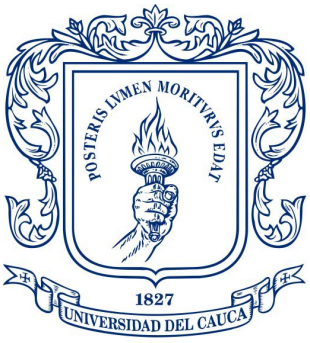
Documentación ágil de arquitecturas de software: un enfoque basado en anotaciones de código



MILTON JAVIER SÁNCHEZ GRUESO

Tesis de maestría en computación

Director:
Ph.D. Julio Ariel Hurtado Alegría

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Grupo de investigación en ingeniería de software IDIS
Popayán, enero 2021

MILTON JAVIER SÁNCHEZ GRUESO

MAESTRÍA EN COMPUTACIÓN

Trabajo de investigación presentado como requisito parcial para optar al título de Magíster en Computación.

Director:
Ph.D. Julio Ariel Hurtado Alegría
Profesor de la universidad del Cauca

Popayán

2021

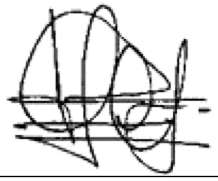
Nota de aceptación

_____

_____

_____

_____

Director 

Ph.D. Julio Ariel Hurtado Alegría

Jurado Libardo Pantoja Y

MSc. Libardo Pantoja Yepez

Jurado 

Ph.D. José García Alonso

Popayán, 25 de enero de 2021.

Dedicatoria

Gracias a mis padres por ser los principales promotores de mis sueños, gracias a ellos por cada día confiar y creer en mí y en mis expectativas, gracias a mi madre por estar dispuesta a acompañarme cada larga y agotadora noche de estudio; gracias a mi padre por siempre desear y anhelar siempre lo mejor para mi vida, gracias por cada consejo y por cada una de sus palabras que me guiaron durante mi vida.

Gracias a Dios por la vida de mis padres, también porque cada día bendice mi vida con la hermosa oportunidad de estar y disfrutar al lado de las personas que sé que más me aman, y a las que yo sé que más amo en mi vida, gracias a Dios por permitirme amar a mis padres, gracias a mis padres por permitirme conocer de Dios y de su infinito amor.

Gracias a la vida por este nuevo triunfo, gracias a todas las personas que me apoyaron y creyeron en la realización de esta tesis.

Agradecimientos

El presente trabajo investigativo lo dedicamos principalmente a Dios, por ser el inspirador y darnos fuerza para continuar en este proceso de obtener uno de los anhelos más deseados.

A la Universidad del Cauca por ser la sede de todo el conocimiento adquirido en estos años.

A mi director de tesis el PhD Julio Ariel Hurtado por su dedicación, apoyo, confianza, paciencia, por darme el impulso y el soporte académico.

Gracias a AIC (Asociación Indígena del Cauca) por permitir realizar la evaluación del enfoque planteado en esta investigación dentro del proceso de desarrollo de software, también a los ingenieros Whilmer Fernández, Jose Trujillo, Silvia Anacona y Richard Campo quienes participaron en la evaluación del mecanismo planteado.

Resumen

El rationale arquitectónico es el conjunto de razones detrás de las decisiones tomadas al diseñar la arquitectura de un sistema o artefacto. Normalmente, dicho rationale se queda en las mentes de los diseñadores y de los demás involucrados en el diseño. Por lo tanto, el razonamiento detrás de las decisiones que sustentan el modelo de arquitectura puede perderse si no se documenta adecuadamente. En la práctica, el rationale no se documenta o se documenta en medio de las descripciones arquitecturales, lo cual dificulta su comprensión y apoyo a las decisiones posteriores dentro del desarrollo y mantenimiento de software. Para abordar este problema, en este trabajo, se propone un enfoque de documentación que combina el modelado del rationale con un modelo de decisiones, con el fin de especificar un lenguaje que expresamos a través de un meta-modelo que hemos denominado DRML (Decisions and Rationale Modeling Language), que fundamenta las bases para generar anotaciones de rationale en el código con la herramienta RADAR (Rationale Architectural Decision Annotations). También, aplicando ingeniería dirigida por modelos el metamodelo DRML sienta los elementos para la construcción de la herramienta DRMLTool, que permite representar el fundamento arquitectónico de las decisiones de diseño que no se pueden colocar directamente en el código de manera gráfica, generando un modelo de Rationale como un artefacto mas de arquitectura. Este enfoque es aplicable en pequeñas entidades de desarrollo de software que utilizan métodos ágiles en sus procesos. DRML es simple, liviano y adaptable con el fin de hacerlo aplicable en proyectos reales. DRMLTool ha sido definido como un plugin de modelado en EMF con GMF y parcialmente aplicado en un estudio de caso empresarial, RADAR es implementado en el lenguaje Java.

Índice general

Índice de figuras

Índice de tablas

Listado de Acrónimos

ADD Método de Diseño Dirigido por Atributos. 39

ATAM Método de Análisis de Arquitectura. 39

DRML Decisions and Rationale Modeling Language. 83

EMF Eclipse Modeling Framework. 44

MDE Ingeniería Dirigida por Modelos. 42

MOF Meta Object Facility. 44

QAW Taller de Atributos de Calidad. 39

SAD Documento de Arquitectura Software. 122

SEI Instituto de Ingeniería de Software. 39

UML Lenguaje Unificado de Modelado. 43

VAB Vistas y Más Allá. 39

Glosario

Modelo Conceptual Un modelo conceptual busca representar conceptos abstractos que tienen como objetivo explicar una situación o un sistema de la vida real. 82, 85

t-Student En probabilidad y estadística, la distribución t (de Student) es una distribución de probabilidad que surge del problema de estimar la media de una población normalmente distribuida cuando el tamaño de la muestra es pequeño. 132, 138, 143

Capítulo 1

Introducción

En este capítulo se realiza una descripción introductoria sobre el problema a resolver, su justificación, el objetivo general y los específicos que fueron alcanzados en esta investigación, la metodología de investigación que se siguió para cumplir con los objetivos y finalmente se termina con la definición de la estructura de todo el documento.

1.1. Motivación

La urgencia en el mercado por desarrollar rápidamente nuevos sistemas y aplicaciones en la era de internet, ha llevado a las organizaciones a desarrollar software sin considerar en forma disciplinada aspectos fundamentales como las cualidades requeridas [79]. Algunas de esas cualidades, tales como usabilidad, seguridad y desempeño, están directamente relacionados con las necesidades del usuario final, sin embargo, aspectos como la escalabilidad y la modificabilidad están relacionados con la evolución del software y, por tanto, con la duración de su ciclo de vida [12]. Por tanto, la rapidez de los mercados de hoy hacen que se prioricen mayoritariamente las cualidades visibles al usuario final y se dejen a un lado las cualidades que faciliten la evolución del software [79]. En la industria del software ha aumentado significativamente el uso de metodologías ágiles durante la última década. Esto ha llevado a aumentar los esfuerzos para ajustar estas metodologías a los productos de la empresa y desarrollo de sistemas complejos, y concretamente para ajustar el minimalismo con la necesidad de tener artefactos de arquitectura bien definidos [41]. Es por esto que existe una necesidad de técnicas y métodos que soporten

documentación en ambientes ágiles ya que ésta a menudo es descuidada en proyectos ágiles de software, incluso cuando los desarrolladores de software perciben la necesidad de una buena documentación, teniendo en cuenta que ésta debe ser más fácil de trazar, manipular y renovar [91].

1.2. Planteamiento del problema

La arquitectura de software ha sido identificada como un artefacto de gran valor para la evolución de los sistemas de software. Sin embargo, es difícil que las organizaciones de software aprovechen su valor por diferentes razones entre otras porque, no es realizada completamente, es muy extensa o es incomprendida por los diferentes interesados. Por lo anterior, hay una gran distancia entre el valor potencial y el valor real para las organizaciones de software. Además, existen dificultades y asuntos fundamentales que influyen en el diseño de la arquitectura. Estos problemas incluyen dificultad para la toma de decisiones de diseño, ya sea por un razonamiento limitado, por sesgos cognitivos, el conocimiento asociado a las decisiones acerca del problema que se está tratando de resolver y el papel de las primeras decisiones que se han realizado [90]. Alrededor de la arquitectura de software existen muchos mitos y creencias, generalmente sustentados por algún caso de éxito o fracaso, pero que no pueden considerarse reglas para el desarrollo de la arquitectura [88]:

- Tomar decisiones arquitectónicas más rápido conduce a un producto de peor calidad
- Tomar y validar decisiones arquitectónicas más rápido aumenta la velocidad de desarrollo y el ROI de proyecto
- Las personas que codifican el sistema deben diseñar el sistema para lograr mayor velocidad en el desarrollo
- Menos documentación arquitectónica disminuye la calidad del producto

Documentar arquitectura en el enfoque ágil es también importante, debido al poco foco en la documentación, es importante considerar además repercutiendo de la generación de valor al negocio inmediato, documentar la arquitectura para el del largo plazo de la organización desarrolladora, aunque esta sea más liviana[88], de otra forma se verá amenazada su evolución. Los problemas que encuentran los arquitectos y otras partes interesadas cuando se ocupan de la documentación de la arquitectura en el desarrollo

ágil, muestran que el documento que contiene la descripción de la arquitectura suele ser muy largo, complejo y en muchos casos, no es auto-descriptivo, ya que en ocasiones estos pueden crecer de decenas a cientos de páginas; consiste de múltiples documentos que dentro y entre ellos abarcan muchos conceptos y relaciones, vistas múltiples y diferentes niveles de abstracción lo que lleva a que sea más difícil escribir, manejar y actualizar [41].

Proporcionar una trazabilidad entre los aspectos de calidad(requerimientos), las decisiones arquitectónicas(diseño), las justificaciones y las áreas pertinentes del código, es de suma importancia para varios aspectos del proceso de ingeniería de software. Es el caso de: el análisis del impacto de cambios, validación de requisitos, preservación arquitectónica, construcción de casos de seguridad y a largo plazo, el mantenimiento del sistema. Por ejemplo, la práctica ha demostrado que la erosión de la arquitectura ocurre a menudo cuando los desarrolladores realizan cambios en el código sin comprender completamente las decisiones arquitectónicas subyacentes y las preocupaciones relacionadas con la calidad [28][33]. Hoy este hecho desafortunadamente es una realidad ya que la arquitectura de software es definida como la composición de un conjunto de decisiones de diseño arquitectónico, donde lo que se busca es evitar la evaporación del conocimiento de éstas decisiones, ya que se han ido convirtiendo en una parte explícita de la arquitectura. Esto a su vez muestra el alto costo que tiene el cambio en las arquitecturas de software, muestran su complejidad y como se erosionan durante su evolución. Estos problemas se deben a la pérdida del conocimiento. Actualmente las decisiones de diseño sobre las que se basa la arquitectura, están implícitamente abordadas y se carece de una representación explícita. En consecuencia, el conocimiento sobre estas decisiones de diseño arquitectónico se hace invisible para el diseño, desarrollo, evolución, reutilización e interpretación. En el diseño, la principal preocupación es qué decisiones tomar. En desarrollo es importante saber qué y por qué se ha tomado ciertas decisiones de diseño. La evolución de la arquitectura tiene que ver mucho con las nuevas decisiones de diseño considerando las anteriores, para satisfacer los requisitos cambiantes[49]. Bratthall et. al. [18] mencionan que un medio potencial es facilitar los cambios arquitectónicos, proporcionando una lógica de diseño, es decir, una documentación de los porqués la arquitectura se construye tal como está, los cambios serán más rápidos y más correctos si dicha información está disponible durante el análisis de impacto del cambio y los cambios mismos

1.3. Hipótesis de la investigación

- **Hipótesis Alternativa H_1**: Un lenguaje de especificación de decisiones de diseño arquitectónico que reside junto a la arquitectura y código fuente, ayuda a guiar la futura toma de decisiones, en el diseño, lo cual resulta en una mejor comprensión del "rationaleo y una documentación que es más fácil de comunicar en un contexto ágil.
- **Hipótesis Nula H_0**: La disponibilidad de un lenguaje de especificación de decisiones de diseño arquitectónico para mejorar la comprensión del "rationale" no tiene un impacto en la comprensión de dicho conocimiento arquitectónico en el contexto ágil.

1.4. Pregunta de Investigación

Considerando este contexto problemático, en este proyecto de investigación se plantea la pregunta ¿ Cómo consolidar un mecanismo que facilite la documentación de las principales decisiones de diseño arquitectónico en el contexto ágil? En este proyecto se abordan distintas opciones para encontrar y probar un mecanismo idóneo para representar las decisiones arquitecturales y sus razones.

1.5. Objetivos

1.5.1. Objetivo general

Definir un lenguaje basado en anotaciones que permita documentar las principales decisiones de diseño a nivel arquitectónico en un contexto de desarrollo ágil de software.

1.5.2. Objetivos específicos

- Especificar los criterios fundamentales para la documentación de arquitectura en el contexto de proyectos ágiles.

- Analizar los lenguajes que existen para la especificación de decisiones de diseño arquitectónico.
- Especificar un lenguaje de anotaciones de decisiones de diseño a nivel arquitectónico.
- Validar el lenguaje de anotaciones propuesto a través de su aplicación en un estudio de caso real.

1.6. Metodología

Para alcanzar los objetivos propuestos al inicio de este proyecto "Documentación ágil de arquitecturas de software: un enfoque basado en anotaciones de código" se utilizarán las pautas de investigación del método de Mario Bunge [21]. Para la especificación del lenguaje que permite facilitar la documentación de arquitecturas en el contexto ágil, se realizó un estudio secundario a través de la literatura [7]. La evaluación del proceso siguió método de estudios de caso propuesto por Runeson et Al [73]. A continuación, son descritas las actividades de desarrollo del lenguaje:

1.6.1. Etapa exploratoria

- **Planteamiento del problema y Construcción del modelo teórico**
  Reconocimiento de los hechos, descubrimiento y formulación del problema: Estudio del referente teórico de mecanismos para documentar las principales decisiones de diseño arquitectónico en arquitecturas de software en el contexto ágil, lineamientos, técnicas. Formulación de una pregunta de investigación e hipótesis [21].

Producto: Planteamiento del problema, diseño reporte técnico del estudio de la industria.

1.6.2. Etapa de formulación

- **Especificación del Mecanismo:**
  Definición del mecanismo que permite documentar las principales decisiones de

diseño arquitectónico a través de anotaciones a partir de los hallazgos de la revisión de la literatura. Evaluación del mecanismo por medio de un estudio comparativo donde se evaluó si el mecanismo es aplicable a productos y componentes [7].

- **Construcción del Lenguaje y la herramienta de soporte**

  Basado en los conceptos identificados en la especificación del mecanismos, se siguió un enfoque MDE[20] para especificar la sintaxis abstracta (Metamodelo) y la sintaxis concreta (Metáfora Gráfica), así como el uso de la infraestructura EMF para implementar el lenguaje como una herramienta de modelado.

Producto: Mecanismo (Lenguaje) , reporte técnico del estudio de caso.

1.6.3. Etapa de evaluación

- **Evaluación del mecanismo**

  Análisis de calidad del mecanismo, evaluación del mismo utilizando el lenguaje y su herramienta en un estudio de caso embebido, positivista, y en una unidad de desarrollo de software[73].

- **Confrontación de los resultados**

  Los resultados del estudio son analizados para evaluar su utilidad percibida para la documentación de la arquitectura y si facilita la comprensión de las decisiones de diseño.

  Producto: Mecanismo (Lenguaje) para documentar las principales decisiones de diseño arquitectónico en arquitecturas de software en un contexto ágil.

- **Ajustes finales**
  Análisis de resultados y ajustes al mecanismo.

  Producto: Monografía, Artículos, Reporte técnico del estudio de caso, Mecanismo.

  La **Figura 1.1** permite visualizar las etapas seguidas en el proceso, con los aspectos claves de cada una de ellas.

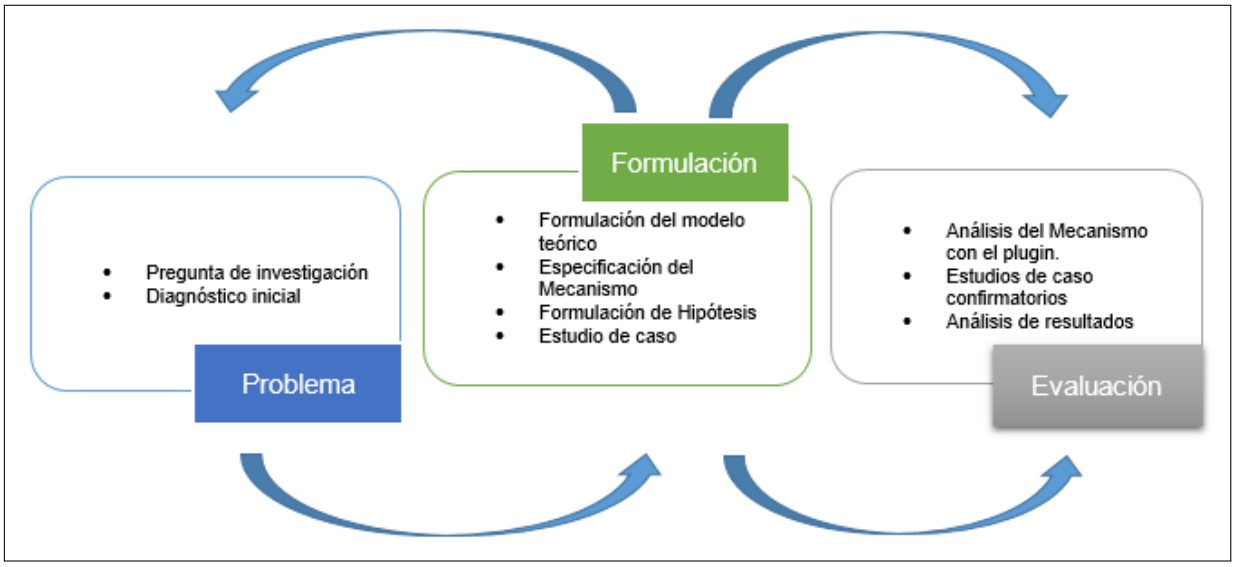


Figura 1.1: Metodología del proyecto

1.7. Estructura y Contenidos de la Tesis

La presente tesis está estructurada de la siguiente forma:

- **Capítulo 1:**

  Se describe el planteamiento del problema, pregunta de investigación, los objetivos de la tesis y metodología. Por último, se presenta el contenido y estructura de este trabajo de tesis.

- **Capítulo 2:**

  Se revisa el significado del término Arquitectura Software. Se describe brevemente por que es importante, los atributos de calidad, patrones y tácticas de arquitectura. Para terminar se presentan los lenguajes de descripción de arquitecturas.

- **Capítulo 3:**

  Se presenta la revisión de la literatura, el estado del arte y los trabajos relacionados.

- **Capítulo 4:**

  Se presenta la propuesta DRML (Modelo racional de decisiones de diseño arquitectónico), lenguaje que permite documentar a nivel conceptual [1] y de anotaciones

[1] Artefacto arquitectónico.

de código el rationale [2] de las principales decisiones de diseño arquitectónico.

- **Capítulo 5:**

  A partir de la especificación del lenguaje conceptual y de anotaciones de código para documentar el rationale de las principales decisiones de diseño arquitectónico ofrecido por el capítulo 3 se realiza la evaluación del mismo mediante un estudio de caso.

- **Capítulo 6:**

  Presenta las conclusiones, limitaciones y posibles trabajos futuros.

- **Anexo I:**

  Estudio de caso

- **Anexo II:**

  Articulo

- Al final del documento se encuentran los apéndices y las referencias utilizadas.

[2] La razón de fondo, se refiere a la "justificación".

Capítulo 2

Marco teórico

En este capítulo se abordan conceptos relevantes para entender el contexto y el dominio del problema que se aborda en este estudio, se especifican temas relacionados con los conceptos fundamentales de la arquitecturas de software, su documentación, la importancia de la arquitectura, tácticas de arquitectura, estrategias y patrones de diseño, atributos de calidad, lenguajes de descripción de arquitectura, métodos de análisis, diseño y evaluación de arquitectura, decisiones de diseño arquitectónico y el *rationale* arquitectónico. Además se abordan los conceptos y tecnologías usadas para validar la hipótesis propuesta, esto incluye la ingeniería dirigida por modelos (MDE, Modelos y Meta-modelos) así como el lenguaje XMI (XML Metadata Interchange).

2.1. Arquitectura Software

La arquitectura de software se define como aquella estructura o estructuras de un sistema, que se organizan a partir de piezas de software y sus relaciones. El énfasis está en que las propiedades visibles al exterior de las piezas de software y cómo se relacionan [29]. La arquitectura de software también se define como la descripción de los subsistemas y componentes de un sistema software y las relaciones entre estos. Los subsistemas son especificados desde diversos puntos de vista para mostrar las propiedades funcionales y no funcionales. Es un artefacto temprano, resultado de la actividad de diseño de software [14]. La arquitectura de software tiene otras definiciones tales como:

- Es un nivel de diseño diferente de los algoritmos y las estructuras de datos. Los elementos estructurales incluyen:
  - La organización y el control globales,
  - Los protocolos de comunicación,
  - La distribución física,
  - La composición de elementos de diseño,
  - La escalabilidad y el rendimiento, y la elección entre distintas alternativas de diseño.
- Un diseño de alto nivel.
- La estructura del sistema.
- Las componentes de un programa o sistema, sus relaciones, y principios que gobiernan su diseño y su evolución en el tiempo.
- Componentes, conectores, configuración y restricciones.

2.1.1. La importancia de la arquitectura

La arquitectura es un mecanismo de establecimiento, documentación y comunicación de las principales decisiones de diseño de un sistema, por ello su importancia es múltiple y diferente a cada participante del proyecto de software. Entre los principales valores de la arquitectura se tienen:

- Es una abstracción primaria del sistema: las personas necesitan pensar, diseñar, codificar, y comunicarse en términos de grandes bloques conceptuales.
- Reutilización de alto nivel: el diseño de la arquitectura permite escapar de los desarrollos excesivamente personalizados y estandarizar el diseño.
- Reutilización de componentes: las piezas de software se construyen para diseñar sistemas de larga vida, llevando a una fácil extensión y reutilización.
- Productividad en el desarrollo: actualmente solamente se reutiliza el código y las estructuras de datos, pensar en arquitecturas facilitaría la reutilización de grandes armazones o soportar la construcción de líneas de productos.

- Abordar otros problemas del ciclo de vida del software:
- Modificabilidad, portabilidad, escalabilidad, seguridad: a medida que el tamaño del sistema crece, las soluciones a estos problemas radican más en la arquitectura.
- Tener un lenguaje común para diseñadores, desarrolladores y usuarios.
- Permitir reutilizar artefactos arquitectónicos; para ayudar a explorar alternativas arquitectónicas; y para apoyar métricas arquitectónicas [51].
- Ayuda a comunicar el cliente con el desarrollador, para ayudar a aclarar los requisitos y su impacto en el diseño del sistema [46].

2.1.2. Patrones y tácticas de arquitectura

El arquitecto (actor principal en la arquitectura) tiene que tomar decisiones para poder satisfacer los atributos de calidad (o una concesión entre éstos) cuando plantea una arquitectura o un diseño arquitectónico. El conjunto de decisiones que debe tomar el arquitecto para poder lograr cada uno de los atributos de calidad importantes en la arquitectura, lo llamamos una estrategia de arquitectura, la cual esta conformada por decisiones concretas que hacen parte de nuestro enfoque arquitectónico. Existen dos tipos de decisiones importantes de diseño de aplicaciones conocidos como patrones de arquitectura y tácticas.

Los diseñadores de arquitectura de software inevitablemente trabajan tanto con patrones de arquitectura como con tácticas. Los patrones de arquitectura describen la estructura y el comportamiento de alto nivel de los sistemas de software como la solución a múltiples requisitos del sistema [42][77], mientras las tácticas son decisiones de diseño que abordan individualmente, los problemas de un atributo de calidad.

- *Los patrones de arquitectura:* son estructuras arquitectónicas comunes que están bien entendidos y documentados. Cada patrón describe la estructura y el comportamiento de alto nivel de un sistema software general y tiene como objetivo satisfacer varias funciones y requisitos no funcionales. Los patrones de arquitectura contienen los principales componentes y conectores del sistema para ser construido. La mayoría de las arquitecturas de software modernas usan un o más patrones de arquitectura. Los patrones de arquitectura se eligen en respuesta a decisiones de diseño tempranas, incluidas decisiones sobre cómo satisfacer requisitos funcionales,

requisitos no funcionales (calidad atributos) y restricciones físicas (como la distancia física entre usuario y proveedor de servicios). Por lo tanto, los patrones de arquitectura proporcionan las principales estructuras en las que se toman múltiples decisiones de diseño. Los patrones de arquitectura generalmente dictan una descomposición particular del sistema modular de alto nivel. Una ventaja importante de los patrones es que las consecuencias del uso del patrón de arquitectura es parte del patrón. Por tanto, el resultado de aplicar un patrón generalmente se documenta como consecuencias y generalmente se indican los aspectos positivos y negativos. Muchos de los beneficios y responsabilidades se refieren atributos de calidad; por ejemplo, un beneficio del patrón de capas indica que Las interfaces estandarizadas entre capas generalmente limitan el efecto de los cambios de código en la capa que se cambia, apoyando la modificabilidad del sistema.

- *Las tácticas:* son decisiones de diseño que tiene como objetivo mejorar una preocupación específica del diseño de un atributo de calidad. Por ejemplo, en un diseño la preocupación sobre el atributo de calidad "modificabilidad.[es] cómo prevenir el efecto onda en una modificación, una decisión de diseño asociada (táctica) es limitar los caminos de comunicación. Al igual que con los patrones, las estructuras y el comportamiento de las tácticas también pueden dar forma a la arquitectura, pero generalmente en una escala más localizada. Así que las tácticas se refieren al abordaje de una cualidad única de un atributo, mientras que los patrones abordan un número variado de cualidades. Las tácticas generalmente se vuelven parte de un patrón junto con otras las otras estructuras que el patrón especifique. Las tácticas son medidas tomadas para mejorar los atributos de calidad e impactan los patrones de arquitectura de varias maneras. En algunos casos, una táctica puede implementarse fácilmente usando las mismas estructuras como un patrón de arquitectura particular. Por otro lado, una táctica puede requerir cambios significativos en la estructura y comportamiento cuando un patrón ha sido seleccionado (decisión previa), o puede requerir completamente de nuevas estructuras y comportamiento. En este caso, la implementación de la táctica, y el mantenimiento futuro del sistema, es muy difícil y propenso a errores.

Las tácticas que se implementan en las arquitecturas existentes pueden tener un impacto significativo en los patrones de arquitectura en el sistema. De manera similar, las tácticas que se seleccionan durante el diseño inicial de la arquitectura tienen un impacto significativo en la arquitectura del sistema que se diseñará: qué patrones usar y cómo se deben cambiar para acomodar las tácticas [42].

2.1.3. Atributos y escenarios de calidad

Los atributos de calidad son características que un sistema debe cumplir a nivel no funcional, a diferencia de las funciones que debe tener. Los atributos de calidad tienen asociadas métricas que definen los niveles de calidad de un producto software como la seguridad, la fiabilidad, el desempeño, entre otros [82]. Los atributos de calidad son características que tiene el sistema como usabilidad, mantenibilidad, desempeño y confiabilidad [74].

Los atributos de calidad definen la calidad del servicio que un producto de software ofrece [1]. El manejo inadecuado de los atributos de calidad se vuelve un elemento importante de riesgo para el proyecto, surgen de las reglas de negocio de alta complejidad y de las preocupaciones de calidad, por ejemplo: modificabilidad, seguridad, rendimiento [35]. El arquitecto debe enfocarse en los problemas sistémicos que existen en los productos software, no tiene que ver con la parte funcional, dado que, en la práctica los sistemas normalmente no fallan por la parte funcional porque mal o bien nosotros estamos haciendo bien el trabajo de identificar los requerimientos funcionales. Los usuarios se quejan principalmente de los atributos de calidad, por ejemplo, el software no esta funcionando con la velocidad adecuada, no es atractivo, es difícil usar, se cae constantemente. Esto lleva a la pregunta: ¿Cual es el origen de estos problemas?. Existe una brecha muy grande de comunicación entre los diferentes participantes, arquitectos y los ingenieros de desarrollo, además, cuando se trabaja con desarrollo de software hay un gran foco en los requerimientos funcionales, olvidando las necesidades de calidad. El análisis y diseño basado únicamente en la funcionalidad no permite expresar la mayoría de necesidades que se tienen desde la perspectiva de calidad del software, de manera que es necesario buscar técnicas que logren plasmar los atributos de calidad de los sistemas [82].

Los atributos de calidad se pueden identificar en diferentes contextos que el arquitecto debe tener en cuenta, por ejemplo: los stakeholders, procesos de negocio, equipos de desarrollo, estructuras organizacionales, tendencias tecnológicas a nivel mundial, la experiencia como arquitecto, leyes, políticas gubernamentales, políticas organizacionales, la visión de negocio y planeación estratégica. La Figura 2.1 permite visualizar tipos de requerimientos no funcionales. El arquitecto debe trabajar con un sinnúmero de variantes que no necesariamente tienen que ver con la parte funcional.

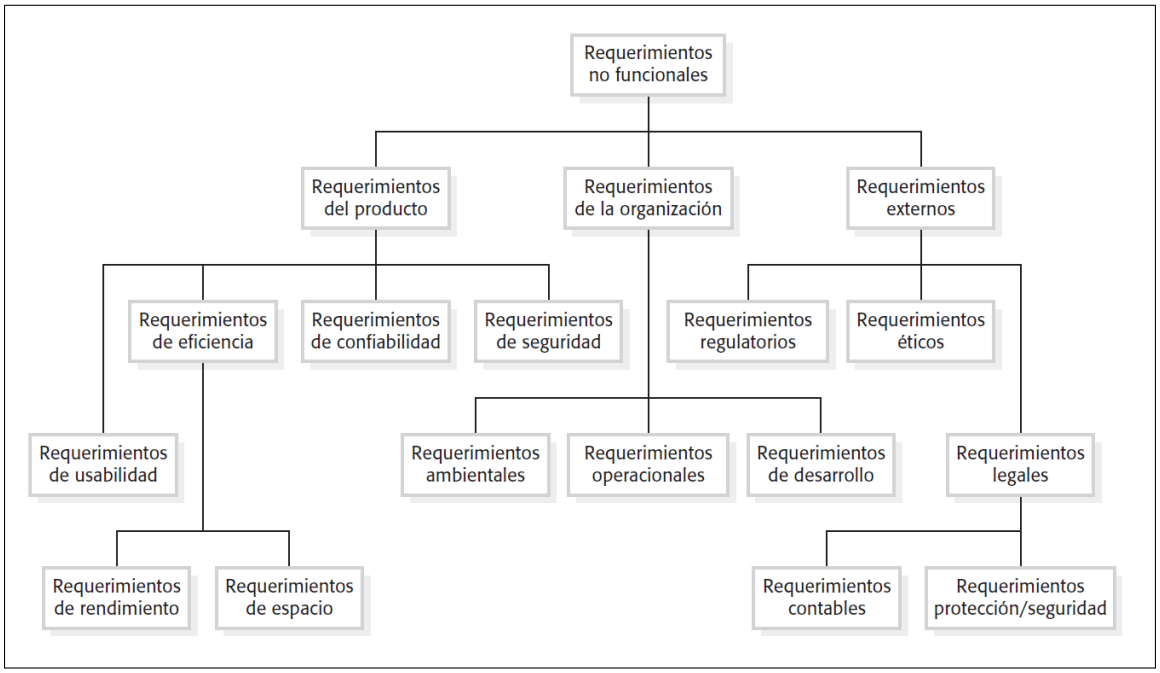


Figura 2.1: Tipos de requerimientos no funcionales. Tomado de [82]

El arquitecto debe tener en cuenta la ortogonalidad de los requerimientos, con el fin de identificar conflictos entre ellos, lo que se conoce como tensión mutua. Un ejemplo mas claro es cuando se trabaja con requerimientos de disponibilidad que pueden afectar la seguridad. Entre mayor disponibilidad se quiera lograr en un producto software, se debe incrementar la redundancia, agregando más puntos de vulnerabilidad a la solución. Por tanto, es muy importante para los arquitectos ubicar todas las tensiones mutuas entre los requerimientos tanto funcionales como no funcionales [82].

Un escenario de atributo de calidad es un mecanismo concreto para desambiguar un requisito específico asociado a un atributo de calidad. Barbacci et al.[11] estructuran un escenario de atributo de calidad en seis partes.

- *Fuente de estímulo:* corresponde a una entidad (un humano, un sistema informático o cualquier otro actuador) que genera el estímulo.
- *Estímulo:* es un evento o condición que debe tenerse en cuenta al llegar al sistema o artefacto analizado
- *Ambiente:* el estímulo ocurre dentro de ciertas condiciones del sistema, el cual

puede estar en una condición de sobrecarga o puede estar funcionando normalmente cuando ocurre el estímulo, o bajo alguna otra condición.

- *Artefacto:* el componente estimulado, este puede ser todo el sistema o algunas partes del mismo.
- *Medida de respuesta:* cuando se produce la respuesta, algún aspecto de la misma debe medirse para establecer los límites bajo los cuales el requisito de calidad se cumple.

2.1.4. Atributos de calidad y tácticas

Normalmente, los sistemas tienen múltiples atributos de calidad importantes, y las decisiones tomadas para satisfacer un atributo de calidad particular pueden afectar a otro atributo de calidad, asimismo, las tácticas son decisiones de alto nivel de abstracción tomadas para mejorar los atributos de calidad. Las tácticas impactan los patrones de arquitectura de varias maneras. En algunos casos, una táctica puede implementarse fácilmente utilizando las mismas estructuras (y comportamiento compatible) como un patrón de arquitectura particular. Por otro lado, una táctica puede requerir cambios significativos en la estructura y el comportamiento del patrón, o puede requerir estructuras y comportamiento completamente nuevos. En este caso, la implementación de la táctica y el mantenimiento futuro del sistema son considerablemente más difíciles y propensos a errores. Las tácticas pueden ser de tiempo de diseño, o enfoques generales de diseño e implementación, como por ejemplo el *ocultamiento de la información* para mejorar la modificabilidad, o pueden ser tácticas de tiempo de ejecución, que son características dirigidas a un aspecto particular de un atributo de calidad, como por ejemplo *autenticar usuarios* para mejorar la seguridad [42]. También manejo de interfaces de usuario para mejorar la usabilidad, uno de los atributos más relevantes al evaluar un producto de software, ya que es importante analizar cómo el diseño de la interacción facilita o dificulta al usuario alcanzar un objetivo concreto [75].

