# Una Herramienta Software basada en Anotaciones de Código para documentar el Rationale Arquitectónico.


Universidad del Cauca

Monografía de Trabajo de Grado para optar el título de
*Ingeniero de sistemas*

Santiago Hyun Dorado

**Director: Ing. Milton Javier Sánchez Grueso**

**Codirector: PhD. Julio Ariel Hurtado Alegría**

*Universidad del Cauca*
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Ingeniería de Sistemas
Investigación y Desarrollo en Ingeniería del Software IDIS
Ingeniería del Software
Popayán, 21 de diciembre de 2018

# Agradecimientos

Este trabajo no sólo reúne los esfuerzos realizados para esta investigación, en él, se encuentran todos los sacrificios y las energías que se han dedicado por más de 5 años de carrera universitaria. En esta carrera, se han construido grandes retos y motivaciones que constantemente me hicieron crecer como persona y profesional. Este estudio me permitió rodearme de personas que constantemente me brindaron su apoyo, sus buenas energías, sus comentarios y sus críticas, y junto con mi familia me apoyaron constantemente para salir adelante y no desfallecer ante la desmotivación, el cansancio o las malas energías.

Quiero agradecer primeramente a mi familia, por ser la base principal de motivación que me hizo sacar adelante este proyecto, por enseñarme que la vida y los esfuerzos que realizamos no tienen un fin diferente a querer brindarles mejores oportunidades y un mejor estilo de vida a las personas que más aprecias y que siempre están contigo brindándote su apoyo. Quiero agradecer a mi madre María Eugenia, por enseñarme que los esfuerzos que realizamos en este mundo siempre tienen que tener como objetivo, aportar a la sociedad y dejar a través de la educación un legado a las nuevas generaciones. También quiero agradecer a mi hermana Susana por siempre estar dispuesta a aprender de los concejos que a mi corta edad le puedo brindar. Agradezco también a mi compañera sentimental Laura, por animarme siempre en los momentos de desmotivación con su cariño y sus consejos.

También quiero agradecer a mi director de tesis Julio Ariel Hurtado por orientarme siempre con sus consejos, por brindarme más que una asesoría, una gran amistad y una gran admiración por su trabajo y sus valores como persona y profesional. Gracias a su apoyo incondicional y a su forma de enseñar, pude conocer los temas académicos que realmente me motivan a seguir estudiando y trabajando por la academia. Y aunque sus enseñanzas académicas fueron muchas, me quedaran para siempre los consejos que me brindó como persona y amigo, con la humildad y la humanidad que lo caracteriza.

A todas las personas que hicieron parte de esta tesis les doy infinitas gracias, en especial a las personas que tuvieron la voluntad de participar en el proceso de evaluación de esta herramienta, gracias a los ingenieros: Wilson Pantoja, Sebastián Landínez, Oscar López, Alejandro Bolaños, Milciades Ordoñez, Eduar Troyano, Daniel Pusil, Santiago Pérez y a los tesistas Santiago Garcia, Carlos Molano, Javier Ágredo y Edwin Marulanda; sin ellos y sin todos los docentes que hicieron parte de este proceso de aprendizaje a través de esta carrera, la culminación de este trabajo no hubiera sido posible.

# Tabla de contenido

# Índice de tablas

# Índice de figuras

# Capítulo 1: Introducción

En esta sección 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.

