *Equipo de Desarrollo:* En este equipo entran todas las personas que hacen e hicieron parte del desarrollo del sistema que se está recuperando. Este equipo participar en todas las actividades del método, pero su principal responsabilidad es la participación en la recuperación del rationale del sistema, aquí ellos tendrán que proveer toda la información que tengan sobre la construcción del sistema que se está analizando.
- *Equipo de Arquitectura:* Este rol está conformado por las personas que van a recibir la arquitectura recuperada, su principal responsabilidad está en la discusión de los resultados obtenidos donde ayudaran a determinar si la información obtenida es suficiente para los objetivos en el método.
- *Equipo de Interesados:* Este equipo está conformado por las personas que están interesadas en realizar la recuperación de la arquitectura del sistema, su responsabilidad es participar en la identificación de los escenarios, donde expondrán sus necesidades e inquietudes sobre el funcionamiento del sistema.

3.6. CONSIDERACIONES SOBRE EV-AR

EV-AR fue diseñado con el objetivo de reunir a las personas involucradas en la construcción de un sistema software para analizar su arquitectura, apoyados de una herramienta de visualización de software y de la documentación existente sobre el sistema, sin embargo el método presenta unas consideraciones para su aplicabilidad:

1. El método puede ser llevado a cabo por mínimo dos personas, donde una de ellas debe llevar el rol de equipo de recuperación, ella será la encargada de controlar la ejecución del método, y la otra persona debe asumir los roles faltantes del método.

2. Es importante que dentro del proceso participe una persona con fuertes conocimientos en arquitecturas de software, la cual tiene como responsabilidad resolver dudas conceptuales que hayan durante el proceso de recuperación.

3. El método plantea que debe haber una fuente de la cual extraer vistas arquitecturales del sistema, para lo cual se presentan dos posibilidades: la primera es usar el código fuente para extraer las vistas arquitecturales, esto se puede hacer usando una herramienta de visualización de arquitecturas, o realizando la recuperación de las vistas de forma manual, la segunda fuente es la documentación del sistema, en esta se espera encontrar con datos relevantes sobre la arquitectura del sistema, las funcionalidades del sistema, los requisitos iniciales sobre los cuales se construyó el sistema.

4. Aunque el método puede ser aplicado con un mínimo de personas, es recomendable que participe un grupo significativo de personas dentro de cada uno de los roles del método, esto con el fin de generar discusión sobre la información recuperada de la arquitectura del sistema.

3.7. ESPECIFICACIÓN DEL PROCESO – EPF

El método EV-AR se diseñó y especificó en la herramienta Eclipse Process Framework Composer – EPF[8], usando SPEM 2.0 que es un estándar de la OMG[9] para la especificación y formalización de procesos.

La estructura global de EPF es la Library, esta contiene a su vez, Plugins y Configurations. Los Plugins tienen Method Contents, que son la estructura principal (ver Figura 14) de EPF para agrupar información y ordenar la información sobre el proceso (nombres, descripciones, tareas, guías, roles, productos de trabajo, etc.). Los Plugins además tienen Processes en donde se definen las actividades establecidas en los Method Content y se describe el workflow y el diagrama en SPEM. Las Configurations sirven para tomar la información establecida en los Method Contents y poderlas adaptar según las necesidades y el entorno en donde se piense ejecutar el proceso.

[8] Eclipse Process Framework Componer - http://www.eclipse.org/epf/
[9] Object Management Group

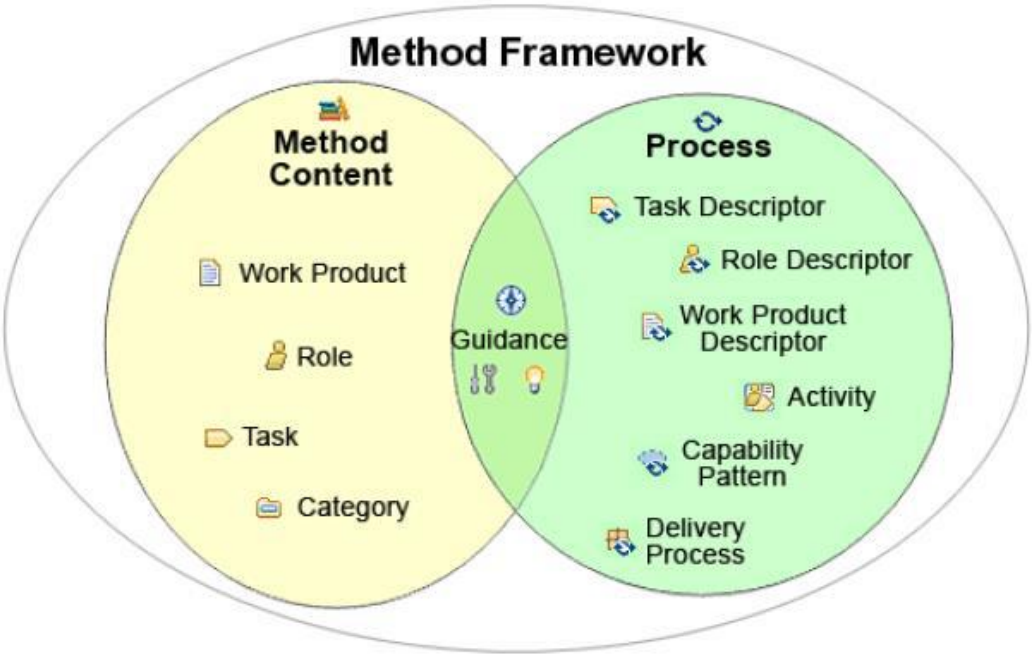


Figura 14. EPF Method Framework tomado de EPF Tutorial User Manual.

Para EV-AR se creó una sola Library, un solo Plugin y una sola Configuration; teniendo en cuenta que el método fue pensado para que haga parte de un proceso de alguna organización se crea un Content Method el cual contiene un Content Package donde se especifican las tareas del método con sus respectivos pasos, los artefactos, los roles y las guías que contiene el método, un Capability Pattern que básicamente muestra el workflow y el diagrama en SPEM con las actividades, tareas, roles y work products asociadas. En la Tabla 13, se muestra el mapeo uno a uno de los elementos que contiene EV-AR con los elementos de EPF. En la figura 15 podemos observar Construcción de EV-AR en EPF Composer.

Tabla 13. Relación entre elementos de EV-AR y APF.

EV-AR	EPF
Método	Method Library, Method Package, Procecess Package, Method Plugin
WorkFlow	Capability Pattern
Tareas	Tasks
Pasos	Steps
Artefactos	Work Products
Roles	Roles

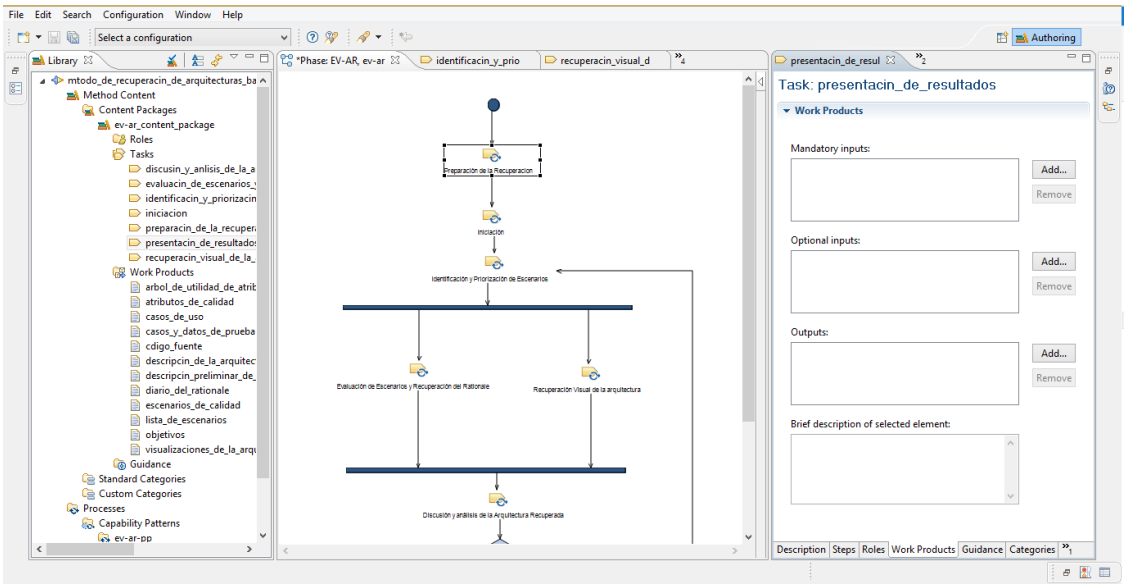


Figura 15. Construcción de EV-AR en EPF Composer

3.8. INSTRUMENTOS DE APOYO

Los instrumentos de apoyo fueron creados con la intención de que las personas interesadas en usar el método, lo puedan conocer y utilizar. Esta decisión se tomó porque se considera que una de las claves para que el método sea usado sería la aceptación, entendimiento y adopción de este.

3.8.1. Documentación Web

Este instrumento, es una aplicación web autogenerada por EPF. Básicamente EPF toma la información que se decide publicar, asociándola a una estructura llamada Custom Category. Para EV-AR se crearon cinco Custom Category en la que se asociaron cada uno de los elementos principales del método. En la Figura 16 se muestra la página inicial de este instrumento. (Ver ANEXO B. Apartado 1).

Figura 16. Página Inicial Documentación Web de EV-AR.

3.8.2. . Guía De EV-AR

Como instrumento para el fácil entendimiento del método se diseñó una guía que describe a EV-AR. A este documento se trabajó para que describa el método de una manera fácil y que de ayudas e información clave para que las personas que estén interesadas en usar el método tengan información clave para su correcta ejecución. Esa guía se puede ver en el Anexo (Ver ANEXO C. Apartado 1).

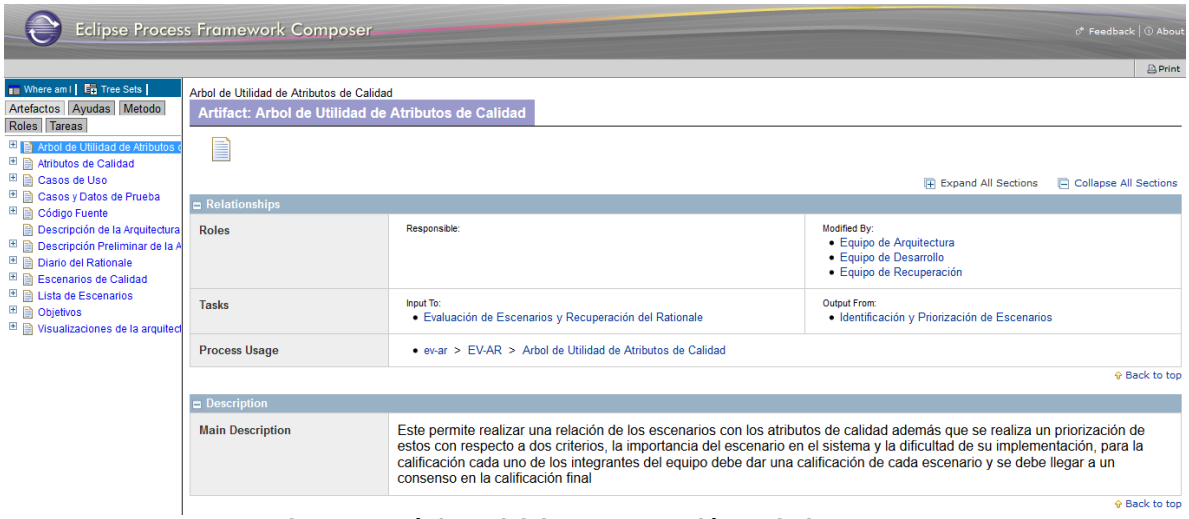


Figura 17. Página Inicial Documentación Web de EV-AR.

EVALUACIÓN DEL METODO EV-AR

4.1. INTRODUCCIÓN

Al proponer un método de evaluación de arquitecturas como facilitador de la recuperación del conocimiento implícito, como es el caso en este proyecto, es necesario evaluarlo tanto de una perspectiva de calidad de su especificación, así como su aplicabilidad práctica en proyectos de desarrollo reales.

El método en el cual nos inspiramos para la creación del método EV-AR para recuperación de arquitecturas basado en la visualización, fue ATAM (Architecture Trade-off Analysis Method) [30] que evalúa con más profundidad, en relación con otros métodos, cuestiones referentes a la arquitectura, como son los atributos de calidad y está conformado por un conjunto de pasos importantes. También se tienen en cuenta diferentes herramientas de recuperación, que sirven como soporte tecnológico para la respectiva recuperación.

Para realizar una evaluación de arquitecturas que permita prever si es o no aplicable en pequeñas entidades productoras de software, fue utilizado el método de investigación de estudios de caso, con aplicaciones a diferentes contextos. La investigación en la ingeniería de software mediante estudios de caso tiene como objetivo estudiar como el proceso de desarrollo es realizado por los ingenieros de software y las partes interesadas en su contexto [18].

4.2. METODOLOGÍA DE LOS ESTUDIOS DE CASO EMPLEADOS EN LA EVALUACIÓN DEL MÉTODO EV-AR

La segunda parte de la evaluación del método EV-AR, siguió el método de estudios de caso propuesto por Runeson y Höst [18]. El estudio de caso es una metodología de investigación adecuada para la ingeniería del software que estudia un fenómeno contemporáneo en su contexto real, buscando mantener la integridad y las características significativas de los eventos, y es ejecutado cuando el investigador tiene poco control sobre los eventos y cuando los sujetos de estudio son más fáciles de observar en grupo que de manera aislada
Se consideraron 4 estudios de casos:

Estudio de Caso 1 – Recuperación Arquitectura de Argouml:
Holístico[10], Exploratorio[11], 1 unidad de análisis[12]

[10] Un estudio de caso holístico, estudia el caso en conjunto [66].
[11] Exploratorio, se emplea para descubrir lo que está sucediendo, en busca de nuevas ideas y la generación de ideas e hipótesis para nuevas investigaciones [67]**.**
[12] En ingeniería de software la unidad de análisis puede ser un proyecto de desarrollo, un equipo o una persona individual.

Se tiene como objetivo hacer la recuperación de la arquitectura de un sistema software que ha sido desarrollado desde la comunidad open source, donde se hace uso de herramientas de ingeniería inversa para la recuperación de la arquitectura de un sistema, las cuales nos proveen diferentes vistas arquitectónicas del sistema.

Estudio de Caso 2 – Aplicación del método de evaluación ATAM-AR, en un sistema E-LearningSync:
Holístico, Exploratorio, 1 unidad de análisis.
El objetivo de este estudio de caso fue determinar si el método de recuperación de arquitecturas basado en la focalización evaluativa, sirve de guía a un arquitecto de forma práctica e intuitiva en un sistema cuyo objetivo es la integración de nuevas plataformas de E-Learning.

Estudio de Caso 3 – Aplicación del método de evaluación EV-AR, en una Pequeña Organización:
Holístico, Descriptivo[13]-Revelatorio[14], 1 unidad de análisis
El objetivo de este estudio de caso fue determinar si el método de recuperación de arquitecturas basado en la focalización evaluativa y visualización de software, sirve de guía a un arquitecto de forma práctica e intuitiva en una pequeña organización de software.

Estudio de Caso 4 – Aplicación del método de evaluación EV-AR, en el sistema FUNCAVO:
Holístico, confirmatorio[15]-Descriptivo, 1 unidad de análisis
El objetivo de este estudio de caso fue determinar si el método de recuperación de arquitecturas en conjunto con la herramienta iplasma ayuda a un arquitecto de forma práctica e intuitiva en un sistema de información de una fundación.
El diseño de los estudios de caso siguió la propuesta de Runenson y Host [18]. El diseño de la investigación partió de varias preguntas de investigación muy relacionadas. Primero, la pregunta básica de investigación planteada en el proyecto ¿Cómo guiar a un arquitecto de software en la tarea de recuperar arquitecturas de sistemas usando un enfoque evaluativo y mecanismos de recuperación visual de tal forma que esto se logre bajo abordaje práctico e intuitivo? y de ésta fueron derivadas las preguntas concretas para cada uno los cuatro estudios de caso.

[13] Tiene como objetivo la descripción precisa del evento de estudio. Este tipo de investigación se asocia al diagnóstico. En la investigación descriptiva se hace enumeración detallada de las características del evento de estudio.
[14] Un caso que genera indicios; es decir, permite explorar un fenómeno, un estudio inicial o previo [68].
[15] Este tipo de investigación requiere de una explicación previa o una serie de supuestos o hipótesis. Su objetivo es verificar una o más hipótesis derivadas de una teoría, a partir de la experiencia directa.

Para la realización de los estudios de caso fueron seguidos los pasos presentados en la tabla 14, se presentan los pasos generales seguidos en los Estudios de Caso empleados en la evaluación del método EV-AR.

Tabla 14. Pasos Generales de un Estudio de Caso

PASOS GENERALES	
Diseño	•Objetivo •Tipo de estudio •Unidad de análisis •Indicadores, métricas e instrumentos
Preparación	•Diseño de instrumentos
Ejecución y Recolección de información	•Desarrollo del caso •Aplicación de instrumentos •Observación
Análisis y reporte	•Análisis de datos cuantitativos y cualitativos •Documentación del estudio de caso.

4.2.1. Estudios De Caso Exploratorios

Para la realización de este proyecto era necesario el estudio de diferentes herramientas y métodos de recuperación de arquitecturas, para no tener únicamente un enfoque teórico, los dos primeros casos de tipo exploratorio permitieron una aproximación más real y visible al enfoque.

4.2.1.1 Estudio de Caso Recuperación Arquitectura de Argouml:

1) Pregunta de Investigación: la pregunta de investigación del proyecto evidenció la necesidad de identificar las herramientas visuales de recuperación que se deben tener en cuenta para la creación del método EV-AR. Así la pregunta que éste caso busca responder es ¿Qué herramientas visuales de recuperación deben considerarse para hacer efectivo el método EV-AR para guiar a un arquitecto de software de forma práctica e intuitiva?

2) Objetivo del Estudio de Caso: identificar las herramientas visuales de recuperación que pueden ayudar en la recuperación de arquitecturas para guiar a un arquitecto de software.

3) Selección del Estudio de Caso: de acuerdo a Runeson y Höst [18], exploratorio y holístico con una unidad de análisis, la cual corresponde a un proyecto de recuperación de la arquitectura de un producto software. La selección del caso siguió un criterio revelatorio.