2.1.5. Lenguajes de descripción de arquitecturas

Con frecuencia para el diseño de las arquitecturas se utilizan lenguajes de definición de arquitecturas o ADLs (Architecture Description Languages) los cuales proporciona características para modelar la arquitectura conceptual de un sistema de software, que se

distingue de la implementación del sistema. Los ADLs proporcionan una sintaxis concreta y un marco conceptual para caracterizar las arquitecturas [63]. Hernández et. al. [66] mencionan algunos ADLs genéricos, como el UML (Unified Modeling Language) yotros específicos al dominio de las aplicaciones, los más conocidos lenguajes son: ACME [39], AESOP, C2, ABACUS (UTS), ADML, ByADL, Darwin, LePUS3, Class-Z, Rapide, Wright y Unicon. Dado que una arquitectura desempeña varios roles en el desarrollo de proyectos, todos ellos importantes, una representación formal de la arquitectura con un ADL ayuda a disminuir la ambigüedad en su comunicación. Es más probable que se mantenga y siga una representación de arquitectura formal que una informal, se puede consultar y tratar más fácilmente, así como autorizarse y además, se puede transferir más fácilmente a otros proyectos como un activo de desarrollo central[30]. Existe una gran variedad de ADLs desarrolladas por grupos académicos o industriales. Muchos lenguajes no estaban destinados a ser una ADL, pero resultan adecuados para representar y analizar una arquitectura. En principio, los ADLs difieren de los lenguajes de requisitos, porque los ADLs están enraizadas en el espacio de la solución, mientras que los requisitos describen el espacio del problema. Se diferencian de los lenguajes de programación, porque los ADLs no vinculan abstracciones arquitectónicas a soluciones puntuales específicas(dependientes de la tecnología). Los lenguajes de modelado representan comportamientos, donde los ADLs se centran en la representación de componentes. Sin embargo, existen lenguajes de modelado específicos de dominio (DSML) que se centran también en la representación de componentes. Los requisitos mínimos de un ADL son:

- Adecuado para comunicar una arquitectura a todas las partes interesadas.
- Apoye las tareas de creación, refinamiento y validación de arquitectura.
- Proporcionar una base para una implementación, por lo que debe poder agregar información a la especificación ADL para permitir que la especificación final del sistema se derive de la ADL.
- Capaz de representar la mayoría de los estilos arquitectónicos comunes.
- Apoyar las capacidades analíticas o proporcione implementaciones de prototipos de generación rápida

Los ADLs tienen en común una sintaxis gráfica con frecuencia una forma textual y una sintaxis y semántica formalmente definidas, permiten modelar sistemas distribuidos, hay poco soporte para capturar información de diseño, excepto a través de mecanismos de anotación de propósito general, y cuentan con capacidad para representar niveles jerárquicos de detalle, incluida la creación de subestructuras mediante la instanciación de plantillas.

2.1.6. Documentación de arquitectura software 4+1 vistas de Kruchten

Para la documentación de arquitectura de software se utilizan los modelos de documentación por vistas. Es lo mas recomendado en esta disciplina por marcos de trabajo o mecanismos para documentar arquitectura. Existe un modelo desde hace muchos años que sirve como esquema general de documentación pensando en las diferentes perspectivas de los participantes. Este esquema se llama el modelo de 4+1 vistas, fue creado en 1995 por Philippe Kruchten [57] el cual facilita a los diseñadores, desarrolladores y arquitectos tener un esquema claro y concreto de como se documenta un diseño de software basado en diferentes perspectivas. En su modelo se incorporan cuatro elementos claves, cuatro perspectivas claves y una central que guía todas las decisiones de software. La arquitectura de software se centra en 2 elementos. Primero esta el elemento de estructura o lógico el cual da la forma, distribución y descomposición. Segundo el elemento dinámico o de comportamiento que permite mostrar como interactúa el software en tiempo de ejecución. Los 2 elementos o grandes perspectivas se incorporan dentro del modelo de 4+1 vistas con la parte estructural y comportamiento del software [57]. También dentro del diseño se debe tener en cuenta la parte física y mostrar la relación con la parte lógica; al hacer la intersección de los cuatro elementos: estructural, dinámica, lógica y física se obtienen las partes principales del modelo de 4+1 vistas y el +1 que es el elemento central del diseño que corresponde a los escenarios donde lo que se busca es darle relevancia relevancia a la funcionalidad, esta vista es la que permite entender como la toma decisiones de las otras cuatro se mantienen consistentes. En el caso del diseño detallado seria una vista de casos de uso como lo acogió RUP - Rational Unified Process y para un esquema ágil se podría tener una vista de features o historias de usuario [57].

La vista central son los elementos arquitectónicamente significativos, por ejemplo, casos de uso, escenarios de atributos de calidad entre otros. La vista tendrá las necesidades que van a permitir darle forma y tomar decisiones alrededor de las otras vistas; es importante tener en cuenta que no se trabaja con todos los requerimientos, solamente los mas relevantes desde el punto de vista arquitectónicos[30].

2.2. Métodos de análisis, diseño y evaluación arquitectónica

2.2.1. Métodos de Arquitectura Software del SEI

El SEI [1] propone la incorporación de los métodos: QAW, ADD, VaB y ATAM en el ciclo de vida de la arquitectura. Usualmente los arquitectos de software realizan el descubrimiento de los elementos que componen a la arquitectura basándose en su experiencia(entre muchos otros factores), pero en las últimas décadas han surgido métodos de desarrollo arquitectónico que permiten realizar este descubrimiento de forma más sistemática y controlada; tal vez el esfuerzo más importante en la creación de estos sea el realizado por el Instituto de Ingeniería de Software (SEI), el cual propone los siguientes:

El método QAW [11] está enfocado en el análisis y la documentación de los requerimientos que determinan la arquitectura. Estos requerimientos incluyen los atributos de calidad que son descritos a través de escenarios que facilitan información medible para analizar el comportamiento del sistema o de un artefacto esperado frente a un estímulo relevante para el atributo de calidad analizado.

El método ADD [93], una vez que se han establecido y priorizado los escenarios, facilita el diseño de la arquitectura siguiendo un enfoque recursivo de descomposición del sistema en componentes cada vez más pequeños. Durante el ADD se van tomando escenarios y se van tomando decisiones de diseño, usando soluciones conceptuales llamadas *tácticas* y *patrones*, que permitan satisfacer los requerimientos descritos por los escenarios a través de unas estrategias arquitectónicas.

VAB [10], brinda las estrategias visuales y documentales clave para comunicar la arquitectura de software resultado de la aplicación de ADD y QAW. Por tanto, se obtienen distintas estructuras del sistema, formadas por los componentes y sus relaciones. Estas estructuras se documentan a través de *vistas* y *tipos de vistas* que muestran una perspectiva particular del sistema.

ATAM [52], es un método que permite revelar qué tan bien una solución arquitectónica satisface los atributos de calidad establecidos y revela, además, qué riesgos, puntos sensibles y compromisos que están involucrados en la arquitectura propuesta respecto de las expectativas de calidad bajo evaluación.

[1] https://www.sei.cmu.edu/

La Figura 2.2 permite visualizar como interactúan cada uno de los métodos entre si:

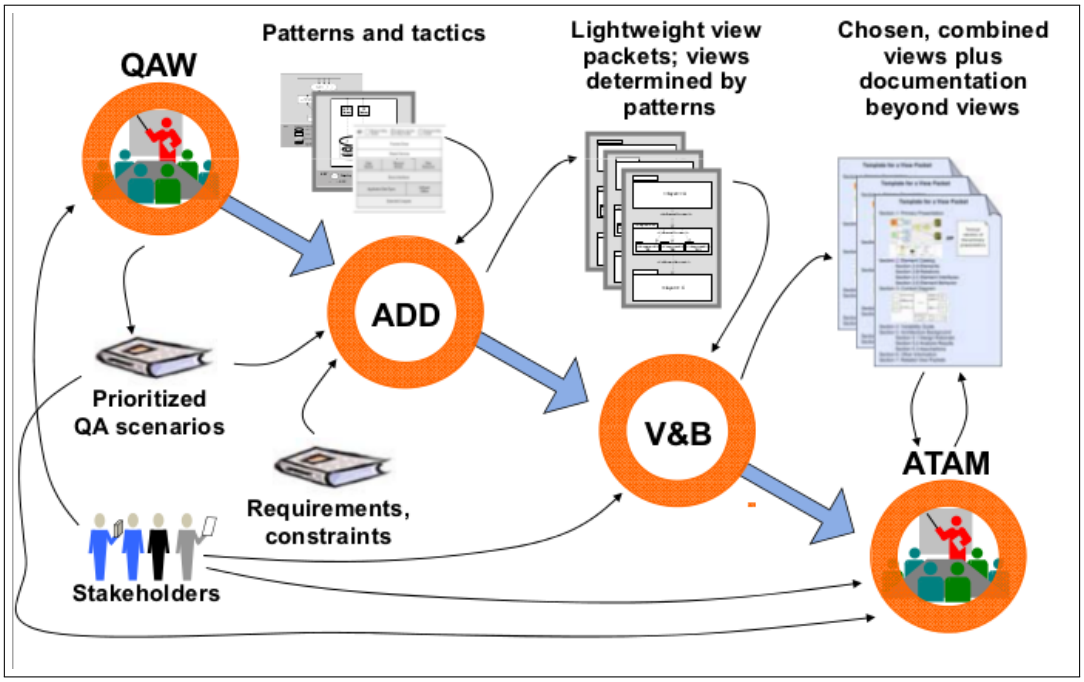


Figura 2.2: Tomado de Instituto de Ingeniería de Software[2]

2.3. Decisiones de diseño arquitectónico

Son decisiones de diseño de alto nivel que un arquitecto toma para satisfacer los requisitos funcionales y no funcionales (atributos de calidad). El proceso de diseño de arquitectura es una actividad de toma de decisiones y una decisión de arquitectura es una elección entre las opciones de diseño basadas en ciertos criterios. Puede haber interdependencia entre varias decisiones. Por ejemplo, una decisión anterior puede limitar las opciones disponibles o imponer algunas restricciones en las decisiones posteriores sobre el sistema [9].

Jansen y Bosch [49] definen una decisión de diseño arquitectónico como la descripción del conjunto de adiciones, sustracciones y modificaciones a la arquitectura del software, su lógica, reglas de diseño, restricciones de diseño y los requisitos nuevos que traen consigo

[2]https://www.sei.cmu.edu

nuevas decisiones. Con la definición de decisiones de diseño arquitectónico se utilizan los siguientes elementos:

- *Justificación:* Es la razón detrás de una decisión de diseño arquitectónico también describe por qué se realiza un cambio en la arquitectura del software.
- *Reglas y Restricciones de Diseño:* Son prescripciones para otras decisiones de diseño. Las reglas son pautas obligatorias, mientras que las restricciones limitan el diseño para permanecer vigentes.
- *Restricciones de Diseño:* Describen el lado opuesto de las reglas de diseño. Describen lo que no está permitido en el futuro del diseño, es decir, que prohíben ciertos comportamientos y estructuras.
- *Requisitos adicionales:* Una decisión de diseño puede resultar en requisitos adicionales que la arquitectura debe satisfacer.

Estos nuevos requisitos necesitan ser abordados por decisiones de diseño adicionales. Una decisión de diseño arquitectónico es, por lo tanto, el resultado de un proceso de diseño durante la construcción inicial o la evolución de un sistema de software. Las decisiones de diseño arquitectónico, entre otras, pueden estar relacionadas con la aplicación el dominio del sistema, los estilos arquitectónicos y patrones utilizados en el sistema, los componentes COTS y otras selecciones de infraestructura, así como otros aspectos necesarios para satisfacer los requisitos del sistema. Proponen ver una arquitectura de software como un conjunto de decisiones explícitas de diseño arquitectónico. Desde esta perspectiva, la arquitectura del software es el resultado de las decisiones de diseño arquitectónico tomadas a lo largo del tiempo.

2.4. Rationale Arquitectónico

La percepción general es que los diseñadores y arquitectos generalmente no entienden completamente el papel crítico del uso sistemático y la captura de la justificación del diseño. Sin embargo, hasta la fecha hay poca evidencia empírica disponible sobre qué significan los motivos de diseño para los profesionales, cuán valiosos los consideran y cómo los utilizan y documentan durante el proceso de diseño [85]. El rationale se refiere a la justificación de una decisión. Son las razones de la toma de decisiones de diseño arquitectónico, las cuales comprenden numerosos tipos, por ejemplo, elección de plataformas, frameworks, estilos arquitectónicos, tácticas arquitectónicas, mecanismos de

comunicación y patrones de diseño de alto nivel, entre otros[28]. Estas decisiones permanecen implícitas durante el inicio y evolución de todo el proyecto [3]. Las decisiones constituyen la lógica del sistema, esta información es valiosa, ya que brinda información útil sobre la intención de los interesados y es vital que ésta sea explícita, puesto que, facilita la evolución del software [72].

También es visto como la base subyacente para la arquitectura en términos de restricciones derivadas de los requerimientos del sistems [70][37]. El rationale, como componente de la arquitectura software, se convierte en un punto crítico para diseñarla, describir las decisiones, evolucionarla y tomar nuevas decisiones [22]. Antony Tang et. al. [85] exploran el valor del rationale desde el punto de vista del diseño, buscando documentar el conocimiento de fondo con sus decisiones de diseño y se refleja la importancia que tiene para los arquitectos documentar el rationale como alternativa para analizar las decisiones de diseño. Por otra parte el rationale es una de las alternativas que tienen los arquitectos para justificar el diseño de la arquitectura. Este tipo de alternativa le permite al profesional de diseño razonar sobre sus decisiones. Sin embargo, la mayor parte de procesos de diseño de arquitectura de un producto software no consideran explícitamente la necesidad de documentar y justificar las decisiones de diseño.

2.5. Ingeniería dirigida por modelos (MDE)

MDE es un paradigma que apunta a elevar el nivel de abstracción del desarrollo de software al enfocarse en actividades de modelado en lugar de codificación. Según el paradigma MDE, a partir de un modelo y mediante transformaciones del modelo, es posible obtener automáticamente una variedad de artefactos, como nuevos modelos, código, etc. En este contexto, el desarrollo de software puede verse como una cadena de transformaciones, proceso donde los modelos de abstracción de bajo nivel se obtienen automáticamente a partir de la transformación de modelos de abstracción de alto nivel [20]. Últimamente, el MDE ha obtenido reconocimiento académico e industrial como una práctica efectiva para lidiar con la creciente complejidad del software embebido moderno. MDE es un paradigma de ingeniería que aborda el desarrollo de software como el proceso de diseñar modelos y refinarlos, comenzando desde niveles más altos y avanzando hacia niveles más bajos de abstracción, a través de las llamadas transformaciones de modelos [20].

2.5.1. Modelos

Los modelos consisten en conjuntos de elementos que describen alguna realidad física, abstracta o hipotética. Los buenos modelos sirven como medios de comunicación; son más baratos de construir que los reales; y se pueden transformar en una implementación. Los modelos pueden recorrer toda la gama, a partir de bocetos llegar a planos bastante detallados y modelos totalmente ejecutables. Todos son útiles en el contexto apropiado [64].

2.5.2. Meta-modelos

Un meta-modelo es un modelo de un lenguaje de modelado que nos permiten representar abstracciones de un dominio en particular, dando significado a los modelos y restringiendo características irrelevantes [64]. Al describir un modelo por medio de meta-modelos es posible eliminar la ambiguedad y falta de precisión, además los meta-modelos cuentan con información relevante que permite ser utilizada en el momento de realizar una transformación. Un meta-modelo define la estructura, la semántica y las restricciones para una familia[3] de modelos. Por ejemplo, un modelo que emplea diagramas UML es capturado por el meta-modelo UML, el cual describe cómo se pueden estructurar los modelos UML, los elementos que pueden contener y las propiedades que exhiben esos elementos. Un meta-modelo puede describir algunas propiedades de cualquier plataforma en particular, no solo UML, y las propiedades de una plataforma pueden ser descritas por más de un meta-modelo [64].

Los modelo para especificar meta-modelos se conocen como meta-metamodelos [62]:

- Meta-modelo: modelo que permiten crear otros modelos, que son la representación de los elementos de dominio y mundo real.
- Meta-metamodelo: modelo con un alto grado de abstracción a partir del cual se pueden especificar meta-modelos.

En la Figura 2.3 se pueden observar los diferentes modelos con su interacción entre cada uno de ellos.

[3]El término familia se usa aquí para agrupar modelos que comparten una sintaxis y una semántica comunes.

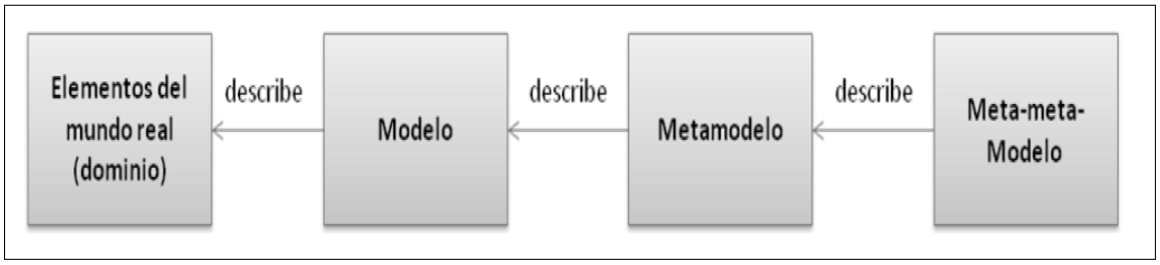


Figura 2.3: Modelo, meta-modelo y meta-metamodelo. Tomado de [62]

Marín et al. [62] mencionan cuatro niveles de abstracción en modelado, el nivel base M0, representa los elementos del mundo real, el nivel M1 que representaría los programas informáticos especificados en un lenguaje, el nivel M2 que sería la especificación del lenguaje utilizado para la representación de los programas, es decir el meta-modelo. Finalmente el nivel M3 que es el de mayor abstracción, que define el modelo que permite especificar meta-modelos(incluyendo éste mismo), es decir el meta-metamodelo. Existen varios meta-metamodelos (M3), planteado por la OMG, está MOF, el estándard para gramáticas libres de contexto, ISO/IEC 14977 EBNF (Extended Backus-Naur Form), ECore planteado por el EMF, y GOPRR (Grafo, Objeto, Propiedad, Relación, y Rol). El objetivo de generar niveles de abstracción tan altos, es proveer un mecanismo común que permita expresar modeloes y transformaciones de un modelo a otro para garantizar la interoperabilidad de las herramientas de modelado y transformaciones.

La noción de meta-modelo se basa en la arquitectura de meta-datos que se muestra en la Figura 2.4, adoptada por el consorcio OMG en la especificación del MOF. Ésta figura muestra la diferencia entre información, modelos, meta-modelos y meta-metamodelos. MOF nombra cada una de estas capas o niveles con los nombres M0, M1, M2 y M3 respectivamente. Es importante fijarse en que estos conceptos están en niveles diferentes y poseen significados diferentes aunque usen la misma notación [61].

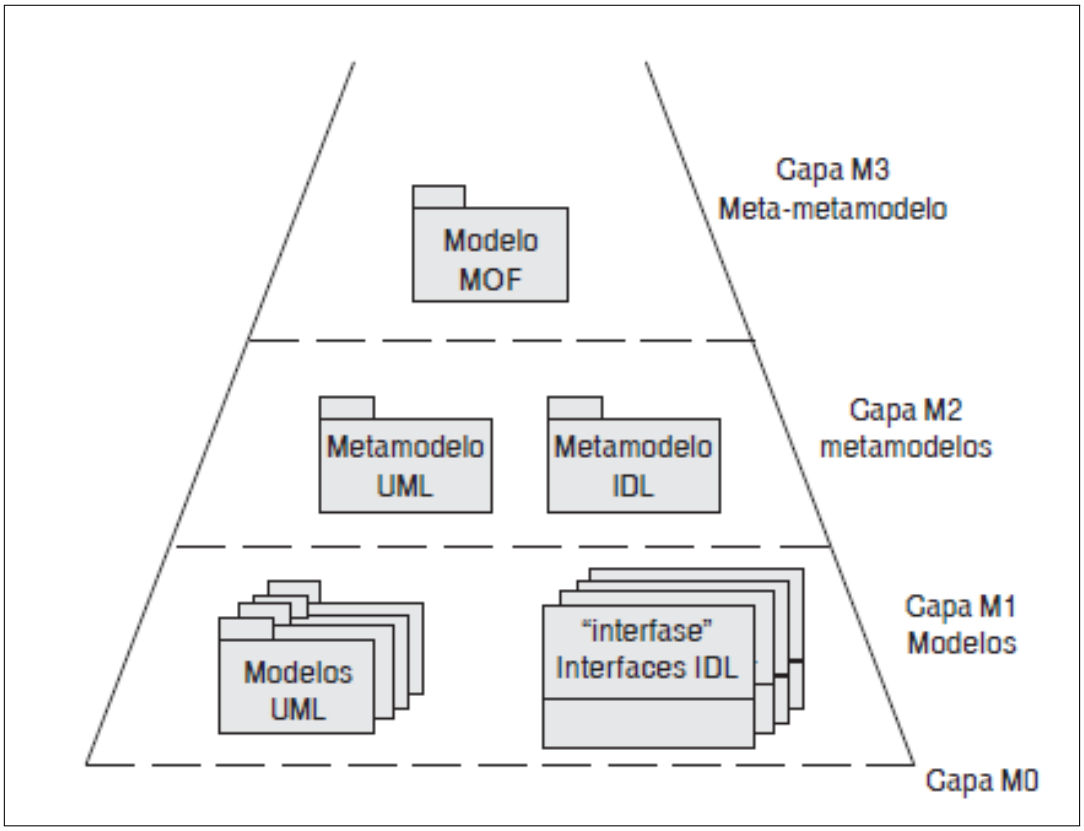


Figura 2.4: Arquitectura de Metadatos en MOF. Tomado de [61]

2.5.3. XML Metadata Interchange (XMI)

El estándar de la OMG XMI es una especificación basada en XML para compartir metadatos MDA. Actualmente XMI es la base para conseguir interoperabilidad entre herramientas de apoyo a los enfoques MDE. Una de las grandes ventajas de XMI es que sobre él se pude almacenar todos los modelos basados en MOF utilizando un mismo esquema. Sin embargo, tiene inconvenientes, como su complejo formato para ser comprendido por humanos o que muchas herramientas que trabajan con XMI no se ajustan a las especificaciones de la recomendación [61].

Capítulo 3

Estado del Arte y Trabajos Relacionados

3.1. Revisión de la Literatura

Con el objetivo de dar una visión global, objetiva y transparente a la investigación, se realizó una revisión sistemática de la literatura con el propósito de identificar trabajos relacionados con la documentación de las decisiones del diseño arquitectónico, particularmente, el *rationale* en el contexto de un enfoque liviano de documentación para el desarrollo ágil con prácticas de arquitectura. En esta revisión se categoriza, de acuerdo a su contenido, para evaluar la disponibilidad existente de información en la literatura, con el objetivo de identificar dónde hacer aportes la comunidad científica y la industria. Para la revisión sistemática, se siguen los protocolos y metodologías aplicados en los estudios de tipo secundario [16, 7].

El protocolo para aplicar la revisión sistemática se define a partir de los elementos que incluyen en sus trabajos los autores anteriormente mencionados. Así, la metodología resultante consta de cinco fases generales cada una con elementos específicos a ejecutar:

1. **Formulación de la pregunta y la cadena de búsqueda**
   - Preguntas de investigación asociadas al estudio.
   - Cadena de Búsqueda
2. **Selección de las fuentes y adaptación de la cadena de búsqueda**

- Fuente de datos.
- Estrategia de búsqueda.

3. **Selección de los estudios**
   - Criterios de inclusión y exclusión.
   - Evaluación.
   - Clasificación de estudios.
4. **Extracción de información**
5. **Resumen de los resultados**
   - Resultados de la búsqueda.

A continuación se muestra en detalle la ejecución del protocolo de acuerdo a la metodología establecida.

3.1.1. Formulación de la pregunta

En relación con la pregunta de investigación es necesario considerar la existencia de trabajos relacionados con el análisis de métodos orientados a la documentación de las decisiones de diseño arquitectónico y su rationale, la creación de métodos orientados a la trazabilidad que ayuden a lograrlo y trabajos existentes en las mismas áreas. La tabla 3.1 ilustra el conjunto de preguntas de investigación asociadas al estudio.

Id	Descripción
PI1	¿Que trabajos de investigación existen sobre documentación de decisiones de diseño arquitectónico, específicamente el "rationale" y sus estrategias?
PI2	¿Que trabajos existen sobre el uso de métodos para análisis de las decisiones de diseño arquitectónico?
PI3	¿Cuales son los principales mecanismos para documentar y especificar la arquitectura en proyectos ágiles?
PI4	¿Que trabajos de investigación existe sobre el uso de anotaciones en el código para documentar la arquitectura, el rationale y decisiones de diseño arquitectónico?

Tabla 3.1: Preguntas de investigación del estudio

3.1.2. Selección de las fuentes

A continuación, se formulan las cadenas básicas de búsqueda y palabras clave, las cuales están formadas por los principales tópicos de la investigación. Conviene subrayar, que las cadenas deben adecuarse para cada base de datos antes de ejecutarse. Con el propósito de obtener mayor precisión en la búsqueda de artículos.

La Tabla 3.2 ilustra la cadena de búsqueda para la revisión sistemática como aproximación inicial. Las palabras clave pueden cambiar en relación al estudio de Brereton et al.[19].

Id	Cadenas generales básicas de búsqueda
1	((software **OR** implementation) **AND** (documentation **OR** documenting) **AND** design **AND** (decisions) **AND** rationale **AND** (architecture **OR** architectural))
2	(((((documentation) **AND** architecture) **OR** architectural) **AND** rationale) **AND** decision)
3	(((software **OR** engineering **OR** recovery) **AND** (architecture **OR** architectural) **AND** design **AND** pattern **AND** attributes **AND** tactics **AND** annotations) **AND** rationale)

Tabla 3.2: Cadenas de búsqueda

Los términos claves principales se relacionan con el operador booleano **AND**, mientras que los términos alternativos (sinónimos, subcategorías, etc.) se ligan mediante el operador lógico **OR**.

Para la búsqueda de trabajos relacionados con las cadenas de búsqueda anteriormente descritas se consulto principalmente en bases de datos académicas, así como también, se realiza la inclusión de recursos no académicos. Así mismo, se definen unos criterios mínimos para aceptar una fuente de información. Ademas, por la cantidad de resultados que arroja el motor de búsqueda de google, se limitó la exploración a las primeras 2 páginas de resultados. La Tabla 3.3 ilustra la estrategia de búsqueda del presente trabajo.

Tipo de búsqueda	Base de datos de trabajos
Bases de datos Académicas	- Scopus - Science Direct - IEEExplore - Librería Digital ACM - Base de Datos Springer
Fuentes no académicas	- Google - Academia.edu - ResearchGate
Tipos de trabajo	- Journals - Workshops - Artículos en Conferencias - Contribuyentes en conferencias de industria y/ó profesional - Publicaciones en linea no academicas
Búsqueda aplicada a	- Titulo - Resumen - Palabras Clave
Idiomas	- Ingles y español
Ventana de tiempo	- Artículos desde 2005 a 2017

Tabla 3.3: Estrategia de búsqueda

3.1.3. Selección de los estudios

La revisión sistemática cuenta con un flujo de ejecución que hemos resumido en cuatro pasos, donde se ilustra la gestión de los estudios hallados, nos basamos en la estrategia de Arias et. al. [7]. Conviene subrayar, que se definió un proceso de ejecución de acuerdo a la necesidad de nuestra investigación. Primero, se busca en las fuentes académicas y no académicas. Segundo, se analiza y adaptan las cadenas de búsqueda. Tercero, evaluar el artículo, con los criterios para artículos primarios o secundarios. Cuarto, una vez el artículo se evalué positivamente, se clasifica como primario o secundario. La Figura 3.1 describe el flujo de ejecución.

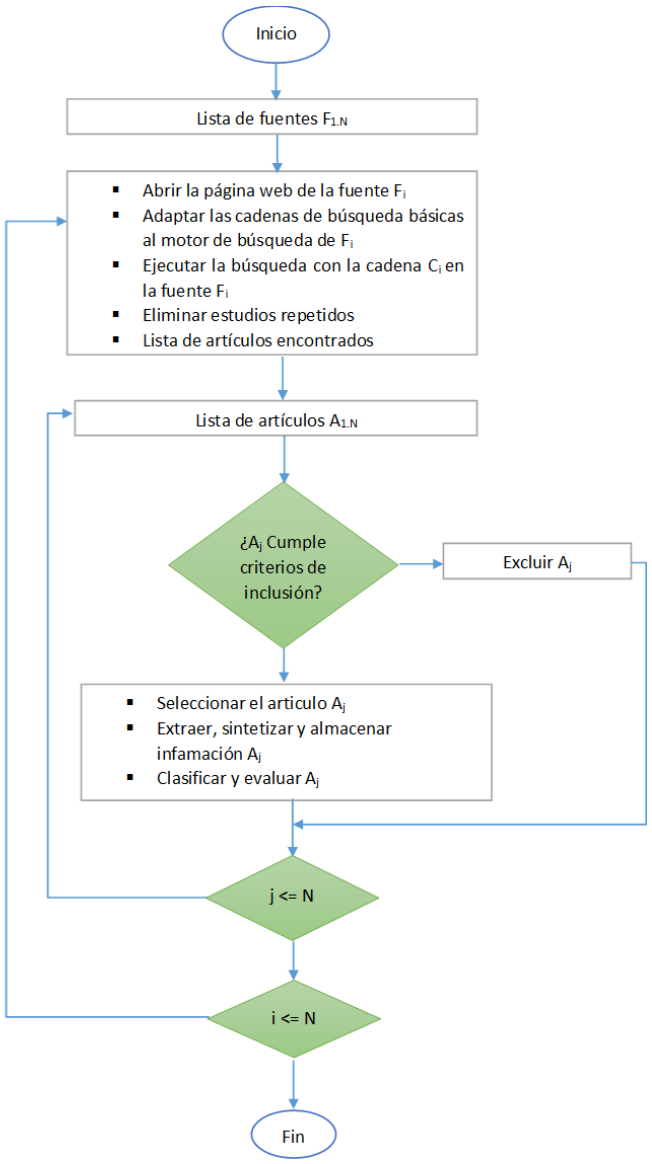


Figura 3.1: Ejecución revisión sistemática

3.1.3.1. Criterios de inclusión y exclusión

El criterio de inclusión de los artículos es basado en el titulo, palabras clave y resumen, con la finalidad de determinar si la propuesta estaba relacionada con la documentación del rationale de las decisiones de diseño arquitectónico. También, deben estar relacionados con técnicas, métodos o herramientas de soporte. Como criterio de exclusión, se

define que los estudios que no representan un artículo de investigación, como reportes técnicos, posters, diapositivas, capacitaciones, tutoriales y cualquier otro recurso que no represente un artículo formal de investigación. Así mismo, se excluyen todos los artículos que se relacionan con el rationale en diferentes fases o etapas del ciclo de desarrollo de software que no sea el diseño o el mantenimiento de la arquitectura software.

3.1.3.2. Evaluación

Para la evaluación de cada uno de los trabajos se han diseñado unas preguntas de evaluación como filtro de aceptación. Cada pregunta de evaluación se compone de una escala de calificación, el rango va desde **(0.0)** a **(5.0)** donde el mino valor para ser tomado en cuenta es **(3.0)**. Estos rangos ilustran la puntuación obtenida de un trabajo para el revisor del estudio. El nivel de calificación otorgado para cada artículo ilustra el grado de pertenencia de los mismos al trabajo. La calificación consiste en el promedio de cada una de las calificaciones otorgadas en cada pregunta de investigación. Los artículos escogidos tienen 3 calificaciones dependiendo de unas reglas especificas:

- El estudio es aceptado si obtiene una calificación promediada igual o superior a **(4.0)** sobre cada una de las preguntas de evaluación de estudios primarios ilustradas en la Tabla 3.4.

Id	Descripción
PE1	¿El estudio presenta una metodología sistemática de la documentación del rationale de las decisiones de diseño arquitectónico que pueda ser aplicado a otros trabajos?
PE2	¿La propuesta es trabajada con herramientas de análisis y/o documentación de decisiones de diseño arquitectónico, específicamente herramientas para colocar el rationale de las principales decisiones de diseño explícito en la arquitectura, es decir, como un artefacto más?

Tabla 3.4: Preguntas de evaluación de trabajos primarios revisión sistemática

- La calificación promediada que debe obtener el estudio para ser aceptado debe ser igual o superior a **(3.0)** sobre cada una de las preguntas de evaluación de estudios definidas en la Tabla 3.5.

- El estudio es descartado si obtiene una calificación promediada inferior a **(3.0)** sobre cada una de las preguntas de evaluación de estudios definidas en la Tabla 3.5.

Id	Descripción
PE3	¿El estudio presenta evidencia positiva del impacto de documentar el rationale de las decisiones de diseño arquitectónico en el compromiso de la evolución de desarrollo de software?
PE4	¿El estudio presenta evidencia positiva del impacto de documentar el rationale de las decisiones de diseño arquitectónico en el producto?
PE5	¿El estudio presenta una metodología sistemática para documentar el rationale de las decisiones de diseño arquitectónico que pueda ser aplicado a otros trabajos?

Tabla 3.5: Preguntas de evaluación de trabajos revisión sistemática

3.1.3.3. Clasificación de estudios

La clasificación de los estudios relacionados se realiza de acuerdo a las preguntas de investigación, el nivel de pertinencia se formula con relación a la orientación del estudio hacia la creación, edición y mejora de modelos de comunicación del rationale de las decisiones de diseño arquitectónico, así como las herramientas utilizadas para su documentación, análisis y evaluación. El esquema de clasificación de los estudios es:

1. **Documentación del rationale y decisiones de diseño arquitectónico:** Esta categoría se relaciona con la pregunta de investigación **PI1** donde se clasifican todos los estudios encontrados sobre "rationale" y documentación del mismo o de herramientas de soporte usadas para éste fin.

2. **Documentación y visualización de decisiones de diseño arquitectónico:** Las herramientas de documentación y visualización de las decisiones de diseño arquitectónico y los modelos usados para definir los elementos de documentación. Categoría dirigida a la pregunta de investigación **PI2**.

3. **Arquitectura de software y métodos ágiles:** Esta categoría esta dirigida a indexar trabajos de la pregunta de investigación **PI3**. Basándonos en la siguiente

declaración: la arquitectura de software se perfila en etapas tempranas del desarrollo y tienen un papel fundamental para lograr la satisfacción de los atributos de calidad del sistema y para guiar su desarrollo, entre otras cosas. Es necesario encontrar criterios que permitan definir practicas de arquitectura en los métodos ágiles de desarrollo de software, sin ir en contra de sus postulados, por ejemplo, el manifiesto ágil y sus pilares.

4. **Enfoques que utilizan anotaciones en el código para documentar distintos tipos de decisiones:** Enfoques que utilicen las anotaciones en el código para documentar distintos tipos de preocupaciones relacionados con la arquitectura, por ejemplo, estilos arquitectónicos, patrones de diseño, estrategias, tácticas de arquitectura, etc. Están catalogados en dirección a la pregunta de investigación **PI4**.

3.1.4. Extracción de información

Con respecto a la ejecución del flujo anterior, una vez escogidos los estudios principales se realizó la extracción de la información relevante para la revisión sistemática. El criterio de inclusión de información a partir de los estudios consistió en obtener información sobre la estrategia de documentación del rationale y las decisiones de diseño, los criterios mínimos que se deben tener en cuenta en la documentación de arquitecturas ágiles que implementan prácticas de arquitectura. Así mismo, se registran las ideas más importantes del estudio. La información de las publicaciones se almacenó en una formato de extracción de datos. La Tabla 3.6 muestra la estructura.

Autor	Información de los autores
Título	Título del estudio
Año	Año de publicación
Base de datos	Corresponde a la fuentes de búsqueda
Estrategia	Corresponde al enfoque propuesto
Aportes	Aportes de la investigación

Tabla 3.6: Formato extracción de datos revisión sistemática

3.1.5. Resumen de los resultados

3.1.5.1. Resultados de la búsqueda

El proceso de búsqueda se realizó siguiendo los criterios y estrategias descritos en la sección anterior. A partir de la información extraída de los estudios, se realizó un análisis estadístico para mostrar descubrimientos relevantes de la revisión sistemática. A continuación se muestran los resultados desde diferentes puntos de vista.

La Figura 3.2 muestra un resumen del número de artículos obtenidos en cada paso del proceso de búsqueda. Como el resultado muestra, el número de estudios, el cual es relativamente pequeño, todos estos artículos fueron publicados entre el 2002 y 2019.