## Problemática y justificación

El desarrollo de software es una actividad compleja, la cual requiere mucho conocimiento implícito y explícito por parte de los involucrados. Este conocimiento es generado y utilizado durante el diseño de la estructura de un sistema intensivo en software y necesita ser compartido y reutilizado por otros participantes en diferentes fases del ciclo de vida del software [1]. En la actualidad existen diferentes modelos de desarrollo de software que tratan de aprovechar ese conocimiento implícito y explícito, con el objetivo de adaptarse a los procesos de las organizaciones y a los cambios constantes exigidos por los clientes y usuarios finales. Sin embargo, una actividad común en todos los procesos de desarrollo es el diseño de la arquitectura del software. Según P. Kruchten et al. [2], la arquitectura de software involucra la estructura y organización de los componentes software, cómo éstos se comunican entre sí, creando subsistemas que interactúan unos con otros, con el objetivo de conformar sistemas más complejos. Por otra parte, A. Jansen et al. [3] definen la arquitectura como un conjunto de decisiones que involucran la adición, eliminación o modificación a la arquitectura, el Rationale (los porqué de las decisiones), las reglas de diseño, las restricciones de diseño y nuevas funcionalidades que representan uno o más requerimientos en una arquitectura software planteada. Generalmente estas decisiones arquitecturales son el resultado de un proceso de diseño durante las fases iniciales de construcción, pero se retoman e impactan la evolución de la solución, por lo cual es indispensable comenzar modelando una buena arquitectura dado que la escalabilidad y mantenibilidad de un sistema, entre otros, son factores clave para conseguir una aplicación con un ciclo de vida mucho más largo [1].

Según P. Kruchten et al. [2], las empresas de software requieren sistemas abiertos-cerrados que puedan ser fáciles de extender, idealmente sin necesidad de cambiar el código fuente ya escrito y que al mismo tiempo sean fáciles de mantener en caso de algún fallo. Un software que pueda extender sus funcionalidades a futuro, representa un gran valor para las empresas, ya que las necesidades de las organizaciones y de las personas cambian en el transcurso del ciclo de vida de la aplicación. Sin embargo, la demanda acelerada de productos software obliga a los arquitectos y desarrolladores de software, a incurrir en malas prácticas de desarrollo y a pasar a un segundo plano la documentación adecuada, y cuando se realiza ésta suele ser muy extensa y pesada, causando que no se utilice, actualice o mejore.

Un problema muy común en la definición de la arquitectura es la falta de entendimiento de los requerimientos funcionales y de calidad, esto generalmente provoca con el tiempo una desviación de las funciones para las que fue diseñada la arquitectura originalmente [4], esto termina causando que el mal entendimiento de la arquitectura derive en la erosión arquitectónica [5], esta se refleja cuando la arquitectura de un sistema no ha sido entendida y se provoca el desgaste funcional de un componente causando que con el tiempo no se pueda modificar, o que la modificación de este implique cambios en el resto del sistema.

Las decisiones de diseño arquitecturales que se toman en el transcurso del proceso de desarrollo quedan implícitas en el código [6], haciendo que el mantenimiento y la recuperación de la arquitectura se vuelva mucho más compleja a futuro y traiga consigo los siguientes problemas [3]:

1. Cruce de decisiones de diseño: Algunas decisiones de diseño son transversales a toda la arquitectura causando que cualquier cambio afecte otras partes de ésta.

2. Violación de reglas y restricciones de diseño: Frecuentemente en la evolución de la arquitectura se violan las reglas y restricciones de diseño causando una desviación de la arquitectura, lo que comúnmente produce problemas como el aumento en los costos del mantenimiento [5].

3. Decisiones de diseño obsoletas que no son removidas: Si no se eliminan las decisiones de diseño obsoletas la arquitectura software tenderá a erosionarse más rápido.

Las decisiones realizadas en el proceso de diseño de la arquitectura tienen un alto impacto en el cumplimento de los objetivos de calidad [7], ya que generalmente las decisiones de diseño que se toman sobre una arquitectura van dirigidas a cumplir con algunos atributos de calidad. Sin embargo, no es posible encontrar un diseño arquitectural definitivo que cumpla con todas las cualidades necesarias, es por eso que los arquitectos tienen que escoger de un gran número de posibilidades de diseño, uno que sea óptimo para cumplir con los requisitos del contexto y el dominio del problema, esta es una actividad que frecuentemente va más allá de las capacidades de las personas debido a la alta complejidad de los sistemas en la actualidad [8].