4) Contexto del caso: Se tiene como objetivo hacer la recuperación de la arquitectura del sistema software ArgoUML, que ha sido desarrollado desde la comunidad open source y proponer unos modelos para la comunicación de la arquitectura de un sistema software. Durante este desarrollo se hace uso de herramientas de ingeniería inversa para la recuperación de la arquitectura de un sistema, las cuales nos proveen vistas arquitectónicas del sistema donde se puede observar desde un nivel general hasta llegar a un nivel de detalle observando una clase y su comportamiento. El objetivo general de este estudio es documentar la arquitectura recuperada de ArgoUML a un nivel de diseño. ArgoUML fue concebido como una herramienta para soportar el análisis y diseño de sistemas de software orientado a objetos. Es similar a muchas de las herramientas CASE comerciales que se venden como herramientas para el modelado de sistemas de software.

   ArgoUML es potente y fácil de usar, soporta el diseño, desarrollo y documentación de las aplicaciones de software orientadas a objetos. ArgoUML pertenece a la familia de aplicaciones de software llamado Computer Aided Software Engineering (CASE).

   Los usuarios de ArgoUML son diseñadores de software, arquitectos, desarrolladores de software, analistas de negocios, analistas de sistemas y otros profesionales involucrados en el análisis, diseño y desarrollo de aplicaciones de software [64].

   ArgoUML fue concebido como un entorno y herramienta para usar en el análisis y diseño de sistemas de software orientados a objeto. Es similar a muchos de las herramientas CASE comerciales que son vendidas como herramientas para modelar sistemas software. ArgoUML tiene las siguientes características [64].
   - Es un software libre.
   - ArgoUML se enfoca en investigación sobre psicología cognitiva para proporcionar nuevas características que incrementen la productividad soportando las necesidades cognitivas de diseñadores y arquitectos dedicados al análisis y diseño de software orientado a objeto.
   - ArgoUML soporta estándares abiertos extensivamente—UML, XMI, SVG, OCL y otros.
   - ArgoUML es una aplicación Java pura 100%. Esto permite a ArgoUML funcionar en todas las plataformas que soporte java.
   - ArgoUML es un proyecto de código abierto. La disponibilidad del código fuente asegura que una nueva generación de diseñadores de software e

investigadores ahora tienen un entorno de trabajo probado desde el que pueden conducir el desarrollo y evolución de tecnologías de herramientas CASE.

Estándares abiertos: XMI, SVG y PGML - ArgoUML soporta estándares abiertos - UML, XMI, SVG, OCL y otros.
XML Metadata Interchange (XMI) es el estándar para guardar los meta-datos que conforman un modelo UML particular. En principio, esto le permitirá tomar el modelo que ha creado en ArgoUML e importarlo en otra herramienta.

Diseño UML - ArgoUML utiliza GEF, UCI Graph Editing Framework para editar diagramas UML. Se admiten los siguientes tipos de diagramas

- Diagramas de clases.
- Diagramas de máquina de estados.
- Diagramas de actividades.
- Diagramas de casos de uso.
- Diagramas de colaboración.
- Diagramas de implementación.
- Diagramas de secuencia.

5) Diseño del Estudio de Caso: de acuerdo al objetivo del estudio de caso, fueron diseñados los indicadores, métricas e instrumentos a emplear, la Tabla 15 relaciona estos elementos para el estudio de caso exploratorio.

Tabla 15. Diseño de Estudio Caso Exploratorio.

Objetivo	Indicadores	Métricas	Instrumentos
Identificar las herramientas visuales de recuperación que pueden ayudar en la recuperación de arquitecturas para guiar a un arquitecto de software.	Eficiencia = Efectividad Vs Esfuerzo	Efectividad (Porcentaje de conocimiento Recuperado respecto a conocimiento total) Conocimiento (Vistas) Esfuerzo (horas Persona).	Artefactos Previos (código ArgoUML) Documento de Arquitectura
	Comprensibilidad y Facilidad de uso	Percepción sobre la claridad de las herramientas de recuperación visual. Percepción sobre la facilidad de uso de las herramientas de recuperación visual	Observación directa

6) Desarrollo del caso: Para la ejecución de este estudio de caso se plantearon cuatro tareas para obtener la arquitectura de ArgoUML. La primera tarea fue la definición de un objetivo claro para la recuperación de esta arquitectura, el cual fue “Apoyar actividades de evolución y mantenimiento de un sistema”, la

segunda tarea consistió en explorar y seleccionar las herramientas que apoyarían la recuperación visual de la arquitectura de ArgoUML, la tercera tarea fue recuperar la arquitectura de ArgoUML usando las herramientas seleccionadas, para lo cual nos basamos en el proceso que plantea Panas et al. [41], la última tarea consistió en documentar la arquitectura de ArgoUML para lo cual usamos el método practico para la documentación de arquitecturas que plantea Clements et al. [33].

7) Dinámica del proyecto: Las actividades del análisis de herramientas visuales de recuperación fueron desarrolladas en su gran mayoría en horas de clase de la electiva Ingeniería Inversa perteneciente al programa de pregrado Ingeniería Electrónica y Telecomunicaciones de la Universidad de Cauca dirigida por tres docentes expertos en el tema. La modalidad seguida fue lluvia de ideas, lo que permitió la participación de todo el grupo y la consideración de varios aspectos como son complejidad de las herramientas, experiencia del grupo con ellas y el conocimiento de las herramientas. Una vez establecido el alcance del estudio de las herramientas del grupo, se distribuyó el análisis de ellas de acuerdo a las destrezas de cada uno de los integrantes. Las herramientas de recuperación fueron analizadas y posteriormente se presentaron en conjunto, una vez identificados fueron examinándose las características de estas, el grupo da sugerencias y se obtienen resultados finales del análisis.

   En el análisis de nuestro estudio se inició con la lectura de procesos, técnicas y métodos para la recuperación de arquitecturas, donde se toma como referencia el proceso unificado de Panas et al. [41], Para la posterior recuperación de la arquitectura de ArgoUML por medio de herramientas visuales.

   Durante el desarrollo del proyecto fue recolectada información siguiendo el diseño del estudio de caso y usando los resultados obtenidos con las herramientas visuales. Para escoger las herramientas que se usaron en este caso, primero se realizó una exploración de las posibles herramientas que se podían usar donde se encontraron 8 candidatas, se procede a obtener documentación relacionada cada una de las herramientas y hacer pruebas sobre ellas para conocerlas y obtener observaciones sobre estas, en este proceso encontramos muchos inconvenientes como la falta de documentación, la cual en ocasiones fue incompleta y para saber sobre la herramienta se optó por consultar en foros para poder usar la herramienta. Ya teniendo en cuenta la experiencia que se tuvo al probar las herramientas y descartando algunas ya que no se pudo probarlas, se toma la decisión de usar Softwarenaut, iPlasma y Moose como criterio principal que se pudo usar en totalidad la herramienta y que los resultados que arrojaron eran posibles de entender. En la Tabla 16 se muestran observaciones sobre las herramientas probadas.

Tabla 16. Observaciones sobre las herramientas probadas.

Herramienta	Observaciones
Archimetrix	• Trabaja sobre código fuente Java, C++ y Delphi. • Su principal funcionalidad es la detección de deficiencias sobre el código fuente. • Se presentaron inconvenientes para instalar la aplicación, esta debía ser instalada usando la plataforma Eclipse, por la cual nos debíamos conectar a una lista de repositorios para instalar los componentes que la herramienta necesitaba y uno de estos componentes no estaba en los repositorios para su instalación.
SemmleCode	• Analiza código Java • Ayuda a calcular métricas, encontrar errores y la exploración del código mediante un lenguaje de consulta llamado QL. • Se encontrón información sobre la herramienta pero no estaba disponible el instalador.
Softwarenaut	• Trabaja sobre el metamodelo FAMIX. • Depende de herramientas que realicen la traducción del código fuente al metamodelo FAMIX. • Permite ver los componentes de un sistema y sus relaciones, además de proveer un mecanismo para navegar dentro de cada componente. • Para poder usar esta herramienta, usamos la herramienta verveinJ. Esta nos permitió transformar el código fuente de ArgoUML en su representación en el metamodelo FAMIX
iPlasma	• Trabaja sobre código fuente C++ y Java • Esta herramienta permite el análisis de código mediante el cálculo de métricas, la detección de duplicación de código.
Imagix4D	• Esta herramienta permite el análisis de código escrito en C, C++ y Java. • Permite la visualización de estructuras de control, uso de datos, herencia y la dependencia de funciones. • Esta herramienta cuenta con una licencia comercial, por lo que no fue posible su uso.
JTransformer	• La documentación del sistema daba un repositorio accesible desde el instalador de la plataforma eclipse, pero no se tuvo acceso a este para la instalación de la herramienta.
CCFinder	• Esta herramienta permite encontrar fragmentos de código repetidos. • Funciona con código Java, C/C++, COBOL, VB, C#. • La aplicación no funcionó.
MOOSE	• Es una plataforma que permite el análisis de aplicaciones y datos, mediante la visualización y exploración. • Moose contiene importadores internos que le permiten recibir datos o código fuente en Smalltalk, XML, MSE. • Para este caso verveinJ permite transformar el código fuente de ArgoUML en su representación en el metamodelo FAMIX generando datos en formato MSE los cuales permiten importar el código ArgoUML en Moose.

8) Resultados Obtenidos:

a. Resultados cuantitativos: La Tabla 17 relaciona los resultados cuantitativos obtenidos en el estudio de caso exploratorio, estos permiten ver que las herramientas usadas nos permitieron recuperar un porcentaje importante sobre ArgoUML, sin embargo la información que recuperan puede ser difícil de entender por la cantidad de datos que pueden representar en una sola vista.

Este producto de trabajo proporciona una visión general arquitectónica completa del sistema, mediante una serie de vistas arquitectónicas diferentes para representar diversos aspectos del sistema donde cada una de las herramientas permitió recuperar diferentes vistas. Hemos decidido representar la arquitectura de software en varias vistas de la arquitectura asignándole un peso representativo a cada una de ellas. Cada vista de la arquitectura trata un conjunto concreto de problemas específicos de interesados en el proceso de desarrollo: usuarios, diseñadores, gestores, ingenieros de sistemas, mantenedores, etc, a cada una de estas vistas le hemos asignado un porcentaje proporcional para poder realizar un mejor análisis de lo que se pudo recuperar exceptuando la vista lógica a la que se le dio un peso un poco más significativo, ya que podemos observar que la vista lógica muestra un subconjunto significativo arquitectónicamente del modelo de diseño, es decir, un subconjunto de las clases, subsistemas y paquetes.

Tabla 17. Resultados Cuantitativos.

Vistas	**Pit**	**%iPl**	**(%Sof)**	**(%Mo)**
Escenarios	17,5%	0%	0%	0%
Lógica	30%	12,5%	12,5%	15%
Procesos	17,5%	0%	0%	0%
Implementación	17,5%	3%	8,8%	9%
Instalación	17,5%	0%	0%	0%
Efectividad		**15%**	**21%**	**24%**
Esfuerzo(Horas persona)		**3**	**5**	**4**

Vistas: Representan un aspecto parcial de la arquitectura recuperada donde muestran propiedades específicas del sistema.
(Pit) Peso representativo a cada una de las vitas
(%iPl): Porcentaje de recuperación herramienta iPlasma con respecto al peso Item de cada vista.
(%Sof): Porcentaje de recuperación herramienta Softwarenaut con respecto al peso Item de cada vista.
(%Mo): Porcentaje de recuperación herramienta Moose con respecto al peso Item de casa vista.

La Tabla 17 relaciona las vistas que se pueden obtener sobre una arquitectura con un porcentaje de recuperación que se obtiene con las herramientas utilizadas en este estudio de caso, podemos observar que estas herramientas nos ayudan a recuperar información sobre la vista lógica y de implementación

del sistema, también se puede concluir que por medio de las herramientas no fue posible identificar las decisiones arquitectónicas que se tomaron en cuenta para el desarrollo de ArgoUML. La Tabla 17 se caracteriza porque se tomó en cuenta el porcentaje de recuperación de cada una de las herramientas según su peso y se calculó la efectividad de cada una de ellas sumando los porcentajes recuperados en cada una de las vistas, también se tuvo en cuenta el tiempo en horas persona en el proceso de recuperación.

b. Resultados Cualitativos: Durante la fase del análisis de herramientas, el grupo logró trabajar en equipo, aprovechando los espacios en conjunto, esta fue una fortaleza del estudio de caso.

iPlasma nos muestra como primer resultado una pirámide de resumen del sistema, ver figura 18, esta nos muestra una recopilación de métricas acerca del sistema en general, donde se muestran indicadores para observar si estas están o no en el rango de la industria, al observar el resumen de complejidad nos damos cuenta que la Complejidad ciclomatica (CYCLO), Líneas de código (LOC) y el Numero de clases (NOC) según las métricas establecidas para aplicaciones en JAVA están en un rango medio, el número de descendientes directos (NDD) se encuentra en un rango bajo y el número de métodos(NOM), la altura del árbol de herencia (HIT) y llamadas por métodos (CALL) se encuentran en un rango alto lo cual indica que la aplicación ArgoUML presenta niveles altos que no son recomendables con respecto a lo que recomienda la industria, las métricas asignadas en cada rango me ayudan a identificar los colores, donde el color rojo sobrepasa el promedio que ha establecido al industria, el verde está dentro del promedio de lo establecido de la industria y el azul esta debajo del promedio.

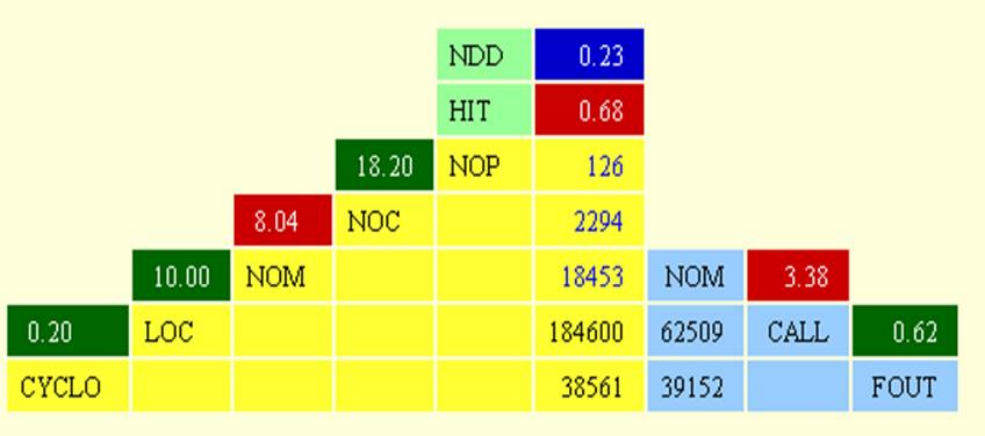


Figura 18. Pirámide de Resumen

También nos muestra una vista llamada “Resumen de la complejidad del sistema”, ver figura 19, esta vista nos permite ver la totalidad de las clases del sistema junto con sus relaciones de herencia, además de incluir en la

representación de cada clase tres métricas que representan el número de atributos, número de métodos y número de líneas de código.

Podemos observar que se representan a los métodos y atributos usando cajas con una tonalidad alta o baja, cuando el color es más oscuro, nos indica que esa clase tiene más líneas de código ya que la vista de “Resumen de complejidad” es muy muy extensa de decidió hacer un análisis del extracto Figura 20 donde se puede observar que la clase que se encuentra oscura presenta un alto número de líneas de código y se considera que puede ser de riesgo para la mantenibilidad del sistema y eso lo convierte en un componente crítico que debe ser auditado constantemente para evitar inconvenientes en el sistema.

Con respecto al segundo extracto Figura 21 se pueden observar dos relaciones de herencia, donde se muestra que las clases que se encuentra en la parte inferior heredan de una clase en común y la otra hereda de otra clase, la clase más general o clase padre se ve que tiene un numero de métodos y clases la segunda va a contener estos mismos métodos y clases, pero en esta se agregan más atributos las dos tienen una misma tonalidad lo que significa que no hay diferencias significativas en líneas de código, pero si vemos las clases de la parte inferior estas ya presentan un aumento en el número de atributos, además que su tonalidad indica que hubo un cambio significativo en el número de líneas de código con respecto a sus antecesores.

Figura 19. Extracto 1 del Resumen de la complejidad del sistema

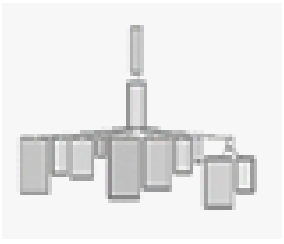


Figura 20. Extracto 2 del Resumen de complejidad del sistema

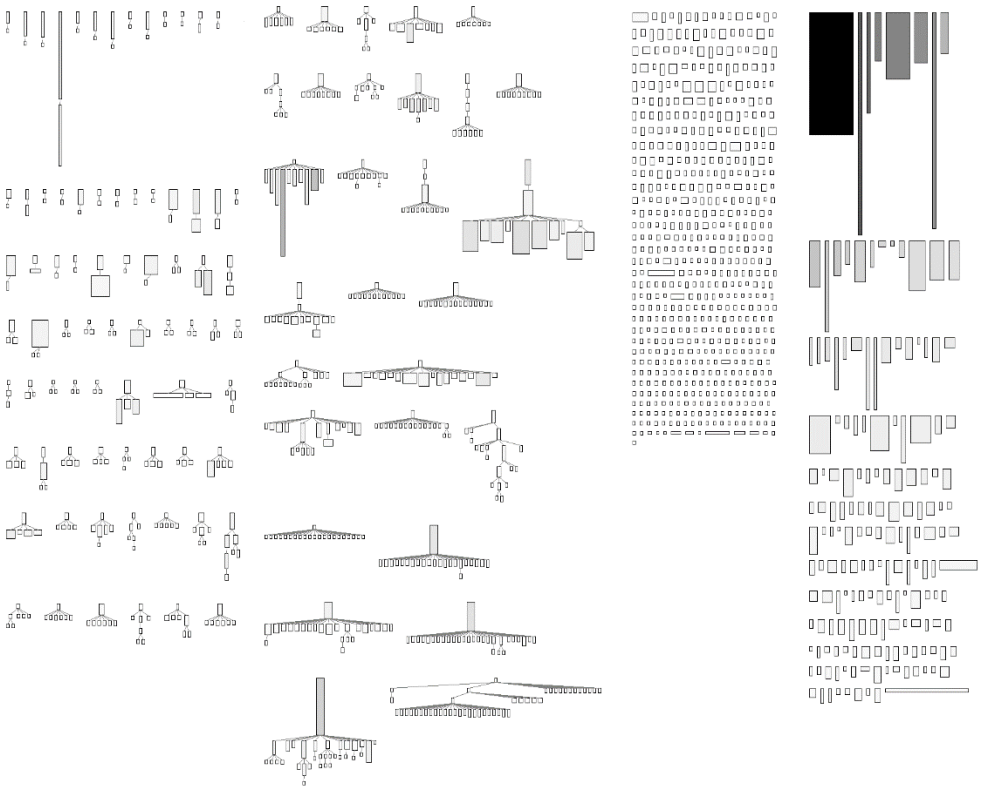


Figura 21. Resumen de la complejidad del sistema

La Figura 22 se muestra cómo se representan las métricas número de atributos, número de métodos y número de líneas de código.