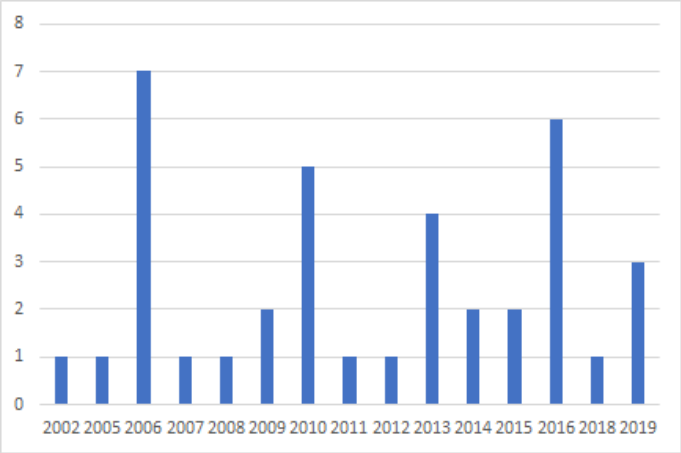


Figura 3.2: Distribución de estudios por año

3.1.5.2. Análisis de los estudios

En esta sección se realiza un análisis de los estudios principales obtenidos siguiendo los criterios de clasificación y las preguntas de investigación que se han esbozado anteriormente. Las respuestas a las preguntas de investigación indicadas, de acuerdo con el análisis realizado en los estudios primarios seleccionados, son las siguientes:

PI1. ¿Que trabajos de investigación existen sobre documentación de decisiones de diseño arquitectónico, específicamente el rationale y métodos?

La distribución de los estudios primarios en términos de las estrategias que consideran se muestran en la Fig. 3.3. La Tabla 3.7 proporciona detalles de qué estrategias fueron consideradas por cada uno de los estudios primarios.

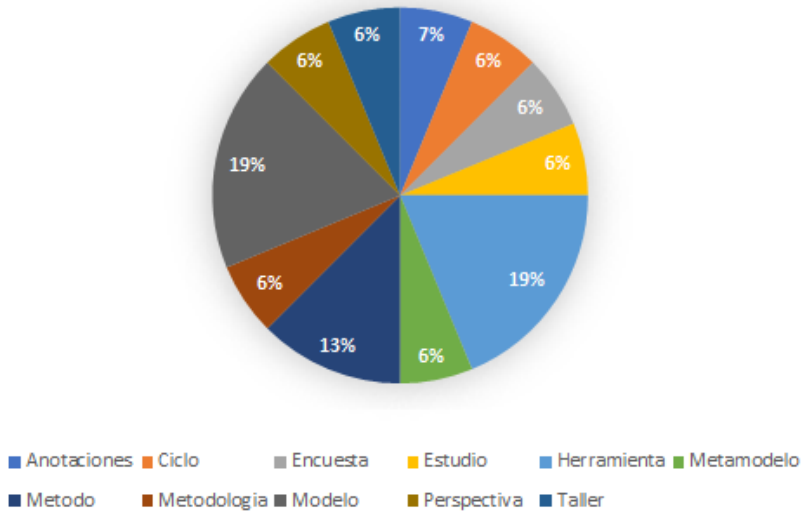


Figura 3.3: Distribución de estudios sobre el rationale arquitectónico por estrategia

Estrategia	Estudio	Puntuación
Metamodelo	[68]	5.0
Anotaciones	[36]	5.0
Modelo	[40] [55] [25]	4.0
Ciclo	[32]	5.0
Herramienta	[71] [80] [92]	4.5
Perspectiva	[47]	4.0
Metodología	[78]	4.0
Método	[5] [83]	4.0
Encuesta	[85]	4.5
Taller	[8]	4.5
Estudio	[38]	4.4

Tabla 3.7: Estrategias consideradas por cada estudio primario de Rationale

PI2. ¿Que trabajos existen sobre el uso de métodos para análisis de las decisiones de diseño arquitectónico?

La Figura 3.4 muestra los resultados de las estrategias utilizadas para documentar las decisiones de diseño arquitectónico. La Tabla 3.8 proporciona detalles de qué estrategias fueron consideradas por cada uno de los estudios primarios.

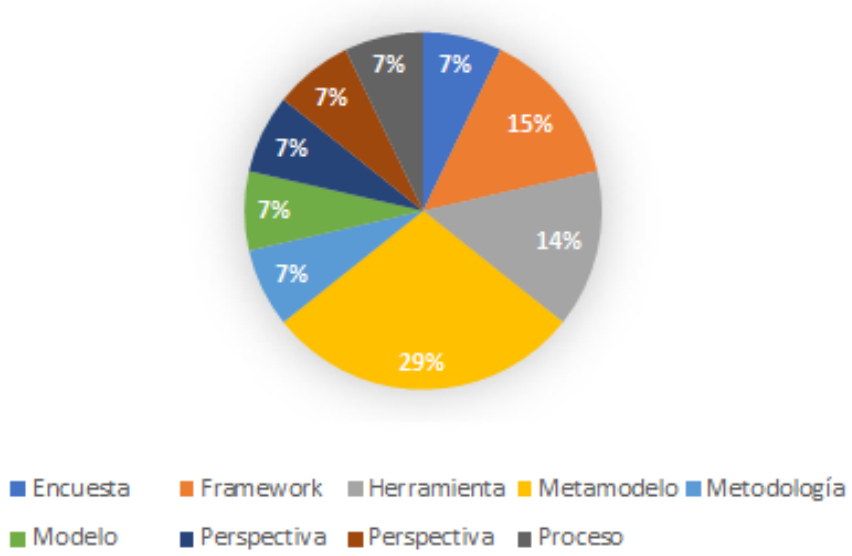


Figura 3.4: Distribución de estudios de decisiones de diseño arquitectónico por estrategia

Estrategia	Estudio	Puntuación
Proceso	[89]	4.5
Metodología	[27]	4.0
Metamodelo	[34] [23] [86] [60]	5.0
Modelo	[28]	4.0
Herramienta	[45] [24]	4.5
Encuesta	[88]	4.5
Perspectiva	[49] [87]	4.0
Framework	[26] [15]	3.9

Tabla 3.8: Estrategias consideradas por cada estudio primario de decisiones de diseño arquitectónico

PI3. ¿Cuales son los principales mecanismos para documentar y especificar la arquitectura en proyectos ágiles?

El objetivo de esta sección es encontrar estudios y bases que se deben tener en cuenta a la hora de proponer algún tipo de mecanismo para documentar la arquitectura en ambientes ágiles de desarrollo de software. La Figura 3.5 muestra la distribución de los estudios principales según su estrategia de investigación. La Tabla 3.9 muestra los detalles de cada estudio principal. Los resultados de esta clasificación muestran que la mayoría de los estudios presentan algún tipo de propuesta sobre la documentación de arquitectura con criterios ágiles.

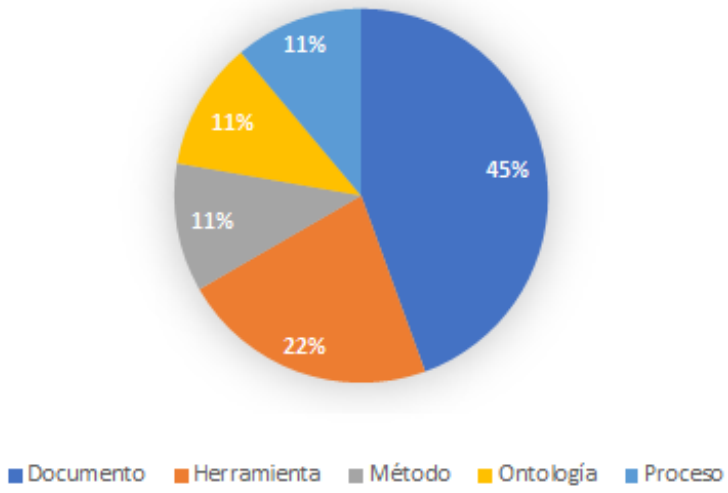


Figura 3.5: Distribución de estudios de criterios para una arquitectura ágil por estrategia

Estrategia	Estudio	Puntuación
Documento	[41] [48] [3] [56]	4.5
Herramienta	[91] [4]	4.0
Proceso	[90]	4.0
Ontología	[17]	4.5
Método	[29]	4.0

Tabla 3.9: Estrategias consideradas por cada estudio primario de criterios para una arquitectura ágil

PI4. ¿Que trabajos de investigación existen sobre el uso de anotaciones en el código para documentar la arquitectura, el rationale o decisiones de diseño arquitectónico?

Esta sección esta enfocada a encontrar estudios que utilicen anotaciones en el código para documentar distintos tipos de decisiones, por ejemplo, estilos arquitectónicos, patrones de diseño, estrategias, tácticas, etc.

La Figura 3.6 muestra la distribución de los estudios principales relacionados con la documentación en el código mediante anotaciones que se han publicado. La Tabla 3.10 muestra los estudios principales por estrategia.

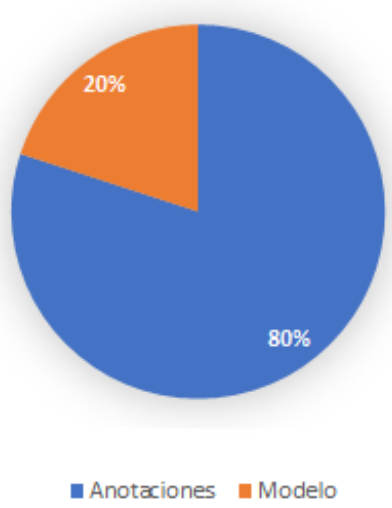


Figura 3.6: Distribución de estudios con enfoque de anotaciones en el código

Estrategia	Estudio	Puntuación
Modelo	[42]	4.0
Anotaciones	[44] [54] [84] [69]	5.0

Tabla 3.10: Estudios principales con enfoque en anotaciones en el código

3.1.6. Discusión

La revisión sistemática proporciona una comprensión estructurada del estado del arte de documentación de las decisiones de diseño arquitectónico y su rationale, centrado en arquitecturas de software con enfoques ágiles. Esto se logró mediante la identificación de estudios más relevantes relacionados con distintos tipos de estrategias, por ejemplo, anotaciones en el código, modelos, procesos, ciclos, herramientas, metamodelos, ontologías, frameworks y perspectivas. Para la realización de este estudio, se han utilizado las siguientes bases de datos: biblioteca digital ACM, IEEE Xplore, y Scopus, ya que se sabe que estas fuentes devuelven la mayoría de las publicaciones.

Los resultados que obtuvimos durante el análisis de los estudios más relevantes muestran que la investigación existente sobre documentar el rationale de las decisiones de diseño arquitectónico de manera explicita en la arquitectura es muy preliminar o incluso inmadura, ya que la mayoría de los estudios se centran en modelo de decisiones de diseño y dejan de lado el rationale, y pocos de ellos ofrecen evidencia empírica sólida de documentar el rationale y las decisiones junto a la arquitectura y de su impacto en la evolución de la arquitectura y por ende en el producto software.

También encontramos un conjunto de criterios, que se deben tener en cuenta para trabajar con enfoques de documentación dentro de los procesos que trabajan métodos ágiles, esto con el objetivo de no generar mecanismos invasivos de documentación a la hora de abordar el tema del rationale como un artefacto mas de arquitectura, es decir, los métodos que se propongan en esta área deben tener en cuenta los pilares que fomentan las metodologías ágiles de desarrollo de software y ser capaces de convivir juntos.

La investigación que realizamos acerca del rationale arquitectónico y su documentación, nos ha servido para darnos cuenta que existen diferentes mecanismos que ayudan a capturar ese conocimiento, por ejemplo, anotaciones en el código, metamodelos, plantillas, herramientas visuales, metodologías etc. Unas con más atributos que otras, ya sea por tecnología, facilidad de uso, atributos más sofisticados, etc. Todos estos métodos muy buenos cada uno aporta desde su enfoque; todos pretenden ser las balas de plata. Sin embargo, la demanda de software en la última era a incrementado notablemente, cada día se necesita software funcionando y su tiempo de construcción sea corto, lo que conlleva a omitir los mecanismos de documentación y calidad, aun cuando arquitectos, desarrolladores y de más interesados reconocen la necesidad de utilizarlos. Razón que nos lleva a pensar que aun carecemos de una estrategia efectiva que nos permita enlazarnos de manera concreta con el proceso de desarrollo de software como es ahora, solamente basado en la parte funcional y con tiempos muy mínimos en construcción.

3.2. Trabajos Relacionados Derivados de la Revisión de la Literatura

3.2.1. Documentación de decisiones arquitectónicas al código

Jonathan et. al.[4] presentan ArchJava, una pequeña extensión Java que integra las especificaciones de arquitectura de software en el código de implementación Java, buscando que la implementación se ajuste a las restricciones arquitectónicas:

- Unifica la estructura e implementación de arquitectura en un lenguaje, permitiendo técnicas de implementación flexibles, lo que garantiza la trazabilidad entre la arquitectura y el código, y el apoyo a la co-evolución de la arquitectura e implementación.
- Busca la integridad de la comunicación entre una arquitectura y su aplicación, incluso en la presencia de elementos arquitectónicos avanzados como el tiempo de funcionamiento de los componentes.

Para evaluar su enfoque aplican la extensión a una aplicación de diseño de circuitos pequeños. Usando un diagrama informal de la arquitectura dibujado a mano, el cual usan como guía, para hacer esta arquitectura explícita en el código. Finalmente, concluyen que ArchJava permite a los programadores expresar la estructura arquitectónica para luego ser llenada en la aplicación con código Java. Se busca en cada etapa del ciclo de vida del software, la integridad de la comunicación, asegurando que la aplicación se adapta a la arquitectura especificada. Un estudio de caso sugiere que ArchJava, se puede aplicar para modelar programas Java de tamaño pequeño con relativamente poco esfuerzo, lo que resulta en una estructura de programa que se aproxima más a la arquitectura conceptual del diseñador. Por lo tanto, ArchJava ayuda a promover el diseño eficaz basado en la arquitectura, la implementación, la comprensión del programa, y la evolución [4]. Este trabajo muestra la importancia de poner la arquitectura de manifiesto en el código, pero usa otro enfoque, el de representar las abstracciones. Nuestro enfoque busca representar las principales decisiones de diseño arquitectónico a nivel de modelado.

Hoy la arquitectura de software carece de la noción de decisiones de diseño arquitectónico, las cuales desempeñan un papel crucial, por ejemplo, durante el diseño, desarrollo, evolución reutilización e interpretación. En el diseño, la principal preocupación es qué

decisiones de diseño tomar. En desarrollo es importante saber qué y por qué se ha tomado ciertas decisiones de diseño. La evolución de la arquitectura tiene que ver mucho con las nuevas decisiones de diseño o eliminar las obsoletas para satisfacer los requisitos cambiantes. El desafío es hacer esto en armonía con las decisiones de diseño previamente tomado [49]. Así, Anton Jansen y Jan Bosch [49] proponen una nueva perspectiva sobre la arquitectura de software, definiéndola como la composición de un conjunto de decisiones de diseño arquitectónico, donde lo que se busca es evitar la evaporación del conocimiento de las decisiones de diseño, ya que se han ido convertido en una parte explícita de la arquitectura.

Por otro lado, evidencian el alto costo que tiene el cambio en las arquitecturas de software, muestran su complejidad y como se erosiona durante su evolución [49]. Estos problemas se deben a la pérdida del conocimiento acerca de las decisiones de diseño sobre las que se basa la arquitectura, que aunque está implícitamente dado, normalmente se carece de una representación explícita de las decisiones y su rationale como ciudadanos de primera clase. En consecuencia, el conocimiento sobre estas decisiones de diseño desaparece en la arquitectura, lo que evidencia los problemas antes mencionados. La reutilización de la arquitectura de software es el uso de las mezclas de decisiones de diseño, por ejemplo patrones de diseño o componentes. Este trabajo nos da información muy útil acerca del papel que juegan las decisiones de diseño arquitectónico tanto en el diseño, desarrollo, integración, evolución y reutilización del software. Permiten que se identifiquen algunos problemas como altos costos de cambio de diseño, erosión y reutilización limitada, que son causadas principalmente por la vaporización de estas decisiones de diseño en la arquitectura.

T. M. Hesseet et al. [45] desarrollan una herramienta para la documentación colaborativa e incremental de las decisiones de diseño plasmadas en diferentes artefactos generados en el proceso de desarrollo. Esta herramienta permite registrar el conocimiento relacionado con las decisiones de diseño que se toman en el transcurso del ciclo de vida del software, permitiendo enlazar diferentes artefactos tales como especificaciones de requisitos, diagramas de diseño y una herramienta basada en anotaciones de código para documentar el rationale arquitectónico, código fuente, consiguiendo una trazabilidad entre el diseño y la implementación. T. M. Hesseet et al. [45] utilizan como ejemplo un sistema de gestión de ventas e inventario llamado CoCoME, para el cual se requiere la migración de las partes principales del sistema a la nube. Esta decisión implica cambios drásticos en la estructura y en algunas características de calidad, como la escalabilidad, la fiabilidad y la seguridad. Los responsables de asumir estos cambios utilizan una herramienta de documentación para gestionar las decisiones desde la especificación de requisitos, hasta la actualización del código fuente, permitiendo que interactúen en tiempo real con otros integrantes del grupo de desarrollo, debido a su naturaleza colaborativa e incremental,

con el fin de obtener retroalimentación directa por parte de todos los involucrados en la realización de los cambios y las actualizaciones a los artefactos. Finalmente se obtienen 42 decisiones con 380 elementos de conocimiento de decisión, con lo que concluyen que es posible documentar estructuras complejas de conocimiento de forma colaborativa e incremental. El principal aporte de este trabajo es la manera en que gestionan las decisiones de diseño que impactan en la estructura del sistema. En nuestro trabajo haremos énfasis en intregrar las decisiones con su rationale y no en el problema de la gestión de las decisiones.

3.2.2. Trazando las decisiones de arquitectura

Jane Cleland et. al. [28] presentan un enfoque centrado en la arquitectura para hacer seguimiento a las preocupaciones de calidad de las partes interesadas, tales como fiabilidad, disponibilidad, seguridad, integridad, rendimiento, portabilidad, los requisitos arquitectónicamente significativos, las razones de diseño y el código fuente. En la trazabilidad de preocupaciones arquitectónicas centrada/basada en decisiones (DCT), todos los vínculos de rastreo se centran en decisiones arquitectónicas que incluyen factores tan variados como plataformas, lenguajes, frameworks, patrones de diseño de alto nivel, mecanismos de comunicación, tácticas arquitectónicas de bajo nivel, estilos arquitectónicos, entre otros. Hoy los sistemas de software están diseñados para satisfacer los requisitos funcionales, así como una amplia escala de intereses de calidad relacionados con los atributos antes mencionados [13]. La Figura 3.7 muestra el modelo de trazabilidad centrado en la decisión (DCT).

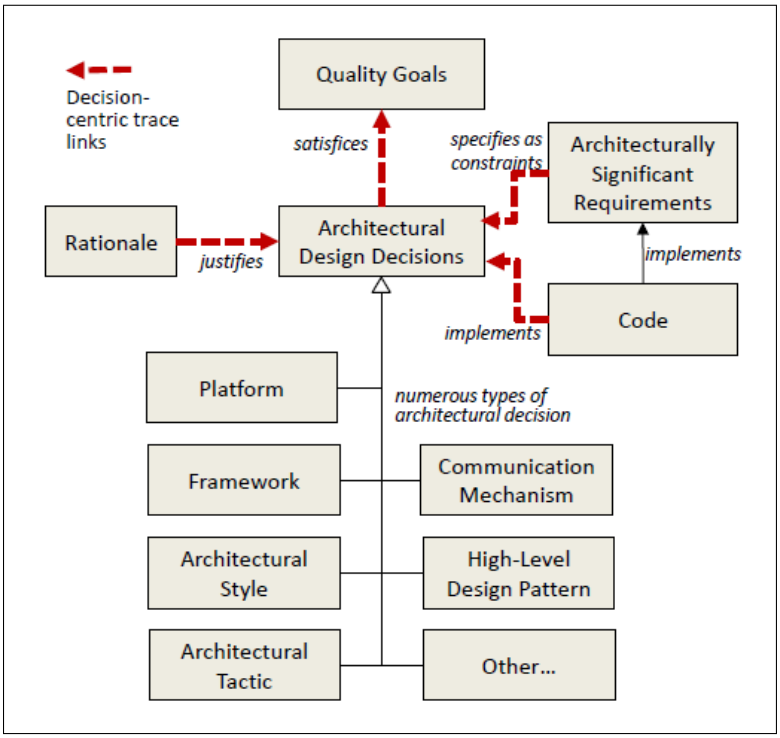


Figura 3.7: Modelo de trazabilidad centrado en la decisión. Tomado de [28]

Por ejemplo, un arquitecto puede decidir abordar una meta de portabilidad mediante el uso de un estricto enfoque en capas, el cual simplifica el proceso de creación de nuevas GUIs específicas de la plataforma o para lograr una meta de disponibilidad mediante la utilización de la táctica de control para supervisar el estado de salud de un componente crítico, o cumplir una meta de rendimiento a través de la utilización de un pool de hilos para administrar recursos compartidos [28]. Facilitar la trazabilidad bidireccional entre las preocupaciones de calidad, las decisiones arquitectónicas, las justificaciones y las áreas pertinentes del código brinda apoyo crítico para varios aspectos del proceso de ingeniería de software, incluyendo análisis de impacto de cambios, validación de requisitos, preservación arquitectónica, construcción de casos de seguridad y a largo plazo mantenimiento del sistema [28]. Por ejemplo, la práctica ha demostrado que la erosión de la arquitectura ocurre a menudo cuando los desarrolladores realizan cambios en el código sin comprender completamente las decisiones arquitectónicas subyacentes y sus preocupaciones relacionadas con la calidad [28].

Los enlaces de rastreo que utilizan para comunicar las principales decisiones de diseño a nivel arquitectónico, van en línea con el enfoque establecido por este trabajo donde, más que documentarlas con una herramienta es establecer un mecanismo que facilite

documentar las principales decisiones de diseño arquitectónico en la arquitectura de software de manera ágil a través de un plugin que permita escribir las decisiones a nivel de código donde los interesados las puedan encontrar más rápido y a partir de esas anotaciones generar un árbol de rastreabilidad donde es más fácil comunicarla.

3.2.3. Relacionando el conocimiento arquitectónico

Los autores Neil Harrison y Paris Avgeriou [42] desarrollan un modelo para la interacción de patrones y tácticas que permite a los arquitectos de software anotar diagramas de arquitectura con información sobre las tácticas utilizadas y su impacto en la estructura general. Este modelo se basa en un análisis en profundidad de los tipos de interacciones involucradas, y muestra varios ejemplos de cómo el modelo se puede utilizar para anotar diferentes tipos de diagramas de arquitectura. Ilustra el modelo y la anotación mostrando ejemplos tomados de sistemas reales, y describe cómo se usó la anotación en revisiones de arquitectura. Las tácticas y los patrones son conceptos arquitectónicos conocidos; el trabajo proporciona una comprensión más específica y en profundidad de cómo interactúan. Su otra contribución clave es que explora el problema más amplio de entender la relación entre las decisiones estratégicas y cómo deben ser adaptadas a la luz de decisiones que involucran nuevas tácticas. Esta propuesta agrega información muy útil a los modelos arquitectónicos para facilitar su análisis y evolución, como consecuencia, aumenta la cantidad de información y amenaza la comprensibilidad. En este trabajo tendremos en cuenta los conceptos que ellos involucran para mejorar la descripción arquitectónica a nivel de modelo.

Hans van Vliet y Antony Tang [90] señalan que hay cuestiones fundamentales que influyen en el diseño de la arquitectura de software. Estos problemas tienen que ver con la toma de decisiones de diseño. Incluyen racionalidad limitada, todo tipo de sesgos cognitivos, las decisiones acerca de cuál es el problema que estamos tratando de resolver, y el papel de las primeras decisiones que se están adoptando. Por ejemplo, un arquitecto de software puede verse influenciado por el contexto en el que se encuentra. Supongamos que él arquitecto está capacitando en el framework Angular que lanzó recientemente Google para el diseño de aplicaciones y en paralelo el dueño del producto le encarga escoger el framework para la nueva aplicación. Según Hans Van hay un alto porcentaje de posibilidad que el escoja este framework aunque este puede ser una solución no óptima dejándose sesgar por el contexto. Es por esto que este trabajo nos muestra la importancia de llevar un control del porqué de la solución, el conjunto de decisiones de diseño y su justificación. Las anotaciones arquitectónicas a nivel de modelos van en línea con el enfoque establecido por este trabajo donde, más que la documentación de las estructuras del sistema, es la

documentación de las decisiones de diseño que se toman a lo largo de la vida del producto de software.

Jan Salvador van der Ven y Jan Bosch [88] estudian las creencias que rodean la arquitectura del software. Las creencias van desde la cantidad de esfuerzo necesario para la documentación de la arquitectura, hasta el tamaño del equipo o las personas responsables de tomar las decisiones arquitectónicas. La mayoría de las creencias se basan en la idea de que el resultado del proyecto depende en gran medida de los métodos utilizados durante el diseño y desarrollo de software. Ellos realizaron una encuesta a 39 arquitectos donde evaluaron 54 decisiones arquitectónicas. En la encuesta se evaluó la forma en que se tomaron las decisiones, los factores de éxito de las decisiones, así como las propiedades de los proyectos. Realizan análisis estadísticos para evaluar algunas de las creencias que existen actualmente en el desarrollo de software. Finalmente concluyen que, para la mayoría de las creencias, no se puede encontrar evidencia estadística, haciendo que estas creencias sean inadecuadas para los mitos, en lugar de guías útiles para predecir el éxito o el fracaso de los proyectos.

3.2.4. Documentación de arquitectura en el contexto ágil

Irit Hadar et. al. [41], presentan un estudio de caso que tiene como objetivo identificar las dificultades que encuentran los arquitectos y otras partes interesadas cuando se ocupan de la documentación de la arquitectura en el desarrollo ágil. Los hallazgos muestran que el documento que contiene la especificación de la arquitectura suele ser muy extenso, complejo y en muchos casos no es capaz de explicarse a sí mismo. Con el fin de ajustar la documentación de la arquitectura al enfoque de documentación ligera de los procesos ágiles, ellos proponen un documento de especificación más corto, que requiera esfuerzos reducidos de documentación el cual resulta en una documentación simplificada que es más fácil revisar, actualizar y comunicar.

Los documentos de arquitectura por lo general suelen ser grandes y complejos, y pueden crecer de decenas a cientos de páginas; consiste en múltiples documentos que dentro y entre ellos abarcan muchos conceptos, relaciones, vistas múltiples y diferentes niveles de abstracción. Jason et al. [48] identificó una lista de desafíos relacionados con la documentación de arquitectura [41], el documento que proponen se basa en los siguientes tres desafíos:

1. Comprensibilidad de los documentos por parte de los arquitectos, revisores y otras

partes interesadas relevantes en los casos de:

- Sistemas grandes y complejos [41].
- Diferentes antecedentes de las partes interesadas, que pueden obstaculizar su comprensión de la lengua y los conceptos utilizados en el documento [41].

2. Localización del conocimiento relevante de la arquitectura
   - Encontrar la información pertinente[41] y el conocimiento relevante[41] dentro de un conjunto de documentos de arquitectura (incluidos los documentos formales, documentos informales, correos electrónicos, etc.)
3. Confianza
   - Mantener la documentación de la arquitectura actualizada, de lo contrario las partes interesadas perderán su credibilidad [41].

Al mirar estos desafíos a través del lente de las metodologías ágiles de desarrollo, se enfatiza la necesidad de reducir la longitud y complejidad de la documentación de la arquitectura. El enfoque de Irit Hadar [41] apoya el minimalismo en el desarrollo ágil y es asistido por una herramienta automática y una plantilla para crear el documento, facilitando un lenguaje común, apoyando la colaboración entre las partes interesadas y fomentando el fácil acceso y la localización del conocimiento de la arquitectura.

Hurtado y Muñoz [50], en su documento presentan algunas aproximaciones para trabajar las arquitecturas, hacen un ligero análisis de ellas; describen también cuales deben ser los criterios mínimos para una arquitectura, sin salirse de los esquemas de agilidad. Plantean algunos lineamientos y prácticas útiles para ser utilizados en cualquier proceso de desarrollo de software: Ante todo, el arquitecto debe:

- Conocer el dominio del problema a resolver.
- Ser un buen comunicador (implica saber escuchar).
- Saber persuadir (si se tiene razón).
- Tener habilidades de gestión, pero no enredarse con estas tareas en su desempeño diario.
- Pensar que siempre se puede mejorar, tanto en los aspectos técnicos como en el humano [50].

La arquitectura debe cumplir también con unos requisitos, pues en la nemotecnia y la sintaxis, es importantes estandarizar conceptos que podrán entender tanto usuarios como profesionales del diseño:

- La prestación de un "vocabulario de elementos de diseño", "los componentes y el conector de tipo".
- La definición de un çonjunto de reglas de configuración".
- La definición de una interpretación "semántica", que da algún significado bien definido a todas las configuraciones de diseño de elementos que cumplan las normas de configuración.
- La definición de .ªnálisis"para las configuraciones de que estilo.

Stefan Voigt y Detlef Hüttemann [91] ven una necesidad de técnicas y métodos que apoyen la documentación en ambientes ágiles teniendo en cuenta que ésta debe ser más fácil de escribir, manejable y actualizable. Hablan sobre los conceptos que se deben tener en cuenta para llevar a cabo una herramienta de documentación ágil, ya que ésta a menudo es descuidada en proyectos ágiles de software, incluso cuando los desarrolladores de software, perciben la necesidad de una buena documentación. Una razón puede encontrarse en herramientas de documentación inadecuadas. El trabajo de ellos ofrece una visión general de las ideas conceptuales centrales para una herramienta de documentación ágil. Su análisis consistió en cuatro talleres (similares a retrospectivas de proyectos), nueve entrevistas con usuarios pilotos y una encuesta en línea de otras compañías desarrollando software (104 encuestados). Aunque cabe aclarar que el análisis no es el objetivo principal de su documento, algunos de los resultados los presentan para apoyar la argumentación del concepto planteado. La identificación de una falta de tiempo como argumento principal contra la documentación en sus entrevistas, indica que debe ser integrado en el proceso de desarrollo incluso a nivel de herramientas. Con base en su investigación, determinan que una de las estrategias de documentación empleadas en proyectos ágiles son las tarjetas y pizarras que registran información de manera temporal, donde las wikis se han identificado como la mejor herramienta electrónica para esto. Sin embargo, identificaron que no es un instrumento adecuado para la trazabilidad de la información ya que no proporcionan capacidades para las correlaciones entre las necesidades y soluciones, convendría que interactuaran con otros sistemas como parte del proceso de desarrollo por ejemplo entornos de desarrollo (IDE), sistemas de control de versión (VCS) y de control de incidencias (JIRA), la información tiene que ser trazable entre las diferentes

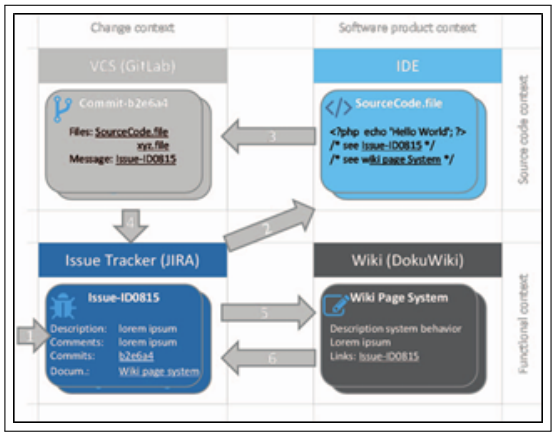


Figura 3.8: Prototipo basado en Wikis. Tomado de [91]

herramientas, para esto diseñan un prototipo que las integra brindando trazabilidad como se indica en la Figura 3.8

El prototipo planteado en consecuencia debería ser capaz de eliminar los principales problemas de la documentación ágil y así hacer más fácil mantener la documentación actualizada, encontrar el lugar correcto para compilar y evitar duplicar la documentación[91]. De acuerdo a los autores, el prototipo seguía siendo evaluado en dos compañías. La primera reacción se reporta como prometedora, según la encuesta que realizaron en línea. Arrojó que la documentación está actualizada, mantiene la información trazable y promete calidad en la documentación como lo indican los requisitos principales para las herramientas de documentación ágil.

3.2.5. Documentación de decisiones de implementación

Hesse et. al. [44] hablan de la importancia que tiene para un equipo de desarrollo de software documentar las decisiones de implementación. Cuando se revisa el código durante el mantenimiento, las decisiones subyacentes deben entenderse y posiblemente ajustarse a la situación actual. En consecuencia, el conocimiento de la decisión raramente se documenta y, por lo tanto, es inaccesible, especialmente cuando los desarrolladores han abandonado el equipo. Por lo tanto, el mantenimiento eficaz se ve obstaculizado. Ellos han desarrollado un modelo de anotación para el conocimiento y decisión integrado con la herramienta de gestión del conocimiento UNICASE ver Figura 3.9. El enfoque permite a los desarrolladores documentar decisiones dentro del código. En este proyecto, se

busca de manera similar documentar las decisiones de diseño arquitectónico a través de la incorporación de un lenguaje de descripción integrada de decisiones de diseño y su rationale.

```
// @Decision Implement input UI using a wizard
// @Issue Complex user input
// @Alternative Use a dialog
// @Contra Need for step-wise user guidance
```

Figura 3.9: Ejemplo de anotación de decisión. Tomado de [44]

Borrego [17] describe el desarrollo global de software ágil como una realidad, ya que hoy en día los productos de software están obligados a entrar en el mercado con más velocidad que antes. Esta situación ha empujado a las empresas de desarrollo global de software, a adoptar maneras más ligeras de desarrollar software para satisfacer las demandas del mercado. Sin embargo, las empresas de desarrollo global de software se han enfrentado a un aumento de la deuda técnica y del conocimiento arquitectónico, principalmente debido a las diferencias inherentes entre desarrollo ágil de software y desarrollo global de software, especialmente en la manipulación de la documentación [17]. En su proyecto de investigación Borrego propone explotar el conocimiento arquitectónico que se registra en medios electrónicos textuales no estructurados generalmente utilizados en el desarrollo global de software ágil, con el fin de disminuir los problemas de evolución y mantenimiento del software causado por la falta de conocimiento arquitectónico. Hasta ahora, los resultados preliminares de esta investigación demuestran que el enfoque propuesto podría ser factible en ambientes de desarrollo global de software ágil.

3.2.6. Rationale arquitectónico

3.2.6.1. Rationale arquitectónico en la arquitectura empresarial

De Jong et. al. [32] resaltan la importancia que tiene el fundamento arquitectónico para la arquitectura empresarial, el cual se refieren a la organización o empresa, sus sistemas de TI [1], unidades (funcionales), procesos de negocio y contactos externos, como clientes y proveedores. Mencionan que durante el proceso de diseño se toman decisiones que obedecen a un razonamiento, en el cual se tiene que identificar el contexto y los requisitos de la empresa, formular y estructurar el problema de diseño, pensar en posibles opciones de

[1] Administración de los sistemas de tecnología de la información en un centro de datos empresarial.

solución y hacer un balance entre diferentes opciones. Diseñar arquitecturas complejas para grandes empresas puede considerarse un problema complejo, esto se debe a que el problema de diseño está mal definido; no hay una regla de detención que indique cuándo se ha alcanzado una solución aceptable; es difícil saber qué es verdadero o falso en un diseño (al menos no antes de la implementación); resolver un problema de diseño puede dar lugar a otros problemas de diseño relacionados; y entender el problema depende de cómo el diseñador enfrente su resolución. Los autores hacen énfasis en el caso de problemas de diseño complejos, es importante capturar los fundamentos del diseño, las razones subyacentes a las decisiones de diseño y documentar adecuadamente las decisiones de diseño arquitectónico y su fundamento es esencial para cualquier descripción arquitectónica, ya que explica el razonamiento detrás de por qué la arquitectura es como es [32]. Comentan que sin una justificación explícita adecuada, la arquitectura se degrada con el paso del tiempo debido a los cambios y las arquitecturas se vuelven cada vez más frágiles si no se conserva la documentación de las decisiones [32]. Por ejemplo, los arquitectos pueden dejar la organización, cambiar roles o cambiar proyectos. Además, la documentación arquitectónica con justificación facilita a los arquitectos comprender un diseño, especialmente si no participaron en el proceso de diseño original [32]. Por otra parte, indican que en trabajos encontrados en la literatura, los arquitectos están totalmente de acuerdo con la premisa de que no pueden entender un diseño de manera efectiva sin su justificación [32].