El diseño de la arquitectura en los proyectos desarrollados con un enfoque ágil presenta un problema adicional, cuando no se documenta adecuadamente y se quieren tomar decisiones posteriormente durante el mantenimiento [3]. Las decisiones de diseño tomadas en ocasiones por los desarrolladores quedan implícitas en el código fuente, o en muchas ocasiones quedan plasmadas en artefactos no formales, como correos electrónicos, documentos de texto planos, chats, etc [6]. Normalmente, cuando se documenta la arquitectura de un sistema, el documento termina siendo extenso y poco conexo con los elementos de la implementación, lo que resulta en que no se consulte y actualice durante la evolución del producto software [2], causando que las modificaciones a un sistema durante su ciclo de vida erosionen su arquitectura, cambiando su estructura y disminuyendo su rendimiento [5]. La documentación de la arquitectura es un aspecto que no se toma con la importancia que se debe por los implicados del software, debido a que no hace parte de los artefactos que generan valor inmediato al cliente. Por lo tanto, las decisiones de arquitectura no se ven reflejadas en modelos arquitectónicos o documentos estructurados. Esto es aún más crítico cuando no se ven reflejadas las razones de la toma de ciertas decisiones de diseño que dieron forma a la arquitectura. A estas razones o justificaciones de diseño , A. Dutoit et al. [9] les denominaron Rationale Arquitectónico y son la base para el entendimiento de la organización de los componentes en una arquitectura de software y por tanto clave para mantener en el tiempo la integridad conceptual [10].

La comprensión del diseño actuales el principal requisito para lograr una buena capacidad para el mantenimiento y la evolución de un sistema, debido a que los cambios realizados en el diseño dependen en gran medida del diseño anterior [11]. Algunas propuestas se orientan hacia la descripción liviana de la arquitectura, ofreciendo alternativas híbridas, donde el enfoque disciplinado de la arquitectura se incorpora en el proceso ágil [3]. En muchas ocasiones las

decisiones arquitecturales se encuentran implícitas en sentencias de código muy puntuales y generalmente esto no se ve reflejado en ninguna parte de la documentación [12].

Por lo anterior, en este trabajo se presenta una herramienta basada en anotaciones de código fuente para capturar el Rationale Arquitectónico, con el objetivo de ser utilizada por arquitectos y desarrolladores de software que tengan la intención de documentar de una manera ágil y efectiva las razones detrás de las decisiones de diseño que se toman a nivel arquitectural. Además, el modelo definido por la herramienta de anotaciones de código se basa en atributos de calidad, tácticas, patrones de arquitectura y diferentes alternativas de diseño propuestas por los arquitectos, las cuales sirven de base para realizar la evolución de una arquitectura a través del conocimiento implícito en el desarrollo de una manera más eficiente y segura.

# Objetivos

## Objetivo general

Construir una herramienta software basada en anotaciones de código fuente para documentar el Rationale Arquitectónico ARAT (Architectural Rationale Annotations Tool:  por sus siglas en inglés), que soporte en un entorno integrado de desarrollo (IDE), el registro y administración de decisiones de diseño arquitectónicas a nivel de código fuente.

## Objetivos específicos

OE1: Caracterizar las estrategias, estructuras y lenguajes utilizados para documentar el Rationale Arquitectónico a nivel de código fuente.

OE2: Diseñar un modelo de anotaciones ARAT en el código fuente en un entorno de desarrollo que permita el registro y administración de decisiones de diseño arquitectónicas a nivel de código fuente.

OE3: Implementar ARAT como un plugin en un entorno integrado de desarrollo (IDE) y validar su utilidad práctica a través de un experimento controlado con ingenieros de software de la región.

# Metodología de investigación

Para el desarrollo de este proyecto se definieron 4 etapas, las cuales buscaban seguir el modelo científico que comienza con una observación sistemática para apreciar los nexos que hay en el estado del arte, seguido de una formulación de hechos no precisamente correctos, con el fin de ser contrastados con la realidad y determinar su veracidad. Estas etapas adaptaron el enfoque presentado por Mario Bunge [13]  a la ingeniería de software. . Esta metodología permitió definir el curso de acción para alcanzar los objetivos de investigación propuestos.