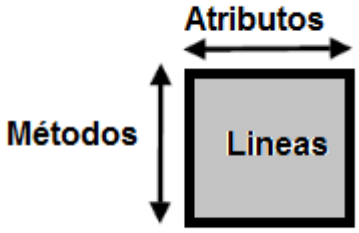


Figura 22. Representación Métrica

Moose también nos entrega un resultado con la mismas características del obtenido por iPlasma, solo que este no representa la métrica número de líneas

de código, en la Figura 23 se muestra una vista parcial de la vista de complejidad del sistema donde se refleja que no se representa la métrica número de líneas de código, Moose través de visualizaciones simples hace un amplio uso de métricas, que permite obtener una visión general de ArgoUML en un corto periodo, esta visión es representada de forma estática donde se generan diagramas que describen su estructura, como el diagramas de clases y de paquetes. Información que se puede de rescatar sobre esta vista es que nos permite identificar y relacionar con la métrica altura del árbol de herencia (ADD), ya que este nos permite identificar el número de niveles de herencia que hay en el sistema.

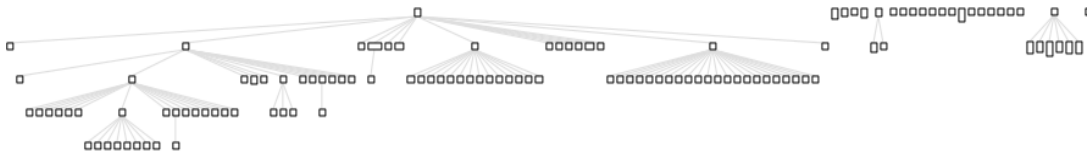


Figura 23. Vista parcial complejidad sistema en Moose.

Cuando ya hemos obtenido la representación a nivel de clases del sistema procedemos realizar la representación a nivel de arquitectura, esta se puede obtener a partir de la herramienta Softwarenaut y la plataforma Moose.

Softwarenaut nos provee de una vista general del sistema donde se representan los módulos del sistema y las relaciones que hay entre estos, además inserta unas métricas donde el tamaño de cada módulo es proporcional a su tamaño real en líneas de código y la anchura de cada relación es proporcional al número de invocaciones entre los módulos correspondientes, además de representar los módulos que contiene un módulo, en la figura 22 muestra una captura de pantalla de Softwarenaut siendo utilizado para analizar ArgoUML, Los módulos se representan como diagramas con sus respectivas relaciones. Las vistas son producidas de forma interactiva, que permite inspeccionar los detalles y hacer zoom en los módulos y las relaciones individuales.

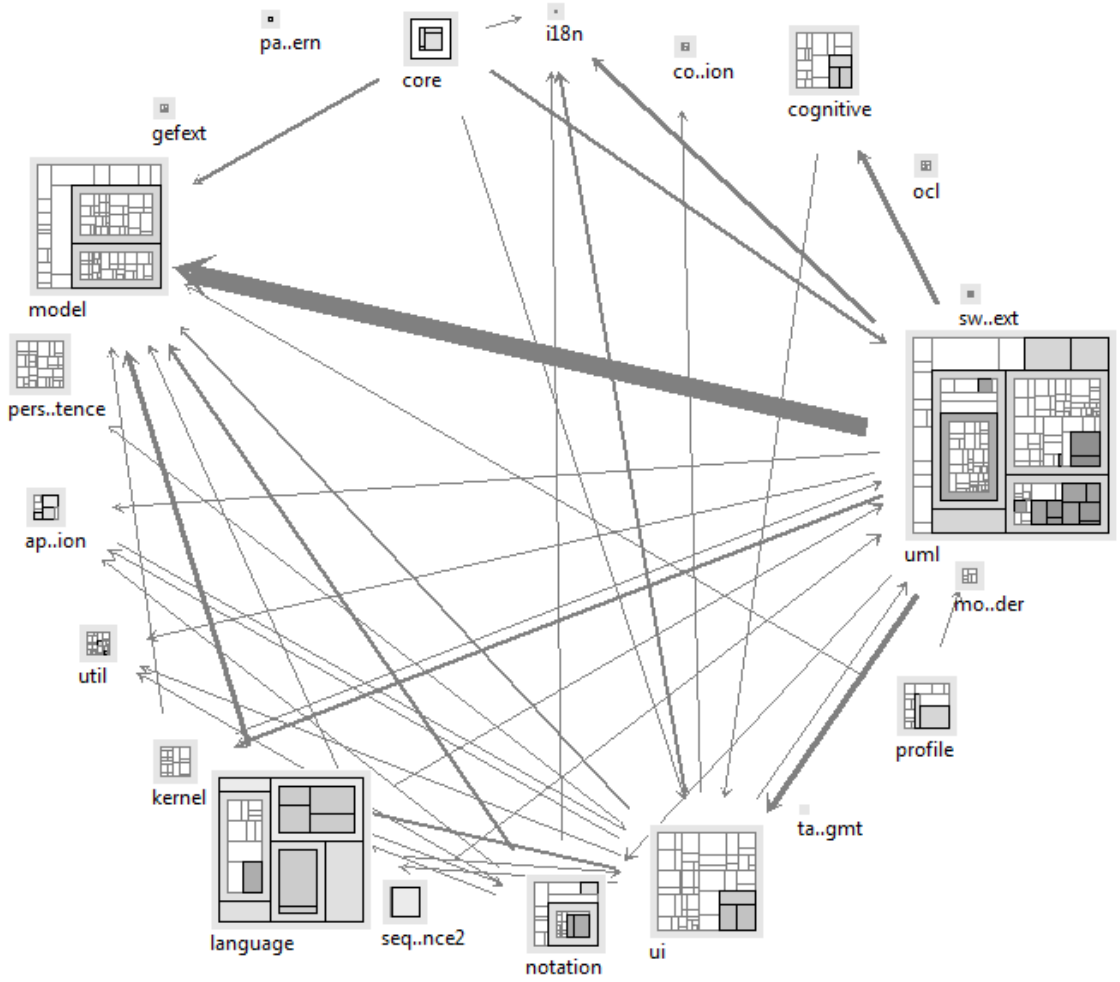


Figura 24. Vista de módulos en Softwarenaut.

Moose nos permite ver los paquetes del sistema como un árbol, el cual nos proporciona una vista muy general del sistema, esta vista es de mucha utilidad ya que nos permite ubicarnos cuando los Sistemas son tan grandes, en la figura 25. Se puede ver el árbol correspondiente al sistema ArgoUml.

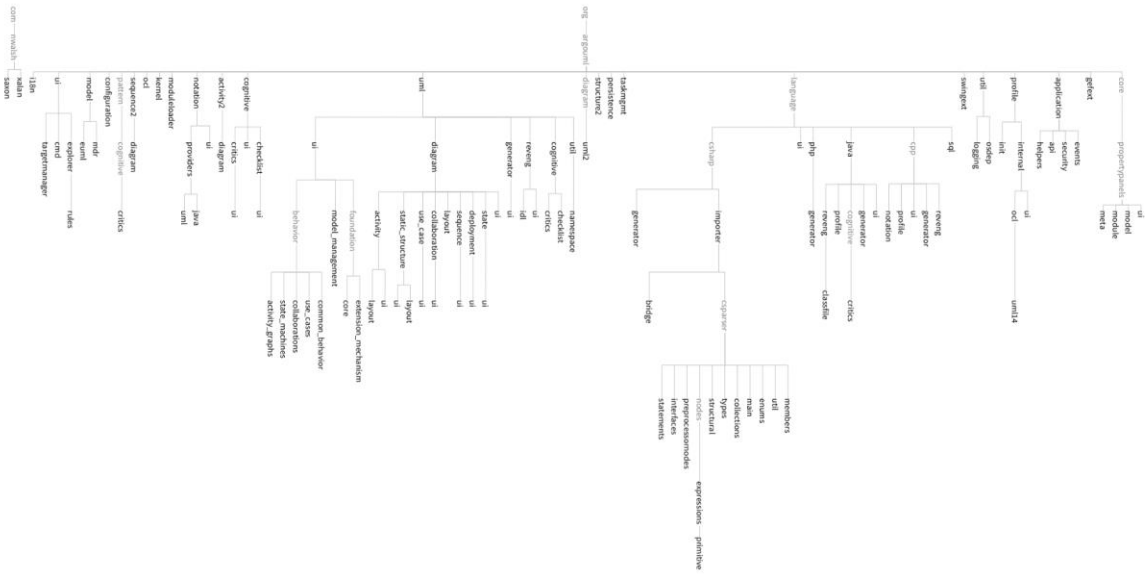


Figura 25. Paquetes ArgoUML.

En el proceso que estamos siguiendo se hace un análisis de la arquitectura y se visualizan componentes del sistema en un nivel de detalle más alto, IPlasma nos permite ver los componentes del sistema partiendo desde una vista de la complejidad de cada uno de los paquetes del sistema hasta una representación de cada clase donde podemos ver cuáles son los métodos, funciones y atributos que esta tiene.

En la Figura 26 podemos ver la vista de complejidad del paquete UML del sistema, esta vista tiene la misma representación que la vista de resumen de la complejidad del sistema completo, pero solo nos concentramos en cada uno de los paquetes.

En un modelo de clase las métricas se asignan de la anchura y la altura de un nodo. Los métodos y atributos están colocados de acuerdo con la capa a la que han sido asignados. La anchura y la altura de los nodos reflejan mediciones métricas de las entidades representadas en el contexto de un modelo de clase, las métricas utilizadas para los nodos de método son líneas de código para la altura y el número de invocaciones (es decir, número de invocación estática de salida del nodo representado) para el ancho. Los parámetros utilizados para el atributo son el número de nodos, los accesos directos de métodos dentro de la clase para la anchura y el número de accesos directos de los métodos definido en otras clases para la altura. Esto permite identificar cómo se accede a los atributos.

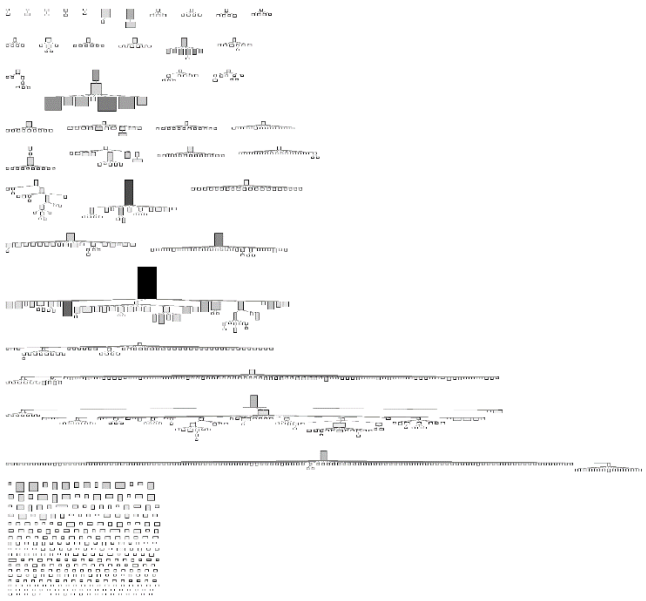


Figura 26. Complejidad del paquete UML.

Mirando el resultado arrojado en la figura 26, podemos ver que una de las clases que contiene este paquete presenta un alto grado de complejidad, en la herramienta se identifica que es la clase FigNodeModelElement. Con el objetivo que ver detalladamente esta clase, se genera el blueprint, ver figura 27, el cual facilita la visualización de los atributos y métodos de la clase, los colores que toman los métodos son para identificar si es un método de lectura, escritura, abstracto, sobrescrito, delegado, extendido o constante, en la figura 28 se puede observar cómo se describen las clases y los métodos en un blueprint.

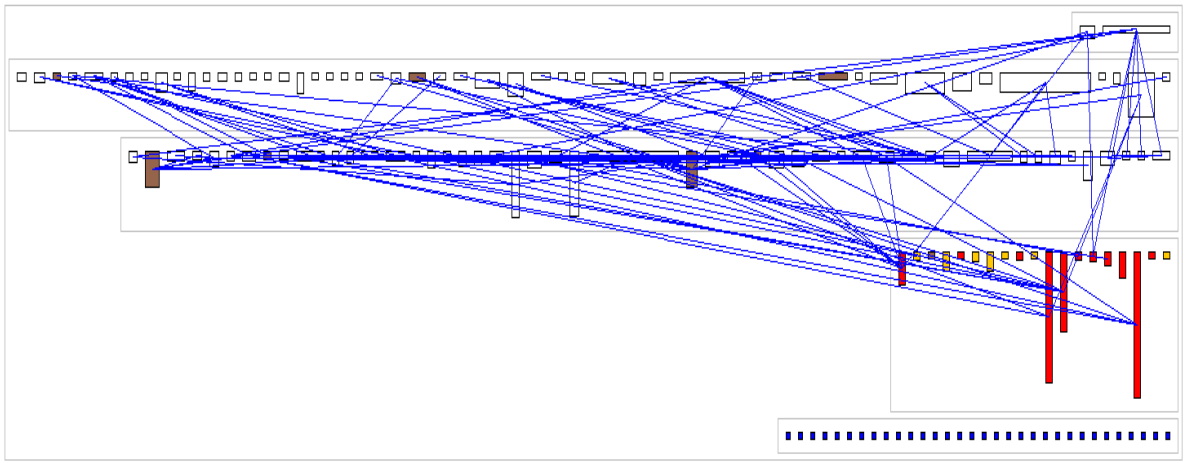


Figura 27. Class BluePrint FigNodeModelElement.

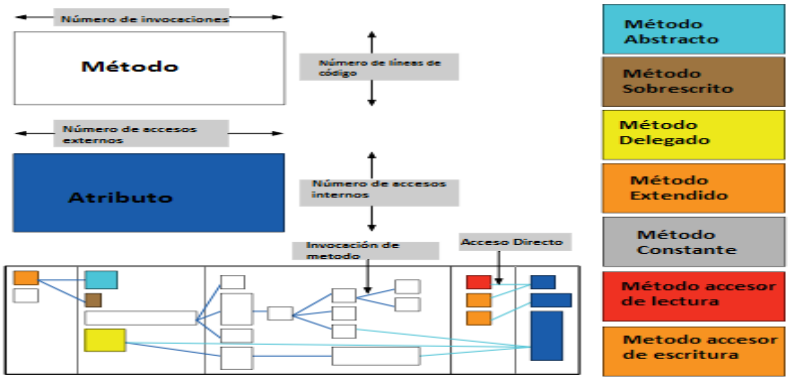


Figura 28. Identificación de colores y relaciones entre clases y accesos.

Softwarenaut también nos ayuda hacer una representación de paquetes por los cuales podemos navegar a través de ellos, en la figura 29 podemos observar la visualización del paquete UML, donde se puede ver que hay cuatro paquetes dentro del paquete llamado UML, cada uno de estos tiene más paquetes que están identificados con una tonalidad diferente lo cual también se refiere al número de líneas de código, una parte interesante para observar en es que ”reveng y útil” no tienen ninguna relación con otros paquetes mientras que el “diagram y ui” si y esto se puede denotar de acuerdo a la anchura de la flechas que los relacionan.

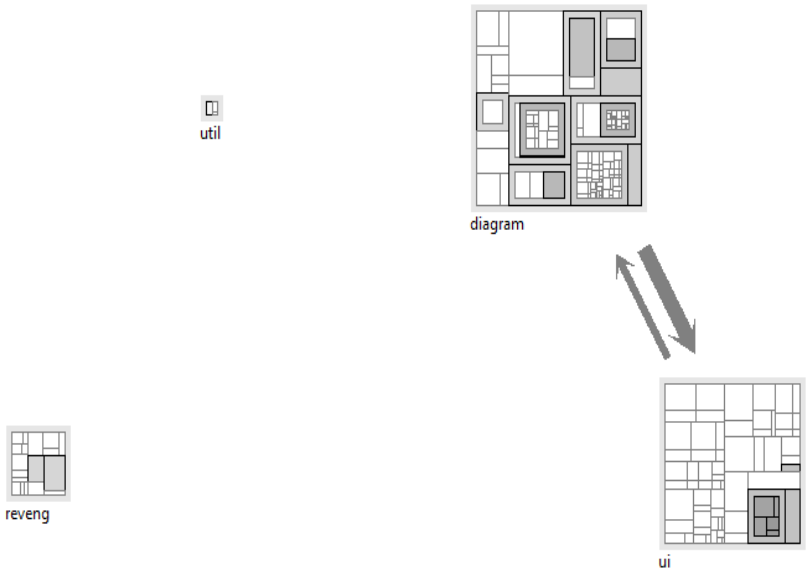


Figura 29. Representación de paquetes de UML.

Esta herramienta nos permite tener una visualización de las clases, la cual nos ayuda en el análisis de estas dándonos una serie de datos correspondientes a las clases como se pueden ver en la figura 30, donde vemos los datos que pertenecen a la clase FigNodeModelElement. En este caso podemos observar que esta clase tiene 416 de invocaciones hacia otras clases, se observa que esta clase es heredada por 54 clases, también cuenta con 33 atributos y 113 métodos, por ultimo podemos obtener métricas sobre la clase tales como:

- HNL: Nivel de herencia de la clase.
- NAI: Número d atributos inherentes.
- NC: Número de métodos constructores.
- NMA: Número de métodos agregados en la clase.
- NMI: Número de métodos inherentes.
- NMO: Número de métodos sobrescritos.
- NOA: Número de atributos sobrescritos.