En su estudio sobre el diseño arquitectónico, encontraron muchos métodos y técnicas para razonar sobre los diseños de arquitectura. Técnicas específicas de solución en diferentes campos, por ejemplo, análisis de restricciones, análisis de riesgos y análisis de supuestos entre otros. Sin embargo, hallaron que la investigación y la presentación de estas técnicas están fragmentadas y falta un método único que las comprenda a todas. Proponen un método denominado ***Ciclo de captura de rationale (RCC)*** , que ayuda a los arquitectos a capturarlo durante el proceso de diseño en el contexto de la empresa. El punto de partida de su investigación es la percepción general de que, aunque el fundamento del diseño debe ser capturado en la documentación, falta una metodología adecuada. Esa premisa, es el punto de partida de su investigación. Para guiar su trabajo realizan la siguiente pregunta: ¿Cuál es un método eficaz para respaldar la provisión de la lógica del diseño en el proceso de diseño de la arquitectura? para responder esta pregunta, estudian el conocimiento existente en arquitectura empresarial y razonamiento de diseño.

Por otra parte Jong et. al. [32] mencionan en su investigación que el fundamento arquitectónico inicia con la identificación de un problema, como la satisfacción de un requisito. Para esta solución, se pueden generar y explorar muchas opciones diferentes, que deben analizarse. Para cada opción, el arquitecto debe analizar los posibles beneficios y debilidades, qué suposiciones son fundamentales, qué restricciones impone o la obstaculizan,

los riesgos involucrados y una decisión final basada en un análisis con múltiples opciones. Para todas las actividades, el arquitecto debe evaluar y reflexionar en qué medida se ha seguido el proceso, y evaluar críticamente lo que podría mejorarse en la lógica y el proceso en sí. El RCC captura cada una de estas actividades de manera cíclica.

El **Método de rationale arquitectónico (RCC)** propuesto se basa en 8 pasos iterativos como se puede ver en la Figura 3.10 cada uno de los ciclos es detallado a continuación.

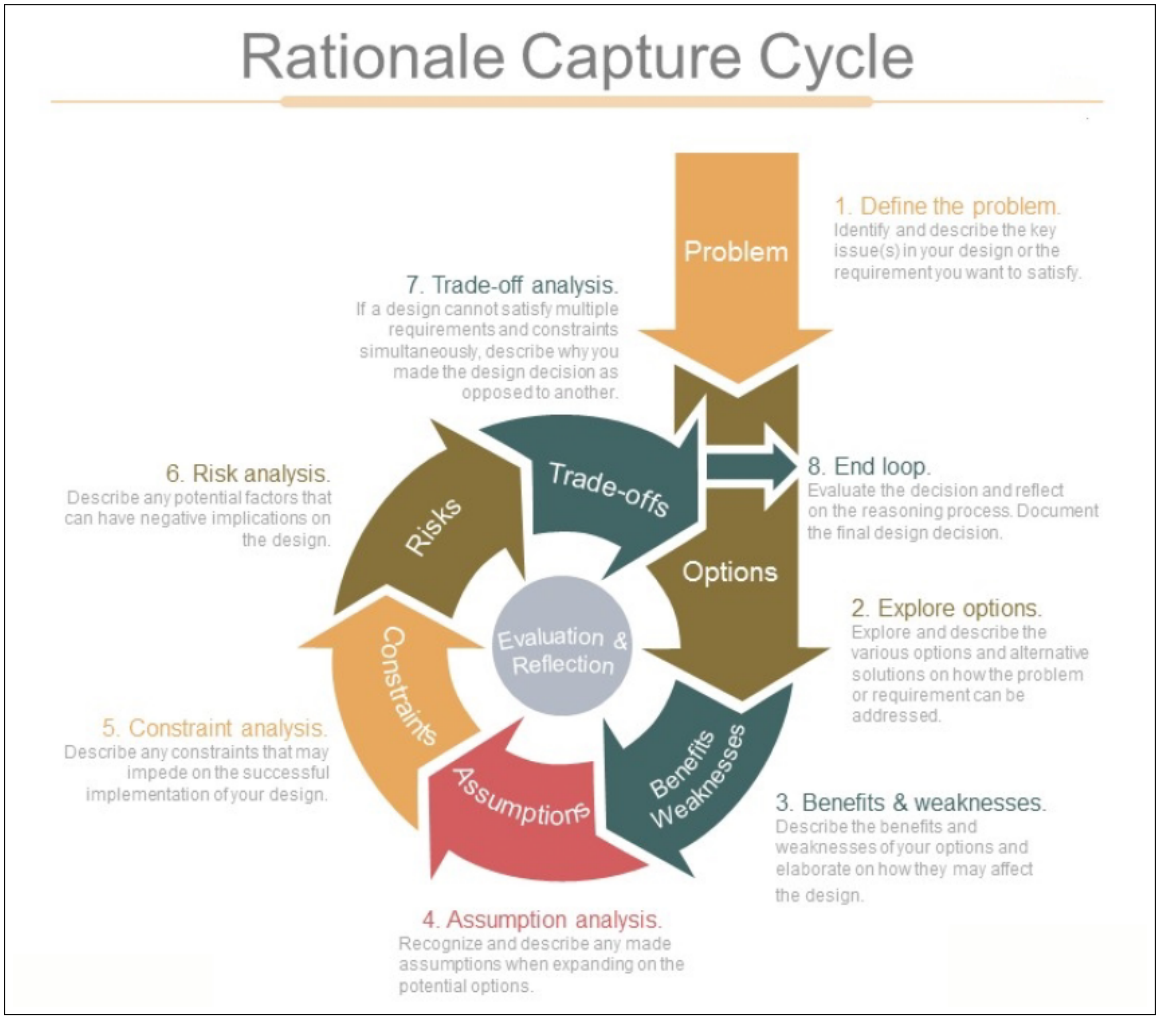


Figura 3.10: Ciclo de captura de rationale (RCC). Tomado de [32]

1. Definir el problema: Identificar y describir los principales requerimientos en su diseño o los requerimientos que se quiere satisfacer.

2. Explorar opciones: Explorar y describir opciones y alternativas de solución, acerca de como el problema o requerimiento puede ser orientado.

3. Beneficios y Debilidades: Describir los beneficios y debilidades de sus opciones y revisar como pueden afectar su diseño.

4. Análisis de suposiciones: Reconocer y describir cualquier suposición hecha al ampliar las posibles opciones.

5. Análisis de restricciones: Describir alguna restricción que puede impedir la implementación exitosa de su diseño.

6. Análisis de riesgo: Describir factores potencial que pueda tener implicaciones negativas sobre su diseño.

7. Análisis de Trade-off: Si un diseño no satisface múltiples requerimientos y restricciones simultáneamente, describe por que tomo la decisión en contra posición a otra.

8. Fin del ciclo: Evaluar la decisión y reflexionar sobre el proceso de razonamiento. Documentar la decisión final de diseño.

El objetivo final de su estudio es proporcionar un medio por el cual los diseñadores de arquitectura puedan emplear sistemáticamente el razonamiento de diseño. Para evaluar su método realizan un experimento controlado con 10 arquitectos experimentados que trabajan en empresa. Los resultados de su experimento muestran que los arquitectos que utilizan el método obtienen un rendimiento significativamente mejor. Al mismo tiempo, el experimento muestra que los participantes no toman mucho más tiempo para producir la arquitectura, contrarrestando la creencia general de que documentar el fundamento arquitectónico es demasiado costoso en tiempo y/o presupuesto.

Aunque los resultados parecen prometedores, el experimento muestra que no todos los arquitectos utilizaron completamente el CCR. Como consecuencia, los resultados son más pequeños de lo que podrían haber sido. Solo 2 arquitectos hicieron un uso completo del ciclo, causando una gran diferencia en la comparación. Se necesita más investigación sobre cómo dicho modelo puede implementarse completamente durante el diseño de la arquitectura. Los autores creen que con más investigación para incorporar activamente el modelo de razonamiento en los marcos de arquitectura, se pueden realizar mejoras adicionales [32].

3.2.6.2. Rationale efectivo del diseño: comprensión de las barreras

J. Horner et al. [47], analizan una serie de barreras que impiden el Rationale en el diseño como mecanismo de advertencia, comunicación y análisis. Las dificultades que mencionan los autores se centran principalmente en limitaciones de tipo: cognitivas, captura, recuperación y uso posterior. Un objetivo del Rationale en el diseño, es transmitir información útil de un diseñador que trabaja en un momento y contexto dado a otro diseñador que trabaja en otro momento y contexto completamente diferentes. También busca facilitar la comunicación entre los diseñadores que trabajan al mismo tiempo. La finalidad de su investigación sobre el rationale del diseño es mejorar la calidad de los diseños. En nuestro trabajo es importante identificar de manera eficaz y clara las dificultades que encuentran los diseñadores al momento de realizar cambios en diseños existentes, lo que tendremos en cuenta para nuestro lenguaje de documentación de rationale.

3.2.6.3. Rationale como subproducto

K. Schneider [78] se refiere al rationale como un activo en ingeniería del software, que se comunica en distintas fases del proyecto, específicamente en el diseño y creación de prototipos. El cual es normalmente olvidado por los diseñadores, debido a su dificultad para ser documentado en las fases mencionadas. En su investigación el autor define el rationale como un enfoque de subproductos que se define a través de siete principios, los cuales identifica al construir dos aplicaciones.

Los principios son:

1. Enfocarse en una tarea del proyecto en la que está emergiendo la lógica.
2. Capturar el rationale en esa actividad, no en una actividad diferente.
3. Poner la menor carga adicional posible en el portador del rationale, pero puede delegar este conocimiento a otras personas.
4. Concéntrarse en guardar una grabación durante la actividad original, posponga la indexación, la estructuración, etc. a una actividad de seguimiento llevada a cabo por otros.
5. Usar herramientas software para registrar y capturar información específica de tareas para la estructuración.

6. Analizar grabaciones, buscar patrones
7. Alentar, pero no insistir en una gestión adicional de rationale.

El autor explica que el rationale aparece cuando se toman decisiones, es decir, en las primeras fases del proyecto, por ejemplo, durante reuniones informales, presentación de diapositivas y discusiones orales. Estas decisiones por lo general son tomadas por expertos técnicos y arquitectos durante la fase de diseño, las cuales se comunican a través de gráficos generales, bocetos de arquitectura y explicaciones orales. Otra fuente interesante de rationale que menciona son los prototipos que generan ideas que se suman al rationale de las decisiones técnicas. Además, los prototipos de demostración obtienen los requisitos y el rationale del cliente. También durante todo el proyecto, los requisitos se negocian, priorizan y reorganizan. En algunas de estas actividades, es necesario propuestas de diseño iniciales o prototipos. Reducir los riesgos del proyecto es una tarea constante en la gestión del proyecto, por tanto los riesgos identificados pueden causar decisiones de diseño. Diferentes partes interesadas en ocasiones están en desacuerdo sobre los requisitos o riesgos, probablemente también en desacuerdo sobre suposiciones y fundamentos más profundos. Se deben encontrar compromisos que irán acompañados de un rationale. Gran parte del rationale mencionado anteriormente reside en los jefes de los participantes del proyecto. Finalmente, el rationale rara vez se documenta.

Finalmente, los autores concluyen que siguiendo los 7 principios, se puede facilitar el desarrollo de herramientas y estrategias que permitan capturar el rationale en cualquier actividad que involucra un proceso de decisión. En nuestra investigación se trata de definir un lenguaje que sienta las bases para una herramienta de representación integrada de las decisiones de diseño y su rationale.

3.2.6.4. Enseñanza de la gestión del rationale en cursos de proyectos ágiles

Kleebaum et al. [55] en su trabajo consideran la administración del rationale como beneficiosa, dado que, apoya la toma de decisiones y evita la vaporización del conocimiento. Reportan una conferencia en donde enseñan, sensibilizan y motivan la administración adecuada del rationale a un grupo de estudiantes que trabaja con enfoques ágiles. En la conferencia, presentan a los estudiantes un modelo de rationale, métodos de captura y explotación, y herramientas de apoyo para la administración de rationale. La conferencia se dicta a estudiantes como parte de un curso de desarrollo ágil en la universidad Técnica de Munich, en donde existen equipos de estudiantes que trabajan en diferentes proyectos de software durante un semestre. Como herramientas de control y seguimiento para los

proyectos utilizan JIRA y una WIKI. El modelo de Rationale que proponen representa el conocimiento con los siguientes elementos: problema, alternativa, pro, con y decisión. En donde se relación entre si para ir dándole forma a un modelo de documentación, por ejemplo, enlazan problema con alternativas y con pro y cont. La herramienta que utilizan se llama ConDec que mediante plugin se enlaza a JIRA. Finalmente, los autores concluyen que los estudiantes comprenden el uso del modelo de rationale y que están motivados para aplicar la gestión del rationale.

En nuestro trabajo, de manera similar buscamos documentar el rationale mediante una herramienta que se basa en un meta-modelo que especifica las bases de un lenguaje de rationale, donde buscamos un énfasis en el rationale de las decisiones, el lenguaje permite expresar elementos como tácticas, patrones y estrategias arquitecturales que impactan la arquitectura de software.

3.2.6.5. Soporte hipermedia para Rationale basado en argumentos

Shum et al. [80] definen un enfoque de argumentación basado en los enfoques de IBIS y QOC. El objetivo principal de su perspectiva se encuentra en capturar en tiempo real, la mediación de las apreciaciones que se generan en las reuniones con todas las partes interesadas en el desarrollo del proyecto. Para mostrar el enfoque propuesto desarrollan un entorno distribuido basado en hipermedia denominado Compendium, éste proporciona un entorno robusto y abierto para la documentación del rationale de diseño y está basado en el enfoque de argumentación IBIS. Este proceso de captura sirve a las necesidades de los interesados para comprenderse mutuamente y saber que se ha escuchado su punto de vista, además soporta la integración de diferente software que generan datos a gran escala. Este enfoque permite documentar información principalmente de reuniones, artefactos generados en estas reuniones e información adicional de otras fuentes.

Sin embargo, esta no es actualmente una herramienta de modelado completa, ya que no se pueden especificar restricciones entre nodos y enlaces: se pueden vincular dos nodos utilizando cualquier tipo de enlace. En nuestro lenguaje de rationale una de las principales características es una mayor expresividad.

3.2.6.6. ¿Cómo discuten los desarrolladores el rationale?

En su trabajo Alkadhi et al. [5], presentan los resultados de un estudio empírico que toma como base los registros de Internet Relay Chat (IRC) en proyectos de Operations

Support Systems (OOS) de código abierto. Su objetivo es investigar cómo los desarrolladores discuten los fundamentos mientras se comunican en los canales IRC, para analizar el potencial de detectar el Rationale en estos mensajes. Para mostrar el enfoque propuesto analizan los mensajes de IRC. Este proceso sirve para proporcionar evidencia cuantitativa de la existencia de Rationale en los mensajes de IRC, la frecuencia de los diferentes elementos de Rationale, identifican que desarrolladores aportan Rationale en los mensajes. También analizan el desempeño de diferentes algoritmos de aprendizaje automático cuando detectan automáticamente el rationale en los mensajes de IRC y los clasifican en elementos de Rationale más específicos: problemas, alternativas, argumentos y decisiones. Este trabajo va de la mano con nuestra investigación ya que nos permite identificar la estructura que debe tener una lenguaje de representación de rationale asociado a las decisiones de diseño arquitectónico.

3.2.6.7. Un modelo para representar el rationale del diseño arquitectónico

Carignano et al. [25] proponen un modelo de rationale que especifican utilizando UML 2.0. El centro de este enfoque se encuentra en representar el rationale generado por los arquitectos durante el diseño arquitectónico, para que persista en el tiempo y pueda recuperarse, analizarse y reutilizarse cuando sea necesario. El modelo incluye conceptos que representan artefactos arquitectónicos, razones, suposiciones, decisiones y el estado de los elementos de razonamiento. Para representar el modelo propuesto, se propone capturar el diseño arquitectónico de un sistema de registro de pedidos a través de dispositivos móviles. Sus funcionalidades son: (i) el vendedor debe registrar los pedidos de los clientes a través de dispositivos móviles; y (ii) el vendedor debe sincronizar fuera de línea los datos de pedidos, productos y clientes con el servidor central. Como resultado, obtienen un conjunto de productos arquitectónicos (objetos, estilos , propiedades y elementos de documentación) y los detalles del rationale asociado con sus definiciones para cada elemento. De manera similar en nuestra investigación buscamos representar el rationale a través de un meta-modelo que especifica sus elementos; con el meta-modelo también buscamos documentar aquel rationale que no se puede colocar directamente en el código utilizando una herramienta que permite de manera gráfica visualizar y documentar el rationale con elementos como: decisiones, estrategias, patrones arquitectónicos, pros, contras, contexto, entre otros.

3.2.6.8. Reutilización del conocimiento de las decisiones de diseño de la arquitectura de software y el rationale dentro de la empresa

Sundaravadivelu et al. [83] presentan un método para documentar el conocimiento arquitectónico de manera estructurada. Los autores siguen un mecanismo de captura basado en una plantilla de arquitectura, que le permite ahorrar tiempo y esfuerzo al arquitecto y las partes interesadas del proyecto. En el enfoque propuesto, busca que el arquitecto no tenga que tomar tiempo adicional para capturar la lógica del diseño. El método propuesto implica dos partes. La primera parte es una herramienta desarrollada para capturar y escribir los datos en un repositorio de conocimiento, la plantilla utiliza meta-etiquetas ocultas que son legibles por un programa que a su ves muestra una plantilla de arquitectura y diseño sencilla. La segunda parte es para la visualización estructurada y el acceso de datos relevantes almacenados en un repositorio central de documentación de arquitectura. En nuestro enfoque representamos el rationale de las decisiones de diseño de manera visual y con anotaciones en el código, con base en un lenguaje que tiene elementos explícitos para capturar rationale, donde se puede revisar al momento de realizar cambios.

3.3. Mecanismos y estrategias para la documentación del rationale arquitectónico

La documentación del rationale arquitectónico como problema fundamental de esta tesis ha sido estudiada por varios investigadores como lo reporta la literatura. La Tabla 3.11 presenta una clasificación de los trabajos relacionados basados en la documentación del rationale arquitectónico de las decisiones de diseño arquitectónico y la información de contexto.

Autor	Mecanismo/Estrategia	Propuesta
Plataniotis et al. [68]	Meta-modelo	Meta-modelo formal que captura el razonamiento y las interrelaciones de las decisiones de diseño.
Dorado et al. [36]	Anotaciones	Documentan el fundamento arquitectónico mediante anotaciones en el código.
Van Der Ven et al. [89]	Proceso	Proceso de construcción del fundamento arquitectónico.

Continua en la siguiente página

Continuación de la página anterior

Autor	Mecanismo/Estrategia	Propuesta
Gilson et al. [40]	Modelo	Enfoque de diseño que combina el modelado de requisitos arquitectónicamente significativos y el modelado de arquitectura.
Che et al. [27]	Metodología	Metodología basada en escenarios.
Dermeval et al. [34]	Plantilla	Plantilla basada en un metamodelo para registrar el fundamento de las decisiones de diseño arquitectónico.
De Jong et al. [32]	Ciclo Iterativo	Ciclo Iterativo, que ayuda a los arquitectos a capturar el fundamento arquitectónico durante el proceso de diseño.
Cleland-Huang et al. [28]	Trazabilidad centrada en decisiones DCT (Herramienta)	DCT establece enlaces de rastreo centrados en una amplia variedad de decisiones arquitectónicas.
Hesseet et al. [45]	DecDoc (Herramienta)	Esta herramienta permite registrar el conocimiento relacionado con las decisiones de diseño que se toman en el transcurso del ciclo de vida del software.
Capilla et al. [24]	ADDSS (Herramienta Web)	Herramienta basada en la Web para administrar las decisiones de diseño de la arquitectura.
Hadar et al. [41]	Documento de especificación de arquitectura abstracta (AAS)	Incluye la información más relevante y actualizada sobre la solución de la arquitectura diseñada para la versión actual, así como el estado de la arquitectura existente del producto.
Rogers et al. [71]	Exploran técnicas para la extracción de fundamento arquitectónico de documentos existentes	Este documento compara dos técnicas para la identificación racional: extracción de texto, usando ontologías y el kit de herramientas WEKA.

Continua en la siguiente página

Continuación de la página anterior

Autor	Mecanismo/Estrategia	Propuesta
Voigt et al. [91]	sprintDoc(Herramienta)	Este documento describe un concepto para una herramienta de documentación ágil.
Van Der Ven et al. [88]	Encuesta	Estudian las creencias que rodean la arquitectura de software. Las creencias varían desde la cantidad de esfuerzo necesario para la documentación de la arquitectura, hasta el tamaño del equipo o las personas responsables de tomar las decisiones arquitectónicas.

Tabla 3.11: Clasificación de trabajos relacionados con la documentación del rationale arquitectónico.

Realizamos un análisis de los estudios para determinar cuáles de estos tenían mayor relevancia para la investigación en curso, es decir, que brinden elementos conceptuales y teóricos para construir el lenguaje de documentación de rationale. La Tabla 3.12. Muestra los estudios que influyeron en la construcción del lenguaje y sus abstracciones más importantes.

En la Tabla 3.12 se muestra una comparativa de las agrupaciones que propone el lenguaje DRML(Decisions and rationale modeling language) con respecto a los elementos de documentación que plantea cada uno de los estudios, en la primera fila vemos las principales abstracciones que el lenguaje DRML agrupa mediante constructos explícitos, la segunda fila muestra los elementos de cada una de las propuestas, en ese orden, comparamos y agrupamos: Rationale (R), Motivation (M), Problem (P), Cause (C); Trade-off (To), Con (Cn), Pro (Pr), Consequence (Cq); Alternative (Alt), Architectural Design Decisions (ADD), Decision (D); Quality Attribute (Qa), Business Rule (BR), Quality Goal (QG); Tactic (T), Architectural Pattern (P) y Strategy (E). En nuestro lenguaje DRML algunos elementos son ciudadanos de primera clase, es decir, existen constructos para modelarlos y los otros conceptos quedan englobados en ellos, aunque no sean explícitos, sin embargo, cuenta con un espacio general que ofrece la posibilidad de describirlos, por ejemplo, los pros y los contras se relatan libremente en un elemento de tipo consecuencia (Cq).

En la revisión de la literatura hallamos distintos tipos de enfoques que buscan docu-

Agrupaciones lenguaje DRML	Rationale	Contexto			Consecuencia				Alternativa	Decisión arquitectónica		Justificación			Táctica	Patrón	Estrategia
	R	M	P	C	T_0	C_n	P_r	C_q	Alt	ADD	D	Q_a	BR	QG	T	P	E
Elementos por el estudio																	
Cleand-Huang et al, 2013	X									X				X			
Van Der Ven et al, 2008		X	X	X	X	X	X	X									
Dermeval et al, 2013								X	X		X						
Hyun & Hurtado, 2019	X	X	X		X	X	X				X	X	X	X	X	X	
Gilson & Englebert, 2011									X		X						
Plataniotis et al, 2015									X		X						
Harrison & Avgeriou, 2010											X	X		X	X	X	
Jansen & Bosch, 2005		X	X	X	X					X	X						
Che, 2014	X				X				X							X	
Lenguaje DRML	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Tabla 3.12: Agrupaciones que propone el lenguaje DRML con respecto a los elementos de documentación que plantea cada uno de los estudios.

mentar el rationale de las decisiones de diseño arquitectónico, por ejemplo, encontramos herramientas, plantillas, anotaciones en el código, modelos, metodologías, plantillas, encuestas y ciclos de captura, muchos basados en meta-modelos, sin embargo, estas propuestas generan modelos de decisiones. Es decir, se centran en generar elementos para documentar la decisión, generando distancia entre el rationale y la decisión. Por otra parte, las propuestas pueden llegar a generar demasiada carga cognitiva al diseñador, esto debido a que incluyen demasiados elementos de documentación, es necesario tener un balance, porque esto puede causar que sea difícil mantener una documentación actualizada en un contexto ágil. El enfoque de documentación del rationale con anotaciones en el código presenta un gran valor, dado que el código es la fuente más confiable de la información a la hora de hacer mantenimiento de software, sin embargo no todas las decisiones y su rationale se puede localizar en un solo lugar del código, dado que decisiones arquitecturales y su rationale pueden impactar muchas partes del sistema, por lo que un modelo que permita documentar en forma explícita y transversal este conocimiento, brindaría una vista más del diseño arquitectónico del sistema que puede resultar más práctico para el mantenimiento incluso que grandes documentos de arquitectura y diseño que se tornan poco confiables en el tiempo [81]. Así que el enfoque de este trabajo de maestría, es complementar las anotaciones de código, las cuales son dos estrategias que pueden integrarse a través del MDE de tal forma que brinden una estrategia práctica y simplificada, particularmente para soportar en lo ágil con foco en la arquitectura. El primer paso sería extender las anotaciones de código para que los lenguajes sean equivalentes semánticamente para facilitar transformaciones MDE M2T (Modelo a texto/código). Así mismo, el lenguaje ofrece los constructos mínimos para integrar las decisiones con el rationale, esto permite modelar una vista integradora que es muy necesaria en el modelado de la arquitectura. Por ejemplo, Kruchten et al. [58] en su enfoque de 4+1 vistas habla de una quinta vista (+1), son los escenarios que permiten evaluar la consistencia entre las vistas, este modelo cumple este rol, de hacer explícitas las decisiones de diseño que son verificables en las demás vistas y el rationale detrás de éstas decisiones.

3.3.1. Discusión

Este trabajo contribuye al área de documentación de la arquitectura, particularmente al enfoque en el cual la Arquitectura de Software es tratarla como un conjunto de decisiones de diseño arquitectónico [81]. Es decir, el trabajo de modelar las decisiones y su rationale, es parte del modelado arquitectónico que se ocupa de la representación, captura, gestión y documentación de las decisiones de diseño tomadas en todo el ciclo de vida del software Kruchten et al. [58]. Diseñar una arquitectura de software es un proceso de toma de

decisiones [59]. Los métodos ágiles cambiaron drásticamente la forma de diseñar una Arquitectura de Software, en proyectos que utilizan métodos ágiles, por ejemplo, Scrum, tomar decisiones de diseño arquitectónico no es responsabilidad de una sola persona, sino de todo el equipo de desarrollo, sumado a esto se centran en la entrega continua del producto, sin embargo, vemos como hacen un esfuerzo en incorporar prácticas de arquitectura en sus procesos [59]. Nuestro enfoque se ajusta de manera apropiada en este contexto ya que permite capturar las decisiones y su rationale considerando el contexto ágil.

A partir de un Modelo Conceptual integrador de decisiones de diseño arquitectónico y su rationale, en este trabajo se propone una especificación a nivel de meta-modelo DRML, el cual va a ser usado como sintaxis abstracta para el modelado de las decisiones y el rational de Arquitecturas de Software específicas. El meta-modelo planteado toma como referencia algunos conceptos y definiciones de los siguientes trabajos:

- La investigación de Plataniotis et al. [68], su principal contribución es un meta-modelo formal que permite especificar el rationale y las interrelaciones de las decisiones de diseño.
- La perspectiva de Dorado et al. [36], donde documentan el rationale mediante anotaciones en el código basado en un modelo conceptual.
- La definición de Van Der Ven et al. [89], proponen unir el rationale y el concepto de una decisión de diseño, combina el rationale con la arquitectura software.
- El enfoque propuesto por Gilson et al. [40], combina el modelado de requisitos arquitectónicamente significativos y el modelado de arquitectura.
- La metodología de Che et al. [27], documenta explícitamente las decisiones de diseño arquitectónico utilizando un enfoque basado en escenarios, que abarca tres vistas de una arquitectura de software, para registrar el conocimiento arquitectónico, e incorpora características centradas en la evolución para gestionar la evolución de las decisiones, para reducir la evaporación del conocimiento arquitectónico.
- La plantilla propuesta por Dermeval et al. [34], donde documenta el rationale de las decisiones de diseño arquitectónico, la documentación de las decisiones permite la evaluación de alternativas de diseño arquitectónico cuando los requisitos evolucionan o cuando se diseñan nuevas alternativas. Además, el meta-modelo proporciona una relación entre los requisitos y los fragmentos de diseño arquitectónico, lo que facilita el mantenimiento de la trazabilidad entre el problema y la solución.

Capítulo 4

Lenguaje Basado en Anotaciones DRML (Decisions and Rationale Modeling Language)

4.1. Introducción

Este capítulo presenta un modelo de dominio específico para el lenguaje que permite la representación del rationale de las principales decisiones de diseño arquitectónico, orientado a proyectos que utilizan métodos ágiles con prácticas de Arquitectura Software en sus procesos de desarrollo de software.

El modelo de dominio especifico del lenguaje, busca mediar entre las exigencias de los postulados del manifiesto ágil y las especificaciones mínimas que debe tener la documentación de la Arquitectura Software, por lo que se caracteriza por ser simple, liviano y adaptable para que las empresas puedan aplicarlo de forma efectiva. Es un modelo listo para ser aplicado en proyectos reales en contextos livianos.

Para especificar el lenguaje con su modelo de dominio, se define un modelo conceptual que representa de forma abstracta los componentes más destacados de una decisión de diseño arquitectónico y su rationale, se hace el desarrollo de un prototipo en eclipse para dar solución al problema mediante la creación de un meta-modelo denominado DRML usando el meta-metamodelo ecore de EMF. Posteriormente, la construcción de la herramienta

DRMLTool gráfica basada en el meta-modelo planteado y con apoyo en Eclipse Modeling Framework (EMF) y Graphical Modeling Framework (GMF) para la representación de un modelo que posteriormente se pueda instanciar a cualquier plataforma. Finalmente, con base en los constructos definidos por el meta-modelo se crea la herramienta RADAR, un plugin escrito en el lenguaje java que permite crear anotaciones de código.

Las secciones siguientes describen la definición de la solución propuesta, un modelo conceptual, el lenguaje incluyendo sus elementos y las anotaciones de código. La Figura 4.1 muestra la construcción de la propuesta.

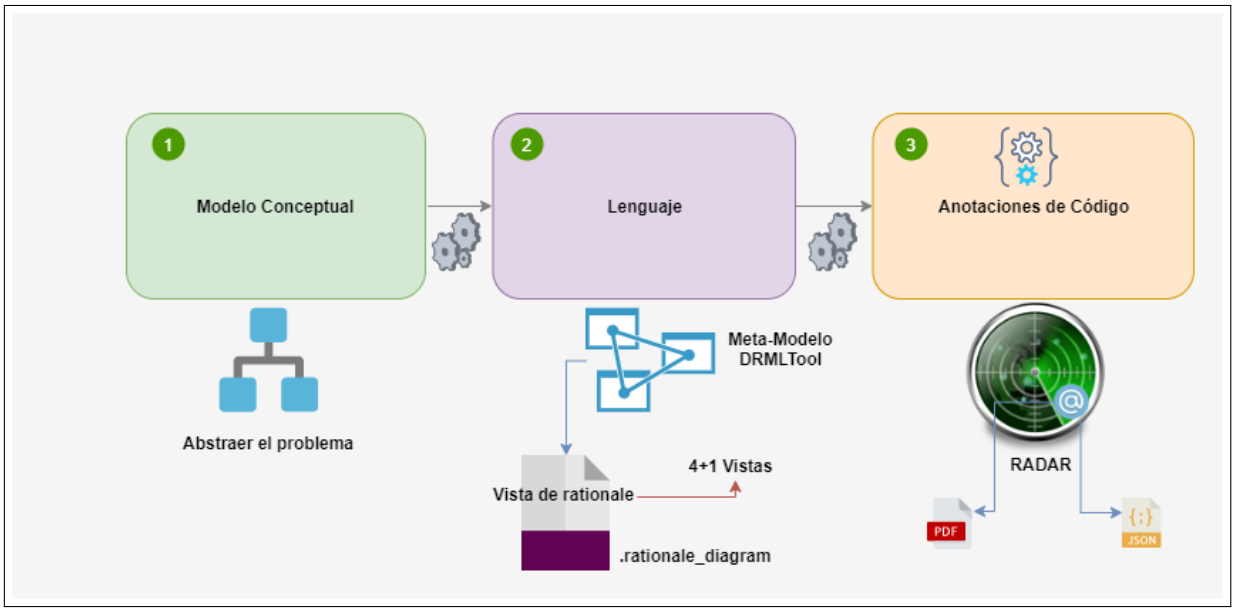


Figura 4.1: Construcción Propuesta

4.2. Definición de la solución propuesta

Previamente en Dorado et al. [36], nosotros intentamos especificar las decisiones de diseño en el código fuente, sin embargo, el código es una sola perspectiva del software de las muchas posibles. Una decisión de diseño que involucre un atributo de tipo ejecución, como es el desempeño, ¿Dónde iría?, por ejemplo, qué afecta el desempeño, la forma en que se instalen los componentes, si estos componentes tienen constante comunicación, si van a estar en una misma máquina o servidor, pero, si están en servidores geográficamente dispersos, hay que considerar otros aspectos, debe hacerse repetición, envió de

información, paquetes, encriptación de información, aspectos que hacen lento el proceso, lo cual no es localizable en un punto del código. Entonces, el código tiene ciertas limitaciones. En lo ágil una de las prácticas es tener código auto-documentado, por ejemplo, que se pueda hacer usando anotaciones. El problema con la Arquitectura Software es que no todo se puede especificar a ese nivel, por lo tanto, necesitamos una especificación simple, liviana y adaptable que se pueda agregar por separado como un artefacto más de arquitectura y se pueda asociar con la documentación nueva o existente. Aquí es donde entra nuestro lenguaje, al tratarse de un lenguaje lo podemos utilizar a nivel conceptual como a nivel de anotaciones, nos basamos en los estudios previos de Dorado et al. [36] donde se ha dicho que las anotaciones de código tienen un impacto positivo sobre la comprensión de la Arquitectura Software y su rationale en la realización de cambios arquitectónicos. Entonces basados en este trabajo y la literatura complementamos el modelo de anotaciones con nuestro modelo de dominio especifico que parte de un Modelo Conceptual que luego se reifica para dar paso al meta-modelo del lenguaje. Usamos el meta-modelo para construir las anotaciones de código.