## Exploración técnica

Se realizó una búsqueda preliminar de artículos relacionados con el uso de anotaciones de código fuente como herramienta para documentar conocimiento implícito en el desarrollo de software, con el objetivo de tener un acercamiento al planteamiento del problema.

## Establecimiento de hechos

Para establecer las hipótesis relacionadas con el tema de investigación se realizó una revisión de la literatura, con el objetivo de revisar la problemática actual relacionada con la documentación del

Rationale Arquitectónico. Además, se revisó en la literatura si las anotaciones de código fuente han sido utilizadas para capturar decisiones de diseño arquitecturales, el Rationale Arquitectónico o cualquier otra información implícita resultante del diseño, el desarrollo o la evolución de la arquitectura. De esta fase se desprendieron dos actividades:

1. Revisión de la literatura con el objetivo de buscar, analizar y seleccionar los trabajos relacionados a temas de anotaciones de código fuente y documentación del Rationale arquitectónico.
2. Establecimiento de hipótesis mediante el análisis previo de los trabajos relacionados.

Para esta fase, se tomó y adaptó la guía para realizar revisiones sistemáticas de la literatura en ingeniería de software propuestas por Kitchenham et al. [14] con una menor formalidad en la conducción y el reporte de la revisión, acompañada de una búsqueda basada en la traza de las referencias. A continuación, se muestran las actividades realizadas en el proceso de revisión de la literatura:

1. Planeando la revisión
   1.1. Identificar la necesidad de una revisión
   1.2. Especificar la pregunta de investigación
2. Conduciendo la revisión
   2.1. Identificación de la investigación
   2.2. Selección de estudios
   2.3. Síntesis de datos
3. Resultados

## Construcción del modelo teórico

En esta etapa se define los pasos seguidos para analizar, diseñar, implementar y probar el modelo de anotaciones de código fuente con información del Rationale arquitectónico. Esta fase siguió un enfoque constructivista (comprender haciendo) [15] a través del refinamiento y prototipado incremental:

1. Documentación y capacitación sobre programación con anotaciones de código.
2. Acercamiento al primer modelo de anotaciones de código con información del Rationale arquitectónico, a través de los trabajos relacionados que fueron seleccionados.
3. Implementación del modelo a través de una herramienta software (plugin .jar) encargada de capturar la información de las anotaciones y generar reportes estructurados.
4. Pruebas de la herramienta con ingenieros de software.
5. Evolución del modelo de anotaciones mediante las sugerencias y los estudios relacionados.

Para el desarrollo del componente software (plugin .jar) se siguió un enfoque ágil siguiendo las pautas de documentación dela metodología ICONIX [16]. Esta metodología de desarrollo de software es un proceso liviano cuya estructura tiene el estilo de una receta de cocina, la cual describe cómo ir desde los casos de uso hasta el código fuente. Este proceso tiene como premisa “Eliminar la ambigüedad de los requisitos para realizar un diseño limpio”.

## Evaluación del modelo

Esta fase de la metodología de investigación tiene como objetivo evaluar la implementación del modelo de anotaciones con información del Rationale, propuesto en la fase descrita anteriormente.

Para evaluar el modelo, se definen siguientes actividades centrales que nos permiten evaluar la utilidad de la herramienta con respecto a las necesidades de los desarrolladores y arquitectos de software:

1. Desarrollo de un estudio exploratorio en el cual se realiza un quasi-experimento, con el objetivo de obtener rápidamente retroalimentación por parte de ingenieros de software que tienen relación con la industria y la academia, con un modelo de anotaciones sencillo que permitió evolucionar a futuro según los resultados y la experiencia de los participantes del quasi-experimento.
2. Se realiza un experimento controlado con ingenieros de software relacionados con la industria y la academia, con el modelo de anotaciones modificado con algunas de las sugerencias de los participantes del estudio exploratorio y los trabajos relacionados en el estado del arte.

Para la planificación, el diseño, la ejecución y el análisis de resultados del quasi-experimento y el experimento controlado, se siguen las guías propuestas por Jedlitschka et al. [17], para la realización y el reporte de experimentos en ingeniería de software.