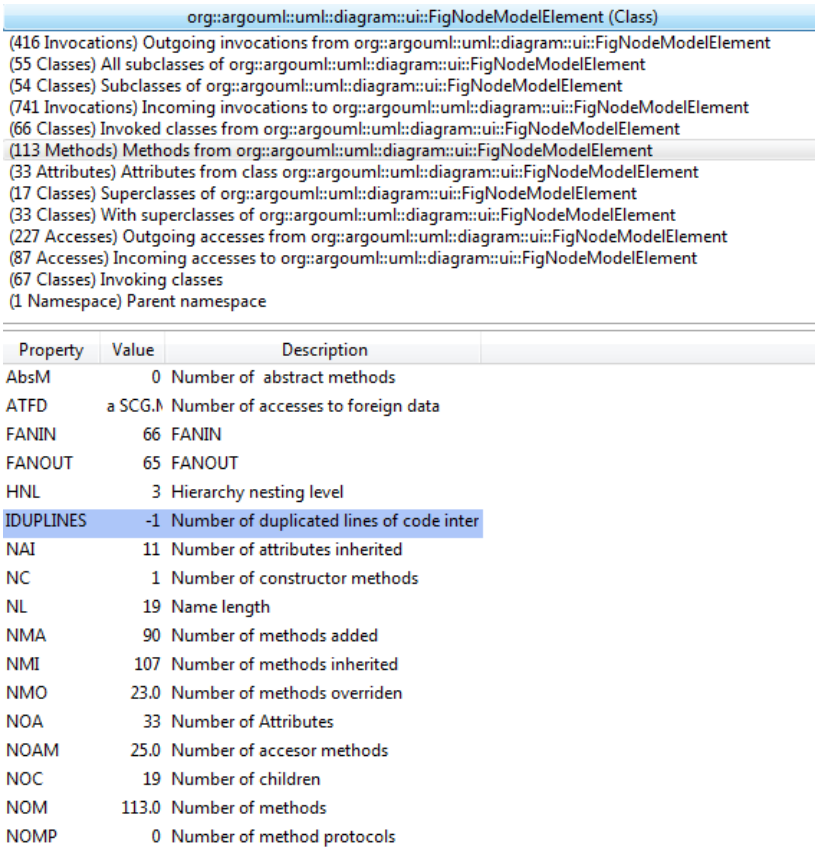


Figura 30. Clase FigNodeModelElement en Softwarenaut.

Moose nos posibilita hacer visualizaciones a nivel de paquetes, donde podemos observar todas las clases que contiene este paquete y sus sub-paquetes y las relaciones que entre ellas, en la figura 31 se puede observar una vista parcial del resultado arrojado por la plataforma.

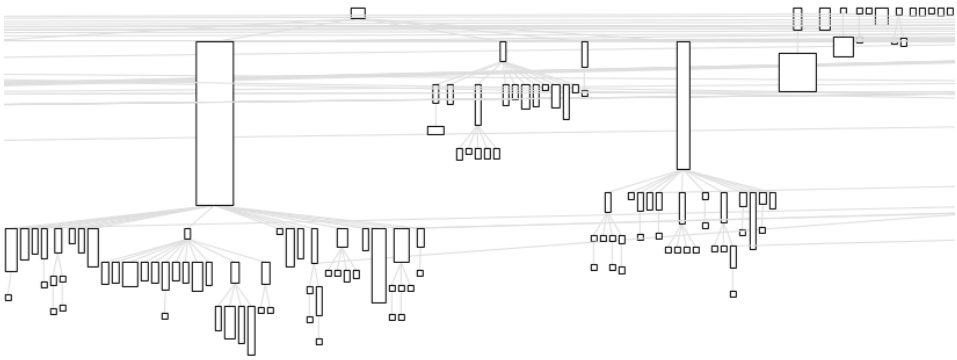


Figura 31. Complejidad del paquete UML en Softwarenaut.

A nivel de clase Moose posibilita la visualización de estas. En la figura 32 se observa el ClassBlueprint de la clase FigNodeModelElement. Además permite la visualización de las clases individualmente en un diagrama UML aunque no permite exportar este.

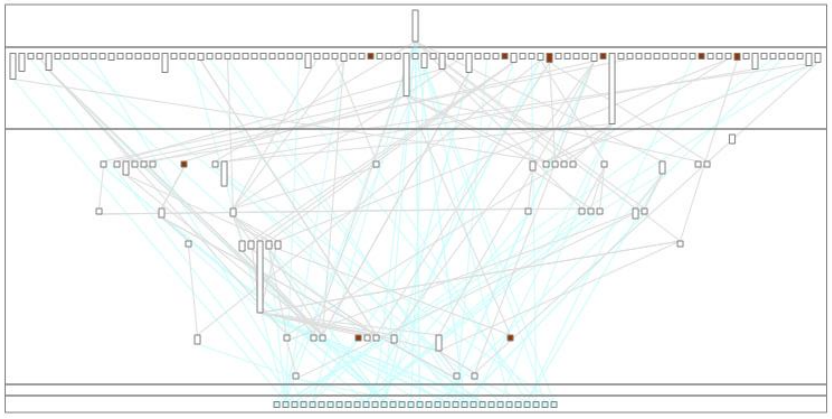


Figura 32. Complejidad del paquete UML en Softwarenaut.

Por último Moose permite generar una vista que muestra los paquetes del sistema y sus relaciones de dependencia entre ellos, además los organiza por capas de tal manera que los paquetes cliente de otro se ubican en una capa inferior, de esta forma permite identificar los paquetes con más dependencias y así ir subiendo por la gráfica para encontrar los paquetes con menos

dependencias del sistema en la figura 33 se puede ver esta vista para el sistema que estamos evaluando.

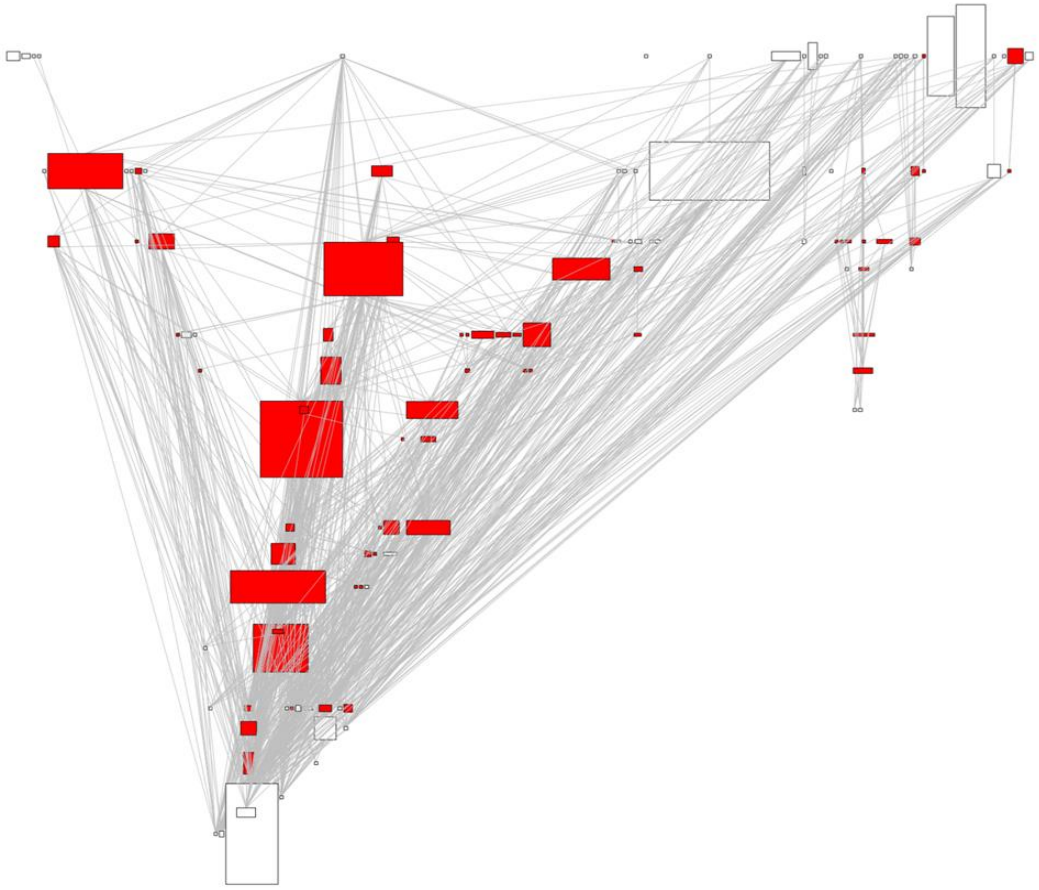


Figura 33. Dependencias de paquetes en Moose.

En la Figura 34 se identifican unos paquetes del sistema los cuales presentan dependencias cíclicas entre ellos, esto nos indica que estos componentes presentan un fuerte acoplamiento y esto produce que sean difíciles de modificar debido al efecto dominó que puede ocasionar que se haga un cambio en una de estos.

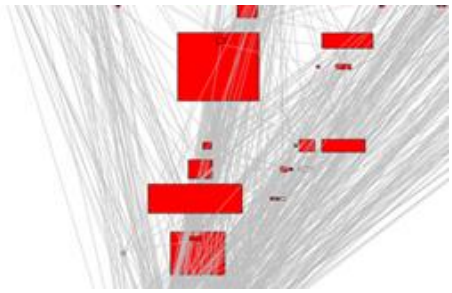


Figura 34. Extracto vista de dependencia de paquetes en Moose

9) Análisis de resultados

Con los resultados presentados en la sección anterior podemos verificar que se cumple en un porcentaje muy mínimo ya que en la recuperación con las herramientas arrojó un promedio del 20% de efectividad con respecto a las vistas. Las herramientas y la plataforma que usamos nos permitieron hacer una exploración del sistema donde nos podemos concentrar en cada una de las partes del sistema con el que estamos trabajando y por último se realizó la visualización del sistema donde se generan vistas que muestran comportamientos del sistema y sus componentes además de sacar la información correspondiente a sus métricas.

El proceso de recuperación de arquitecturas permitió obtener resultados de utilidad para realizar el análisis de sistema, para esto Clements et. al. [60], presentan un método practico para la documentación de la arquitectura, donde se debe identificar a quien o quienes va dirigida la documentación de la arquitectura y que deseamos documentar de la arquitectura del sistema y luego relacionar estas decisiones con una tabla que presentan, ver Tabla 18 para poder identificar que estilos de vistas son las más apropiadas para poder cumplir con el objetivo de la documentación.

Esta documentación se realiza con el objetivo de apoyar actividades de evolución y mantenimiento de un sistema, a partir de esto podemos identificar como interesados a los arquitectos de un sistema y los equipos de mantención del sistema y teniendo en cuenta que la finalidad de la recuperación del sistema ArgoUML fue documentar su arquitectura, podemos ver en la figura 29 que las vistas que les interesa conocer a los interesados que identificamos son: las vista de descomposición, la vista de generalización, la vista de uso y la vista de capas.

Tabla 18. Vistas útiles para los interesados

Vistas / Interesados	Vistas de Modulo				Vistas de Componentes y Conectores					Vistas de Distribución		
	Descomposición	Generalización	Uso	Capas	Tuberias y Filtros	Datos Compartidos	Cliente Servidor	Par a Par	Comunicación de Procesos	Despliegue	Implementación	Asignación de Trabajo
Actual/Futuro Arquitecto	D	D	D	D	S	D	D	D	D	D	S	S
Director del proyecto del gobierno	D	O	O	S	O	S	O	O	O	S		D
Director del proyecto contratistas	S	O	S	S	O	S	S	S	O	D	S	D
Miembro del equipo de desarrollo	D	D	D	D	O	D	D	D	D	S	S	D
Probadores e integradores	S	S	D	S	O	D	D	D	S	S	D	
Mantenedores	D	D	D	D	O	D	D	D	D	S	S	S
Ingenieros Comerciales	D	S		D		D	D	D	S	D		D
Analistas de Rendimiento	D	S	D	S	O	D	D	D	D	D		
Analistas de Integración de Datos	S	S	S	D	O	D	D	D	D	D		
Analistas de Seguridad	D	S	D	D	O	S	D	D	D	D	O	O
Analistas de Disponibilidad	D	S	D	D				S	S	D		O
Agencia fundadora	O				O	O				O		
Usuarios de la comunidad científica	O				O	O				O		
D: Información detallada S: Algunos Detalles					O: Información de Resumen							

En la tabla 19 podemos relacionar los resultados obtenidos por las herramientas y la plataforma con las vistas de descomposición, generalización y uso

Tabla 19. Relación de resultados con vistas

	iPlasma			Softwarenaut		Moose			
Respuesta / Vista	Resumen de la complejidad del sistema	ClassBlueprint	Pirámide de Resumen	Vista de módulos	Vista de Clase	complejidad sistema	Paquetes	ClassBlueprint	Dependencia de paquetes
Descomposición				X			X		
Generalización	X					X			
Uso		X						X	X

La pirámide de Resumen y la Vista de Clase no se relacionan con ninguna de las vistas que queremos observar, pero se mantienen dentro de la documentación del sistema ya que se consideran que ofrecen información importante del sistema para su análisis. También en la tabla 20, podemos observar un resumen de lo que me permite recuperar cada una de las herramientas.

Tabla 20. Resumen Comparación de Herramientas.

Softwarenaut	iPlasma	Moose
• Descripción de alto nivel de las relaciones entre la arquitectura de los módulos en un sistema. • Número de líneas de código • Representación de paquetes, en los cuales se puede hacer una navegación a fondo.	• Métricas del sistema en general • Resumen de la Complejidad del sistema • Clases, relaciones de herencia, numero de atributos, numero de métodos, número de líneas de código • Permite ver componentes del sistema partiendo de una vista de complejidad donde se pueden ver métodos, atributos y funciones	• Entrega resultados con las mis características que se obtienen en iPlasma • No representa métricas del número de líneas de código. • Dependencias de paquetes • Permite ver paquetes del sistema como un árbol, clases de cada paquete, sub-paquetes y relaciones entre ellas. • No permite la exploración de cada una de las clases en forma individual

Según el análisis, las 3 herramientas permitieron solo obtener información de dos vistas; la lógica y la de implementación, en el resto de vistas analizadas su recuperación fue nula. La Recuperación con las herramientas ayuda a tener un concepto general de cómo está construido el sistema ArgoUML mostrando de forma visual la relación de los componentes, sus clases importantes y sus paquetes, por esta razón es importante analizar cómo estas herramientas sirven

de apoyo en la creación del método donde se pueda, no solo tener una recuperación visual, sino también poder obtener decisiones arquitecturales que se tuvieron en cuenta en la creación del sistema ayuden a guiar de una a un arquitecto, ya que en este estudio no se pudo recuperar estas decisiones, porque no se contaba con personas que participaron en la creación de ArgoUML. También es importante recalcar que las herramientas permiten realizar una recuperación en un menor tiempo a comparación de una recuperación en forma manual de forma manual, pero no habría certeza de que lo que recupera las herramientas sea viable.

4.2.1.2. Estudio de Caso Aplicación del método de evaluación ATAM-AR, en un sistema E-LearningSync:

1) Pregunta de Investigación: la pregunta de investigación del proyecto evidenció la necesidad de identificar que tan práctico es el método de recuperación de arquitecturas de software ATAM-AR, el cual se va ha tener en cuenta para la creación del método EV-AR. Así la pregunta que éste caso busca responder es ¿Qué tan practico e intuitivo es el método de recuperación de arquitecturas basado en la focalización evaluativa y la visualización de software, para guiar a un arquitecto de software?

2) Objetivo del Estudio de Caso: Determinar si el método de recuperación de arquitecturas basado en la focalización evaluativa y la visualización de software, sirve como guía a un arquitecto de forma práctica e intuitiva.

3) Selección del Estudio de Caso: de acuerdo a Runeson y Höst [18], Descriptivo y holístico con una unidad de análisis, la cual corresponde a un proyecto de desarrollo de software. La selección del caso siguió un criterio revelatorio.

4) Contexto del caso: Esta es una propuesta de arquitectura con el objetivo de soportar la sincronización por demanda de archivos asociados a actividades de usuarios en plataformas de aprendizaje en línea, como parte del trabajo de fin de carrera de Zambrano y Chagüendo [65]. El método ATAM-AR fue aplicado a esta propuesta arquitectónica para ver los resultados que se pueden obtener, en la cual participaron los diseñadores de la arquitectura quienes tomaron los roles del equipo de arquitectura, equipo desarrollador y grupo de interesados.

5) Diseño del Estudio de Caso: de acuerdo al objetivo del estudio de caso, fueron diseñados los indicadores, métricas e instrumentos a emplear, la Tabla 21 relaciona estos elementos para el estudio de caso exploratorio.

Tabla 21. Estudio de caso sistema E-learning sync

Pregunta de Investigación	Indicadores	Mediciones	Instrumentos	Aplicado a
¿Qué tan práctico es el método de recuperación de arquitecturas basado en la focalización evaluativa y la visualización de software, para guiar a un arquitecto de software?	Eficiencia = Esfuerzo vs Efectividad	Esfuerzo [Horas-Personas] Efectividad [Porcentaje de Conocimiento Recuperado respecto al conocimiento total estimado] Conocimiento [Vistas, Rationale]	Artefactos Previos Documento de Arquitectura Entrevista	Repositorio del Proyecto de Recuperación de Arquitectura en sistema E-LearningSync
¿Qué tan intuitivo es el método de recuperación de arquitecturas basado en la focalización evaluativa y la visualización de software, para guiar a un arquitecto de software?	Comprensibilidad Facilidad de Aprendizaje	• Percepción sobre la claridad del método. • Percepción sobre la facilidad en el uso del método. • Percepción sobre la Facilidad del Aprendizaje del Método	Observación Directa	Arquitecto Recuperador Stakeholders

.

6) Desarrollo del Caso: Siguiendo la primera tarea se realizó la presentación del método y la presentación de la arquitectura propuesta la cual se puede ver en la Figura 35.

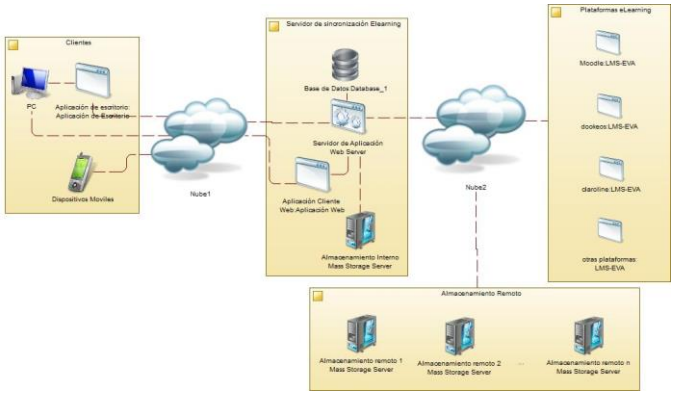


Figura 35 Arquitectura E-LearningSync

En la tarea Investigación y Análisis, se generaron los escenarios y se priorizaron, en la Tabla 22 se puede ver el árbol de utilidad de atributos de calidad que fue generado a partir de estos dos pasos. En el paso Generar árbol de utilidad de atributos de calidad de la tarea de Investigación y Análisis se analizó el escenario más relevante de los escenarios generados (importancia y complejidad de desarrollo), el cual corresponde con la integración de nuevas plataformas de E-Learning, debido que el principal compromiso del mecanismo fue la integración con distintas plataformas de aprendizaje en línea.