4.3. Modelo Conceptual

Con base en la revisión de la literatura y trabajos relacionados, formalizamos el Modelo Conceptual de rationale, que nos permite representar con elementos abstractos la interacción entre el rationale y las decisiones de diseño arquitectónico al momento de afectar la Arquitectura de Software o un componente bajo descomposición, ver la Figura 4.2

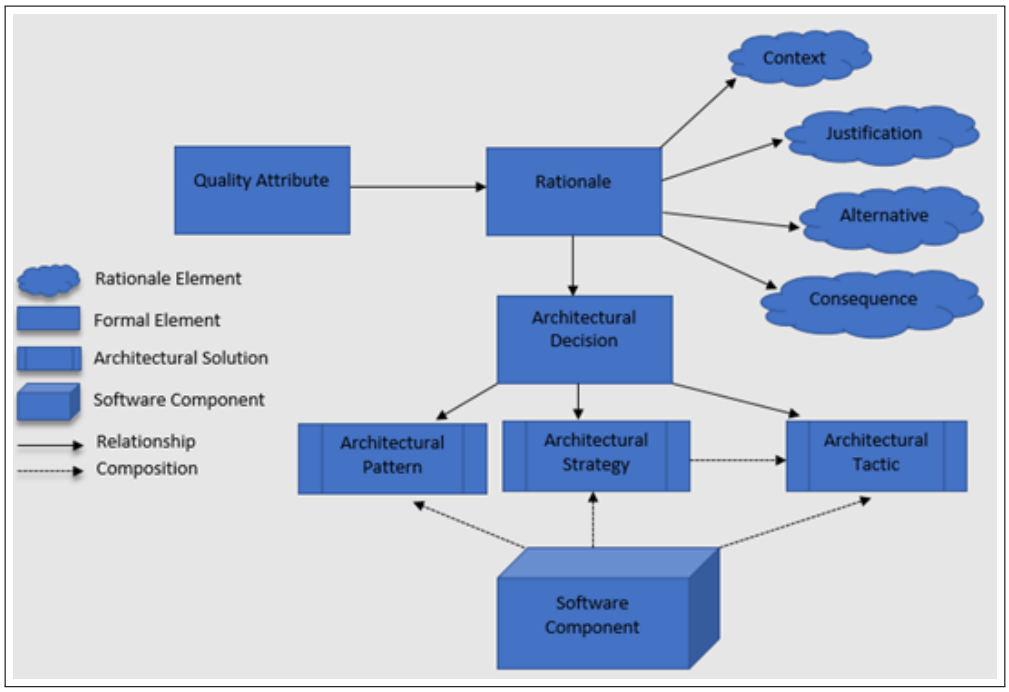


Figura 4.2: Modelo Conceptual

A continuación se definen los elementos del modelo conceptual:

- *Quality Attribute(Formal Element):* Corresponde a los escenarios de calidad consolidados y priorizados.
- *Rationale(Formal Element):* Representa de manera explicita los elementos que rodean a una justificación de una decisión de diseño arquitectónico.
- *Context(Rationale Element):* Contexto bajo el cual se toma una decisión de diseño arquitectónico.
- *Justification (Rationale Element):* Es la justificación de una determinada decisión de diseño arquitectónico
- *Alternative(Rationale Element):* Representa un conjunto de posibles soluciones de la necesidad a resolver.
- *Consequence (Rationale Element):* Son las posibles consecuencias de tomar una determinada decisión de diseño arquitectónico.

- *Architectural Decision (Formal Element):* Corresponde a la decisión de diseño arquitectónico final.
- *Architectural Pattern(Architectural Solution):* Son los patrones arquitectónicos aplicados que cumplirán con los atributos de calidad.
- *Architectural Strategy (Architectural Solution):* Corresponde al conjunto de decisiones de diseño que debe tomar el arquitecto para poder lograr cada uno de los atributos de calidad, es decir, estrategias arquitectónicas.
- *Architectural Tactic (Architectural Solution):* Las estrategias estan conformadas por unas decisiones concretas que hacen parte de nuestro enfoque arquitectónico que se llaman tácticas de arquitectura.
- *Software Component(Software Component):* Es el componente software que sera afectado por la decisión de diseño arquitectónico.

4.4. Lenguaje

4.4.1. Meta-modelo DRML

El meta-modelo propuesto, especificado en ecore se presenta en la Figura 4.3 donde se visualizan los elementos base del lenguaje de modelado para documentar el rationale arquitectónico de las principales decisiones de diseño arquitectónico. El meta-modelo por naturaleza está compuesto de diez meta-clases instancias de la meta-meta clase EClass en ecore, las cuales son: ArchitecturalRationale, Rationale, ArchitecturalDecision, Justification, Context, ArchitecturalTactic, ArchitecturalPattern, ArchitecturalStrategy y Consequence todos están relacionados con la meta-clase Rationale, puesto que la meta-clase Rationale puede tener cero o muchas meta-clases de los anteriores, la meta-class Rationale hereda de la meta-class ArchitecturalRationale y todas las meta-clases están contenidas en la meta-class ArchitecturalRationale, esta relación es obligatoria en todo meta-modelo pues es quien representara el contenedor de las demás meta-clases, en este lugar es donde se desplieguen los módulos a modelar, es decir, donde estarán contenidos, es necesario aclarar que la meta-clase ArchitecturalRationale solo almacenará cero o una instancia de la meta-clase Rationale, esto tiene sentido pues no puede haber más de una instancia de la meta-clase Rationale en un modelo de dominio específico, ya que este a su vez almacena los elementos de documentación tales como (Justificacion, Context,

Consequence, Alternative). Por último, cada meta-clase tiene sus propios atributos que la compone, ver la Figura 4.3.

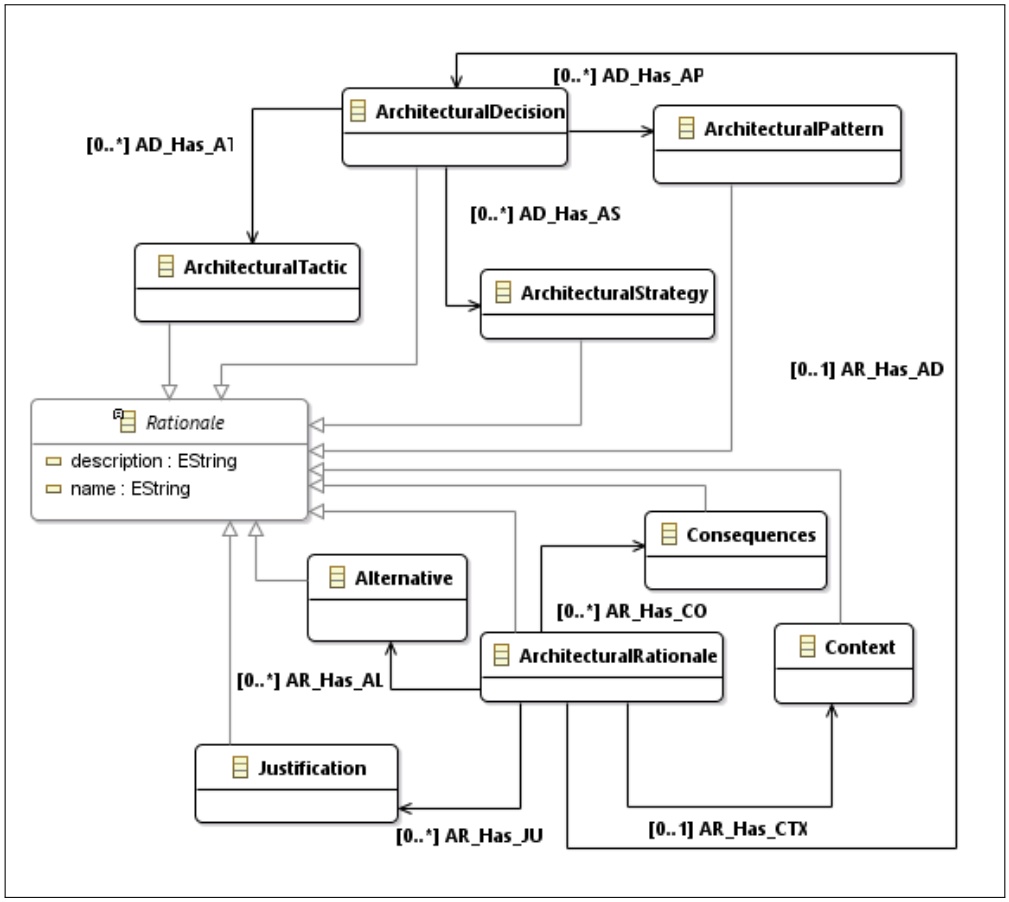


Figura 4.3: Meta-modelo DRML de Rationale Arquitectónico.

Rationale

Superclase: $<<$ *metaclass* $>>$ *"Element"*
Descripción: *"Rationale"* es una generalización abstracta que representa los elementos de documentación. Atributos:

"Description: String" característica que permite especificar la descripción del fundamento arquitectónico.

"name: String" característica que permite especificar el nombre del fundamento arquitectónico.

Semántica: *"Rationale"* representa una estructura que provee a sus subclases, características, que permiten poseer campos para gestionar información importante para el fundamento arquitectónico, en la identificación del elemento de documentación, es decir, atributos tales como nombre ("name") y descripción ('Description").

Notación Gráfica: Ninguna.

ArchitecturalRationale

Superclase: *"Rationale"*

Descripción: *"ArchitecturalRationale"* es una clase que relaciona los elementos que se encuentran dentro de la justificación de la decisión de diseño arquitectónico que se desea documentar.

Propiedades Asociadas:

"AD_Has_AR: El rationale tiene una decisión arquitectónica". Está asociación de referencia representa que cada decisión tiene un fundamento asociado.

"AR_Has_AL: El rationale tiene alternativa". Está asociación de referencia representa que el fundamento tiene una o mas alternativas.

"AR_Has_JU: El rationale tiene justificación". Almacena un conjunto de justificaciones que se relacionan al fundamento.

"AR_Has_CO: El rationale tiene consecuencia". Está asociación representa que cada decisión de diseño arquitectónico tiene una posible consecuencia con sus pros y con-tras.

"AR_Has_CTX: El rationale tiene contexto". Esta asociación representa que el fundamento de una determinada decisión de diseño arquitectónico se lleva a cabo bajo un contexto o situación en especifico.

Semántica: *"ArchitecturalRationale"* representa una estructura de almacenamiento que permite organizar y agrupar un conjunto de elementos que permiten documentar la justificación de una determinada decisión de diseño arquitectónico, basándose en el nivel de detalle del diseñador, respecto a las particularidades de un proyecto o componente software.

Notación Gráfica: 

Alternative

Superclase: *"Rationale"*

Descripción: *"Alternative"* es una estructura que permite definir un elemento del *Rationale* con su respectivo nombre y descripción.

Semántica: *"Alternative"* es la representación de un conjunto de alternativas al tomar una decisión de diseño arquitectónico.

Notación Gráfica: 

Justification

Superclase: *"Rationale"*

Descripción: *"Justification"* es una estructura que permite definir un elemento del *Rationale* con su respectivo nombre y descripción.

Semántica: *"Justification"* es la representación de un conjunto de justificaciones al tomar una decisión de diseño arquitectónico.

Notación Gráfica: 

Consequence

Superclase: *"Rationale"*

Descripción: *"Consequence"* es una estructura que permite definir un elemento del *Rationale* con su respectivo nombre y descripción.

Semántica: *"Consequence"* representa las consecuencias de una decisión de diseño arquitectónico.

Notación Gráfica: 

Context

Superclase: *"Rationale"*

Descripción: *"Context"* es una estructura que permite definir un elemento del *Rationale* con su respectivo nombre y descripción.

Semántica: *"Context"* representa el contexto en que una posible solución es planteada.

Notación Gráfica: 

ArchitecturalDecision

Superclase: *"Rationale"*

Descripción: *"ArchitecturalDecision"* es una clase que relaciona los elementos concretos de la decisión con el rationale arquitectónico y su propiedades.

Propiedades Asociadas:

"AD_Has_AT: La decisión arquitectónica tiene una táctica arquitectónica". Está asociación de referencia representa que cada decisión tiene asociado un conjunto de tácticas.

"AD_Has_AS: La decisión arquitectónica tiene una estrategia arquitectónica". Está asociación de referencia representa las estrategias de arquitectura.

"AD_Has_AP: La decisión arquitectónica tiene un patrón arquitectónico". Almacena el conjunto de patrones arquitectónicos planteados en la decisión de diseño.

Semántica: *"ArchitecturalDecision"* representa una estructura de almacenamiento que permite organizar y agrupar un conjunto de decisiones de diseño arquitectónico, basándose en el nivel de detalle del diseñador.

Notación Gráfica: 

ArchitecturalTactic

Superclase: *"Rationale"*

Descripción: *"ArchitecturalTactic"* es una estructura que permite definir un elemento de *ArchitecturalDecision* con su respectivo nombre y descripción.

Semántica: *"ArchitecturalTactic"* representa las decisiones concretas que hacen parte del enfoque arquitectónico, que se encargan de dar control a cada uno de los atributos de

calidad.

Notación Gráfica: 

ArchitecturalStrategy

Superclase: *"Rationale"*

Descripción: *"ArchitecturalStrategy"* es una estructura que permite definir un elemento de *ArchitecturalDecision* con su respectivo nombre y descripción.

Semántica: *"ArchitecturalStrategy"* representa el conjunto de decisiones de diseño arquitectónico que el arquitecto debe tomar para satisfacer todos los atributos de calidad importantes en la arquitectura.

Notación Gráfica: 

ArchitecturalPattern

Superclase: *"Rationale"*

Descripción: *"ArchitecturalPattern"* es una estructura que permite definir un elemento de *ArchitecturalDecision* con su respectivo nombre y descripción.

Semántica: *"ArchitecturalPattern"* representa el conjunto de patrones arquitectónicos que cumplen con un atributo de calidad en específico.

Notación Gráfica: 

4.4.2. Tecnología de Desarrollo DRMLTool

El desarrollo del prototipo DRMLTool se basó en el meta-modelo DRML donde los modelos abstractos son creados y transformados sistemáticamente en implementaciones concretas. La base del enfoque MDE es la implementación del modelo de dominio conceptual el cual se ha definido a un nivel de abstracción muy alto, para la generación de este meta-modelo existen varios lenguajes de modelado entre los que se encuentran KM3, MOF y Ecore. Este meta-modelo es procesado por distintas herramientas para

la generación semi-automatizada de modelos que ayudarán en la generación de editores gráficos. Por esta razón, DRMLTool fué construido mediante Eclipse Modeling Framework y Graphical Modeling Framework, dado que la naturaleza de nuestra propuesta se fundamenta en el meta-modelo DRML y Ecore satisface esta pretensión.

4.4.2.1. Eclipse Modeling Framework

El proyecto EMF [1] es un marco de modelado y una instalación de generación de código para crear herramientas y otras aplicaciones basadas en un modelo de datos estructurados. A partir de una especificación de modelo descrita en XMI, EMF proporciona herramientas y soporte de tiempo de ejecución para producir un conjunto de clases Java para el modelo, junto con un conjunto de clases de adaptador que permiten la visualización y la edición del modelo basado en comandos y un editor básico.

EMF (core) es un estándar común para modelos de datos, en el que se basan muchas tecnologías y marcos. Esto incluye soluciones de servidor, marcos de persistencia, marcos de interfaz de usuario y soporte para transformaciones.

Para el uso de Ecore en Eclipse es esencial tener instalado el plugin de EMF (Eclipse Modeling Framework), este plugin provee básicamente dos herramientas para construir un modelo basado en ecore, una el Ecore Model que es un editor manual que funciona en un estilo de árbol de navegación para la creación del modelo basado en ecore, la otra es el Ecore Diagram siendo este un editor gráfico similar a las herramientas gráficas para la creación de diagramas de clases UML. Cualquiera de las dos formas que se utilice para crear el diagrama basado en ecore, genera un fichero XMI [2] (XML Metadata Interchange) que es una especificación para el intercambio de diagramas, en este caso se utilizará Ecore Model.

4.4.2.2. Graphical Modeling Framework

Graphical Modeling Framework (GMF) [3] es un marco de trabajo de Eclipse que permite crear un editor gráfico, sobre un metamodelo especifico, siguiendo patrón arquitectóni-

[1] EMF: Eclipse Modeling Framework. Más información puede encontrarse en: http://www.eclipse.org/modeling/emf/

[2] https://www.eclipse.org/modeling/emf/

[3] GMF: Graphical Modeling Framework. Más información puede encontrarse en: https://www.eclipse.org/modeling/gmp/

co Modelo-Vista-Controlador (MVC). GMF proporciona componentes e infraestructura para el desarrollo de editores gráficos basado en EMF (Eclipse Modeling Framework) y GEF (Graphical Editing Framework). GMF utiliza seis archivos específicos para la generación del editor gráfico, el modelo del dominio (Domain Model) el cual es el metamodelo principal que contiene toda la información de las clases y objetos para la creación del editor gráfico, la generación de este metamodelo es posible mediante varios tipos de metamodelos diferentes: Código Java con anotaciones, Modelo Ecore, Modelos UML, Esquemas XML. El Modelo de Dominio Generado (Domain Gen Model - .genmodel) es el metamodelo usado para la generación del código del modelo de dominio. El Modelo de Definición Gráfica (Graphical Def Model - .gmfgraph) es el metamodelo usado para definir los elementos gráficos para el modelo de dominio, en este archivo es posible definir que clases del metamodelo será usadas como nodos y enlaces dentro del editor gráfico. El Modelo de Definición de Herramientas (Tooling Def Model - .gmftool) es usado para definir la paleta de herramientas que será usado en el editor gráfico. El Modelo de Mapeo (Mapping Model - .gmfmap) es usado para relacionar el modelo de Dominio, el Modelo de Definición Gráfica y el Modelo de Herramientas. Por último el Modelo de Generación de Diagrama del Editor es usado para generar el editor gráfico GMF adicional al código EMF generado por el archivo que contiene el Modelo de Dominio Generado (.genmodel).

4.4.3. Construcción del editor para el modelo

Para esta fase se empleó, Graphical Modeling Project (GMP) [4] usando Eclipse Luna. Los pasos para la construcción del modelo Gráfico se visualiza en la Figura 4.4. Un tutorial para el desarrollo de herramientas de este tipo se encuentra en su Web oficial [5].

[4]https://www.eclipse.org/modeling/gmp/
[5]https://wiki.eclipse.org/GraphicalModelingFramework

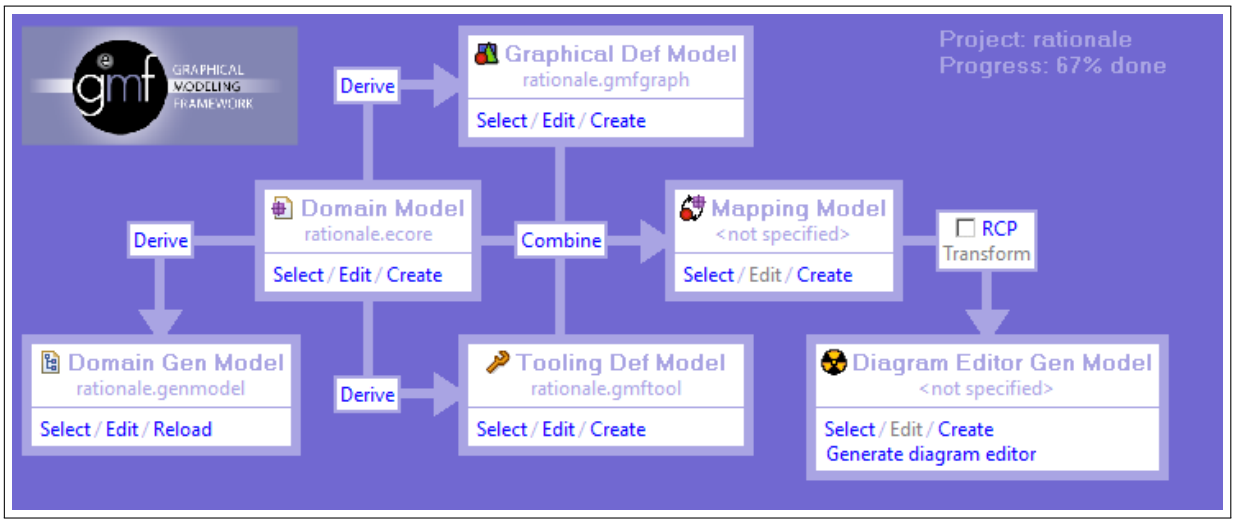


Figura 4.4: GMF Dashboard.

Según el dashboard de la Figura 4.4 lo primero que se debe crear es el Domain Model, este corresponde al meta-modelo descrito en la sección anterior. El siguiente paso es crear el Domain Gen Model, que es un modelo que permite transformar automáticamente el modelo ecore a código fuente. El código se genera aplicando patrones de transformación. El resultado es un conjunto de clases java, que serán utilizadas más adelante en la herramienta DRMLTool.

A continuación, se crea el Graphical Def Model, este es usado para definir las figuras, nodos, conexiones, etc. El resultado es un archivo con la siguiente estructura; un Canvas (lienzo) en la raíz con una galería de figuras base que contiene elementos de Rectángulos, Etiquetas y Conexiones de Polilineas. Estas son usadas por el correspondiente elemento Nodo, Etiqueta del diagrama y Conexión para representar los temas del domain model.

El siguiente paso es la creación del Tooling Def Model, este es usado para especificar la paleta (Pallete) de herramientas de creación, acciones, etc, para los elementos gráficos. Allí existe un elemento en el nivel superior “Tool Registry”, en el que se encuentra una paleta (Palette). La “Palette” contiene un “Tools Group” con elementos de tipo “Creation Tool” para los nodos tema y conexiones para elementos de subtemas.

El Mapping Model es el siguiente paso en el dashboard, el Mapping Model combina los tres modelos: Domain Model, Graphical Def Model y Tooling Def Model.

Finalmente, el último paso es la elaboración del Diagram Editor Gen Model, en este punto

se establecen las propiedades para la generación de código, similar al EMF genmodel. A partir de este modelo se obtiene un plugin para eclipse que contiene la herramienta DRMLTool construida. La apariencia de la herramienta se muestra en la Figura 4.5.

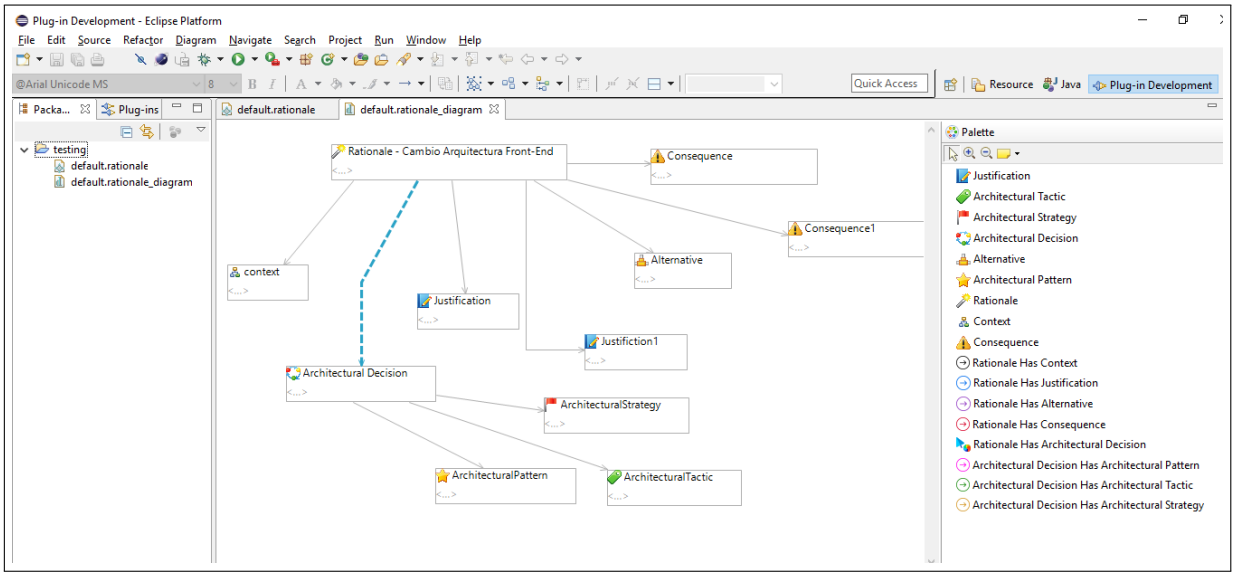


Figura 4.5: Herramienta para modelar el rationale de las principales decisiones de diseño arquitectónico.

4.4.4. Diagrama de Casos de Uso del Editor Gráfico DRMLTool

En esta sección se explica la funcionalidad del editor gráfico DRMLTool, descrito a través del diagrama de casos de uso de la Figura 4.6, que explica la funcionalidad del editor.

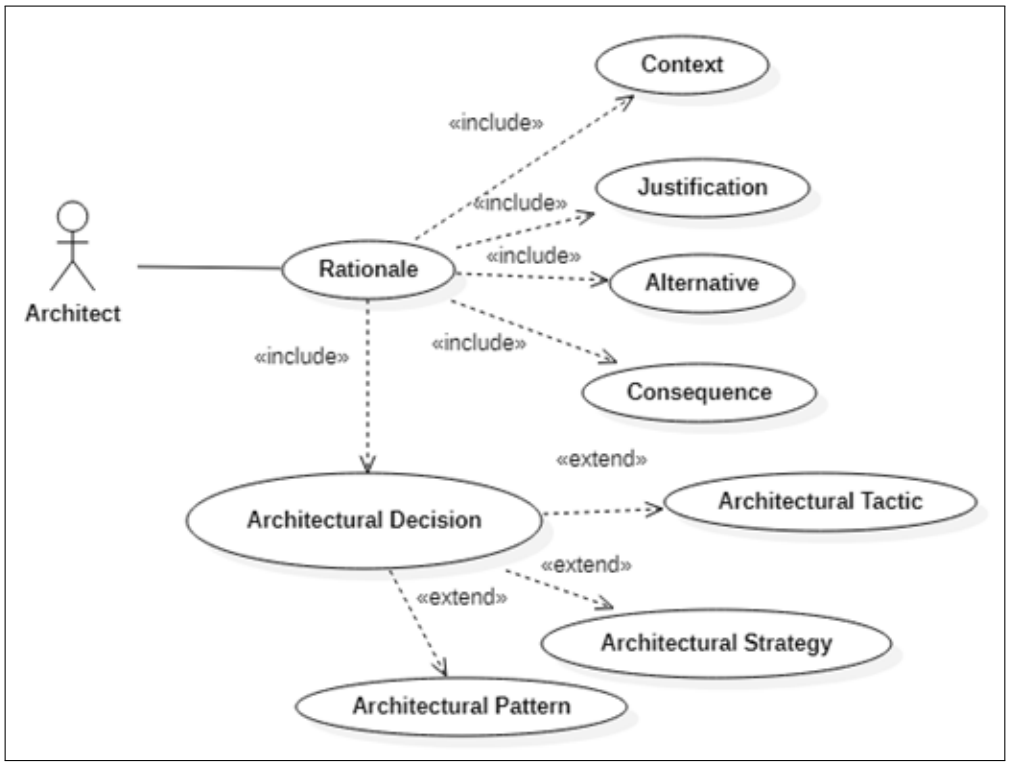


Figura 4.6: Diagrama de Casos Uso

DRMLTool es una herramienta dirigida a arquitectos de software (diseñadores de la arquitectura) y a usuarios que participan activamente del diseño y ejecución de un proyecto de desarrollo software. El arquitecto de software tiene la función de diseñar y modelar la arquitectura y establecer guías que permitan trasmitir el conocimiento de forma explícita de las decisiones finales que permitan la comprensión de la arquitectura. El diseñador de la arquitectura debe definir el rationale arquitectónico a través del caso de uso Rationale. Así mismo, debe colocar información de un contexto específico, a través del caso de uso Context. Indicar las justificaciones que considere pertinentes, a través del caso de uso Justification. Las alternativas que consideró, mediante el caso de uso Alternative. Mencionar las consecuencias de optar por una decisión en particular, con el caso de uso, Consequence. Finalmente, a través del caso de uso ArchitecturalDecision, que a su vez incluye ArchitecturalTactic, ArchitecturalStrategy y ArchitecturalPattern documenta la decisión final. Documentar las decisiones de diseño y su rationale es una tarea sencilla que puede ser realizada por el desarrollador dentro del editor gráfico, sin embargo, se considera que esta tarea solo debe ser hecha por diseñadores con experiencia debido al bagaje que poseen, puesto que se debe hacer un análisis de cómo se relacionan las estrategias, tácticas y patrones del contexto modelado.

4.4.5. Modelado del Rationale con DRMLTool

El funcionamiento de la herramienta se describe a continuación. Para crear los módulos basta con arrastrar los nodos de la "Pallete" al área de trabajo, rellenar los campos y conectarlos respetando las siguientes reglas:

- El elemento "Rationale" tiene como mínimo una "ArchitecturalDecision" y máximo una "ArchitecturalDecision". Y se denota por una flecha punteada de color azul. Figura 4.7.

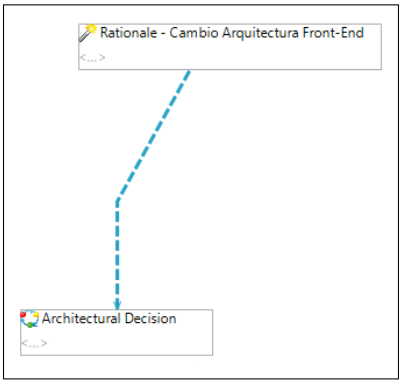


Figura 4.7: Rationale tiene Architectural Decision.

- El elemento "Rationale" tiene cero o un solo "Context". Figura 4.8.

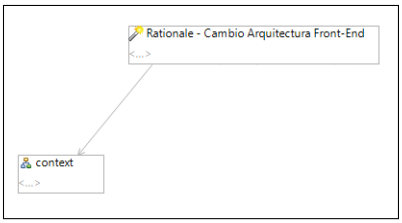


Figura 4.8: Rationale tiene Context.

- El elemento "Rationale" tiene cero o muchos "Consequence". Figura 4.9.

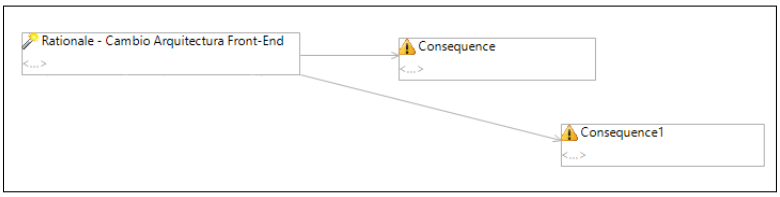


Figura 4.9: Rationale tiene Consequence.

- El elemento "Rationale" tiene cero o muchos "Justification". Figura 4.10.

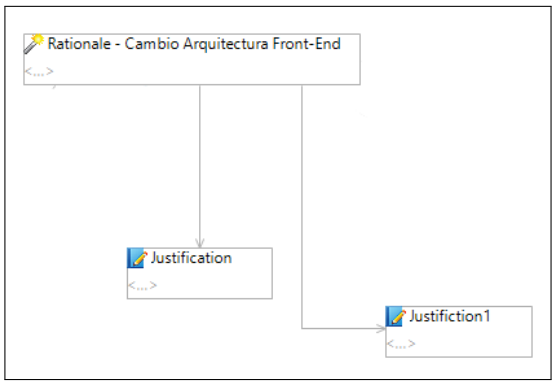


Figura 4.10: Rationale tiene Justification.

- El elemento "Rationale" tiene cero o muchos "Alternative". Figura 4.11.

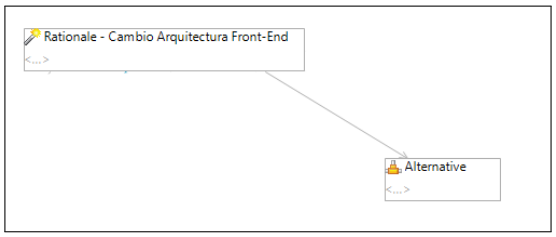


Figura 4.11: Rationale tiene Alternative.

- El elemento "ArchitecturalDecision" tiene cero o muchos "ArchitecturalPattern". Figura 4.12.

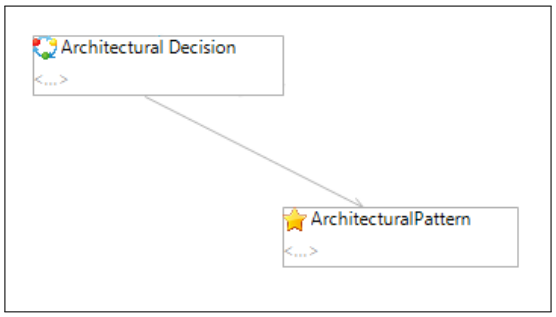


Figura 4.12: ArchitecturalDecision tiene ArchitecturalPattern.

- El elemento "ArchitecturalDecision" tiene cero o muchos "ArchitecturalTactic". Figura 4.13.

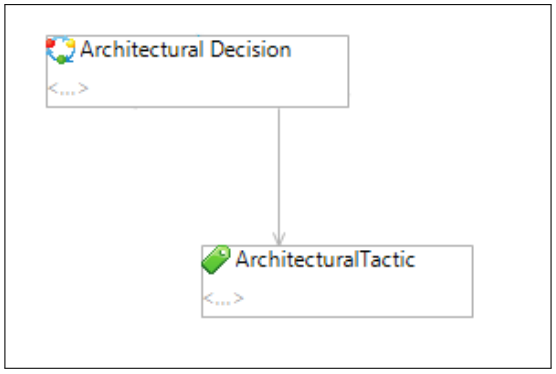


Figura 4.13: ArchitecturalDecision tiene ArchitecturalTactic.

- El elemento "ArchitecturalDecision" tiene cero o muchos "ArchitecturalStrategy". Figura 4.14.

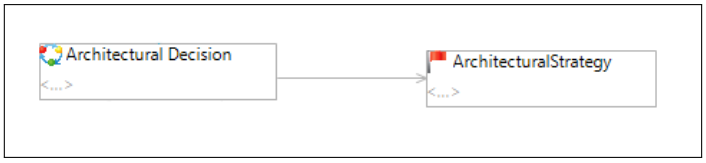


Figura 4.14: ArchitecturalDecision tiene ArchitecturalStrategy.

En resumen, hasta el momento se ha hablado de cómo se creó el modelo del rationale, basado en un meta-modelo generado y que a su vez se estructura sobre EMF(Ecore) [6].

[6]https://www.eclipse.org/modeling/emf/

4.5. Características del Lenguaje

- **Liviano:**

  El lenguaje con su modelo de dominio combina la idea de una documentación liviana para organizaciones de software pequeñas y medianas, planteando un lenguaje sólo con los elementos necesarios que permita documentar el rationale de las principales decisiones de diseño arquitectónico, pero buscando disminuir el trabajo pesado asociado a las actividades de los arquitectos de software y diseñadores a la hora de documentar, trazar y mantener la documentación de la arquitectura.

- **Simple:**

  El lenguaje plantea un conjunto de elementos y relaciones mínimas, para lograr una especificación de valor del rationale arquitectónico, incluyendo y extendiendo los elementos de proceso más comunes encontrados en la literatura científica y los elementos más usados por los arquitectos de software.

- **Adaptable:**

  Cada entidad de desarrollo y cada proyecto, es diferente, el lenguaje abstrae los elementos para cubrir los principales escenarios posibles para documentar un cambio de diseño arquitectónico.

4.6. Anotaciones de Código

La implementación del modelo de anotaciones de código, se hace en base a la propuesta de Dorado et al. [36] y el meta-modelo DRML, que provee un mecanismo simple para documentar el rationale de las decisiones de diseño que se toman sobre un componente software, así como también los atributos de calidad, patrones satisfactorios, las estrategias arquitectónicas asociadas que están conformadas por unas decisiones concretas que hacen parte del enfoque arquitectónico que se llaman tácticas de arquitectura.

En la Figura 4.15 se muestra la declaración del modelo de anotaciones que permite registrar los elementos del rationale y la decisión de diseño arquitectónico.