## Documentación y socialización

En esta fase se realiza toda la documentación que sustenta el desarrollo del modelo de anotaciones de código con información del Rationale Arquitectónico y se sustenta de manera presencial todos los aportes investigativos realizados. Las actividades que componen esta fase son:

1. Elaboración de la monografía: El proceso investigativo se registra a través de la monografía de grado, en donde también se documentan los anexos que dan soporte a los resultados y datos que se muestran a lo largo de este documento.
2. Elaboración de artículos: Se elabora un artículo corto en español para documentar los resultados del estudio exploratorio (Ver Anexo N° 5: *Artículo corto en español*) y otro artículo completo en inglés para publicar el procedimiento experimental y los resultados en una revista indexada (Ver Anexo N° 6: *Artículo completo en inglés*), estos artículos se hicieron en la realización y la aplicación del prototipo final de la herramienta de anotaciones de código con información del Rationale Arquitectónico.
3. Socialización: Se hace la preparación de los resultados, con el objetivo de realizar la sustentación de los aportes investigativos en el desarrollo del modelo de anotaciones. La primera socialización se hace dentro de las instalaciones de la Universidad del Cauca en el semillero de investigación IRIS, frente a los integrantes del semillero, entre ellos estudiantes y docentes de la facultad. Los resultados de este estudio también se comparten en el primer encuentro interno de semilleros organizado por la Vicerrectoría de Investigaciones en el 2018 y del cual se obtiene el octavo lugar con más de 50 ponencias participantes. Finalmente se hace la ponencia de este trabajo en representación de la Universidad del Cauca, en el evento más importante de informática en el país, el 13 Congreso Colombiano de Computación. También se realiza la socialización del trabajo de investigación a profesores invitados desde diferentes universidades, entre ellos David Benavides de la universidad de Sevilla España y David Garlan de la universidad Carnegie Mellon Pensilvania.

## Estructura del documento

Este documento se encuentra organizado mediante 5 capítulos, en los cuales se describe el proceso investigativo realizado para la implementación del modelo de anotaciones de código fuente con información del Rationale Arquitectónico. A continuación, se describen los capítulos siguientes a la Introducción:

Capítulo 2 - Marco teórico y trabajos relacionados. En este capítulo se explican los conceptos clave para entender el contexto y el dominio del problema que se quiere abordar en este estudio, se especifican los términos relacionados con arquitecturas de software, la documentación del Rationale Arquitectónico y las anotaciones de código fuente. Además, se realiza una revisión de la literatura con el objetivo de buscar, analizar y clasificar los trabajos directamente relacionados con la documentación del Rationale Arquitectónico o el uso de anotaciones de código fuente para registrar aspectos relacionados con las decisiones de diseño arquitecturales. Finalmente se presentan los resultados del análisis de los estudios relacionados con el objetivo de diferenciar los aportes investigativos entre las propuestas que están en el estado del arte y el presente trabajo de investigación.

Capítulo 3 – Definición del modelo de anotaciones de código con información del Rationale Arquitectónico. Este capítulo describe el proceso que parte del análisis de los trabajos relacionados con Rationale Arquitectónico y las anotaciones de código, se obtiene un modelo conceptual que representa el diseño del Rationale Arquitectónico y finalmente se realiza la implementación del modelo a través de una herramienta software desarrollada como plugin .jar mediante el lenguaje de programación Java.

Capítulo 4 – Evaluación del modelo: En este capítulo se muestra el proceso investigativo realizado para la elaboración de un estudio exploratorio y un experimento controlado, los cuales nos permitieron validar la utilidad de la herramienta software de manera cualitativa y cuantitativa.

Capítulo 5 – Conclusiones, limitaciones y trabajos futuros: Para finalizar, en este capítulo se presentan las principales conclusiones derivadas del estudio investigativo, también se escriben las limitaciones que restringen el uso y el desarrollo de la herramienta y finalmente se especifican los trabajos pendientes a realizar en el futuro.