7) Dinámica del Proyecto

Para la realización de este acercamiento se buscó el apoyo y la asesoría del Doctor en ciencias de la computación Julio Ariel Hurtado profesor del Departamento de Sistemas de la Universidad del Cauca, quien hizo su acompañamiento en una sesión que permitió establecer los escenarios problemáticos que podrían presentarse en la arquitectura propuesta y realizar el análisis de por lo menos uno de los escenarios más críticos encontrados.

La sesión de validación de la arquitectura fue organizada y dirigida por el equipo de evaluación ATAM compuesto por el Doctor Julio Ariel Hurtado y los estudiantes de ingeniería de sistemas en trabajo de grado Edwar Alejandro Giraldo y Yuli Andrea Ordoñez Guzmán.

8) Resultados

Después de generar los escenarios y de priorízalos se tiene una tabla de resultados que es representada en la Tabla 22 (Ver ANEXO D. Apartado 2)

Tabla 22. Árbol de Utilidad de Atributos de Calidad

	Atributo de Calidad	**Escenario**	**Prio-ridad**
E-LearningSync	Escalabilidad	Almacenamiento	(H, M)
		Crecimiento de usuarios	(H, L)
	Integrabilidad	Integración de nuevas plataformas de E-Learning	(H, H)
	Modificabilidad	Evolución del Servidor de aplicaciones	(H,H)
		Tiempos de respuesta	(H, L)
	Desempeño	Fallos en el proceso de resolución de conflictos	(H, H)
	Confiabilidad	Fallos en el servidor de aplicaciones	(H, L)
		Fallos en el servidor de Almacenamiento	(H, L)

Para este caso no se realizaron más refinamientos, por lo que se procedió a analizar resultados y a su reporte, en la que se determina que la arquitectura está preparada

significativamente para la interoperabilidad, pero que se depende de las librerías de acceso a los servicios de cada plataforma de aprendizaje en línea, Tabla 23. (Ver ANEXO E, F y G. Apartado 2)

Tabla 23. Análisis Escenario Integración Plataformas De E-Learning

Descripción del Escenario	Mejorar la interoperabilidad[16] del mecanismo con las plataformas de aprendizaje en línea de tal manera que sea posible detectar la creación y actualización de recursos dentro de un curso e informar al servidor de sincronización para que este actualice los cambios si es necesario hacia los usuarios.		
Atributo	Integrabilidad/Interoperabilidad		
Estimulo	Lograr que los recursos creados desde las plataformas de aprendizaje puedan ser compartidos hacia los usuarios del mecanismo		
Respuesta	Para lograr esto se requiere: 3.5 personas/ mes para realizar las implementaciones propias para una nueva plataforma (El servidor de sincronización no será modificado)		
Decisiones Arquitecturales	**Puntos Sensibles**	**Trade Off**	**Riesgos**
Abstraer Servicios Web con el protocolo Rest	Servidor Web ElearningSync	Portabilidad	Ninguno relevante
Abstraer Servicios para cada plataforma de aprendizaje en línea	Servidor Web en la plataforma	Portabilidad	Algunas plataformas pueden restringir por seguridad la realización de algunas acciones por medio de servidores web
La comunicación entre componentes está enfocada en conexiones cliente / servidor ó P2P	Conexión	Disponibilidad	Posibles pérdidas de información
Rationale	Transparencia en el uso de las plataformas, dado que si el docente usa el mecanismo, los recursos pueden verse desde la plataforma como si hubiese sido creado desde ella misma. Interoperabilidad con distintas plataformas		
Vista de componentes del sistema	En la Figura 3 se presenta una vista de tipo componentes y conectores que incluye los servicios web, la plataforma de aprendizaje en línea y las conexiones que siguen el estilo arquitectónico cliente servidor.		

El esfuerzo dedicado a la ejecución del estudio de caso con ATAM-AR fue de 34 horas-persona con un equipo asesor de 3 personas, un experto y dos aprendices, y 2 personas de las interesadas en la arquitectura (gerencia y desarrollo). El tiempo invertido fue de 3 días incluyendo preparación (motivación y presentación del método), ejecución (incluyendo el modelado de la vista recuperada) y recolección de información del caso.

[16] la habilidad de los sistemas para trabajar juntos, en general gracias a la adopción de estándares.

A través de los resultados de la aplicación de ATAM-AR en un caso de recuperación de un modelo arquitectónico se puede concluir que las decisiones que se toman al momento de definir una arquitectura de un sistema proveen información relevante para su entendimiento y recuperación, por lo que considerar los interesados que han participado en el desarrollo de la arquitectura es clave. Por ello la inclusión de los desarrolladores o arquitectos que estuvieron a cargo del desarrollo del sistema, fue en este caso una decisión clave para obtener el rationale de una arquitectura Figura 36.

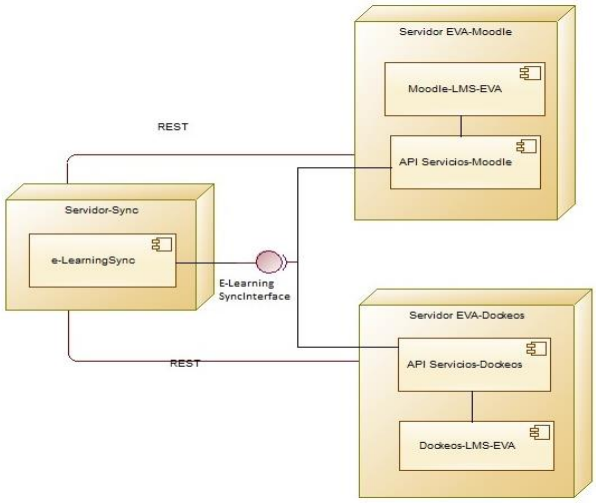


Figura 36. Vista de Componentes del sistema

Es importante rescatar el valor de la definición de escenarios como un mecanismo útil para la concentración de esfuerzos de análisis al momento de recuperar una arquitectura, de tal forma que el equipo pueda trabajar cohesivamente alrededor de una parte del problema.

La principal limitante de la recuperación aquí presentada está relacionada al no recuperar las vistas arquitecturales desde los artefactos del sistema (por ejemplo el código) en forma automatizada, aunque se quería presentar la recuperación de la arquitectura desde el conocimiento de los arquitectos, no hubo información concreta sobre la vista arquitectónica más allá de un modelo de despliegue como el presentado en el modelo relacional.

Teniendo en cuenta el esfuerzo necesario para la ejecución del proceso de recuperación del sistema, observamos que el uso de un método de recuperación basado en la focalización evaluativa nos ayudaba en gran medida a dar un orden a este proceso, se observa que el uso de los escenarios sirve para que el esfuerzo dedicado a la recuperación de un sistema sea realmente efectivo, esto nos permite centrarse en el análisis de una parte del sistema y la priorización de los escenarios

ayuda a que esas partes del sistema sea realmente importantes para los interesados en la recuperación del sistema.

Por último un factor importante para que el esfuerzo que se dedicó a esta recuperación fuera poco, es que los interesados y al mismo tiempo diseñadores de la arquitectura, tenían una base sólida de conocimientos sobre los conceptos utilizados en el área de las arquitecturas.

4.2.2. Estudio de Caso Aplicación del método de evaluación EV-AR, en una Pequeña Organización:

1) Pregunta de Investigación: la pregunta de investigación del proyecto evidenció la necesidad de determinar si el método de recuperación de arquitecturas basado en la focalización evaluativa y visualización de software, sirve de guía a un arquitecto de forma práctica e intuitiva. Así la pregunta que este caso busca responder es: ¿Qué tan práctico es el método de recuperación de arquitecturas basado en la focalización evaluativa y la visualización de software, para guiar a un arquitecto de software? ¿Qué tan intuitivo es el método de recuperación de arquitecturas basado en la focalización evaluativa y la visualización de software, para guiar a un arquitecto de software?

2) Objetivo del Estudio de Caso: Determinar si el método de recuperación de arquitecturas basado en la focalización evaluativa y la visualización de software, sirve de guía a un arquitecto de forma práctica e intuitiva.

3) Selección del Estudio de Caso: de acuerdo a Runeson y Höst [18], Revelatorio-Descriptivo y holístico con una unidad de análisis, la cual corresponde a un proyecto de desarrollo de software.

4) Contexto del caso: Nexura Internacional[17] es una pequeña empresa productora de software colombiana con sedes en Bogotá, Cali y Medellín, con más de 14 años de experiencia. Desde el año 2002 se convirtió en la compañía con el mayor número de implementaciones de la Estrategia de Gobierno en Línea en Colombia. Este conocimiento les permite entender y atender oportunamente las necesidades de cada organización, apoyándolos de la mejor forma a realizar esta implementación en sus sitios web de una manera acertada y eficiente para lograr sus objetivos. A finales del 2015 cuenta con 63 empleados. Esta empresa está centrada en la creación de soluciones tecnológicas de gobierno en línea (GEL), teniendo como clientes principales las entidades públicas.

[17] https://www.nexura.com/

Durante sus años de operación ha construido soluciones para diferentes tipos de clientes orientadas a la implementación de Gobierno en Línea, la empresa cuenta con premios y reconocimientos como : Nexura Platform e-gov[18] La Tinatec 2014 - Solución Informática para Gobierno, esta plataforma fue presentada en la convocatoria 648 de 2014 de COLCIENCIAS, llamada Locomotora de la Innovación[19], la cual fue seleccionada para el proceso de innovación de la plataforma, en donde se implementó el enfoque de líneas de producto de software.

Dentro de sus necesidades para la implementación de su línea de productos, estaba la recuperación de la arquitectura de su plataforma, lo que permitió el planteamiento de este estudio de caso.

5) Diseño del Estudio de Caso: de acuerdo al objetivo del estudio de caso, fueron diseñados los indicadores, métricas e instrumentos a emplear, la Tabla 24 relaciona estos elementos para el estudio de caso.

Tabla 24. Métricas Estudio de Caso Nexura

Pregunta de Investigación	Indicadores	Mediciones	Instrumentos	Aplicado a
¿Qué tan práctico es el método de recuperación de arquitecturas basado en la focalización evaluativa y la visualización de software, para guiar a un arquitecto de software?	Eficiencia = Esfuerzo vs Efectividad	Esfuerzo [Horas-Personas] Efectividad [Porcentaje de Conocimiento Recuperado respecto al conocimiento total estimado] Conocimiento [Vistas, Rationale]	Artefactos Previos Documento de Arquitectura	Repositorio del Proyecto de Recuperación de Arquitectura en una SPL.
¿Qué tan intuitivo es el método de recuperación de arquitecturas basado en la focalización evaluativa y la visualización de software, para guiar a un arquitecto de software?	Comprensibilidad Facilidad de Aprendizaje	• Percepción sobre la claridad del método. • Percepción sobre la facilidad en el uso del método. • Percepción sobre la Facilidad del Aprendizaje del Método	Observación Directa	Arquitecto Recuperador Stakeholders

[18] https://www.nexura.com/Publicaciones/nexura_platform_e-gov
[19] http://www.colciencias.gov.co/node/661

6) Desarrollo del Estudio de Caso**:** El proceso de la recuperación de la arquitectura se realizó durante el periodo comprendido entre Junio y Noviembre del 2015, donde se realizaron aproximadamente 30 sesiones con una duración de 2 horas cada una en el cual participaron: El gerente de Nexura, El gerente de desarrollo, gerente de requisitos y dos desarrolladores.

   Para iniciar con la recuperación de la arquitectura del sistema, se realiza una socialización, en estas sesiones de socialización se contó con el gerente de la empresa, los gerentes de las áreas técnicas y el equipo de recuperación, donde los integrantes de la empresa planteaban su conocimiento sobre el sistema y cuál es el dominio al cual apunta este, donde se llega a un dominio en común y se identifican las diferentes implementaciones de la plataforma que van a ser analizadas para la definición de la arquitectura de la línea de productos de la empresa.

   Ya teniendo conocimiento sobre el sistema y las expectativas que tenía la empresa en cuanto a lo que se quería saber sobre su sistema, el arquitecto inicia la exploración de las posibles herramientas de recuperación de arquitecturas que podrían ayudar en el proceso, en esta exploración sólo se identifica una herramienta candidata para el proceso, esto debido a las características de los sistemas a los cuales se iba recuperar su arquitectura, debido a que la plataforma ha sido construida durante aproximadamente 10 años tiende a tener diferentes paradigmas de programación, también ha sido construida con diferentes tecnologías, para lo cual no se encontraron herramientas de recuperación que soportaran estas características, debido a esto se limita a encontrar una herramienta que pueda recuperar sobre la tecnología con la cual se ha construido la mayoría del sistema, se realizó el análisis de la herramienta phpModeler, pero al probar la herramienta no fue posible su instalación ya que se generaron excepciones que no se solucionaron aun cuando se consultó sobre las posibles soluciones a estos problemas en diferentes foros.

   Esta situación se expone en la siguiente sesión y se propone la recuperación manual del sistema, para lo cual los integrantes de la organización acceden, se definen sesiones adicionales con los desarrolladores que cuentan con más experiencia en la aplicación para realizar la recuperación manual de la arquitectura del sistema, en la Figura 37 se muestra el proceso de recuperación manual que se siguió para obtener las vistas del sistema.

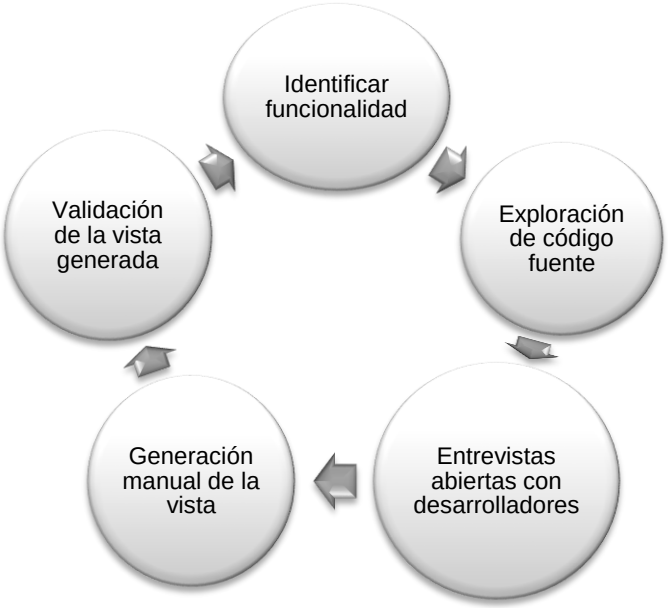


Figura 37. Proceso de Recuperación Manual de la Arquitectura de Nexura Platform.

De forma paralela se realizan sesiones con los gerentes de las áreas técnicas para identificar las necesidades que tenían para realizar la recuperación del sistema, teniendo en cuenta estas necesidades el arquitecto y los desarrolladores plantean, describen y priorizan los escenarios para la recuperación.

Teniendo los escenarios priorizados inician las tareas de recuperación visual de la arquitectura, análisis y recuperación del rationale, el proceso que se siguió para ejecución de estas tareas, fue mediante entrevistas abiertas, el arquitecto y los desarrolladores tomaban un escenario, aquí los desarrolladores procedían a identificar los diferentes componentes del sistema que estaban involucrados en el escenario, además de explicar las responsabilidades de cada uno, luego el arquitecto generaba las visualizaciones teniendo en cuenta la explicación de los desarrolladores y el código fuente de la aplicación.

En sesiones posteriores el arquitecto indagaba sobre las posibles decisiones arquitecturales que se tomaron para la construcción del sistema, en ocasiones los desarrolladores no contaban con esta información, y se presentaban dos situaciones, que se tiene acceso al desarrollador que estuvo involucrado en ese escenario, o que no se tenía acceso ni información adicional para adicionar a la recuperación, en el primer caso se contactaba al involucrado y se hacía la respectiva entrevista sobre las decisiones que se tomaron para construir el sistema, en el segundo caso se cierra el escenario quedando sin posibles decisiones arquitecturales, solo con la descripción de cómo funciona el sistema en este escenario.

Luego el arquitecto y los desarrolladores, se reunieron para la construcción del diario del rationale, para esto se toma cada escenario y el arquitecto presenta a los desarrolladores la vista que generada, para que estos aprueben o realicen modificaciones sobre la vista, una vez se tiene consenso sobre la vista generada, se empiezan a registrar el escenario en el diario del rationale, luego se plasma la vista generada, seguido de la lista de posibles decisiones arquitecturales que se tuvieron en cuenta para el desarrollo de este escenario, por último se registra cuál fue la decisión tomada para el desarrollo de este escenario, con su respectiva justificación.

Dentro de la realización de esta tarea se presentó un escenario donde los desarrolladores sugirieron realizar una sesión donde se invitara al gerente de la empresa, ya que él fue quien realizó el desarrollo de este escenario para el sistema, para esta sesión la ejecución fue diferente debido a la disponibilidad del gerente, en esta se presentó el escenario al gerente, para lo que él expuso cómo fue la experiencia para el desarrollo de dicho escenario, esta información fue capturada y luego con ayuda de los desarrolladores fue depurada y se generó la vista para este escenario.

Por último, se realiza una presentación de todos los escenarios que fueron recuperados, junto con un documento de arquitectura que ayude a soportar el proceso de definición la línea de productos de la organización.

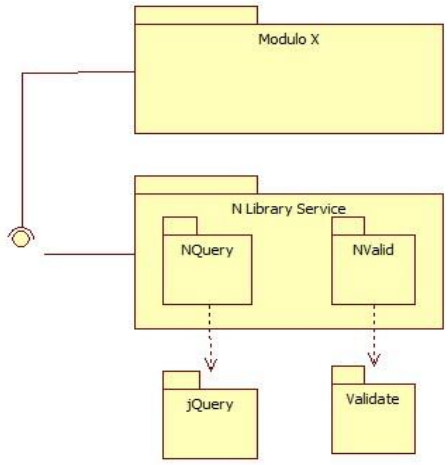
En la tabla 25 se presenta uno de los resultados obtenidos en la recuperación de la arquitectura.

7) Resultados Obtenidos:

a) Resultados cuantitativos: La Tabla 26 relaciona los resultados cuantitativos obtenidos en el estudio de caso, estos permiten ver que la recuperación de la arquitectura es bastante compleja ya que la empresa de desarrollo manejaba tres paradigmas de Programación muy diferentes (estructurado, funcional y orientado a objetos). Después de realizar la recuperación de forma manual se obtienen los resultados de la tabla 25, donde se distribuyen 6 variables las cuales se identifican así: Vistas, Peso del ítem, total de vistas detectadas, total de vistas analizadas, el porcentaje de recuperación con respecto al peso y porcentaje de recuperación con respecto a la totalidad de vistas. Se le dio a cada vista un porcentaje significativo, donde se identificaron 14 escenarios de calidad, de estos se decidió trabajar con 4 escenarios donde se realizó la descripción detallada y la recuperación de la arquitectura para cada escenario, también se encontraron 94 vistas lógicas de las cuales se analizaron 54 y por último se descubrieron 49 vistas de implementación donde se analizaron 24.