Rationale.java

```java
package org.radar.bo.annotation;

import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PACKAGE;
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

@Documented
@Retention(RUNTIME)
@Target({ TYPE, METHOD, PACKAGE })

public @interface Rationale {
    String id() default "";

    String[] context() default {};

    String[] justification() default {};

    String[] consequence() default {};

    String[] alternative() default {};

    String[] decision() default {};

    String[] pattern() default {};

    String[] tactic() default {};

    String[] strategy() default {};
}
```

Figura 4.15: Anotaciones de Código

A continuación, se especifica cada uno de los atributos de la anotación. Nótese que los elementos se dividen en atributos de rationale y de decisión. Los atributos son un conjunto de listas.

1. **Atributos de rationale (Opcionales):**
    - context (opcional / recomendado): Contexto en el que se toma la decisión.
    - justification (opcional / recomendado): Lista de justificaciones de la decisión.
    - consequence (opcional / recomendado): Lista de consecuencias al tomar una decisión.

- alternative (opcional / recomendado): Lista de alternativas que se consideraron para una decisión.

2. **Atributos de decisión (Opcionales):**
   - decision (opcional / recomendado): Lista de decisiones que se toman para alcanzar los atributos de calidad.
   - pattern: (opcional / recomendado): Lista de patrones satisfactores y/o decisiones de diseño que se asocian a una táctica.
   - tactic: (opcional / recomendado): Lista de tácticas que se asocian con los atributos de calidad.
   - strategy: (opcional / recomendado): Lista de estrategias que agrupan tácticas de arquitectura.

Para visualizar el uso de la anotación de código fuente, en el momento que se usa, se plantea un caso con requisitos funcionales y no funcionales.

Caso: Suponga que en su empresa le plantean el siguiente escenario, se requiere la integración entre el sistema financiero y de salud para obtener los datos de las facturas auditadas por parte de los médicos y posteriormente aplicarlas en la aplicación contable.

Consideraciones:

- Al tratarse de facturación en salud es información sensible, es importante contar con seguridad.
- No solo el sistema financiero necesita acceder a esta información, otros sistemas también, es decir, interoperabilidad.
- El volumen de información (Factura auditadas) es alto, se debe tener un buen rendimiento y procesamiento simultaneo.

En la Figura 4.16 se muestra cómo se visualiza la anotación en el código fuente con la solución al caso planteado:

```
3  import org.radar.bo.annotation.Rationale;
4
5  @Rationale(
6          id = "1.0",
7          context = {"API REST de facturación, Conexión a proyecto con BD firebird"},
8          justification = {"Spring boot brinda características como: balanceo de carga,"
9                  + " servidor de descubrimiento Eureka Server, GateWay Zuul",
10                 "Spring Security provee autenticación basada en tokens"},
11         consequence = {"EJBs y NodeJs no brindan herramientas para balanceo de carga y "
12                 + "trazabilidad distribuida se deben implementar"},
13         alternative = {"Spring Boot", "EJBs", "NodeJs",
14                 "Para el broker de mensajeria tambien se puede usar rabbitmq, sin embargo, "
15                 + "repite los mensajes revisar"},
16         decision = {"Implementar Microservicio en spring boot, rendimiento, seguridad, interoperabilidad",
17                    "Esta desplegado en un contenedor docker port:80"},
18         pattern = {"Patrón Cliente Servidor,", "N-Tiers", "Patrón publicador subscriptor"},
19         tactic = {"Detección, Heart Beat", "Incluir procesamiento concurrente mediante hilos",
20                  "Almacenamiento en cache", "Métodos asincrónicos",
21                  "Implementar el broker de mensajería Apache Kafka"},
22         strategy = {""}
23 )
```

Figura 4.16: Implementando Anotaciones de Código

La Figura 4.16 muestra la documentación del rationale y de la decisión de diseño para el caso planteado, la solución consiste en implementar un microservicio en Spring Boot para publicar un API REST que permita consultar las facturas del módulo de salud por parte del sistema financiero, vemos que para dar solución a las preocupaciones de calidad se proponen un conjunto de tácticas, también se anota la decisión y las justificaciones de manera explicita al igual que el contexto en el que se toma la decisión, las alternativas y unas posibles consecuencias de elegir un determinado enfoque, por ejemplo, seleccionar EJBs no es recomendado, dado que no cuenta con herramientas como: balanceo de carga, seguridad, servidor de descubrimiento, etc. Esto significa que se deberían implementar estas funcionalidades, por otra parte, el framework de Spring Boot cuenta con un conjunto de herramientas que ayudan a cumplir con las necesidades funcionales y no funcionales.

4.6.1. Implementación del Plugin

El desarrollo del plugin se realiza siguiendo las guías del proceso descrito en Dorado et al. [36] y la metodología de ICONIXI Process [2], como se muestra en la Figura 4.17, es un enfoque minimalista y simplificado que se centra en el área que se encuentra entre los casos de uso y el código. Su énfasis está en lo que tiene que suceder en ese momento del ciclo de vida donde está empezando, se inicia con algunos casos de uso, después se

necesita hacer un buen análisis y diseño.

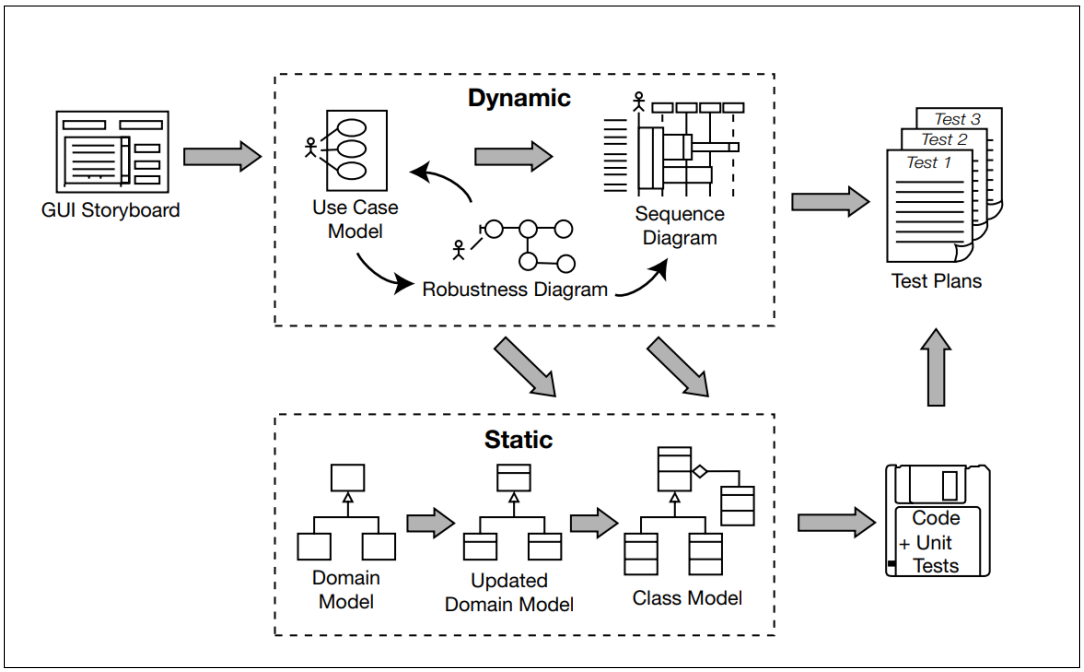


Figura 4.17: ICONIX Process. Tomado de [2]

Fases de ICONIX:

1. **Análisis de Requisitos**
   - Modelo de Dominio
   - Requisitos Funcionales
   - Requisitos No Funcionales
   - Casos de Uso
2. **Análisis/Diseño Preliminar**
   - Análisis de Robustez
   - Diagramas de Secuencia
   - Diagramas de Clase
3. **Implementación y Pruebas**

- Escritura de Código
- Pruebas de Aceptación del Usuario y de Sistema

Modelo de Dominio

- La evolución de la Arquitectura Software tiene que ver mucho con las nuevas decisiones de diseño o eliminar las obsoletas para satisfacer los requisitos cambiantes. El desafío es hacer esto en armonía con las decisiones de diseño previamente tomadas [49].
- La Arquitectura de Software es compleja y erosiona durante su evolución, estos problemas se deben a la pérdida del conocimiento acerca de las decisiones de diseño sobre las que se basa la arquitectura que, aunque esta implícitamente dado, normalmente se carece de una representación explicita de las decisiones y su rationale como ciudadanos de primera clase. En consecuencia, el conocimiento sobre estas decisiones de diseño desaparece en la arquitectura [49].
- La reutilización de la Arquitectura de Software es el uso de las mezclas de decisiones de diseño, por ejemplo, patrones de diseño o componentes.
- Las decisiones de diseño arquitectónico incluyen diferentes factores, por ejemplo, atributos de calidad, plataformas tecnologías, lenguajes, frameworks, patrones de diseño de alto nivel, estilos arquitectónicos, tácticas arquitectónicas, mecanismos de comunicación, entre otros. [28].
- Sin un rationale explicito adecuado, la arquitectura se degrada con el paso del tiempo debido a los cambios, las arquitecturas se vuelven cada vez más frágiles si no se conserva la documentación de las decisiones y su justificación. Por ejemplo, los arquitectos pueden dejar la organización, cambiar roles o cambiar proyectos. Además, la documentación arquitectónica con justificación facilita a los arquitectos comprender un diseño, especialmente si no participaron en el proceso de diseño original [32].
- Un objetivo del rationale en el diseño, es transmitir información útil de un diseñador que trabaja en un momento y contexto dado a otro diseñador que trabaja en otro momento y contexto completamente diferentes [47].
- En un contexto ágil con practicas de Arquitectura Software, es importante contar con un mecanismo de documentación no invasivo que requiera esfuerzos reducidos, con el objetivo de tener una documentación completa y simplificada que sea más fácil de revisar, actualizar y comunicar en el tiempo [41].

Requisitos Funcionales

- El arquitecto debe poder registrar las decisiones de diseño arquitectónico junto con su rationale, es decir, sus justificaciones.
- El arquitecto debe poder modificar las anotaciones y agregar nuevas.
- Los integrantes del equipo deben poder ver las anotaciones de los arquitectos de software.
- Los integrantes del equipo deben poder ver las anotaciones de decisiones de diseño y su rationale a través de un reporte.

Requisitos No Funcionales

- Interoperabilidad: El sistema debe ser capaz de comunicarse con otras herramientas que deseen mostrar las decisiones de diseño y su rationale.
- Mantenibilidad: El sistema debe permitir agregar funcionalidad adicional sin modificar el código ya existente.
- Usabilidad: La herramienta debe permitir su utilización de manera rápida y sencilla.
- Rendimiento: El sistema debe tener la capacidad de generar los reportes en menos de 1 minuto.

Casos de Uso

Los casos de uso describen la forma en que el usuario interactuará con el sistema y cómo responderá el sistema, se definen con base en los requisitos funcionales. En la Figura 4.18 explica la funcionalidad de la herramienta.

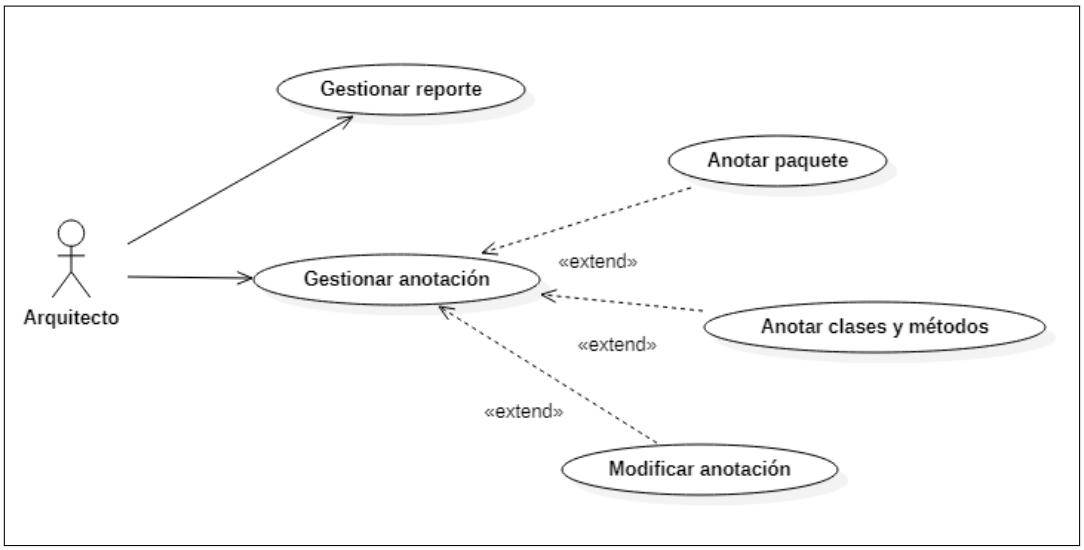


Figura 4.18: Diagrama de Casos de uso de la herramienta RADAR

- **CU01 - Gestionar Anotación:** El Arquitecto gestiona la anotación de código, documentando el rationale de las principales decisiones de diseño arquitectónico y la decisión misma.

  CU01.1 - Anotar paquete: Por lo general los proyectos Java generan un package-info, el Arquitecto debe poder generar anotaciones en estos paquetes.

  CU01.2 - Anotar clases y métodos: El Arquitecto puede generar anotaciones a nivel de clase y métodos si así lo considera.

  CU01.3 - Modificar anotación: El Arquitecto puede modificar las anotaciones generadas.

- **CU02 - Gestionar reporte:** El Arquitecto e integrantes del equipo generan un reporte del rationale y las decisiones de diseño arquitectónico documentadas en el código fuente.

Diagrama de robustez

En la Figura 4.19, se muestra el diagrama de robustez de la herramienta RADAR, permite observar como interactúan, los limites, entidades y controles involucrados entre el Arquitecto y el sistema.

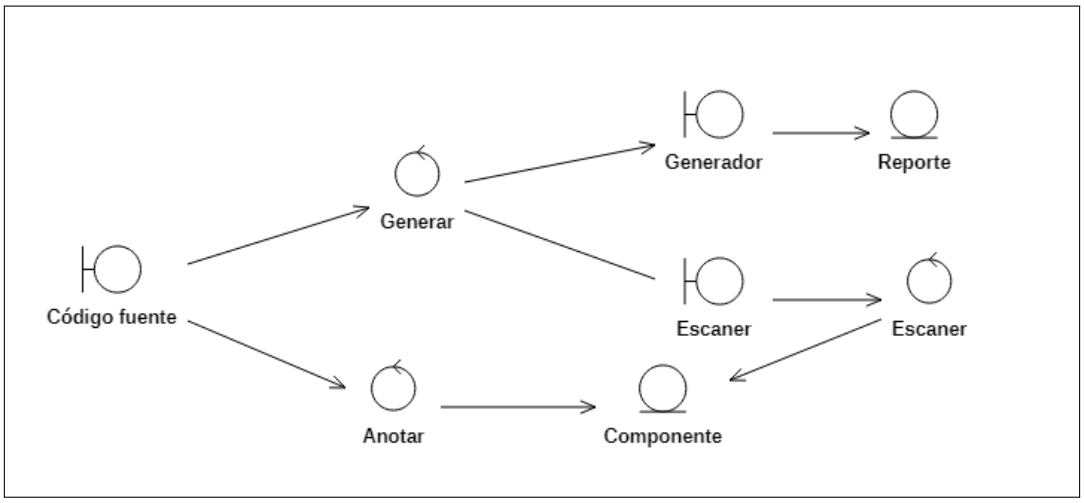


Figura 4.19: Diagrama de Robustez de la herramienta RADAR

Diagrama de secuencia

Este diagrama representa el flujo de actividades que realiza el Arquitecto en la documentación de la decisión de diseño arquitectónico y su rationale. En la Figura 4.20, se expresa la comunicación entre los componentes que estructuran la herramienta RADAR basada en anotaciones de código.

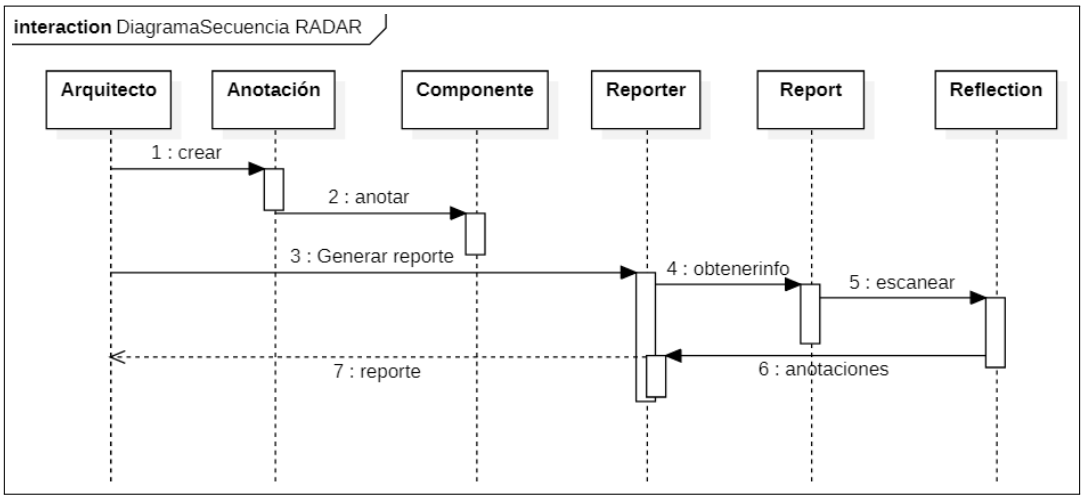


Figura 4.20: Diagrama de secuencia de la herramienta RADAR

Diagrama de clase

Este diagrama describe la estructura del sistema RADAR, muestra sus clases, atributos, operaciones (métodos) y las relaciones entre los diferentes objetos, con los cuales el Arquitecto puede anotar los componentes y generar el reporte de las anotaciones. La Figura 4.21 muestra la iteración de las clases.

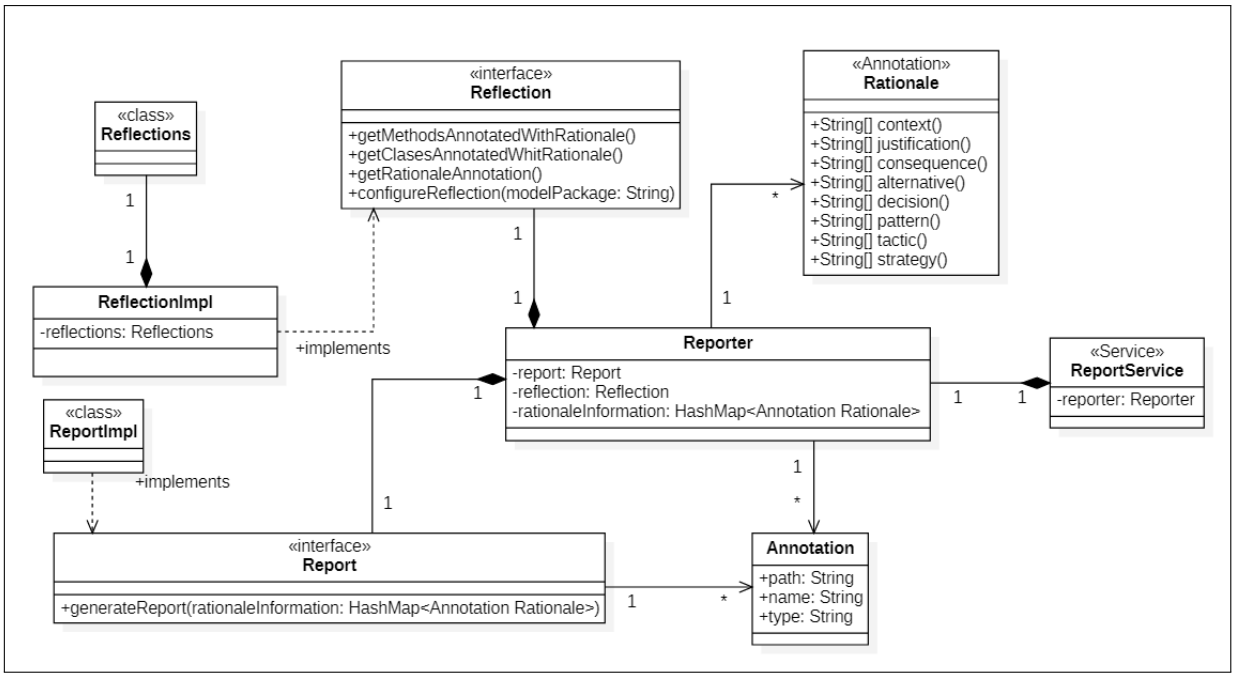


Figura 4.21: Diagrama de Clases de la herramienta RADAR

Diagrama de paquetes

El diagrama de la Figura 4.22 representa las dependencias entre los paquetes que componen el sistema. Es decir, muestra cómo un sistema está dividido en agrupaciones lógicas y las dependencias entre esas agrupaciones.

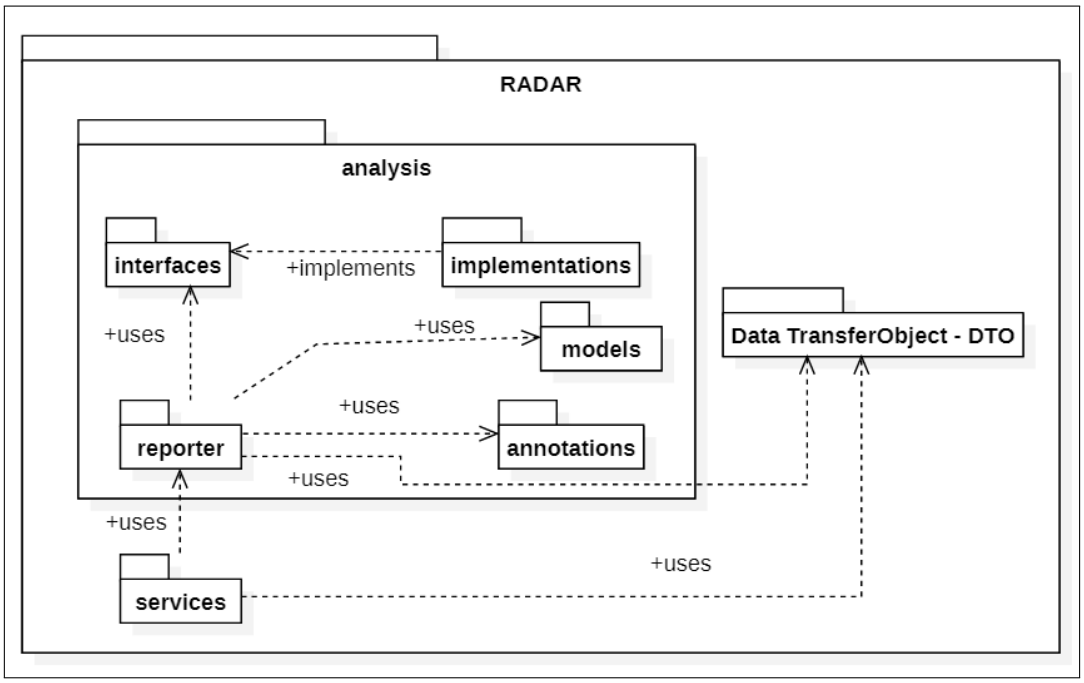


Figura 4.22: Diagrama de Paquetes de la herramienta RADAR

Implementación

La implementación se realiza en el lenguaje Java, tamando como base el repositorio de Dorado et al. [36] https://github.com/zahydo/arat-V1.0, la gestión de dependencias se realiza utilizando maven [7], la versión seleccionada de Java es 1.8.

El código fuente de RADAR con las modificaciones necesarias para este proyecto se encuentra en un repositorio público de github, https://github.com/tomilton/radar. La Figura 4.23 muestra la implementación.

[7]Maven es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002.

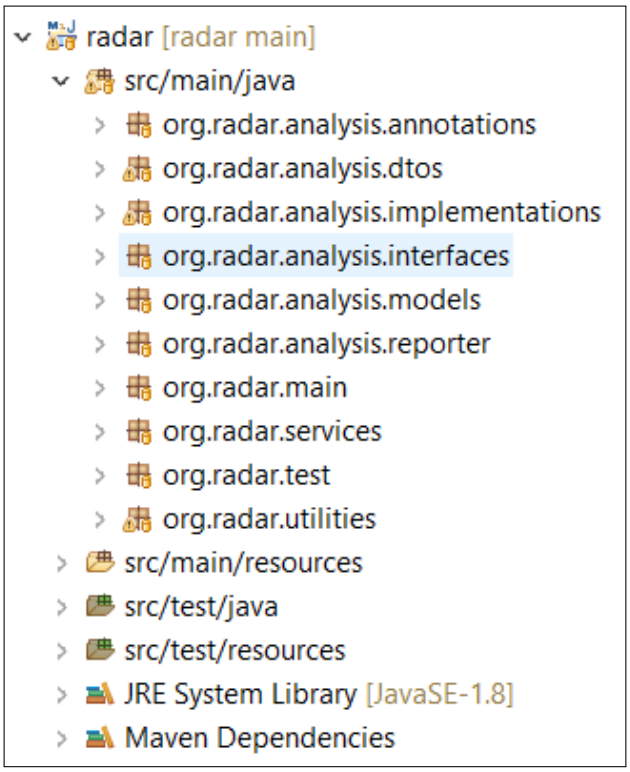


Figura 4.23: Implementación de la herramienta RADAR

Diagrama de despliegue

La herramienta se empaqueta en un archivo .jar, el cual puede ser integrado como una dependencia en proyectos maven o como una librería externa en proyectos java sin gestión de dependencias. Este archivo .jar contiene todas las dependencias necesarias para su funcionamiento y no posee ninguna configuración adicional para su uso e instalación. En la Figura 4.24 se puede observar cómo la herramienta se integra a un sistema y captura los componentes marcados con las anotaciones de código fuente con información de las decisiones de diseño y su rationale arquitectónico, expone un servicio REST, con el objetivo de generar interoperabilidad con otras herramientas.

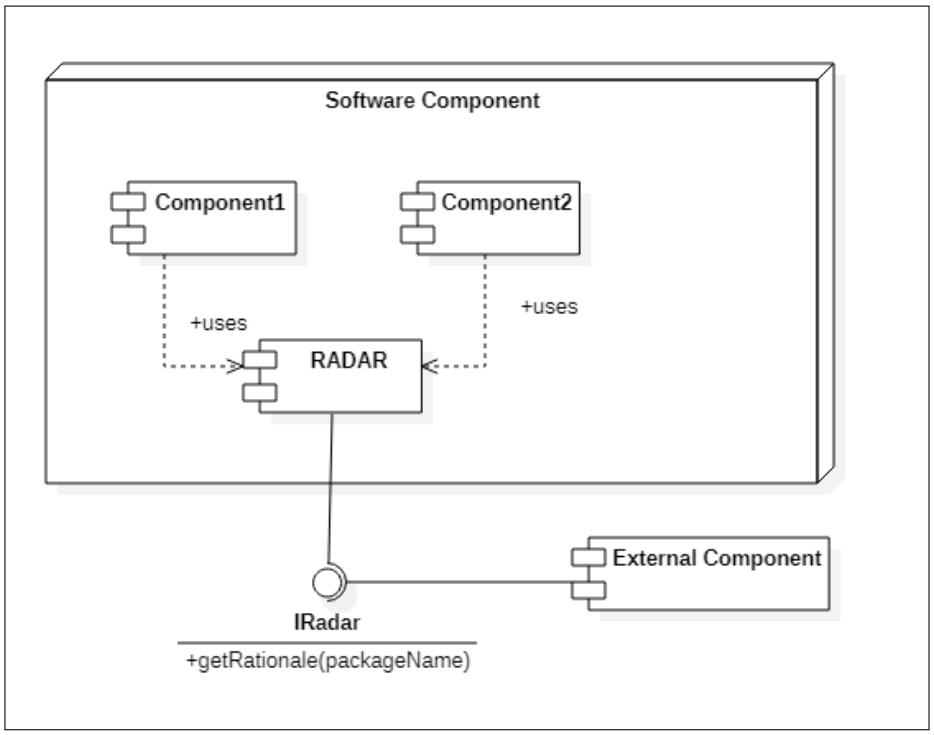


Figura 4.24: Diagrama de Despliegue de la herramienta RADAR

4.7. Discusión

En este capítulo hemos presentado el proceso que permite especificar el lenguaje de anotaciones, mediante un modelo conceptual abstraemos los principales elementos que componen una decisión de diseño y su rationale, los elementos identificados se formalizan a través del meta-modelo DRML, a partir del cual se crea la herramienta DRMLTool que permite documentar de manera visual aquellas decisiones que no se pueden colocar de manera explícita en el código, la herramienta genera un archivo con la extensión .rationale-diagram que se puede agregar por separado como un artefacto más de arquitectura junto con la documentación existente o nueva. Dentro de un enfoque de 4+1 vistas, este corresponde a una quinta vista de rationale. Finalmente, con base en los constructos del meta-modelo y sus reglas de diseño se crea la herramienta RADAR que permite a los arquitectos documentar el fundamento y las decisiones directamente en el código. Con este planteamiento se pretende cubrir la documentación del rationale y las decisiones de diseño arquitectónico desde las diferentes perspectivas de la Arquitectura Software, con el objetivo de que sea fácil de comunicar y mantener en el tiempo.

Capítulo 5

Evaluación del Lenguaje de Anotaciones

5.1. Introducción

Al proponer un lenguaje de anotaciones de código, 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.

Para realizar una evaluación del lenguaje conceptual de anotaciones, que permita prever si es o no aplicable en pequeñas entidades productoras de software que usan métodos ágiles con prácticas arquitecturales en sus procesos, fue utilizado el método de investigación de estudio de caso, con aplicaciones a diferentes contextos. La investigación en la ingeniería de software mediante estudios de caso tiene como objetivo estudiar fenómenos relacionados al desarrollo de software a través de las partes interesadas en un contexto real dado [73].

5.2. Metodología del Estudio de Caso

En el estudio se siguió el método de estudios de caso propuesto por [73]. 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 [73].

El diseño de la investigación partió de varias preguntas de investigación muy relacionadas. Primero, la pregunta básica de investigación planteada en el proyecto **¿Cómo consolidar un mecanismo que facilite la documentación de las principales decisiones de diseño arquitectónico en el contexto ágil ?**, y de ésta fue derivada la pregunta concreta para el estudio de caso. En la Figura 5.1 se puede observar los pasos para el diseño y ejecución del estudio. Con el propósito de analizar la documentación del rationale a través del modelo DRML y su implementación en DRMLTool.

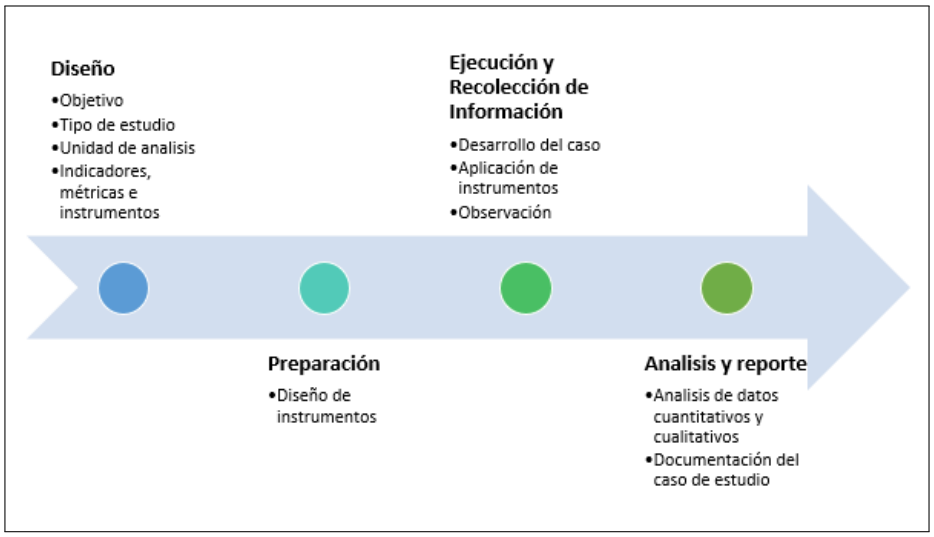


Figura 5.1: Pasos generales seguidos en los Estudios de Caso.

5.3. Estudio de Caso DRML y DRMLTool en la Industria

5.3.1. Pregunta de Investigación

En el marco del proyecto "DRML y DRMLTool EN LAS EMPRESAS DE SOFTWARE DEL CAUCA: UNA EXPERIENCIA DESDE AIC-EPSI" busca introducir el paradigma DRML en las pequeñas empresas desarrolladoras de software en el contexto de la realidad caucana, derivada de ésta búsqueda surge la necesidad de implementar mecanismos de

documentación que ayuden a la comprensión, evolución y preservación de la arquitectura del software. Necesidad derivada de la pregunta: ¿Qué efecto tiene el hecho de utilizar el modelo DRML, a través de su instrumentación en DRMLTool para documentar el rationale de las decisiones de diseño sobre la mantenibilidad de la arquitectura en términos de comprensión? Con el fin de responder ésta pregunta, se determinó que una forma es a través de una experiencia en la industria de software local.

5.3.2. Hipótesis

La hipótesis de este estudio radica en que la existencia de un diseño que representa visualmente la información del rationale arquitectónico y las decisiones podría impactar positivamente la mantenibilidad y evolución de la arquitectura considerando aspectos como la comprensión, eficiencia y efectividad al realizar cambios arquitecturales.

Comprensión

- H_O: La media de comprensión del grupo que utilizo la herramienta es igual al grupo que no la uso.
- H_A: La media de comprensión del grupo que utilizo la herramienta es mayor al grupo que no la uso.

Eficiencia

- H_O: La media de eficiencia del grupo que utilizo la herramienta es igual al grupo que no la uso.
- H_A: La media de eficiencia del grupo que utilizo la herramienta es mayor al grupo que no la uso.

Efectividad

- H_O: La media de efectividad del grupo que utilizo la herramienta es igual al grupo que no la uso.
- H_A: La media de efectividad del grupo que utilizo la herramienta es mayor al grupo que no la uso.

5.3.3. Objetivo del Estudio de Caso

Analizar la documentación del rationale a través del meta-modelo DRML (Decisions and Rationale Modeling Language) y su herramienta DRMLTool, con el propósito de determinar el valor del artefacto generado para comprender, capturar y documentar el rationale de las principales decisiones de diseño en la arquitectura.

5.3.4. Selección del Estudio de Caso

Debido a que la práctica del diseño y documentación de la arquitectura software es desarrollada por los ingenieros de software, son ellos los indicados para evaluar la utilidad de la propuesta. De acuerdo a Runeson et. al. [73], este estudio de caso es holístico [1] y descriptivo [2], considerando una unidad de análisis, la cual corresponde al proyecto de mantenimiento en el que es utilizando el meta-modelo DRML y las fuentes de información son los ingenieros responsables de la toma de decisiones de arquitectura.

5.3.5. Contexto del caso

La Asociación Indígena del Cauca es una entidad promotora de salud pública, que busca fortalecer la capacidad administrativa y organizativa de sus procesos y procedimientos internos, a través de la administración automatizada de la información. Con el fin de prestar un mejor servicio de salud a sus usuarios y optimizar procesos internos que en algunos casos se hacen manualmente. Razones por las cuales, dentro de sus procesos organizativos tiene un subproceso de desarrollo de software. Es un equipo de trabajo conformado por ingenieros de sistemas, con un alto grado de conocimiento técnico, enfocados en la formalización, optimización y sistematización de procesos, mediante un sistema denominado SUIIN (Sistema Único de Información Indígena) que centraliza la información.

Hay que mencionar, además, que implementan buenas prácticas en el desarrollo de sistemas de información, por ejemplo, la metodología que utilizan para desarrollar software es el proceso unificado ágil (AUP), con el objetivo de desarrollar software de manera ágil, pero con buenas prácticas de arquitectura.

[1] Un estudio de caso holístico, estudia el caso en conjunto [94]
[2] Descriptivo detalla una situación o fenómeno [73].