Con el objetivo de complementar la documentación relacionada con este trabajo de investigación, se adjuntan los siguientes anexos que soportan los resultados del estudio:

1. Reporte estudio exploratorio
2. Reporte experimento controlado
3. Manual de usuario de la herramienta
4. Herramienta empaquetada en archivo .jar
5. Artículo corto en español
6. Artículo completo en inglés
7. Certificado de participación en 13CCC
8. Carta aceptación de publicación en Springer
9. Monografía de trabajo de grado en .pdf
10. Links a repositorio y página web de la herramienta

# Capítulo 2: Marco teórico y trabajos relacionados

En esta sección se abordan conceptos relevantes para entender el contexto y el dominio del problema que se quiere abordar en este estudio, se especifican temas relacionados con arquitecturas de software, métodos de arquitectura, la documentación de la arquitectura, el Rationale Arquitectónico y anotaciones de código fuente. También se realiza una revisión de la literatura con el objetivo de buscar, analizar y clasificar los trabajos relacionados con la documentación del Rationale Arquitectónico a través de anotaciones de código fuente. Finalmente se presentan los resultados del análisis de los estudios relacionados con el objetivo de diferenciar los aportes investigativos entre las propuestas que están en el estado del arte y el presente trabajo de investigación.

## Marco teórico

Para la realización de este trabajo se mencionan temas relacionados con arquitecturas de software como los métodos de arquitectura, patrones y tácticas, su documentación, etc. También se menciona el impacto que tienen algunas características de calidad sobre el diseño de la arquitectura de software. Además, se abordan temas como el uso de las anotaciones de código fuente como herramienta para documentar conocimiento implícito.

### Stakeholders

K. Wigers et al [18] definen un Stakeholder como una persona, un grupo o una organización que esta activamente involucrada en el desarrollo de un proyecto, el cual se ve afectado en forma negativa o positiva de acuerdo a los resultados del mismo. De manera similar, I. Sommerville [19] usa el termino Stakeholder para referirse a una persona o un grupo el cual se ve afectado directa o indirectamente por los resultados de un sistema, un stakeholder puede incluir usuarios finales, los cuales pueden interactuar con el sistema o verse afectados por los resultados de este. Los stakeholders también pueden ser ingenieros de desarrollo, de mantenimiento, gestores del negocio, expertos del dominio o personas que custodian los intereses de los patrocinadores del proyecto.

### Atributos de calidad y escenarios

En el desarrollo de sistemas se deben identificar los requisitos funcionales de la aplicación, desarrollar el software encargado de implementar esos requisitos y asignar los recursos físicos necesarios para su funcionamiento. Sin embargo, no es suficiente realizar únicamente los requisitos a nivel funcional, también se requiere que se cumplan ciertas necesidades de calidad de software [20]. 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 son métricas que definen la calidad de un producto software como la seguridad, la fiabilidad, el rendimiento, entre otros [19]. La IEEE 1061 [21] define la calidad de un producto software como el grado en el cual un sistema cumple con una combinación de atributos. Anteriormente el estándar ISO/IEC 9126-1:2001 [22] define un modelo de calidad de software de acuerdo a las siguientes seis categorías de características: funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad, las cuales están divididas en sub características. Estas se definen de acuerdo a las características externamente perceptibles para cada sistema software [23]. En la actualidad se actualiza la norma ISO/IEC 9126-1:2001 [22] por la norma ISO/IEC 25010 [24], esta se compone de las siguientes ocho

características de calidad: Adecuación funcional, Eficiencia de desempeño, Compatibilidad, Usabilidad, Fiabilidad, Seguridad, Mantenibilidad y Portabilidad.

Por otra parte, según L. Bass et al. [25], existen 3 clases de atributos de calidad:

1. Atributos de calidad del sistema, como la disponibilidad, la modificabilidad, el desempeño, la seguridad, la capacidad de ser probado y la usabilidad.
2. Atributos de calidad del negocio, como el tiempo de compra u otros requerimientos de relacionados con el negocio, los cuales son afectados por la arquitectura.
3. Cualidades como la integridad conceptual, la cual afecta indirectamente atributos como la modificabilidad.