Tabla 25.Diario del Rationale obtenido en el proceso de recuperación.

Escenario	Necesidad de facilitar el cambio de la librería Jquery	**Atributo**	Modificabilidad
Estimulo	Equipo de Desarrollo	**Respuesta**	Se espera que este cambio sea realizado con un esfuerzo no mayor a 1.5 personas mes



Decisiones Arquitecturales	**Puntos Sensibles**	**Trade Off**	**Riesgos**
Uso del patrón Bridge	Modulo	Desempeño, Portabilidad	Se puede requerir un esfuerzo mayor para el desarrollo de un nuevo módulo.

Rationale	**Alternativa**	**Justificación**	**Razones**
	Se plantea implementar una capa entre los módulos de la plataforma y la librería	Desacoplar una abstracción de su implementación	El objetivo de esta capa es concentrar los cambios sobre el código que la actualización de librería pueda implicar.

Se calcula el porcentaje de recuperación de cada una de las vistas y podemos finalizar el análisis mostrando en la Figura 38 que de las 5 vistas tenidas en cuenta solo se pudo recuperar las que hacen parte de los escenarios, la lógica y la implementación, en cuanto a las vistas de procesos e instalación su recuperación fue nula. El porcentaje de recuperación se calculó con respecto al

peso que se definió para cada una de las vistas que nos ayuda a calcular el porcentaje total teniendo en cuenta el peso y el porcentaje recuperado de cada una de las vistas donde se procede a calcular la efectividad (porcentaje de Conocimiento Recuperado respecto al conocimiento total estimado), realizando la suma de cada una de estas y por último se hace el cálculo de esfuerzo horas persona de acuerdo al tiempo de cada una de las personas que participaron en la recuperación.

Tabla 26. Porcentaje recuperado en algunas vistas.

Vistas	Peso del Item	Total Detectados	Total de vistas analizadas	Porcentaje de recuperación	%porcentaje con respecto a su totalidad (%PI*%rec)/100%
Escenarios	17,5%	14	4	28,57%	5,00%
Lógica	30%	94	54	57,45%	17,23%
Procesos	17,5%	0	0	0,00%	0,00%
Implementación	17,5%	49	24	48,98%	8,57%
Instalación	17,5%	0	0	0,00%	0,00%
Efectividad					30,81%
Esfuerzo(Horas persona)					60

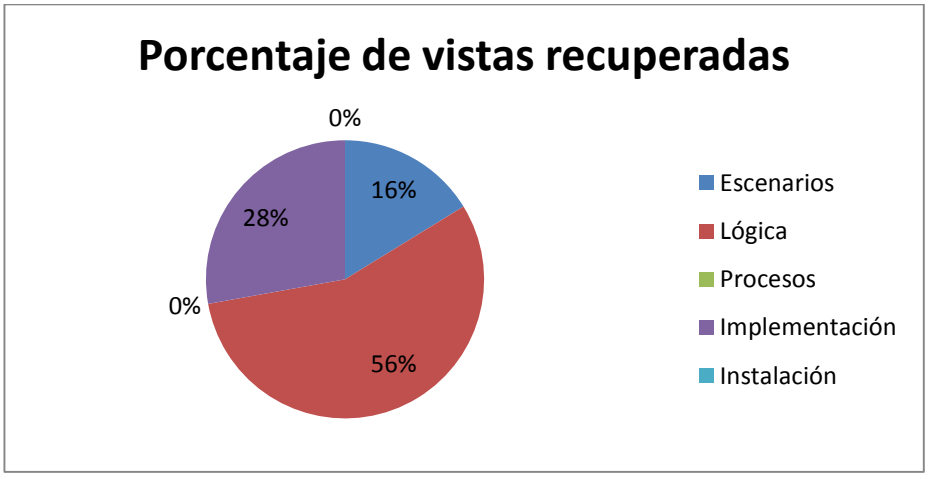


Figura 38. Vistas Recuperadas.

También fueron encontradas diferentes decisiones arquitectónicas donde se identificaron 12 rationale, 6 tácticas y tres patrones arquitecturales en la Figura 39 se pueden ver el detalle de cada uno de estos.

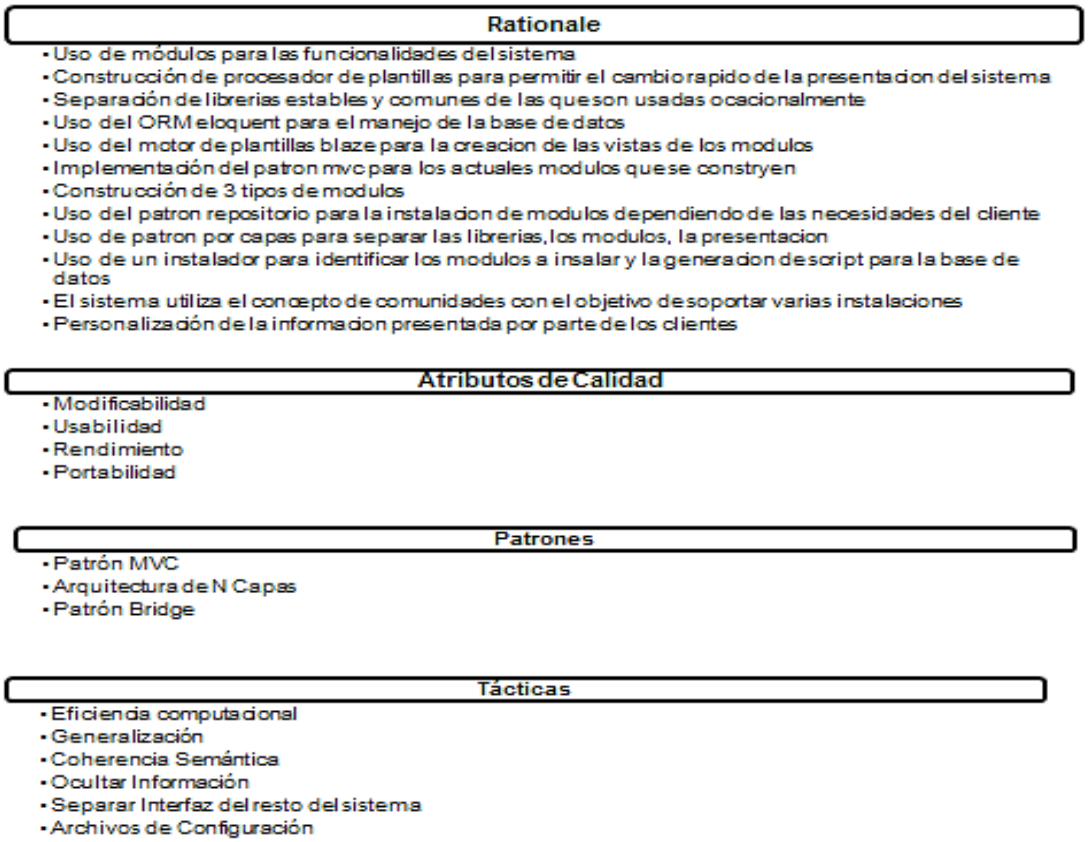


Figura 39. Decisiones Arquitectónicas.

b) Resultados cualitativos: Durante la recuperación de la arquitectura el grupo logró trabajar en equipo, ya que todos estaban muy interesados en la recuperación en especial el gerente, aprovechando los espacios en conjunto, esta fue una fortaleza del estudio de caso. La comunicación con los desarrolladores e interesados en la recuperación fue fácil, lo que permitió identificar los requisitos del conjunto de productos, y considerar los elementos comunes fue un cambio de concepción y de estructura.

La visualización fue el factor que determinó la rapidez con la cual se ejecutó el proceso de recuperación de la arquitectura del sistema, esto debido que no se encontró una herramienta que ayude al proceso de recuperación ya que se deben tener en cuenta muchos aspectos, entre ellos están las diferentes tecnologías que se usan para el desarrollo, los resultados que nos da la herramienta e incluso hay que tener en cuenta los paradigmas de programación en los cuales fue construido el sistema. En este caso no fue posible encontrar una o varias herramientas que ayudaran a este proceso por lo que se vio la necesidad de

realizar la recuperación de forma manual, esta recuperación es mucho más lenta y es más propensa a errores por lo que fue necesario realizar validaciones con los desarrolladores del sistema el cual consistía en darles a conocer las vistas generadas y parte de código que se estaba representando, si los desarrolladores no comprendían las vistas se procedía a revisar con ellos el código de forma iterativa hasta que la visualización de las vistas quedaran claras.

Al realizar la recuperación de la arquitectura de los sistemas nos permitió encontrar diferencias entre los sistemas, estas diferencias fueron analizadas junto con los desarrolladores y teniendo en cuenta los resultados que se iban produciendo en el proceso de definición de la línea de productos en especial el modelo de características de la línea, se pudieron identificar que partes de la arquitectura de los sistemas, iban a ser incluidas en la arquitectura de la línea de productos del sistema y la identificación de los puntos de variabilidad de esta.

El sistema cuenta con una característica importante ya que fue construido para que se puedan colocar módulos independientes y estos sean integrados a la plataforma sin ocasionar grandes cambios al resto de módulos que el sistema tenga instalado, estos módulos tienen diferentes características y para identificarlos, la empresa los clasificó por tipos, la visualización del sistema nos permitió identificar las dependencias que había entre módulos e incluso se identificaron dependencias que los desarrolladores no tenían presentes y que no se encontraban en la documentación del sistema.

Las entrevistas con los desarrolladores nos permitieron entender el funcionamiento del sistema e información sobre las dificultades que se presentaron para el desarrollo de algún componente del sistema y como fue el proceso para solucionar dichas dificultades, que opciones se tuvieron en cuenta para la solución y cuáles fueron las razones que tuvieron para escoger una opción sobre otra.

4.2.3. Estudio de Caso Aplicación del método de evaluación EV-AR, en el sistema FUNCAVO

1) Pregunta de Investigación: la pregunta de investigación del proyecto evidenció la necesidad de determinar si el método de recuperación de arquitecturas basado en la focalización evaluativa y visualización de software, sirve de guía a un arquitecto de forma práctica e intuitiva. así la pregunta que este caso busca responder es ¿Qué tan práctico e intuitivo es el método de recuperación de arquitecturas basado en la focalización evaluativa y la visualización de software, para guiar a un arquitecto de software?

2) Objetivo del Estudio de Caso: Determinar si el método de recuperación de arquitecturas basado en la focalización evaluativa y la visualización de software, sirve de guía a un arquitecto de forma práctica e intuitiva.

3) Selección del Estudio de Caso: de acuerdo a Runeson y Höst [18], Confirmatorio-Descriptivo y holístico con una unidad de análisis, la cual corresponde a un proyecto de desarrollo de software.

4) Contexto del caso: La Fundación Casa del Vocal es una institución dedicada al apoyo, asesoramiento, capacitación y formación de los vocales de control del Cauca, para fortalecer la participación de los usuarios en la prestación de los Servicios Públicos Domiciliarios y las TICS, a través de los Comités de Desarrollo y Control Social. Con el fin de sistematizar los procesos que se llevan dentro de la fundación y tener control sobre los documentos que tienen, se construyó el sistema llamado FUNCAVO. El método EV-AR fue aplicado a este sistema, donde se llevó a cabo cada uno de los pasos de la guía con la participación de dos recuperadores, un experto y el desarrollador de la aplicación. Para la recuperación visual de la arquitectura del sistema, fue usada la herramienta iPlasma.

5) Diseño del Estudio de Caso: de acuerdo al objetivo del estudio de caso, fueron diseñados los indicadores, métricas e instrumentos a emplear, la Tabla 27, relaciona estos elementos para el estudio de caso confirmatorio -Descriptivo.

Tabla 27. Medición estudio de caso FUNCAVO

Pregunta de Investigación	Indicadores	Mediciones
¿Qué tan práctico es el método de recuperación de arquitecturas basado en la focalización evaluativa y la visualización de software, para guiar a un arquitecto de software?	Eficiencia = Esfuerzo vs Efectividad	• Esfuerzo [Horas-Personas] • Efectividad • [Porcentaje de Conocimiento Recuperado respecto al conocimiento total estimado] • Conocimiento [Vistas, Rationale]
¿Qué tan intuitivo es el método de recuperación de arquitecturas basado en la focalización evaluativa y la visualización de software, para guiar a un arquitecto de software?	Comprensibilidad Facilidad de Aprendizaje	• Percepción sobre la claridad del método. • Percepción sobre la facilidad en el uso del método. • Percepción sobre la Facilidad del Aprendizaje del Método • Medidas de Facilidad

6) Desarrollo del Estudio de Caso: La recuperación del sistema FUNCAVO, se realizó en una sesión de 5 horas donde participaron 4 personas distribuidas así: 3 interesados en la recuperación y el desarrollador del sistema FUNCAVO, inicialmente se realiza la presentación del método en donde se explican cada una de las tareas que se van a realizar, a continuación se presenta el sistema donde se muestran sus funcionalidades y la arquitectura propuesta para la construcción del sistema la cual se puede ver en la Figura 40.

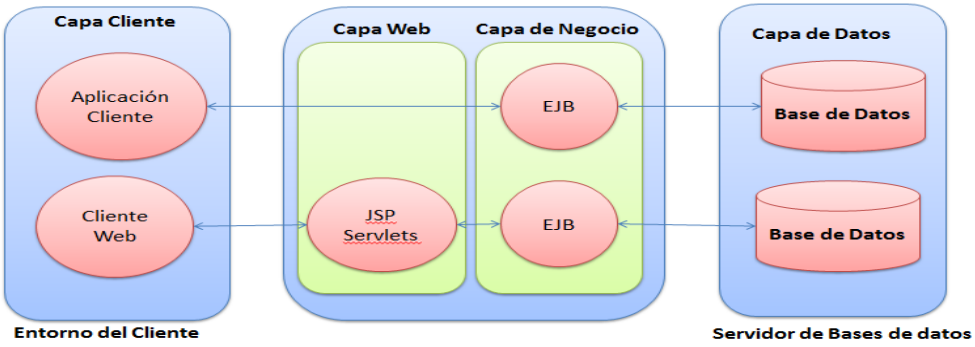


Figura 40. Arquitectura N Capas J2EE

La arquitectura propuesta consiste en una arquitectura de N capas, la cual era soportada por la plataforma de programación J2EE o también llamada Java Empresarial, esta arquitectura presenta una capa cliente donde se encuentra el medio por el cual el cliente se comunica con el resto de capas de la aplicación, para el sistema FUNCAVO se usa un cliente web para este fin. Seguido se encuentran la capa web y la capa de negocio, en estas se encuentran la lógica de negocio de la aplicación y los recursos web que son accedidos por el cliente. Y por último se encentra la capa de datos, donde se almacenan los datos de la aplicación.

Para la parte final de la tarea de iniciación se planteó el objetivo del proceso de recuperación, el cual fue identificar la arquitectura de FUNCAVO para reutilizarla en próximos proyectos que sean construidos bajo la plataforma de programación J2EE.

En la tarea identificación y priorización de escenarios, se realizó una lluvia de ideas con el fin de identificar posibles escenarios, seguido de la priorización de estos, donde se construyó el árbol de atributos de calidad y se identificó que los escenarios analizados fueron: Comportamiento del sistema ante ataques de seguridad y Migración de Datos.

Para la recuperación visual del sistema FUNCAVO se usó la herramienta iPlasma donde se obtuvieron varias vistas, como el resumen de complejidad del sistema, ver figura 41, esta vista nos muestra una recopilación de métricas acerca del sistema en general, donde se muestran indicadores para observar si estas están o no en el rango de la industria, al observar el resumen de complejidad nos damos cuenta que se presentan bajas relaciones de herencia, las clases presentan un numero de métodos que está dentro del promedio de lo establecido en la industria, y que por cada paquete del sistema se encuentran pocas clase, también encontramos que los métodos tienden a tener pocas líneas de código y que su complejidad ciclomática se encuentra dentro del promedio, por ultimo identificamos que se presenta un bajo acoplamiento dentro del sistema al ver que hay pocos llamados hacia otras clases.

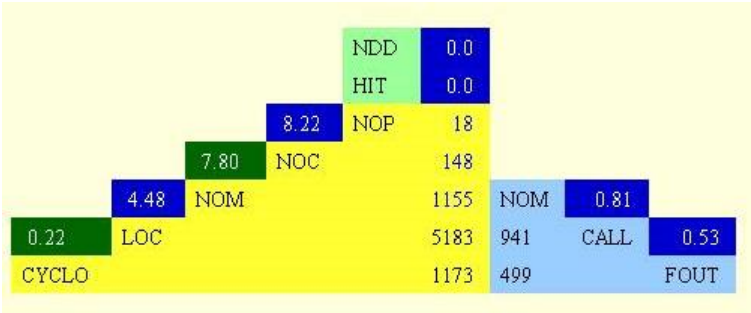


Figura 41. Resumen de Complejidad FUNCAVO

Para analizar la seguridad del sistema, se observa la clase AccionIniciarSesion para esto obtenemos el blueprint de esta, aquí se identificó que la clase cuenta con seis atributos, con un método de acceso para lectura y dos métodos para escritura, finalmente se encuentra que la función getUsuario es la encargada de identificar el usuario dentro del sistema. En la Figura 42 se puede ver el blueprint de la clase analizada.

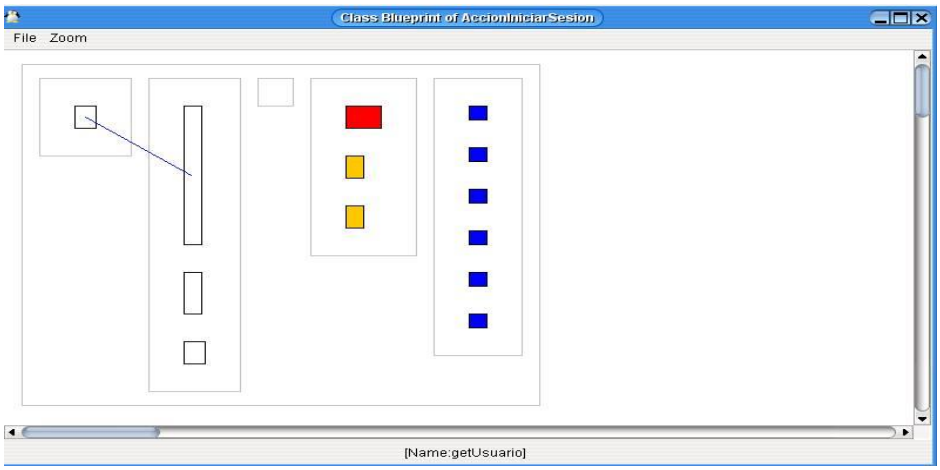


Figura 42. Visualización de la clase AccionIniciarSesion

Por último se identificó con ayuda del desarrollador que aunque la clase AccionIniciarSesion era quien identifica al usuario, la clase LoginFilter es la encargada de aplicar filtros sobre los recursos del sistema, usando la herramienta iPlasma se obtiene el blueprint de esta clase para ver cómo está construida, la clase no presenta atributos y métodos de acceso a los atributos de la clase, solo encontramos el método doFilter el cual hace el llamado del método no proteger, quien identifica si el recurso está protegido para su acceso o no. En la Figura 43 se muestra el blueprint de la clase LoginFilter.