El proceso unificado ágil (AUP)

El Proceso Unificado Ágil de Scott Ambler [6] o Agile Unified Process (AUP) en inglés es un enfoque liviano del Proceso Unificado de Rational (RUP). Este describe de una manera simple y fácil de entender el proceso de desarrollar aplicaciones de software de negocio usando técnicas ágiles, conceptos y artefactos de (RUP), con enfasis en la Arquitectura Software y una buena documentación. El (AUP) aplica técnicas ágiles incluyendo Desarrollo Dirigido por Pruebas (test driven development – TDD), Modelado Ágil, Gestión de Cambios Ágil, y Refactorización de Base de Datos para mejorar la productividad. Además, la organización incorpora otras practicas ágiles, por ejemplo, reuniones diarias (daily), creo un documento híbrido entre (Épica/Historia de usuario) y casos de uso para documentar componentes software complejos, iteraciones mas cortas y documentos livianos.

Es por esto que es un buen candidato para realizar el estudio.

En las Las Figuras 5.2 y 5.3 se puede observar el sistema.

Figura 5.2: Sistema Único de Información Indígena

Figura 5.3: Módulos Suiin

El código fuente del sistema se encuentra en el repositorio de la empresa, con el fin de describir el software con la mayor precisión posible, la estructura se basa en el modelo de vistas de arquitectura "4 +1".

Es importante resaltar que para todas las carpetas de los módulos software se maneja la estructura:

- Análisis y Diseño, contiene toda la información referente a la fase de levantamiento de requisitos, prototipos y documentación asociada a la lógica de negocio.
- Implementación, contiene el código fuente.
- Calidad, contiene la documentación asociada a la fase de pruebas internas y externas para garantizar el proceso de calidad.
- Despliegue, contiene las diferentes versiones de cada aplicativo puestas en producción.
- Mantenimiento, contiene documentación asociada a ajustes o mejoras de cada aplicativo.

5.3.5.1. Cambios Arquitectónicos

1. **Front-End:** Actualmente el sistema está utilizando el framework de presentación primefaces, el cual ha respondido de manera óptima a las necesidades de la organización, sin embargo, en aras de mejorar la accesibilidad de los usuarios a el sistema, ha surgido la necesidad de que los nuevos módulos que se construyan sean adaptables a dispositivos móviles, además de mejorar su aspecto visual. Ver Figura 5.4.

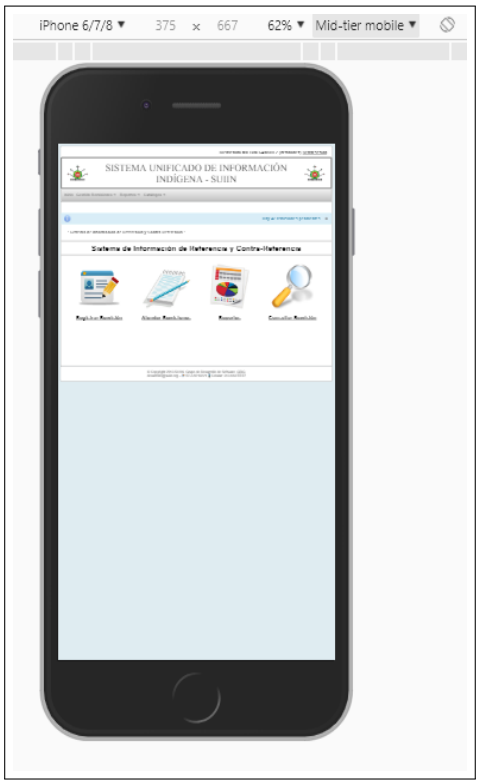


Figura 5.4: Dispositivo móvil Suiin

2. **Back-End:** Actualmente los módulos que se construyen para el sistema no cuentan con un mecanismo que les permita interactuar con otros módulos y al tratarse de una aplicación Java Web el empaquetado se hace mediante un War como se puede ver en la Figura 5.5. Lo que dificulta la reutilización de código, además, se utilizan EJBs de manera incorrecta y es necesario redefinir algunas capas.

   Enterprise JavaBeans (EJBs) : Un Enterprise JavaBean es una clase Java con características que lo hacen mucho más potente y robusto.

   Características (EJBs):

   Al ejecutar un EJB en un contenedor Java EE (Enterprise Edition) con soporte para EJBs, el contenedor brinda los siguientes servicios:

   - Seguridad
   - Llamadas asíncronas
   - Llamadas remotas

- Transacciones
- Inyección de dependencias (CDI)
- Pool de conexiones
- Thread-Safety(Manejo de concurrencia seguro)
- Scheduling (Manejo de tareas programadas)
- Mensajería

Brindan, robustez, reusabilidad y escalabilidad a nuestras aplicaciones, se ejecutan en cualquier servidor de aplicaciones Java.

Es importante que la aplicación dentro de su arquitectura cuente con una capa dedicada a la lógica de negocio, donde se haga uso de EJBs no solo para llamar a métodos transaccionales sino para implementar lógica de negocio importante y aprovechar todas las características y bondades que ofrece un EJB.

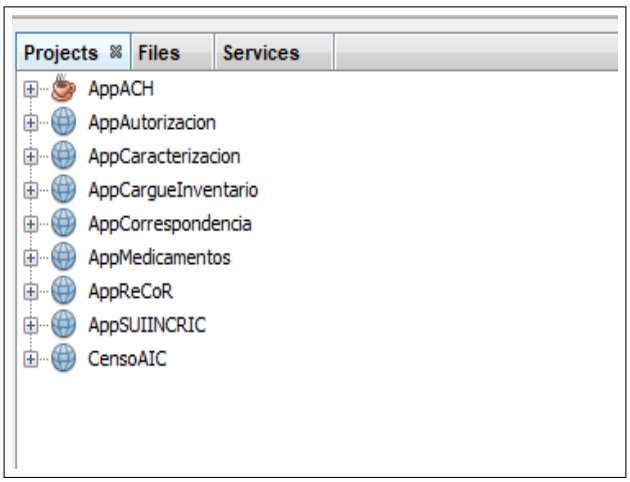


Figura 5.5: Módulos Actualmente

3. **Documentación:** Para verificar la comprensión del rationale y la arquitectura de software, los ingenieros deben documentar las decisiones finales y su fundamento para los cambios arquitectónicos realizados de acuerdo a la estrategia de documentación que le corresponda, documento de arquitectura (SAD) o con la herramienta DRMLTool.

La necesidad de la empresa de formalizar el fundamento arquitectónico de cada uno de sus productos software, enfocado en la arquitectura software en el contexto de un enfoque ágil, permitió el planteamiento de este estudio de caso para una evaluación de la especificación del modelo DRML.

5.3.5.2. Sujetos de investigación

Los participantes fueron ingenieros de software (ingenieros de sistemas y electrónicos) actualmente relacionados con la industria local de software, por lo que todas las personas seleccionadas tienen conocimiento en los campos de la ciencia de la computación y campos relacionados. Una mujer y tres hombres mayores de 27 años contando con experiencia entre 2 y 5 años en la industria desempeñando diferentes roles como analista, ingeniero de producto, desarrollador, pruebas y arquitecto de software.

5.3.6. Diseño del estudio

Los ingenieros se dividen en dos grupos, uno con el modelo de rationale, la herramienta DRMLTool y con un SAD que incluye información del rationale arquitectónico en forma descriptiva (Grupo Experimental) y otro, sólo con el SAD que incluye información del rationale arquitectónico en forma descriptiva (Grupo de Control). Además, los ingenieros recibieron una capacitación sobre los conceptos fundamentales relacionados con: decisiones de diseño, rationale arquitectónico, patrones de arquitectura, patrones de diseño, arquitectura de software, tácticas, estrategias y una introducción al modelado y documentación de la arquitectura software. A continuación, a los ingenieros se les entrega la herramienta de modelado DRMLTool de acuerdo a la selección del grupo que le corresponda, para ello se dividieron los ingenieros en los dos grupos en forma aleatoria.

En vista de que, los ingenieros conocen el sistema a modificar, no es necesario contextualizar los componentes del sistema. Luego, se hace entrega de los cambios arquitectónicos que se deben realizar. Así mismo, empieza a contar el tiempo. A medida que los ingenieros terminan, se retiran los artefactos del estudio y se capturan los tiempos de cada cambio arquitectónico. Por consiguiente, se hace entrega de la encuesta con las preguntas especificadas y generales, para obtener información cuantitativa y cualitativa.

5.3.7. Indicadores y Métricas

Para evaluar de manera objetiva éste estudio de caso, particularmente, para dar respuesta a la pregunta de investigación fue necesario definir un conjunto de métricas e indicadores. La Tabla 5.1 muestra un resumen de los indicadores y métricas identificados.

Indicadores	*Métricas*	*Instrumentos*
Comprensión	Comprensión percibida por los ingenieros al documentar el rationale y realizar los cambios arquitectónicos del estudio.	Encuesta. Protocolo de observación.
Eficiencia	Cambios arquitectónicos diseñados y documentados correctamente. Tiempo utilizado para realizar los cambios arquitectónicos.	Encuesta. Protocolo de Observación.
Efectividad	Cambios arquitectónicos diseñados y documentados correctamente. Correctitud esperada en los cambios realizados.	Encuesta. Protocolo de observación.
Percepción de la usabilidad por parte de los usuarios	Facilidad de uso de los elementos del prototipo implementado a través del metamodelo DRML.	Encuesta.

Tabla 5.1: Métricas e Indicadores

A continuación, se describen en detalle los indicadores y la forma en que estos son calculados a través de las métricas identificadas:

Para realizar el calculo de cada una de las variables nos basamos en la evaluación realizada por Dorado et al. [36].

En el cálculo de la eficiencia y esfuerzo, es necesario medir las variables tiempo y correctitud en los cambios arquitectónicos. Así mismo, se establecen los valores óptimos para las variables. En el caso de la correctitud los ingenieros deben realizar tres cambios arquitectónicos, que se califican de acuerdo a los siguientes criterios:

La Tabla 5.2 muestra la correctitud para el primer y segundo cambio arquitectónico.

Puntuación	Descripción
1	Realizó los cambios pero no cumplen con el resultado deseado.
2	Realizó modificaciones, pero no logró cambios funcionales.
3	Realizó cambios a nivel de diseño y manteniéndose dentro de la arquitectura.
4	Realizó cambios manteniendo el diseño y cambiando la arquitectura.
5	Realizó los cambios a nivel de diseño y la arquitectura.

Tabla 5.2: Correctitud primer y segundo cambio arquitectónico. Tomado de Dorado et al. [36]

La Tabla 5.3 muestra la correctitud para el tercer cambio arquitectónico.

Puntuación	Descripción
1	La documentación esta incompleta.
2	Documentó los cambios a nivel funcional.
3	Documentó los cambios a nivel de diseño.
4	Documentó los cambios a nivel arquitectural sin su Rationale.
5	Documentó los cambios arquitectónicos junto con su Rationale.

Tabla 5.3: Correctitud cambio arquitectónico tres. Tomado de Dorado et al. [36]

La fórmula para hacer el cálculo de la correctitud obtenida (C_o) de los cambios arquitectónicos es definida como:

$$C_o = \sum_{i=1}^{n} C_i,$$

donde C_i es la correctitud evaluada en el cambio arquitectónico i, el cual puede tomar valores de uno a cinco, cuando todos los valores de C_i son iguales a cinco la correctitud obtenida la notaremos C_e la cual corresponde a la correctitud esperada, n es el número de cambios arquitectónicos realizados por los ingenieros.

La fórmula que hemos definido para hacer el cálculo del tiempo total (T_t) en realizar los cambios arquitectónicos está dada por:

$$T_t = \sum_{i=1}^{n} T_i,$$

Donde, T_i el tiempo en que un ingeniero realiza un cambio arquitectónico, n el número de tiempos tomados en los cambios arquitectónicos realizados por los ingenieros.

Los valores óptimos que se establecen para la correctitud y tiempo se muestran en la Tabla 5.4

Cambio	**Correctitud**	**Tiempo (Horas)**
Front-End	5	1
Back-End	5	1
Documentación	5	0.5
Valores óptimos	$C_e = 15$	$T_e = 2.5$

Tabla 5.4: Resultados óptimos para correctitud y tiempo total

El tiempo estimado T_e corresponde a la suma de los tiempos estimados para el primer, segundo y tercer cambio arquitectónico.

5.3.7.1. Comprensión

Es el nivel de comprensión percibido por los ingenieros al realizar cada uno de los cambios arquitectónicos. La evaluación se hace mediante un cuestionario de cinco preguntas, las cuales se enfocan en determinar la **razón fundamental** de cada unas de las decisiones de diseño tomadas. También, se evalúa que la decisión final corresponda al resultado deseado para cada una de las preguntas. Además, se contrasta con las razones iniciales del arquitecto plasmadas en el documento de arquitectura software SAD (Documento de Arquitectura de Software). La fórmula que hemos definido para hacer el cálculo de la comprensión (C) está dada por:

$$C = \frac{\sum_{i=1}^{n} P_i}{n},$$

Donde, P_i es el resultado asimilado para la pregunta i, el cual puede tomar valores de uno a diez, n es el número de ítems evaluados en el cuestionario por los ingenieros.

5.3.7.2. Eficiencia

La eficiencia se define como el grado en que se cumplen los objetivos al menor costo posible. Siendo en este caso de estudio el menor costo posible de tiempo, dado que el recurso es constante. La fórmula que hemos definido para hacer el cálculo de la eficiencia (E_{fc}) es:

$$E_{fc} = \frac{C_o}{T_t},$$

Donde, C_o son los cambios arquitectónicos diseñados correctamente (*correctitud obtenida*), T_t es el tiempo total en horas utilizado para realizar los cambios.

Por otra parte, es necesario establecer un valor referente para la eficiencia (ER_{fc}) y la correctitud (C_r), con el objetivo de dar un valor en términos de porcentaje y contrastar el resultado con un valor deseado, para ello la fórmula que hemos definido es:

$$ER_{fc} = \frac{C_r}{T_e}.$$

En nuestro caso, tenemos que $C_r = C_1(4) + C_2(5) + C_3(5) = 14$ y $T_e = 2.5$, con lo cual,

$$ER_{fc} = \frac{14}{2.5} = 5.6.$$

Aplicando una regla de tres simple, se obtiene el porcentaje de eficiencia (S) por cada ingeniero, el cual esta dado por:

$$S = 100\frac{E_{fc}}{ER_{fc}}.$$

5.3.7.3. Efectividad

La efectividad se define como el grado en que se producen los resultados esperados. La relación entre los resultados previstos, no previstos y los objetivos. La fórmula que hemos definido para hacer el cálculo de la efectividad (E_f) es dada por:

$$E_f = \frac{C_o}{C_e},$$

Donde, C_o es la correctitud obtenida y C_e es la correctitud esperada.

Al igual que la eficiencia, aplicando una regla de tres simple, se obtiene el porcentaje de efectividad (S_e) por cada ingeniero, dado por:

$$S_e = 100\ E_f.$$

5.3.7.4. Percepción de usabilidad por parte de los usuarios

Para evaluar la satisfacción percibida por los usuarios, aplicamos un cuestionario de siete preguntas, nos basamos en los cuestionarios estandarizados de Sauro et al. [76]. Los ítems del formulario producen cuatro puntajes, uno general y tres subescalas. Las reglas para computar son:

- General: promedio de las respuestas para los ítems 1 a 16 (todos los ítems).
- Calidad de la herramienta: elementos promedio del 1 al 6.
- Calidad de la información: elementos promedio de 7 a 12.
- Calidad de interfaz: elementos promedio de 13 a 16.

La fórmula que hemos definido para hacer el cálculo de la usabilidad percibida es:

$$P_u = \frac{\sum_{i=1}^{n} RA_i}{n},$$

Donde P_u es la percepción de usabilidad, RA_i es el resultado asimilado para la pregunta i, el cual puede tomar valores de uno a siete, n es el número de ítems evaluados en la encuesta por los ingenieros.

5.3.8. Desarrollo del Caso

El estudio comienza con una introducción a la teoría de la lógica arquitectónica para presentar los conceptos a los participantes, seguido de la presentación del modelo DRML y la herramienta DRMLTool, haciendo un ejemplo para comprender las estructuras, su uso y algunas restricciones, después el modelo de documentación se muestra con la información del rationale arquitectónico. Además, la aplicación y los cambios arquitectónicos a realizar fueron explicados, después de llevar a cabo todas las modificaciones arquitectónicas y la entrega de los resultados, cada participante debía completar una encuesta. Después de entrenar a los participantes, se les entregó el material del estudio. La configuración del proyecto se realiza en el entorno de desarrollo eclipse. El código fuente de la aplicación se descarga del repositorio SVN (Subversión) de la empresa al igual que el documento de arquitectura - SAD. También se les entrega una guía con los cambios arquitectónicos y una encuesta. Los participantes comenzaron a documentar el rationale arquitectónico de las decisiones de diseño finales, cada participante revisa la documentación previa de un cambio documentado, con el objetivo de evaluar si el fundamento de cada decisión es comprensible por otro ingeniero. A medida que cada ingeniero modela la decisión con su fundamento y evalúa la documentación de las demás cambios se le entrega la encuesta para recopilar información cualitativa que contribuya a la definición de la estructura final del modelo de documentación para la justificación arquitectónica.

5.3.9. Amenazas de Validez

Para garantizar la fiabilidad de los resultados en el estudio de caso y resultara verdadero y evitar que no sea sesgado por el punto de vista del investigador existen diferentes formas de clasificar los aspectos de la validez y las amenazas a la validez. Así que, nosotros optamos por colocar en práctica la validez interna y validez externa. Para validez interna el diseño del instrumento fue validado por expertos en arquitectura de software, analista de sistemas e ingenieros expertos en métodos ágiles. Por otra parte, para la validez externa en la convocatoria a los sujetos se tuvo en cuenta a todos los ingenieros que integran el equipo de desarrollo de software en la empresa, que participan en el proceso de construcción, por ejemplo, equipo de pruebas, analistas de negocio, programadores y

arquitectos, el volumen de participación fue alto, sin embargo, no todos cumplieron con los requisitos mínimos para participar en el estudio.

5.3.10. Resultados

A continuación, son presentados los resultados cuantitativos y cualitativos del estudio de caso; las respuestas más precisas de la encuesta realizada a los ingenieros; registro y calculo de los valores tomados; observaciones; modelos resultantes con su rationale arquitectónico (Figura 5.7) y participantes del estudio.

5.3.10.1. Resultados Cuantitativos

Mediciones directas:

A continuación se muestra el registro de los datos tomados en el caso donde los sujetos investigados realizaron los cambios arquitectónicos basándose en la guía provista, la medida de tiempo ha sido tomada en horas. La Tabla 5.5 muestra la información del tiempo que cada sujeto utilizó para diseñar y documentar los cambios arquitectónicos.

Cambios arquitectónicos (Tiempo en horas)					
	T_1	T_2	T_3	T_t	DRMLTool
Ingeniero 1	1.0	1.0	0.5	2.5	SI
Ingeniero 2	1.0	1.0	0.5	2.5	SI
Ingeniero 3	1.0	1.0	0.5	2.5	NO
Ingeniero 4	1.0	1.0	0.5	2.5	NO

Tabla 5.5: Registro del tiempo empleado por los ingenieros.

La Tabla 5.6 muestra la correctitud de los cambios arquitectónicos diseñados y documentados de manera correcta por los ingenieros.

Correctitud en los cambios arquitectónicos					
	C_1	C_2	C_3	C_o	DRMLTool
Ingeniero 1	4.5	3.5	3.5	11.5	SI
Ingeniero 2	4.0	4.0	3.5	10.5	SI
Ingeniero 3	4.0	3.0	3.0	10.0	NO
Ingeniero 4	4.0	3.0	3.5	10.5	NO

Tabla 5.6: Cambios arquitectónicos diseñados y documentados correctamente por los ingenieros.

Comprensión: Teniendo en cuenta la encuesta entregada al finalizar la sesión de los cambios arquitectónicos realizados por parte de los ingenieros, se analizó los datos recogidos con el fin de obtener un valor cuantitativo de la comprensión al diseñar y documentar las decisiones de diseño con su fundamento en la arquitectura. Las preguntas analizadas en la encuesta respecto a la comprensión se pueden ver en la Tabla 5.7:

Pregunta	**Descripción**	Puntaje
A	¿Cuáles son los atributos de calidad que busca conseguir con la arquitectura plateada?	10
B	¿Qué estrategias y tácticas utilizo para cumplir con los atributos de calidad?	10
C	¿Qué patrones arquitectónicos utilizó para la arquitectura?	10
D	¿Qué patrones a nivel de diseño sugiere para la arquitectura planteada?	10
E	¿La contextualización sobre la importancia del rationale fue de utilidad para realizar los cambios solicitados?	10

Tabla 5.7: Preguntas para evaluar la comprensión. Tomado de Dorado et al. [36]

Comprensión en documentación								
	P_A	P_B	P_C	P_D	P_E	C	Promedio	DRMLTool
Ingeniero 1	10	10	9.0	8.0	10	47	9.4	SI
Ingeniero 2	10	10	7.0	8.0	10	45	9.0	SI
Ingeniero 3	9.0	8.0	9.0	9.0	6.0	41	8.2	NO
Ingeniero 4	9.0	7.0	8.0	7.0	8.0	39	7.8	NO

Tabla 5.8: Indicadores en la comprensión

Como se puede observar en la Tabla 5.8, al aplicar la siguiente regla de tres, $Porcentaje = (((9.4+9.0)/2)*100)/10$, se obtiene el porcentaje de comprensión para el grupo que utilizó la herramienta DRMLTool, en este caso es del 92 %, a diferencia del grupo sin DRMLTool el cual es de 80 %. Esto comprueba que si hay impacto positivo en la comprensión de la arquitectura y su rationale al realizar cambios con un impacto arquitectural mediante el uso de la herramienta.

Eficiencia:

Eficiencia						
	C_o	T_t	$E_{fc} = \frac{C_o}{T_t}$	ER_{fc}	$S = 100\frac{E_{fc}}{ER_{fc}}$	DRMLTool
Ingeniero 1	11.5	2.5	4.6	5.6	82.14	SI
Ingeniero 2	10.5	2.5	4.2	5.6	75.00	SI
Ingeniero 3	10.0	2.5	4.0	5.6	71.42	NO
Ingeniero 4	10.5	2.5	4.2	5.6	75.00	NO

Tabla 5.9: Indicador de eficiencia por cada ingeniero

De acuerdo a los resultados obtenidos en la Tabla 5.9 se puede observar el porcentaje de eficiencia promedio para el grupo con DRMLTool fue del 78.57 % a diferencia del grupo sin DRMLTool el cual tiene un porcentaje promedio de eficiencia del 73.21 %, lo cual indica que el grupo con la herramienta tuvo mejor eficiencia, sin embargo, observamos que los valores se mantienen en un rango aceptable para ambos grupos.

Efectividad:

Efectividad						
	C_o	C_e	$E_f = \frac{C_o}{C_e}$	$S_e = 100\ E_f$	Promedio	DRMLTool
Ingeniero 1	11.5	15	0.76	76.66	73.33	SI
Ingeniero 2	10.5	15	0.7	70.00		SI
Ingeniero 3	10.0	15	0,66	66,66	68.33	NO
Ingeniero 4	10.5	15	0,7	70.00		NO

Tabla 5.10: Indicador de efectividad por cada ingeniero

Los resultados de la Tabla 5.10 muestran que el promedio de efectividad para el grupo con DRMLTool es de 73.33 % y el porcentaje promedio para el grupo sin DRMLTool es

del 68.33 %. De manera similar a la eficiencia, la efectividad parece afectada de manera positiva por el uso de la herramienta con información del rationale arquitectónico.

Percepción de Usabilidad:

La Tabla 5.11 muestra los resultados de la encuesta de percepción de usabilidad, para cada uno de los ingenieros, en los criterios evaluados: calidad de la herramienta, calidad de la información, calidad de la interfaz.

Unidad de Análisis				
	Ingeniero 1	Ingeniero 2	Ingeniero 3	Promedio
Calidad de la herramienta	6.0	6.6	6.1	6.2
Calidad de la información	5.1	6.6	5.0	5.6
Calidad de interfaz	6.0	6.0	3.2	5.0
Promedio general	5.7	6.4	4.8	5.6

Tabla 5.11: Percepción de usabilidad

Los resultados de la 5.11 Muestran que la calidad de la herramienta es aproximadamente del 89 %. La calidad de la información es 80 % aproximadamente y la calidad de la interfaz es 72 % aproximadamente.

Prueba de hipótesis:

Los resultados de medir comprensión, eficiencia y efectividad, son contrastados con la prueba t-Student, para identificar si existe una significancia con respecto a la media de los grupos, los indicadores se muestran a continuación:

Comprensión

El porcentaje de comprensión para el grupo con la herramienta DRMLTool es de 92 % y el grupo sin DRMLTool es del 80 %. Cuando se calcula el valor p da como resultado 0.010412, el resultado es significante a $p < 0.05$, por lo tanto, se acepta la hipótesis alternativa y rechazamos la hipótesis nula. Esto significa que el uso de la herramienta tiene una significación estadística sobre la comprensión de la arquitectura y su rationale al realizar cambios arquitectónicos.

Efectividad

Los resultados muestran que el porcentaje de efectividad para el grupo con la herramienta DRMLTool es de 73,33 % y el grupo sin DRMLTool es del 68,33 %. Cuando se calcula, el valor p da como resultado 0.281419, el resultado no es significante a $p < 0.05$, por lo tanto, se debe rechazar la hipótesis alternativa y aceptar la hipótesis nula, esto significa que el uso de DRMLTool con información del rationale no tiene una significancia estadística sobre la efectividad al realizar cambios arquitectónicos.

Eficiencia

El porcentaje de eficiencia para el grupo con la herramienta DRMLTool fue del 78.57 % y el grupo sin DRMLTool es del 73.21 %. Cuando se calcula el valor p da como resultado 0.270798, el resultado no es significante a $p < 0.05$, por lo tanto, se acepta la hipótesis nula y se rechaza la hipótesis alternativa. Esto significa que el uso de DRMLTool con información del rationale, no tiene una significancia estadística sobre la eficiencia al realizar cambios arquitectónicos.

Finalmente en la Figura 5.6 muestra que hay impacto positivo en la comprensión de la arquitectura y su rationale cuando se hace uso de la herramienta DRMLTool, en términos de comprensión, eficiencia y efectividad.

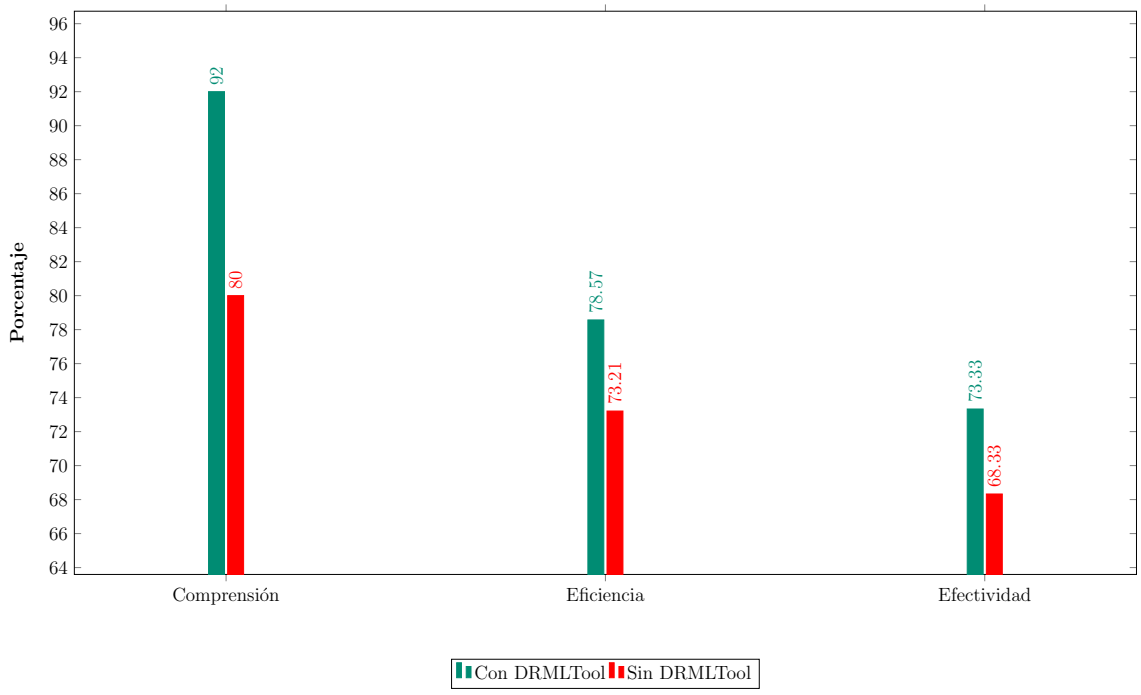


Figura 5.6: Gráfica de comprensión, eficiencia y efectividad

5.3.10.2. Resultados Cualitativos

Encuesta general:

1. ¿Qué uso le daría al lenguaje especificado?
   R: El lenguaje sería muy esencial, para diseñar y crear herramientas para el modelado de mini-patrones de arquitectura y decisiones de cambios a nivel de código.

2. ¿Qué uso le daría a la herramienta diseñada?
   R: La herramienta es muy útil para documentar de una forma muy organizada las razones de cambios a nivel arquitectónico.
   Diseñar y plasmar el conjunto de conocimiento de decisiones, que se generan al momento de construir una arquitectura, facilitando el análisis del modelado, garantizando la evolución adecuada del software.

3. ¿La Información que el lenguaje provee fue útil para la toma de nuevas decisiones de diseño arquitectónico?
   R: Si fue útil, ya que se tuvo como evidencia el porqué de la decisión, proporcionando una visión macro del diseño arquitectónico. Es útil el componente Rationale, Justificación, alternativas y patrones de arquitectura porque se puede saber el estudio y los motivos que se tuvo, para la adopción de una arquitectura en específico.

4. ¿Qué elementos considera que debe tener el lenguaje para mejorar la trazabilidad del conocimiento arquitectónico?
   R:Que se pueda agregar también el requerimiento. Y también permita enlaces a documentos o artefactos externos, por ejemplo, UML.
   Considero que debe permitir relacionar más alternativas y contextos, como también se debe obtener la opción de especificar los requerimientos o tareas.

5. ¿Por qué cree que es importante documentar la justificación arquitectónica en el desarrollo de software?
   R: Porque permite entender la manera de cómo está estructurada la arquitectura de un sistema y llevar un control que ayuda a los arquitectos de software con la toma de nuevas decisiones.Es importante para conocer el funcionamiento de una aplicación y evitar erosionar la arquitectura ya planteada.Es fundamental plasmar contextualizadamente, las razones que permitieron optar por una decisión en particular al momento de diseñar una arquitectura, ya que este tipo de decisiones, facilitan avances o generan limitaciones en el funcionamiento de un software.

6. ¿Qué información es considerada mas relevante para documentar el conocimiento de la arquitectura?
   R: Es información relevante la documentación de los patrones de diseño utilizados en la arquitectura. Su justificación, que conlleva a tomar una decisión, teniendo en cuenta su punto de vista contextual. Para documentar una arquitectura se debe de tener en cuenta las historias de usuario más relevantes que generan impacto, el sprint, las necesidades presentadas por las partes interesadas.

7. ¿Describa las razones por las cuales su estructura arquitectónica tuvo que ser alterada para cumplir los cambios solicitados?
   R: La arquitectura anterior estaba diseñada para aplicaciones monolíticas lo cual era muy difícil de aplicar escalabilidad. También era difícil la reutilización de código, por lo cual redundaba código y las aplicaciones tenían que ser construidas desde cero. Por lo cual se optó por diseñar una arquitectura basada en micro servicios utilizando un gestor de dependencias que permitiera una conexión entre diferentes proyectos. Además, permitiendo también la escalabilidad de la misma. Porque no se comprendían los diseños iniciales, actualmente los requerimientos exigían que se cambiara de tecnología para cumplir las necesidades del negocio. La decisión que se tomó fue diseñar el componente y dotarlo de mecanismos de comunicación como api rest y que fuese interoperable con los componentes existentes.
   No se ajusta, a una futura evolución, ya que al momento de tomar una decisión no se hizo un análisis macro, que permitieran ver a futuro la evolución del sistema y por ende de la arquitectura software.

Observaciones:

- La definición de la arquitectura y su lógica son el eje fundamental para la evolución del software, sería interesante contar con un repositorio donde se pudiera buscar el conocimiento documentado.
- Sería muy útil que la herramienta pudiera subirse a un repositorio y ser instalada como plugin [3] del IDE Eclipse.
- Sería útil que se pudiera representar especificados patrones de diseño estructural y patrones de mini-arquitectura.
- Sería útil que se pudieran representar los patrones de diseño que hayan sido personalizados.

[3]Plugin: Es un complemento para software que agrega una funcionalidad.

- Es necesario que permita enlaces a documentos o artefactos externos, por ejemplo, UML.
- Es importante que en la herramienta se pueda identificar el actor que genera el requerimiento y los diferentes miembros del equipo que participan en el proceso, cada uno con su respectivo rol.
- Cuando se documenta las tácticas y estrategias arquitectónicas, es útil poder anotar que una estrategia consta de un conjunto de tácticas y relacionarla con la decisión.

Modelo Resultante de Fundamento arquitectónico:

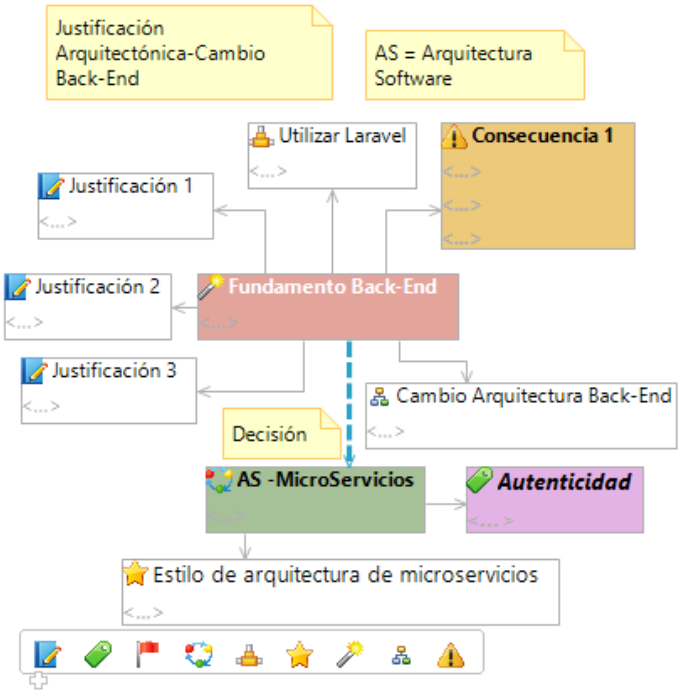


Figura 5.7: Modelo Resultado

Fundamento Back-End : Cambio de arquitectura Monolítica por arquitectura basada en micro-servicios.

Justificación 1 : Se utiliza la herramienta gestión de dependencias Maven, para la creación de arquetipos lo cual garantiza la estructuración del código en proyectos separados, en el cual estará la capa lógica,la capa de acceso a datos utilizando el patrón

de diseño DAO, y por ultimo la capa web donde se exponen los diferentes servicios que serán utilizados por diferentes aplicaciones.

Justificación 2  : Se diseña un arquetipo común basado en Maven, en el cual estará lógica reutilizable como acceso a métodos ,clases comunes que se requieran desde diferentes proyectos.

Justificación 3  : Se crean arquetipos por aplicación ,teniendo como dependencia el arquetipo común "Common".

Consecuencia 1  : Con, Al desarrollar componentes en diferentes tecnologías el mantenimiento es complejo. Pro, No es necesario detener toda la aplicación para realizar ajustes, cada componente funciona por separado.