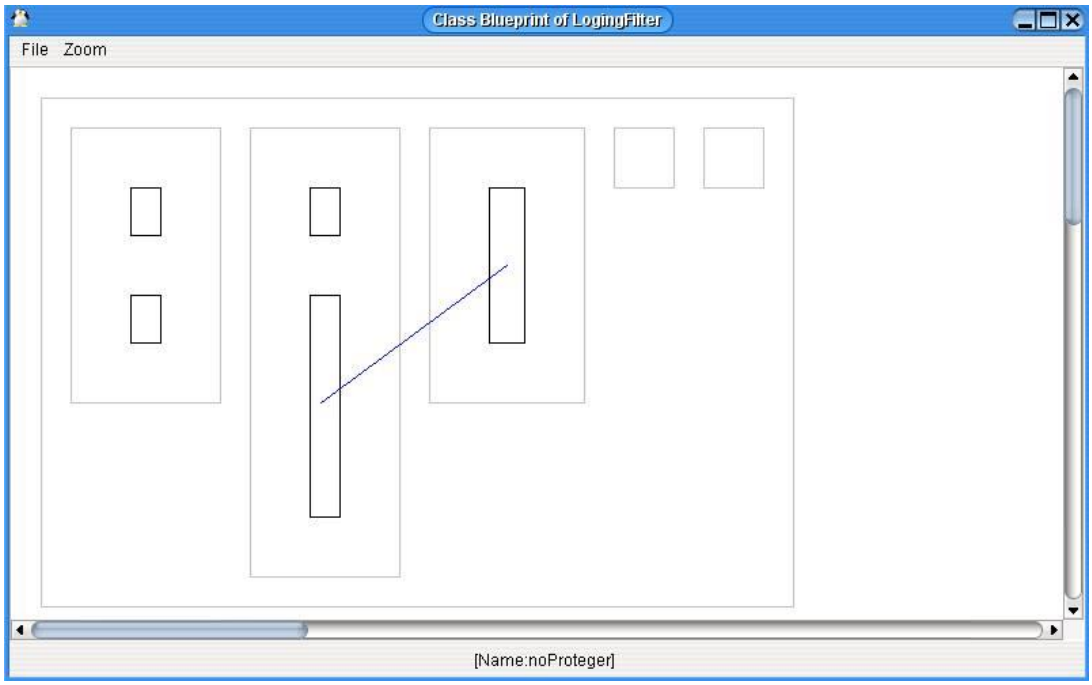


Figura 43. Loginfilter

Como mecanismo de validación se observa el código fuente para identificar la clase LoginFilter y analizar el funcionamiento del método doFilter donde encontramos que se crea una instancia de la clase AccionIniciarSesion, ver Figura 44.

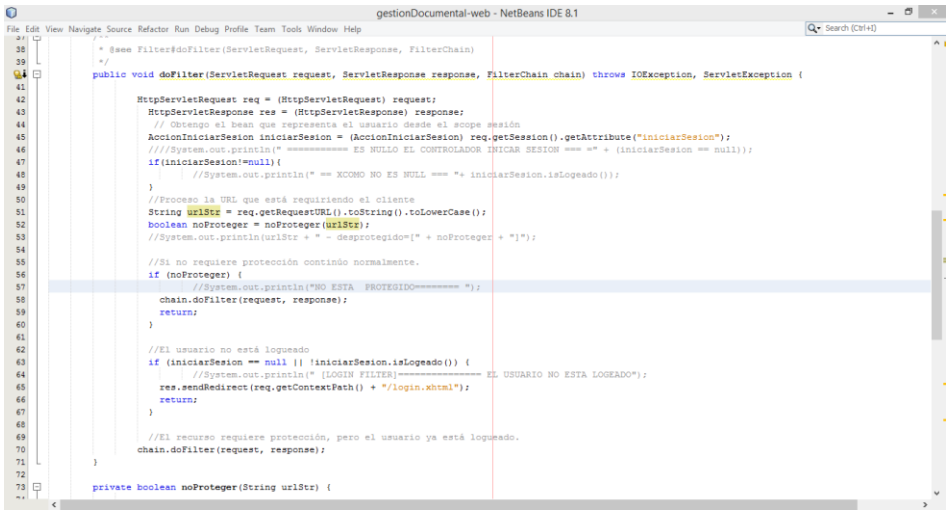


Figura 44. Método doFilter

8) Resultados

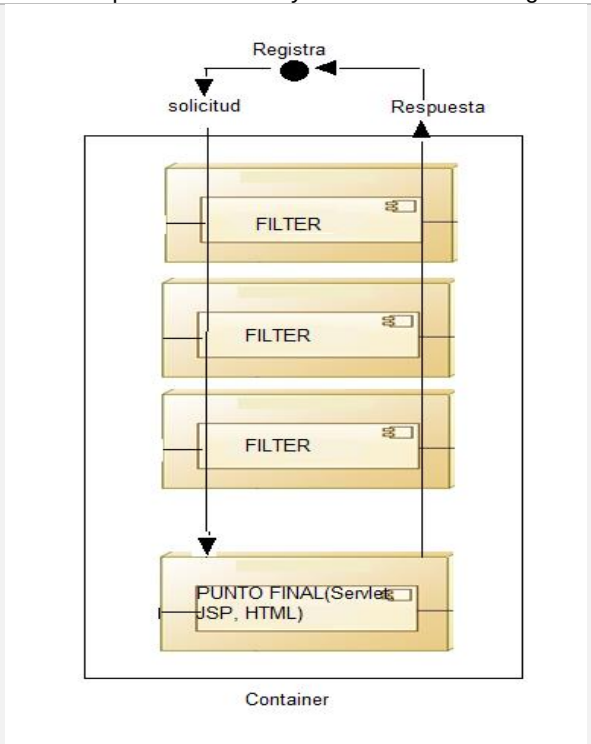
Después de generar los escenarios y de priorizarlos se tiene una tabla de resultados que es representada en la tabla 28. Esta tabla presenta el árbol de utilidad de atributos de calidad el cual fue construido a partir de una lluvia de ideas sobre los posibles escenarios que el desarrollador deseaba analizar en el sistema FUNCAVO, dentro de este análisis el desarrollador indicaba que la forma en que él había desarrollado el mecanismo para que el usuario tuviera permisos sobre un recurso del sistema le había parecido el más correcto, por lo que deseaba recordar cómo se había hecho en ese momento, también expresaba que una acción de mejora que haría para el sistema era la forma en que se estaban almacenando los documentos en el sistema, no le parecía la más correcta en caso de querer soportar una posible migración de datos, así que necesitaba ver que otras posibilidades habían, para el analizar porque tomo la decisión de desarrollar la persistencia de documentos como estaba construida en ese momento. Teniendo en cuenta las razones expuestas por el desarrollador se decide dar la mayor prioridad a los escenarios: Comportamiento del sistema ante ataques de seguridad y Migración de Datos (Ver ANEXO H e I. Apartado 3).

Tabla 28. Árbol Atributos de Calidad

Atributos de calidad	Escenario	Prioridad	Orden de Prioridad
Seguridad	Comportamiento del sistema ante ataques de seguridad	(H,H)	1
	Mejorar el inicio de sesión	(H,M)	2
	Cambiar librería de componentes visuales (Prime Faces)	(M,M)	3
Rendimiento	Optimización de consultas JPQL	(L,M)	4
	Optimizar el rendimiento de la interacción de la aplicación	(H,M)	2
	Migración de Datos	(H,H)	1
Modificabilidad	Cambio de JDK del 16 al 18	(M,H)	2
	Incluir un módulo nuevo al sistema	(L,M)	4

Se procedió a analizar los escenarios ya priorizados junto con los resultados obtenidos a partir de la recuperación visual de la arquitectura del sistema, donde se construyó el diario del rationale para cada uno de estos, ver tablas 29 y 30. (Ver ANEXO J y H. Apartado 3).

Tabla 29. Análisis y Rationale del atributo Seguridad.

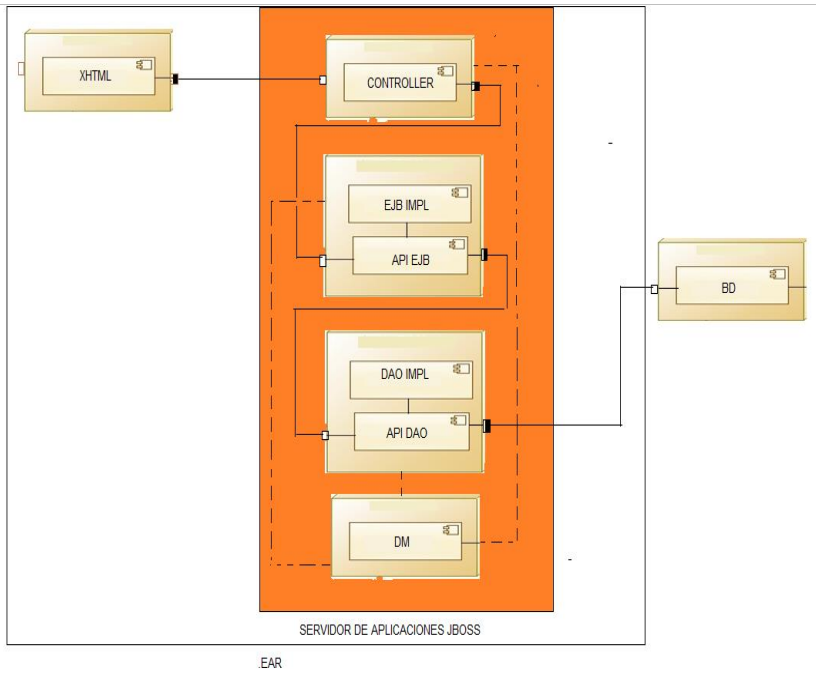
Descripción del Escenario	**Comportamiento del sistema ante ataques de seguridad. Hacker**		
Atributo	Seguridad		
Estimulo	Cliente		
Respuesta	Restricción del acceso durante algún tiempo		
Decisiones Arquitecturales	Puntos Sensibles	Trade-Off	Riesgos
Aplicar seguridad en Jboss, Aplicar JavaServerFilter y separa módulo de seguridad	La app también validaría, muchas peticiones al servidor y demora en el desarrollo	Rendimiento	Más horas de desarrollo y análisis
Alternativa	Aplicar Java server filter		
Justificación	Ofrece buen nivel en la seguridad y permite un desarrollo rápido.		
Razones	Poco tiempo de desarrollo y reutilización del código		
Rationale			

Para el análisis del escenario Comportamiento del sistema ante ataques de seguridad, se inicia con la descripción detallada de este, donde se identifica que el escenario se refiere a el comportamiento del sistema cuando un usuario intenta acceder a recursos del sistema a los cuales no tiene permiso, para esto el desarrollador expresaba que habían tres posibilidades: usar la seguridad del servidor de aplicaciones Jboss, construir y aplicar los Java Server Filters, o construir desde cero un módulo de seguridad.

El desarrollador tomo la decisión de usar los Java Server Filter porque permitían aplicar un nivel alto de seguridad, además de ser un desarrollo muy rápido, en cambio si usa la seguridad de Jboss de igual manera debía construir validaciones de seguridad en la aplicación y si optaba por construir un módulo de seguridad desde cero esto le tomaría tiempo para realizar un análisis de lo que debía tener este módulo y el desarrollo del mismo. Por último se generó una vista donde se explica cómo funcionan los Java Server Filter para complementar la información obtenida a partir de las vistas obtenidas con la herramienta iPlasma.

Tabla 30.Análisis y Rationale del escenario migración de datos.

Descripción del Escenario	Migración de Datos		
Atributo	Modificabilidad		
Estimulo	Cliente		
Respuesta	Poder soportar una correcta migración de datos del sistema, incluyendo los documentos subidos a este		
Decisiones Arquitecturales	**Puntos Sensibles**	**Trade-Off**	**Riesgos**
Bases de datos Multimedia	Desconocimiento		
Campo Blob en la base de datos	Necesidad de mayor almacenamiento	Escalabilidad	Corrupción de Datos
Guardar el archivo en otro servidor y la ruta en base de datos	La migración de los documentos implica el cambio de la ruta en la base de datos.	Modificabilidad	Mayor tiempo de desarrollo y problemas de rutas de los archivos
Alternativa	Campo BloB en base de datos		
Justificación	Mayor agilidad en el desarrollo y cumple ligeramente con las expectativas		
Razones	Facilidad de desarrollo, cumple con el estándar JPA y con la conversión de arrays e bits Facilita la integración primeface de archivos.		

Rationale	
Descripción	El XHTML (eXtensible HyperText Markup Language), hace una petición al servidor por medio del controlador (clases Java) que está en comunicación con la paginas HTML, se hace la comunicación hacia los EJB, el EJB es donde se encuentra toda la lógica del negocio. En el controlador validaciones sencillas, los EJB una vez reciben la petición del servidor se preguntan si es necesario recurrir a la base de datos el módulo DAO (acceso a datos) por medio de consultas Java Persistence query language (JPQL) o por medio SQL, si es el caso. Tanto en el EJB como en el paquete DAO existe un patrón llamado fachada o abstrac Factory que consiste en la representación de interfaces con unos métodos que se utilizan entonces la interface solo muestra el métodos pero a su vez en el paquete DAO existe una implementación de las interfaces el EJB se comunica solo por medio de estas interfaces y luego realiza la llamada a las diferentes implementaciones, lo mismo pasa por el DAO. Por último el DM el dominio o entidades es la representación de la base de datos hecha clases, se constituye con ayuda de representación de anotaciones que ayuda a representación de la base de datos en el paquete DM disponible para todos los paquetes del EJB, DAO y el Controlador

Para el escenario de migración de datos, el desarrollador nos expuso que para el desarrollo se planteó la utilización de una base de datos multimedia, pero debido al desconocimiento de esta tecnología fue descartada de inmediato, como segunda alternativa estaba el almacenamiento de los documentos en un servidor aparte y guardar en la base de datos la ruta donde se encuentran, pero esta solución planteaba que el servidor debía tener mayor almacenamiento, por ultimo estaba la posibilidad de serializar los documentos y guardarlos en un campo de la base de datos, este último mecanismo se usó en la construcción del sistema porque le permitía una mejor integración con PrimeFaces para el guardar y mostrar estos documentos.

7) Análisis de resultados

La recuperación de la arquitectura del sistema FUCAVO se realizó en una sesión de 5 horas por lo tanto el esfuerzo equivale a 20 horas, se tuvieron en cuenta dos atributos de calidad: la seguridad y la modificabilidad., la percepción de efectividad con respecto al conociendo del desarrollador del sistema, fue aceptable ya que considera que aún hace falta más documentos que ayuden o faciliten entender mejor la parte que se evalúa y cómo ésta afecta en su recuperación. Sin embargo, aunque existe un acercamiento de lo que se intenta recuperar, quizás no es muy clara la evaluación de algunos puntos.

Para obtener la percepción de facilidad de uso, comprensibilidad y facilidad de aprendizaje del método, se realizó una encuesta al desarrollador del sistema, en esta se pidió que indicara una calificación entre un rango de valores con respecto a unas preguntas que se le presentaron, la escala estaba descrita de la siguiente forma:

1. Total acuerdo
2. Acuerdo
3. Aceptable
4. Desacuerdo
5. Total desacuerdo

La apreciación sobre la facilidad de uso que tuvo el desarrollador fue “Aceptable”, además manifiesta que existen algunas ambigüedades en la identificación de algunos puntos y en conclusión se reconoce que la guía da una mejor comprensión de la arquitectura que se intenta recuperar, ya que ofrece un panorama general y ayuda a identificar los puntos importantes de esta.

Con respecto a la facilidad de aprendizaje el desarrollador responde, pero manifiesta que el manejo de los conceptos tales como los de atributos de calidad fue difícil de entender y en la priorización de los escenarios consideró ambigua la escala que se propuso para esto.

En el uso de la herramienta de recuperación iPlasma, esta permitió encontrar o entender los puntos importantes de cualquier elemento, de esta manera al realizar la recuperación de la arquitectura de manera visual nos permitió comprender qué partes de ésta arquitectura son importantes para la recuperación y para el sistema. La parte visual nos permite tomar decisiones debido a que analizamos la manera en que se relacionan los componentes. Ahora al existir un método que nos brinde un orden y unas pautas facilita entender lo que es importante recuperar teniendo en cuenta los puntos críticos del sistema relacionados con los atributos de calidad que este contiene. Entendiendo esta parte y trabajando en conjunto el método y la recuperación visual es mucho más fácil entender la arquitectura y por ende permite que esta evolucione según la manera en que está construida.

CONCLUSIONES, LIMITACIONES, TRABAJOS FUTUROS Y LECCIONES APRENDIDAS

En ésta tesis se ha desarrollado y evaluado un método de recuperación de arquitectura enfocado hacia diferentes estudios empíricos donde se toman elementos de los métodos de evaluación arquitectónica existentes y los adapta para su aplicación en diferentes sistemas. Aunque su estructura y mayor inspiración es el método de evaluación de arquitecturas ATAM [63].

5.1. CONCLUSIONES

Para el desarrollo de la propuesta se utilizaron varias fuentes de información. Inicialmente se estudió la literatura con el fin de entender los problemas, métodos, técnicas, metodologías y herramientas existentes en cuanto a la recuperación de arquitectura y las formas en que se solucionan de acuerdo a los reportes y artículos reconocidos por la comunidad mundial en ingeniería de software.

Adicionalmente, se estudiaron las diferentes herramientas de recuperación de arquitectura visuales existentes, donde se identificaron sus características y limitaciones en cuanto a la instalación y adaptación, éste estudio permitió confirmar que las herramientas ayudan a recuperar un porcentaje muy mínimo. Con el fin de adquirir las destrezas técnicas, un conjunto de casos iníciales fueron desarrollados a través de dos estudios de caso exploratorios.

Basado en lo anterior el método EV-AR fue definido y refinado a través de su evaluación desde 4 perspectivas: el análisis de herramientas de recuperación visuales en un sistema de software en el cual solo se pudo obtener información sobre las vistas lógica y de implementación de un sistema, esto nos permitió identificar que estas herramientas de recuperación apoyarían en el método, luego se evaluó la aplicabilidad de un método de recuperación de arquitecturas el cual fue diseñado basándonos en un método de evaluación de arquitecturas, donde se pudo concluir que las decisiones que se toman al momento de definir una arquitectura de un sistema proveen información relevante para su entendimiento y recuperación, por lo que considerar las personas que han participado en el desarrollo de la arquitectura es clave para obtener el rationale de la misma. Posteriormente el análisis del método en un estudio preliminar donde se analizó su aplicabilidad en un contexto industrial a través de una pequeña empresa de software donde se realiza una recuperación visual de forma manual por la dificultad con la instalación de la herramienta de recuperación y por último se propone el modelo completo y se evalúa en un último estudio de caso en el que se hace una recuperación completa pero nos damos cuenta que el método no es aplicable sino se cuenta con una persona que tenga fuertes conocimientos en arquitecturas de software.

La evaluación del método EV-AR realizado en diferentes entornos logró analizar ampliamente las dificultades que se tienen al momento de hacer una recuperación

de la arquitectura software. El método EV-AR aporta varias prácticas basándose en un método de evaluación y en la visualización de la arquitectura, sin embargo, no todas son obligatorias. Por lo anterior, el modelo de proceso incluye actividades opcionales, con el fin de hacer flexible la aplicación del método a distintos productos y empresas de software de acuerdo a sus posibilidades, contexto y realidades.

La evaluación en diferentes casos ayudó a mejorar y a refinar el flujo de trabajo y los artefactos usados en el método de recuperación, además de identificar las habilidades que deben tener las personas que desean utilizar el método EV-AR, también nos ayudó a manifestar su aplicabilidad en diferentes contextos y previendo los beneficios y fortalezas que puede aportar con él, en lo referente a la aplicabilidad del método en una organización, es importante recalcar que el método no puede ser incluido por la falta de tiempo o porque ya tienen establecidos sus procesos de desarrollo y es complicado la adopción del método.
EV-AR incluye orientaciones para generar y discutir escenarios de evaluación iniciales y a partir de las cuáles estudiar las decisiones arquitectónicas consideradas incluyendo a desarrolladores y herramientas de visualización de arquitectónicas de software.

5.2. LIMITACIONES

El método EV-AR, a pesar de haber sido evaluado por 4 diferentes estudios de caso, requiere ser implementado y evaluado en entidades de desarrollo de la industria, que verifique sus prácticas, esta parte no fue desarrollada ni considerada por el alcance de este proyecto, el impacto del método, debe realizarse de manera metódica y acompañada, con personas expertas en arquitectura y con recursos que posibiliten esta experiencia, que permita el enriquecimiento y el fortalecimiento del método de recuperación.

La aplicabilidad del método es un poco compleja con respecto al tamaño de la empresa, su nivel de adaptación puede requerirse con un mayor acompañamiento de personas expertas en arquitectura, el método no se enfoca en este sentido, sino como guía en acompañamiento de pequeñas entidades de software para realizar la documentación y recuperación visual de la arquitectura.

La no existencia de tecnología de recuperación de arquitectura a través del código fuente hace que el método tenga complicaciones en su recuperación ya que se tendría que hacer de forma manual y esto aumentaría el esfuerzo de las personas interesadas en la recuperación.

5.3. TRABAJOS FUTUROS

En el trabajo futuro del método, se buscará estudiar e incluir prácticas en aspectos relacionados con el acompañamiento de personas expertos en arquitecturas, que brinden un soporte completo y efectivo, fortaleciendo la guía del método.

Con el fin de mejorar los aspectos de aplicabilidad de la guía del método es importante adoptar prácticas de planificación, seguimiento y control, para incrementar el éxito del método en la práctica.

Llevar a cabo una práctica desde el momento en que se empiece a desarrollar el sistema, donde a medida que se vaya creando el producto software, este se vaya documentando y esto ayudé a realizar una mejor recuperación.

Se espera poder hacer una integración más profunda entre los procesos de una empresa, métodos y herramientas de la arquitectura con el fin de alcanzar un soporte completo a la compleja actividad de determinarla, evaluarla y especificarla adecuadamente.

5.4. LECCIONES APRENDIDAS

El uso de los escenarios es un mecanismo que ayuda a identificar temas de interés de un sistema software, bien sea una funcionalidad, un atributo de calidad o las tácticas que se usaron para resolver una problemática que puede ser común en un dominio en específico. Enfocar la recuperación de la arquitectura para resolver las inquietudes que se tengan sobre dichos escenarios permite que los esfuerzos dedicados a este proceso de recuperación tengan como resultado la recuperación de información que sea relevante para las personas interesadas en conocer la arquitectura de este sistema.

Las herramientas de visualización de arquitecturas de software pueden ayudar en el proceso de recuperación de una arquitectura, pero para que estas herramientas no entorpezcan el proceso de recuperación se necesita que estas herramientas sean analizadas con respecto a las características del sistema y los resultados que arrojan, esto con el objetivo de tener información necesaria para escoger una herramienta adecuada para el proceso de recuperación.

Para recuperar las decisiones arquitecturales que se tuvieron en la construcción en un sistema software es necesario contar con aquellas personas que participaron en la construcción de dicho sistema, esto se concluye teniendo en cuenta la experiencia obtenida en el desarrollo del estudio de caso realizado en una pequeña organización de software, donde se contaba con estas personas.

Es necesario que dentro del proceso de recuperación participe una persona con fuertes conocimientos en la arquitectura que ayude a indagar sobre las posibles decisiones arquitecturales que se tuvieron en cuenta para el desarrollo del sistema.

REFERENCIAS BIBLIOGRÁFICAS

[1] Y. Ye, “Supporting Software Development as Knowledge-Intensive and Collaborative Activity,” *2006 Int. Work. Work. Interdiscip. Softw. Eng. Res. WISER 06*, pp. 15–21, 2006.

[2] G. Matturro and A. Silva, “A model for capturing and managing software engineering knowledge and experience,” *J. Univers. Comput. Sci.*, vol. 16, no. 3, pp. 479–505, 2010.

[3] B. Nuseibeh, “Weaving together requirements and architectures,” *Computer (Long. Beach. Calif).*, vol. 34, no. 3, pp. 115–117, 2001.

[4] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2e édition., vol. 2nd. 2012.

[5] P. Bengtsson and J. Bosch, “Scenario-based software architecture reengineering,” *Softw. Reuse, 1998. Proceedings. Fifth Int. Conf.*, pp. 308–317, 1998.

[6] I. Hogganvik, E. Molstad, and S. Fordypningsemne, “INCO- Open Source Architecture Recovery,” no. November, 2002.

[7] M. Lungu, M. Lanza, and O. Nierstrasz, “Evolutionary and collaborative software architecture recovery with Softwarenaut,” *Sci. Comput. Program.*, vol. 79, pp. 204–223, 2014.

[8] J. B. Tran, M. W. Godfrey, E. H. S. Lee, and R. C. Holt, “Architectural repair of open source software,” *Proc. - IEEE Work. Progr. Compr.*, vol. 2000–Janua, pp. 48–59, 2000.

[9] O. Maqbool and H. Babri, “Hierarchical clustering for software architecture recovery,” *IEEE Trans. Softw. Eng.*, vol. 33, no. 11, pp. 759–780, 2007.

[10] C.-H. Lung, “Software architecture recovery and restructuring through clustering techniques,” *Proc. third Int. Work. Softw. Archit. - ISAW ’98*, no. 613, pp. 101–104, 1998.

[11] J. E. Gaffney and T. A. Durek, “Software reuse - key to enhanced productivity: some quantitative models,” *Inf. Softw. Technol.*, vol. 31, no. 5, pp. 258–267, 1989.

[12] W. B. Frakes and S. Isoda, “Success factors of systematic reuse,” *Software, IEEE*, vol. 11, no. 5, pp. 14–19, 1994.

[13] H. Mili, A. Mili, S. Yacoub, and E. Addy, *Reuse Based Software Engineering: Techniques, Organizations, and Measurement*. 2002.

[14] M. Shaw, “Architectural Issues in Software Reuse: It’s Not Just the Functionality, It’s the Packaging,” *ACM SIGSOFT Symp. Softw. Reusability*, pp. 3–6, 1995.

[15] M. C. Camacho and J. A. H. Alegría, “Analizing the viability for adopting the software process line approach in small entities,” *2012 7th Colomb. Comput. Congr. CCC 2012 - Conf. Proc.*, 2012.

[16] L. Antovski and F. Imeri, “Review of Software Reuse Processes,” vol. 10, no. 6, pp. 83–88, 2013.

[17] J. A. Hurtado, “Toward a Scientific Method in Software Engineering (Position Paper),” 2011.
[18] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empir. Softw. Eng.*, vol. 14, no. 2, pp. 131–164, 2009.
[19] R. Kazman, “Tool support for architecture analysis and design,” *Int. Softw. Archit. Work. Proceedings, ISAW*, pp. 94–97, 1996.
[20] L. M. N. Paul C. Clements, “Software Architecture: An Executive Overview,” Pittsburgh, Pennsylvania.
[21] D. Garlan, P. Clements, R. Little, R. Nord, and J. Stafford, “Documenting software architectures: views and beyond,” *25th Int. Conf. Softw. Eng. 2003. Proceedings.*, no. November 2001, p. 342, 2010.
[22] J. Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach*. Addison-Wesley Professional; 1 edition (May 29, 2000).
[23] I. Sommerville, “Ingeniería del software,” in *DanielR.Obolog.Es*, 2005th ed., 2005, pp. 220–283.
[24] R. S. Pressman and J. M. Troya, “Ingeniería del software,” Séptima ed., no. 001.64 P74s., P. R. Vázquez, Ed. University of Connecticut, 1988, pp. 235–264.
[25] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. Pearson (April 12, 1996), 1996.
[26] J. B. Karl Wiegers, “Software Requirements,” in *Software Requirements*, (3rd Editi., 978-0735679665, 2013, pp. 230–300.
[27] F. Bachmann, L. Bass, and R. Nord, “Modifiability Tactics,” *Softw. Eng. Inst.*, no. September, 2007.
[28] D. Garlan, P. Clements, R. Little, R. Nord, and J. Stafford, “Documenting software architectures: views and beyond,” *25th Int. Conf. Softw. Eng. 2003. Proceedings.*, p. 342, 2010.
[29] L. Dobrica and E. Niemela, “A survey on software architecture analysis methods,” *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 638–653, 2002.
[30] M. K. Clements, Rick Kazman, *Evaluating Software Architectures” Methods and Case Studies*, 1 edition. Addison-Wesley Professional, 2001.
[31] R. Kazman, L. Bass, M. Webb, G. Abowd, and M. Webb, “SAAM: a method for analyzing the properties of software architectures,” *Proc. 16th Int. Conf. Softw. Eng.*, pp. 81–90, 1994.
[32] I. O. R. Kazman and M. Klein, “Quality-Attribute-Based Economic Valuation of Architectural Patterns - Report,” *Softw. Archit. Technol. Initiat.*, vol. 53, no. May, p. 160, 2007.
[33] P. C. Clements, “Active Reviews for Intermediate Designs,” no. August, pp. 1–25, 2000.
[34] E. J. Chikofsky and J. H. Cross II, “Reverse Engineering and Desingn Recovery: A taxonomy,” *Ieee*. pp. 13–17, 1990.
[35] M. Kong, J. A. Pow-sang, M. F. Tupia, and L. A. Flores, “VI Jornadas Iberoamericanas de Ingenierıa del Software e Ingenierıa del Conocimiento,”

Primera ed., F. de C. e I. de la P. U. C. del Peru, D. de I. de la P. U. C. del Peru, and M. F. T. A. y L. A. F. G. Maynard Kong Wong, JoseAntonio Pow-Sang Portillo, Eds. 2007, pp. 123–201.

[36] C. M. L. W. Aline Pires Vieira de Vasconcelos, “Architectural Elements Recovery and Quality Evaluation to Assist in Reference Architectures Specification.,” *Conference Paper · January 2007*, Boston, Massachusetts, USA.

[37] A. K. Jain, P. W. Duin, and J. Mao, “Statistical pattern recognition: a review,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 1, pp. 4–37, 2000.

[38] “Técnicas de agrupamiento (clustering) Introducción,” 2015. [Online]. Available: https://eva.fing.edu.uy/pluginfile.php/63359/mod_resource/content/6/clase_recpat_clustering2015.pdf.

[39] Y. Cai, H. Wang, S. Wong, and L. Wang, “Architecture Recovery Based on Design Rule Hierarchy,” no. 1, p. 9, 2011.

[40] R. Duddukuri and T. V. Prabhakar, “Helping architects in retrieving architecture documents: A semantic based approach,” *CEUR Workshop Proc.*, vol. 178, pp. 113–120, 2006.

[41] T. Panas, W. Löwe, and U. Aßmann, “Towards the Unified Recovery Architecture for Reverse Engineering,” *Int. Conf. Softw. Eng. Res. Pract.*, pp. 854–860, 2003.

[42] M. E. Monroy, J. L. Arciniegas, and J. C. Rodríguez, “Propuesta Metodológica para Caracterizar y Seleccionar Métodos de Ingeniería Inversa,” *Inf. Tecnol.*, vol. 24, no. 5, pp. 23–30, 2013.

[43] E. Constantinou, G. Kakarontzas, and I. Stamelos, “Towards open source software system architecture recovery using design metrics,” *Proc. - 2011 Panhellenic Conf. Informatics, PCI 2011*, pp. 166–170, 2011.

[44] J. Zhu, J. Huang, D. Zhou, Z. Yin, G. Zhang, and Q. He, “Software Architecture Recovery Through Similarity-Based Graph Clustering,” *Int. J. Softw. Eng. Knowl. Eng.*, vol. 23, no. 4, pp. 559–586, 2013.

[45] A. Vasconcelos and C. Werner, “Evaluating reuse and program understanding in ArchMine architecture recovery approach,” *Inf. Sci. (Ny).*, vol. 181, no. 13, pp. 2761–2786, 2011.

[46] M. Pinzger *et al.*, “Architecture Recovery for Product Families 2 Reference Architecture Construction by Exploiting Related Prior Systems,” pp. 332–351, 2004.

[47] R. Ferreira Barcelos and G. H. Travassos, “Evaluation approaches for software architectural documents: A systematic review,” *Actas IDEAS 2006 - 9th Work. Iberoam. Ing. Requisitos y Ambient. Softw.*, no. January, pp. 443–446, 2006.

[48] A. Pires, V. De Vasconcelos, C. Maria, and L. Werner, “An Approach to Software Architecture Recovery Aiming at Its Reuse in the Context of Domain Engineering,” p. 8.

[49] S. Tichelaar, “Modeling Object-Oriented Software for Reverse Engineering and Refactoring,” *Doktorarbeit, Univ. Berne, Switz.*, 2001.

[50] T. M. Khoshgoftaar and K. Bennett, “International Conference on Software Maintenance,” in *Conference on Software Maintenance*, 1989, no. 7, p. 1993.
[51] F. Deissenboeck and D. Ratiu, “A unified meta-model for concept-based reverse engineering,” *Proc. 3rd Int. Work. Metamodels, Schemas, Grammars Ontol. (ATEM’06*, 2006.
[52] M. Lanza and S. Ducasse, “Polymetric Views---A Lightweight Visual Approach to Reverse Engineering,” *Trans. Softw. Eng.*, vol. 29, no. 9, pp. 782–795, 2003.
[53] A. Bergel, C. Andrei, S. Ducasse, and T. Girba, “Moose,” 2015. [Online]. Available: http://www.moosetechnology.org/.
[54] M. Lungu and M. Lanza, “Package Patterns for Visual Architecture Recovery,” no. Csmr, 2006.
[55] N. Medvidovic and V. Jakobac, “Using software evolution to focus architectural recovery,” *Autom. Softw. Eng.*, vol. 13, no. 2, pp. 225–256, 2006.
[56] G. Y. Guo, J. M. Atlee, and R. Kazman, “A Software Architecture Reconstruction Method BT - Software Architecture,” *Softw. Archit.*, p. 93, 1999.
[57] A. Vasconcelos and C. Werner, “Architecture recovery and evaluation aiming at program understanding and reuse,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 4880 LNCS, pp. 72–89, 2007.
[58] C. Stoermer and L. O’Brien, “MAP - Mining Architectures for Product Line evaluations,” *Proc. - Work. IEEE/IFIP Conf. Softw. Archit. WICSA 2001*, pp. 35–44, 2001.
[59] R. R. Lutz and G. C. Gannod, “Analysis of a software product line architecture: An experience report,” *J. Syst. Softw.*, vol. 66, no. 3, pp. 253–267, 2003.
[60] R. Kazman, G. Abowd, L. Bass, and P. Clements, “Scenario-based analysis of software architecture,” *IEEE Softw.*, vol. 13, no. 6, pp. 47–55, 1996.
[61] J. Hurtado, E. Giraldo, and Y. Ordoñez, “ATAM-AR : ATAM-Based Recovery Architecture Method ATAM-AR : Un método de Recuperación de Arquitecturas basado en ATAM,” 2015.
[62] P. Kruchten, “The 4+ 1 view model of architecture,” *Software, IEEE*, vol. November 1, no. November, p. 9, 1995.
[63] R. Kazman, M. Klein, and P. Clements, “ATAM : Method for Architecture Evaluation,” *Cmusei*, vol. 4, no. August, p. 83, 2000.
[64] A. Ramirez *et al.*, “ArgoUML User Manual A tutorial and reference description,” *ArgoUML Community*, pp. 1–385, 2011.
[65] C. Juan, Z. Cesar, and M. Erwin, “Propuesta De Arquitectura Para Soportar La Sincronización Por Demanda De Archivos Asociados A Actividades De Usuarios En Plataformas De Aprendizaje En Línea,” Universidad del Cauca, 2014.
[66] R. K. Yin, “Case Study Reserach - Design and Methods,” *Clin. Res.*, vol. 2,

pp. 8–13, 2006.
[67] R. Colin, “Real World Research: A Resource for Social Scientists and Practitioner-Researchers,” *Blackwell Publishing*. pp. 1–608, 2002.
[68] W. Peña, “El Estudio De Caso Como Apropiado a La Investigación En Ciencias Sociales,” *Rev. Educ. Y Desarro. Soc.*, vol. 3, no. 2, pp. 180–195, 2009.