Contexto, Cambio Arquitectura Back-End  : Se plantea la necesidad de cambiar estructura de arquitectura monolítica por una basada en micro servicios utilizando herramienta de gestión de dependencias Maven.

Decisión, AS-Micro-Servicios  : Trabajar una arquitectura micro servicios, tomando como base la generación de módulos utilizando arquetipos maven, en casos especiales se podrá construir con otras tecnologías, utilizando como base de comunicación servicios api-rest y autenticación de los servicios con tokens de seguridad.

Táctica, Autenticidad  : Autenticación única utilizando JWT generando un token único y usando un servicio de autenticación que garantice la validez de la conexión.

Patrón Arquitectónico  : Una arquitectura de micro servicios consta de una colección de servicios autónomos y pequeños. Los servicios son independientes entre sí y cada uno debe implementar una funcionalidad de negocio individual.

5.3.11. Análisis de Resultados

En el proyecto se emplearon diferentes estilos arquitectónicos motivados por diferentes decisiones de diseño. Un producto arquitectónico fue generado por cada componente durante las iteraciones realizadas. Diferentes grados de los detalles de las decisiones tomadas fueron registradas, para que pudieran ser utilizadas para una mayor manipulación

o proceso de mantenimiento. Tanto en el primer cambio arquitectónico como en el segundo tuvieron puntos de vista diferentes. La evaluación de la herramienta en ambos cambios nos demostró la utilidad de representar visualmente y gestionar el rationale arquitectónico de las decisiones de diseño. Hasta este punto, la interfaz de la herramienta fue suficiente para navegar fácilmente el fundamento y las decisiones tomadas. Por lo tanto, el arquitecto tiene una vista completa del proceso de diseño general, porque los elementos mostrados por la herramienta describen cronológicamente los eventos más importantes ocurridos durante el proyecto, este enfoque facilita la comprensión del proceso de construcción de arquitectura para los interesados que están directamente relacionados con los problemas técnicos. Además, generamos documentación útil que incluye las decisiones tomadas, los patrones aplicados y los productos de arquitectura generados durante el proceso. La herramienta proporciona el modelo con la extensión Rationale Diagram que se asocia con la documentación nueva o existe de la arquitectura software como una vista más, en un contexto ágil.

Después de aplicar la prueba t-Student, se encontró que hay impacto positivo en la comprensión de la Arquitectura Software y su rationale, sin embargo, para las variables eficiencia y efectividad no se encontró alguna significancia estadística respecto al uso o no uso de la herramienta DRMLTool con información del rationale arquitectónico, no obstante, en la Figura 5.6 se puede observar que incluso para estas dos variables el resultado promedio fue mejor para el grupo con DRMLTool. Creemos que esto se debe a diferentes factores, por ejemplo, experticia en determinadas tecnologías, entornos de desarrollo, habilidad para desarrollar, etc.

Aunque los ingenieros de software encontraron el lenguaje y la herramienta útil, se tienen algunos reparos al respecto. Primero la calidad de la interface fue evaluada con sólo el 72 %, particularmente hay limitaciones para escalar el modelo de decisiones a todo el proyecto, la trazabilidad con los requisitos, en particular los no funcionales, así como su trazabilidad hacia el modelo arquitectónico expresado en otros modelos, como por ejemplo UML. Segundo, al documentar el rationale con la herramienta DRMLTool, se evidencia que la comprensión del rationale y la Arquitectura Software mejora notablemente, sin embargo, la eficiencia y efectividad se ven un poco afectadas, creemos que depende en gran medida de la experiencia, nivel técnico y manejo de las herramientas utilizadas en el estudio por parte de los ingenieros.

5.4. Evaluación de Anotaciones de Código

La Arquitectura Software es representada desde diferentes perspectivas que ayudan a los diferentes stakeholder a entender como está estructurado un componente software de ahí la importancia que cada una de estas vistas se documente de manera clara, que exponga con elementos explícitos las justificaciones de determinadas decisiones que permitan la futura toma de nuevas decisiones sin erosionar los planteamientos iniciales, en el contexto de nuestra investigación es importante evaluar el enfoque de documentar el rationale con anotaciones de código y el de representarlo de manera visual como una vista más de arquitectura, que llamamos "vista de rationale".

Los resultados de evaluar el lenguaje DRML y su implementación DRMLTool indican que representar visualmente el rationale y las decisiones de diseño arquitectónico que no se pueden colocar de manera explícita en el código ayuda a comprender la Arquitectura Software y las razones de porque se ha estructurado de determinada forma de modo efectivo y eficiente, los resultados son contrastados con los resultados de la investigación de Dorado et. al.[36] donde se construyó la herramienta ARAT (Architectural Rationale Annotations Tool) que permite documentar el rationale con anotaciones de código, la evaluación de la herramienta se realizó a través de un cuasi-experimento, se comprobó que las anotaciones de código con información del rationale arquitectónico tienen un impacto positivo sobre la eficiencia, la efectividad y la comprensión de la arquitectura y su rationale en la realización de cambios arquitecturales.

La evaluación de estos enfoques en las diferentes vistas de la Arquitectura Software ha permitido mejorar y evolucionar el lenguaje de anotaciones, que sirve para ser instanciado tanto en anotaciones como a nivel visual, con base en estos resultados se construye la herramienta RADAR (Rationale Architectural Decision Annotations) la cual permite documentar el rationale y las decisiones de diseño con elementos explícitos de documentación no invasivos dentro de procesos que trabajan con métodos ágiles pero con prácticas de Arquitectura Software.

Estos resultados nos permiten establecer que documentar el rationale desde diferentes perspectivas ayudan a guiar la futura toma de decisiones y evitar que el conocimiento arquitectónico se diluya en el tiempo.

5.5. Discusión

En este capítulo se presentó la evaluación de lenguaje basado en anotaciones de código, desde las diferentes perspectivas de la Arquitectura Software como es el código y la vista de rationale. Es importante considerar que el caso en el que se evaluó el lenguaje DRML y su implementación DRMLTool se desarrolló en el contexto de una pequeña organización a través de varias sesiones de modelado, con un conjunto pequeño de decisiones de diseño (menos de tres). Sin embargo, la experiencia reportada por Zimmermann [95], han identificado 35 decisiones recurrentes en el desarrollo de aplicaciones empresariales en el contexto del diseño de arquitectura orientado a servicios. Además, Kellari et al. [53] analizan las decisiones arquitectónicas del software de aviones desde el DC3 hasta el 787, encontrando una disminución en la variación de las decisiones en un conjunto de datos de 157 arquitecturas en el histórico de aviones, llegando a la conclusión de que son 27 las decisiones arquitectónicamente relevantes. Estos dos escenarios uno típico y otro extremo en términos de confiabilidad, permiten evidenciar que el lenguaje de modelado de decisiones y su rationale debe facilitar el modelar el modelado de un número no pequeño de decisiones de 26 diseño arquitectónico, facilitando una estructura de organización de dichas decisiones (relaciones temporales o causa/efecto) y un relacionamiento con elementos externos para facilitar su integración y la trazabilidad. Un enfoque es visualizar decisiones de arquitectura desde diferentes perspectivas. Una perspectiva permitiría abordar un conjunto de preocupaciones relacionadas con la decisión. La idea de mostrar diferentes perspectivas de decisiones permitiría la asociación directa con los puntos de vista con los que describa la arquitectura, aunque esto traería más complejidad para relacionar además las decisiones entre diferentes perspectivas de decisiones, además que no habría una relación muchos a muchos entre ellas.

Por otro lado, una efectiva toma de decisiones requiere de comunicación efectiva y de acciones coordinadas del equipo, lo cual agrega mayor complejidad por ejemplo a los proyectos open source, debido a que los desarrolladores de sistemas open source generalmente no trabajan en el mismo lugar Harrison et al. [43]. Esto aumenta la posibilidad de decisiones arquitectónicas mal entendidas y la motivación detrás de éstas. Además, es probable que tengan existan objetivos diferentes porque no necesariamente trabajan en la misma empresa y para los mismos intereses, por lo que la toma de decisiones en forma unificada es de vital importancia.

En este escenario la herramienta DRMLTool brindaría un mecanismo para eliminar la ambigüedad de las decisiones y la posibilidad de su unificación, además del tema de la escalabilidad, en este escenario habría que explorar la posibilidad de brindarle capacidades de trabajo colaborativo asistido por computador. Y esto no sólo resultaría beneficioso

solamente en proyectos open source, sino en general porque el diseño arquitectónico requiere del trabajo en equipo que normalmente falla por varios motivos, como la falta de flexibilidad, el ego del personal y la lealtad hacia una tecnología preferida, evitando que el equipo logre un consenso en las decisiones y su rationale. Algunos miembros del equipo constantemente intentan forzar su forma de hacer las cosas a los otros, en lugar de participar objetivamente en las discusiones Dasanayake et al. [31].

Finalmente, la construcción de la herramienta RADAR sintetiza de manera efectiva los elementos mínimos y necesarios que debe tener una herramienta de este tipo para documentar el rationale y las decisiones de diseño más importantes que se toman en el proceso de creación y mantenimiento de la Arquitectura Software.

Capítulo 6

Conclusiones, Limitaciones y Trabajo Futuro

En esta tesis se estudia la importancia de documentar el rationale arquitectónico detrás de las decisiones de diseño. Dado que el rationale es una pieza clave para apoyar la evolución del producto de software, este debe documentarse explícitamente y elevar su grado de relevancia para la documentación. Para lograr el objetivo en la documentación de la justificación de las decisiones arquitectónicas, en esta tesis se propone elevar las decisiones y su rationale hemos propuesto un lenguaje y una herramienta basada en sus elementos especificados capaz de registrar, y mantener las decisiones tomadas durante el proceso de construcción y mantenimiento de la arquitectura. Nuestro enfoque conecta los requisitos con las arquitecturas a través del rationale arquitectónico y decisiones de diseño, para que podamos establecer relaciones entre ellos.

6.1. Conclusiones

1. La investigación realizada nos ha permitido establecer que representar el rationale de las decisiones de diseño arquitectónico de manera ágil desde diferentes perspectivas, como es el código y una quinta vista de rationale en un enfoque de 4+1 vistas, ayuda a los arquitectos a comprender las razones de por qué la arquitectura es estructura de determinada forma, optimizando la futura toma de decisiones y permitiendo que estas convivan en armonía con las que permanecen vigentes, lo que resulta en una documentación que es más fácil de comunicar, mantener y

actualizar.

2. La investigación que realizamos acerca del rationale arquitectónico y su documentación, ha servido para entender que existen diferentes mecanismos que ayudan a capturar ese conocimiento, por ejemplo, meta-modelos, plantillas, herramientas visuales, anotaciones en el código, metodologías etc. Unas usando más conceptos que otras, ya sea por tecnología, facilidad de uso, conceptos más sofisticados, etc. Sin embargo, la demanda de software en la últimas décadas ha incrementado notablemente, cada día se necesita software funcionando en un tiempo de construcción corto, lo que ha llevado a la industria de software a omitir los mecanismos de documentación y calidad, aun cuando arquitectos, desarrolladores y demás interesados reconocen la necesidad de utilizarlos. Por ello se puede concluir que aun se carece de una estrategia efectiva que permita enlazar las prácticas de arquitectura con los ciclos de vida de desarrollo de software actuales, los cuales se han focalizado en que el producto alcance los aspectos funcionales y en tiempos muy mínimos en construcción.

3. Con base en los cambios arquitectónicos implementados y documentados por los ingenieros y las respectivas observaciones en este proceso utilizando la herramienta DRMLTool y su lenguaje, se observa que el promedio y la media favorecen la herramienta y su lenguaje con respecto a la eficiencia, efectividad y la comprensión de la Arquitectura Software y su rationale. Sin embargo, la prueba t-Student permitió definir qué resultados presentaban en realidad una significancia estadística, dando como resultado en el estudio de caso que el uso de la herramienta con información del rationale arquitectónico beneficia la comprensión de la Arquitectura Software y su rationale, ayudando a guiar la futura toma de decisiones en el diseño arquitectónico con una documentación que es más fácil de comunicar y mantener en el contexto ágil, por otra parte, la prueba arrojo que la significancia estadística de la eficiencia y efectividad para lograr los cambios arquitectónicos no es relevante.

4. A través del estudio de caso que se desarrollo en una unidad de desarrollo *inhouse*, para evaluar el modelo de rationale y su herramienta de modelado DRMLTool se pudo establecer que su uso ayuda a los arquitectos y partes interesadas a comprender mejor el rationale de las decisiones de diseño que se han tomado en un punto especifico del tiempo. Ayudando a preservar la arquitectura de software sin erosión y en registrar el conocimiento para la toma de futuras decisiones. Para la evaluación del modelo, en ésta tesis se ha desarrollado una herramienta prototipo la cual ha seguido un desarrollo basado en la ingeniería dirigida por modelos MDE, el cual es un enfoque que ha resultado de gran valor tanto para el desarrollo del producto, así como para soportar los principios de la herramienta para documentar el rationale arquitectónico de las principales decisiones de diseño de una manera

más sistemática. Al aplicar la herramienta en dos equipos de desarrollo, se pudo establecer que el lenguaje de modelado hace una diferencia en la comprensión del rationale, así como la efectividad para la toma de decisiones durante cambios significativos desde el punto de vista de la arquitectura.

5. La investigación muestra que trabajar con MDE permite precisar y realizar trasformaciones entre diferentes modelos, facilitando la construcción rápida de aplicación. Como un resultado de la evaluación realizada, se evidencia que este tipo de herramientas y en particular herramientas basadas en modelos, constituye un nuevo paso para compartir mecanismos con elementos estandarizados para la construcción de nuevas herramientas que benefician a la industria del software y comunidad científica. El enfoque MDE permite construir de manera rápida y eficaz un producto software a partir de la especificación de meta-modelos y por medio de generadores de código, generar el código que soporta la gestión de instancias de modelo. Es importante tener en cuenta que en este enfoque si el meta-modelo no es bien diseñado, acorde con los requisitos que demanda un producto, el resultado no será correcto y por supuesto, el tiempo invertido y sus demás recursos demandados habrían sido utilizados en vano. Pero esta es a su vez una ventaja, dado que el factor de intervención humano durante la generación de código es bajo, puesto que si hay un error en la especificación del meta-modelo que no fue advertido, se puede corregir y volver a generar, quitando cierto grado de dependencia del programador, como no sucede en otro tipo de aplicaciones, en las cuales si los modelos no fueron diseñados apropiadamente, el impacto sobre el producto final sería catastrófico, y el grado de dependencia del programador aumentaría para solucionar tales situaciones.

6.2. Limitaciones

El estudio, permitió evidencia que es necesario que el lenguaje permita escalar el modelado para un número grande de decisiones y sus relaciones. En cuanto a DRMLTool es importante que cuente con enlaces a fuentes externas de conocimiento, por ejemplo, UML, documentos de arquitectura, WIKIS, etc. También carece de una estructura que facilite la navegabilidad donde se pueda buscar el conocimiento documentado entre varias decisiones y su rationale. Dado que el estudio es un caso holístico, este también tiene limitaciones tales como las características propias de la empresa que hacen que no se pueda generalizar los resultados, particularmente porque la empresa antes del estudio trabaja con un foco importante en la especificación de la arquitectura de software, lo cual no necesariamente es una característica típica de proyectos ágiles en la región.

6.3. Trabajo Futuro

En esta tesis se han analizado algunos puntos que pueden ser tenidos en cuenta para trabajo futuro a corto y mediano plazo, entre ellos se incluye:

1. Extender el lenguaje para soportar el modelado de paquetes de decisiones (paquetes de preocupaciones y paquetes perspectivas), la trazabilidad con otros artefactos de la arquitectura, así como incorporar en la herramienta aspectos de versionado y modelado colaborativo.

2. Bajo el paradigma de *Documentación de Arquitectura centrada en el Rationale* , se busca construir un mecanismo que genere las anotaciones en el código, a partir del modelo de rationale generado por DRML y de manera viceversa se genere el modelo de rationale arquitectónico con las anotaciones en el código y mostrar estas relaciones visualmente. Para esto se haría uso paradigma de desarrollo MDE para relacionar por transformaciones los elementos de documentación del lenguaje DRML con las anotaciones de código desarrolladas por Dorado et al. [36], dado que el código sigue siendo la fuente más frecuente y confiable, en la que los desarrolladores encuentran información relevante a las justificaciones de las decisiones de diseño, particularmente en el enfoque ágil con foco en la arquitectura.

3. Se espera hacer un experimento controlado sobre la manera de especificar el rationale en DRML para evidenciar qué constructos son necesarios agregar al metamodelo, con el fin de definir reglas más completas semántica y sintácticamente, así como determinar qué elementos DRML dejar por fuera de la especificación formal.

Bibliografía

[1] *Ieee standard for a software quality metrics methodology*, IEEE Std 1061-1992 (1993), 1–96.

[2] *Introduction to iconix process*, pp. 1–20, Apress, Berkeley, CA, 2007.

[3] Pekka Abrahamsson, Muhammad Ali Babar, and Philippe Kruchten, *Agility and architecture: Can they coexist?*, IEEE Software **27** (2010), no. 2.

[4] Jonathan Aldrich, Craig Chambers, and David Notkin, *Archjava: connecting software architecture to implementation*, Proceedings of the 24th international conference on Software engineering, ACM, 2002, pp. 187–197.

[5] Rana Alkadhi, Manuel Nonnenmacher, Emitza Guzman, and Bernd Bruegge, *How do developers discuss rationale?*, 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2018, pp. 357–369.

[6] Scott Ambler et al., *The agile unified process (aup)*, Toronto, Canada. Recuperado de: http://www. ambysoft. com/unifiedprocess/agileUP. html (2005).

[7] Juan José Morales Arias and César Jesús Pardo Calvache, *Revisión sistemática de la integración de modelos de desarrollo de software dirigido por modelos y metodologías ágiles*, Informador técnico **80** (2016), no. 1, 87–99.

[8] Paris Avgeriou, Philippe Kruchten, Patricia Lago, Paul Grisham, and Dewayne Perry, *Architectural knowledge and rationale: issues, trends, challenges*, ACM SIGSOFT Software Engineering Notes **32** (2007), no. 4, 41–46.

[9] Muhammad Ali Babar, Ian Gorton, and Barbara Kitchenham, *A framework for supporting architecture knowledge and rationale management*, Rationale Management in Software Engineering, Springer, 2006, pp. 237–254.

[10] Felix Bachmann, Len Bass, Paul Clements, David Garlan, James Ivers, M. Little, Paulo Merson, Robert Nord, and Judith Stafford, *Documenting software architectures: Views and beyond*, second ed., Addison-Wesley Professional, 2010.

[11] Mario Barbacci, Robert Ellison, Anthony Lattanze, Judith Stafford, Charles Weinstock, and William Wood, *Quality attribute workshops (qaws)*, Tech. Report CMU/SEI-2003-TR-016, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2003.

[12] Mario R Barbacci, Robert J Ellison, Anthony Lattanze, Judith Stafford, Charles B Weinstock, and William Wood, *Quality attribute workshops*, (2002).

[13] Len Bass and Paul Clements, *Rick kazman software architecture in practice chap. 4, 5 boston*, 2003.

[14] Kent Beck, *Embracing change with extreme programming*, (2004).

[15] Manoj Bhat, Klym Shumaiev, and Florian Matthes, *Towards a framework for managing architectural design decisions*, Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings, 2017, pp. 48–51.

[16] Jorge Biolchini, Paula Gomes Mian, Ana Candida Cruz Natali, and Guilherme Horta Travassos, *Systematic review in software engineering*, System Engineering and Computer Science Department COPPE/UFRJ, Technical Report ES **679** (2005), no. 05, 45.

[17] G. Borrego, *Condensing architectural knowledge from unstructured textual media in agile gsd teams*, 2016 IEEE 11th International Conference on Global Software Engineering Workshops (ICGSEW), Aug 2016, pp. 69–72.

[18] Lars Bratthall, Enrico Johansson, and Björn Regnell, *Is a design rationale vital when predicting change impact?–a controlled experiment on software architecture evolution*, International conference on product focused software process improvement, Springer, 2000, pp. 126–139.

[19] Pearl Brereton, Barbara A Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil, *Lessons from applying the systematic literature review process within the software engineering domain*, Journal of systems and software **80** (2007), no. 4, 571–583.

[20] A. Bucaioni, A. Cicchetti, F. Ciccozzi, S. Mubeen, and M. Sjodin, *A metamodel for the rubus component model: Extensions for timing and model transformation from east-adl*, IEEE Access (2017).

[21] Mario Bunge, *La ciencia y su método y su filosofía*, (siglo XXI, 2000).

[22] Janet E Burge, John M Carroll, Raymond McCall, and Ivan Mistrik, *Rationale-based software engineering*, Springer, 2008.

[23] Rafael Capilla, Francisco Nava, and Juan C Duenas, *Modeling and documenting the evolution of architectural design decisions*, Second Workshop on Sharing and Reusing Architectural Knowledge-Architecture, Rationale, and Design Intent (SHARK/ADI'07: ICSE Workshops 2007), IEEE, 2007, pp. 9–9.

[24] Rafael Capilla, Francisco Nava, Sandra Pérez, and Juan Dueñas, *A web-based tool for managing architectural design decisions*, ACM SIGSOFT Software Engineering Notes **31** (2006).

[25] María Celeste Carignano, Silvio Gonnet, and Horacio Leone, *A model to represent architectural design rationale*, 2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture, IEEE, 2009, pp. 301–304.

[26] Meiru Che, *An approach to documenting and evolving architectural design decisions*, 2013 35th International Conference on Software Engineering (ICSE), IEEE, 2013, pp. 1373–1376.

[27] ———, *Managing architectural design decision documentation and evolution*, Ph.D. thesis, 2014.

[28] Jane Cleland-Huang, Mehdi Mirakhorli, Adam Czauderna, and Mateusz Wieloch, *Decision-centric traceability of architectural concerns*, Traceability in Emerging Forms of Software Engineering (TEFSE), 2013 International Workshop on, IEEE, 2013, pp. 5–11.

[29] Paul Clements and Len Bass, *Using business goals to inform a software architecture*, Requirements Engineering Conference (RE), 2010 18th IEEE International, IEEE, 2010, pp. 69–78.

[30] Paul C Clements, *A survey of architecture description languages*, Proceedings of the 8th international workshop on software specification and design, IEEE Computer Society, 1996, p. 16.

[31] Sandun Dasanayake, Jouni Markkula, Sanja Aaramaa, and Markku Oivo, *An empirical study on collaborative architecture decision making in software teams*, European Conference on Software Architecture, Springer, 2016, pp. 238–246.

[32] P. De Jong, J. M. E. M. Van Der Werf, M. Van Steenbergen, F. Bex, and M. Brinkhuis, *Evaluating design rationale in architecture*, 2019 IEEE International Conference on Software Architecture Companion (ICSA-C), March 2019, pp. 145–152.

[33] Lakshitha De Silva and Dharini Balasubramaniam, *Controlling software architecture erosion: A survey*, Journal of Systems and Software **85** (2012), no. 1, 132–151.

[34] Diego Dermeval, Jaelson Castro, Carla Silva, João Pimentel, Ig Ibert Bittencourt, Patrick Brito, Endhe Elias, Thyago Tenório, and Alan Pedro, *On the use of metamodeling for relating requirements and architectural design decisions*, Proceedings of the 28th Annual ACM Symposium on Applied Computing, ACM, 2013, pp. 1278–1283.

[35] L. Dobrica and E. Niemela, *A survey on software architecture analysis methods*, IEEE Transactions on Software Engineering **28** (2002), no. 7, 638–653.

[36] Santiago Hyun Dorado and Julio Ariel Hurtado, *Documenting architectural rationale using source code annotations: An exploratory study*, Proceedings of 28th International Conference on Software Engineering and Data Engineering (Frederick Harris, Sergiu Dascalu, Sharad Sharma, and Rui Wu, eds.), EPiC Series in Computing, vol. 64, EasyChair, 2019, pp. 204–214.

[37] Allen H Dutoit, Raymond McCall, Ivan Mistrík, and Barbara Paech, *Rationale management in software engineering: Concepts and techniques*, Rationale management in software engineering, Springer, 2006, pp. 1–48.

[38] Davide Falessi, Rafael Capilla, and Giovanni Cantone, *A value-based approach for documenting design decisions rationale: a replicated experiment*, Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge, 2008, pp. 63–70.

[39] David Garlan, Robert T Monroe, and David Wile, *Acme: Architectural description of component-based systems*, Foundations of component-based systems **68** (2000), 47–68.

[40] Fabian Gilson and Vincent Englebert, *Rationale, decisions and alternatives traceability for architecture design*, Proceedings of the 5th European Conference on Software Architecture: Companion Volume, ACM, 2011, p. 4.

[41] Irit Hadar, Sofia Sherman, Ethan Hadar, and John J Harrison, *Less is more: Architecture documentation for agile development*, Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on, IEEE, 2013, pp. 121–124.

[42] Neil B Harrison and Paris Avgeriou, *How do architecture patterns and tactics interact? a model and annotation*, Journal of Systems and Software **83** (2010), no. 10, 1735–1758.

[43] Neil B Harrison, Erich Gubler, and Danielle Skinner, *Architectural decision-making in open-source systems–preliminary observations*, 2016 1st International Workshop on Decision Making in Software ARCHitecture (MARCH), IEEE, 2016, pp. 16–21.

[44] Tom-Michael Hesse, Arthur Kuehlwein, Barbara Paech, Tobias Roehm, and Bernd Bruegge, *Documenting implementation decisions with code annotations.*, SEKE, 2015, pp. 152–157.

[45] Tom-Michael Hesse, Arthur Kuehlwein, and Tobias Roehm, *Decdoc: A tool for documenting design decisions collaboratively and incrementally*, Decision Making in Software ARCHitecture (MARCH), 2016 1st International Workshop on, IEEE, 2016, pp. 30–37.

[46] Rich Hilliard, *Ieee-std-1471-2000 recommended practice for architectural description of software-intensive systems*, IEEE, http://standards. ieee. org **12** (2000), no. 16-20, 2000.

[47] John Horner and Michael E Atwood, *Effective design rationale: understanding the barriers*, Rationale management in software engineering, Springer, 2006, pp. 73–90.

[48] Anton Jansen, Paris Avgeriou, and Jan Salvador van der Ven, *Enriching software architecture documentation*, Journal of Systems and Software **82** (2009), no. 8, 1232–1248.

[49] Anton Jansen and Jan Bosch, *Software architecture as a set of architectural design decisions*, Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on, IEEE, 2005, pp. 109–120.

[50] Luis Freddy Muñoz Sanabria Julio Ariel Hurtado, *Arquitectura del software: Criterios para una arquitectura ágil*.

[51] Rick Kazman, *Tool support for architecture analysis and design*, Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints' 96) on SIGSOFT'96 workshops, Citeseer, 1996, pp. 94–97.

[52] Rick Kazman, Mark Klein, and Paul Clements, *Atam: Method for architecture evaluation*, Tech. Report CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2000.

[53] Demetrios Kellari, Edward F Crawley, and Bruce G Cameron, *Architectural decisions in commercial aircraft from the dc-3 to the 787*, Journal of Aircraft **55** (2018), no. 2, 792–804.

[54] Andy Kellens, Carlos Noguera, Kris De Schutter, Coen De Roover, and Theo D'Hondt, *Co-evolving annotations and source code through smart annotations*, 2010 14th European Conference on Software Maintenance and Reengineering, IEEE, 2010, pp. 117–126.

[55] Anja Kleebaum, Jan Ole Johanssen, Barbara Paech, and Bernd Bruegge, *Teaching rationale management in agile project courses*, (2019).

[56] P. Kruchten, H. Obbink, and J. Stafford, *The past, present, and future for software architecture*, IEEE Software **23** (2006), no. 2, 22–30.

[57] P. B. Kruchten, *The 4+1 view model of architecture*, IEEE Software **12** (1995), no. 6, 42–50.

[58] Philippe Kruchten, Rafael Capilla, and Juan Carlos Dueñas, *The decision view's role in software architecture practice*, IEEE software **26** (2009), no. 2, 36–42.

[59] Socrates Veridiano Faria Lopes and Plinio Thomaz Aquino Junior, *Architectural design group decision-making in agile projects*, 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), IEEE, 2017, pp. 210–215.

[60] Ioanna Lytra, Gerhard Engelbrecht, Daniel Schall, and Uwe Zdun, *Reusable architectural decision models for quality-driven decision support: A case study from a smart cities software ecosystem*, 2015 IEEE/ACM 3rd International Workshop on Software Engineering for Systems-of-Systems, IEEE, 2015, pp. 37–43.

[61] Carlos Marín, Juan Cueva Lovelle, Oscar Sanjuán, and Paulo Gaona-García, *Desarrollo de un lenguaje de dominio específico para sistemas de gestión de aprendizaje y su herramienta de implementación "kiwidsm" mediante ingeniería dirigida por modelos*, Ingeniería **15** (2010).

[62] Carlos Marín, Paulo Gaona-García, Juan Cueva Lovelle, and Oscar Sanjuán, *Aplicación de ingeniería dirigida por modelos (mda), para la construcción de una herramienta de modelado de dominio específico (dsm) y la creación de módulos en sistemas de gestión de aprendizaje (lms) independientes de la plataforma*, Dyna **78** (2011), 43–52.

[63] Nenad Medvidovic and Richard N Taylor, *A classification and comparison framework for software architecture description languages*, IEEE Transactions on software engineering **26** (2000), no. 1, 70–93.

[64] Stephen J Mellor, Kendall Scott, Axel Uhl, and Dirk Weise, *Mda distilled: principles of model-driven architecture*, Addison-Wesley Professional, 2004.

[65] Oscar Pedreira, Félix García, Nieves Brisaboa, and Mario Piattini, *Gamification in software engineering–a systematic mapping*, Information and Software Technology **57** (2015), 157–168.

[66] Flor de Maria Hernández Pérez and Julio Ariel Hurtado Algeria, *Difficulties and challenges in the incorporation of architectural practices*, Sistemas & Telemática **14** (2016), no. 38, 74–86.

[67] Francisco J Pino, Félix García, and Mario Piattini, *Revisión sistemática de mejora de procesos software en micro, pequeñas y medianas empresas*, REICIS. Revista Española de Innovación, Calidad e Ingeniería del Software **2** (2006), no. 1.

[68] Georgios Plataniotis, Qin Ma, Erik Proper, and Sybren de Kinderen, *Traceability and modeling of requirements in enterprise architecture from a design rationale perspective*, 2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS), IEEE, 2015, pp. 518–519.

[69] Ghulam Rasool, Ilka Philippow, and Patrick Mäder, *Design pattern recovery based on annotations*, Advances in Engineering Software **41** (2010), no. 4, 519–526.

[70] Carlos Billy Reynoso, *Introducción a la arquitectura de software*, Universidad de Buenos Aires **33** (2004).

[71] B. Rogers, J. Gung, Y. Qiao, and J. E. Burge, *Exploring techniques for rationale extraction from existing documents*, 2012 34th International Conference on Software Engineering (ICSE), June 2012, pp. 1313–1316.

[72] Benjamin Rogers, Yechen Qiao, James Gung, Tanmay Mathur, and Janet E Burge, *Using text mining techniques to extract rationale from existing documentation*, Design Computing and Cognition'14, Springer, 2015, pp. 457–474.

[73] Per Runeson and Martin Höst, *Guidelines for conducting and reporting case study research in software engineering*, Empirical software engineering **14** (2009), no. 2, 131.

[74] Ahmed E Sabry, *Decision model for software architectural tactics selection based on quality attributes requirements*, Procedia Computer Science **65** (2015), 422–431.

[75] Fabiola Sanz, Raúl Galvez, Cristian Rusu, Silvana Roncagliolo, Virginica Rusu, César A Collazos, Juan Pablo Cofré, Aníbal Campos, and Daniela Quiñones, *A set of usability heuristics and design recommendations for u-learning applications*, Information Technology: New Generations, Springer, 2016, pp. 983–993.

[76] Jeff Sauro and James R. Lewis, *Chapter 8 - standardized usability questionnaires*, Quantifying the User Experience (Second Edition) (Jeff Sauro and James R. Lewis, eds.), Morgan Kaufmann, Boston, second edition ed., 2016, pp. 185 – 248.

[77] Douglas C Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann, *Pattern-oriented software architecture, patterns for concurrent and networked objects*, vol. 2, John Wiley & Sons, 2013.

[78] Kurt Schneider, *Rationale as a by-product*, Rationale Management in Software Engineering, Springer, 2006, pp. 91–109.

[79] Mary Shaw and David Garlan, *Software architecture: perspectives on an emerging discipline*, vol. 1, Prentice Hall Englewood Cliffs, 1996.

[80] Simon J Buckingham Shum, Albert M Selvin, Maarten Sierhuis, Jeff Conklin, Charles B Haley, and Bashar Nuseibeh, *Hypermedia support for argumentation-based rationale*, Rationale management in software engineering, Springer, 2006, pp. 111–132.

[81] Janice Singer, *Practices of software maintenance*, Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272), IEEE, 1998, pp. 139–145.

[82] Ian Sommerville, *Software engineering 9th edition*, Addison-Wesley (2011).

[83] Subhashree Sundaravadivelu, Aparajithan Vaidyanathan, and Srini Ramaswamy, *Knowledge reuse of software architecture design decisions and rationale within the enterprise*, 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), IEEE, 2014, pp. 253–261.

[84] Ryo Suzuki, *Poster: Interactive and collaborative source code annotation*, 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 2, IEEE, 2015, pp. 799–800.

[85] Antony Tang, Muhammad Ali Babar, Ian Gorton, and Jun Han, *A survey of architecture design rationale*, Journal of systems and software **79** (2006), no. 12, 1792–1804.

[86] Minh Tu Ton That, Salah Sadou, Flavio Oquendo, and Régis Fleurquin, *Preserving architectural decisions through architectural patterns*, Automated Software Engineering **23** (2016), no. 3, 427–467.

[87] Salvador Trujillo, Maider Azanza, Oscar Diaz, and Rafael Capilla, *Exploring extensibility of architectural design decisions*, Second Workshop on Sharing and Reusing Architectural Knowledge-Architecture, Rationale, and Design Intent (SHARK/ADI'07: ICSE Workshops 2007), IEEE, 2007, pp. 10–10.

[88] Jan Salvador van der Ven and Jan Bosch, *Busting software architecture beliefs: A survey on success factors in architecture decision making*, Software Engineering and Advanced Applications (SEAA), 2016 42th Euromicro Conference on, IEEE, 2016, pp. 42–49.

[89] Jan Salvador van der Ven, Anton G. J. Jansen, Jos A. G. Nijhuis, and Jan Bosch, *Design decisions: The bridge between rationale and architecture*, pp. 329–348, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[90] Hans van Vliet and Antony Tang, *Decision making in software architecture*, Journal of Systems and Software **117** (2016), 638–644.

[91] Stefan Voigt, Detlef Hüttemann, and Andreas Gohr, *sprintdoc: concept for an agile documentation tool*, Information Systems and Technologies (CISTI), 2016 11th Iberian Conference on, IEEE, 2016, pp. 1–6.

[92] Wei Wang and Janet E Burge, *Using rationale to support pattern-based architectural design*, Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge, 2010, pp. 1–8.

[93] Rob Wojcik, Felix Bachmann, Len Bass, Paul Clements, Paulo Merson, Robert Nord, and William Wood, *Attribute-driven design (add), version 2.0*, Tech. Report CMU/SEI-2006-TR-023, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2006.

[94] Robert K. Yin, *Case study research desing and methods third edition*, vol. 5, 2003.

[95] Olaf Zimmermann, *Architectural decision identification in architectural patterns*, Proceedings of the WICSA/ECSA 2012 Companion Volume, 2012, pp. 96–103.