Sin embargo, los atributos de calidad no siempre son fáciles de detectar debido a que son definiciones de funcionalidad no operacional y generalmente se sobrescriben debido a las diferentes preocupaciones de calidad involucradas. L. Bass et al. [25] definen un escenario de atributo de calidad como una solución a los problemas anteriormente mencionados, este escenario especifica un requerimiento de calidad de manera formal compuesto por las siguientes seis partes:

1. Fuente del estímulo: Puede ser una entidad como una persona, una unidad de cómputo o cualquier otro actor que genere un estímulo.
2. Estímulo: Es una condición que necesita ser considerada cuando ocurre en un sistema.
3. Ambiente: Son las condiciones en las cuales ocurre un estímulo.
4. Artefacto: Puede ser el sistema completo o partes de él. En ocasiones este artefacto puede ser simulado.
5. Respuesta: Son las actividades que se llevan a cabo después de que se presenta un estímulo.
6. Medida de respuesta: Una respuesta debe ser medida inmediatamente después de que ocurra, debido a que en algunas ocasiones se necesita probar el requerimiento.

## Arquitecturas de software

Según la norma ISO/IEC/IEEE 42010 [26], encargada de estandarizar la descripción de arquitectura, define arquitectura de software como los conceptos fundamentales de un sistema o sus propiedades expresadas en sus elementos, las relaciones y los principios que guían su diseño y evolución. Otra definición planteada por L. Bass et al [25] describe la arquitectura de software de un programa o sistema de cómputo como la estructura o estructuras del sistema, la cual compromete elementos software, propiedades externas visibles de los elementos y la relación entre ellos. A. Jansen et al. [3] junto con H. Van et al. [27] definen la arquitectura de software como un conjunto de decisiones de diseño que involucran la adición, eliminación o modificación a la arquitectura, las reglas, las restricciones, las razones de diseño y nuevas funcionalidades que representan uno o más requerimientos en una arquitectura de software planteada. Esta última definición es la más acertada para los propósitos relacionados con el cumplimiento de los objetivos de investigación de este estudio.

Podemos ver la arquitectura de software de una manera análoga a la arquitectura tradicional de edificaciones. Generalmente en el diseño de una arquitectura tradicional se pueden observar diferentes planos de acuerdo a las perspectivas necesarias para cada tipo de rol en la construcción de una edificación, esto quiere decir que se requiere, por ejemplo, un diagrama que especifique la organización externa del edificio, otro diagrama que describa la organización interna de las

habitaciones y otro en el cual se especifique la organización interna de las tuberías. Cada diagrama representa la misma edificación, sin embargo, cada uno está estructurado de una manera diferente, con el objetivo de ser atendido por diferentes audiencias encargadas de interpretarlos y materializarlos. En la construcción de software se tienen diferentes perspectivas de manera análoga, en el modelo de 4 + 1 vistas propuesto por P. Kruntchen [28], se definen diferentes vistas encargadas de describir la arquitectura de software para diferentes audiencias en términos de funcionalidad, procesos y actividades, desarrollo, despliegue y escenarios de casos de uso. Unified Modeling Language (UML) [29] define algunos diagramas para cada vista de arquitectura, permitiendo que se puedan comunicar e interpretar por los roles encargados en el proceso de desarrollo. Algunos de estos modelos son diagramas de clases, de componentes, de secuencia, de despliegue, entre otros.

Las decisiones son acciones que se toman respecto a una preocupación, estas decisiones en el desarrollo de software están clasificadas según P. Abrahamsson et al [30] en decisiones generales, de diseño, de arquitectura y las restricciones de las que dependen todas las decisiones. Generalmente, las decisiones de diseño que se toman sobre una arquitectura van dirigidas a cumplir con ciertas consideraciones del negocio, las cuales determinan algunas características de calidad que generalmente seleccionan y priorizan un grupo de atributos de calidad [25], debido a que no es posible definir un diseño arquitectural definitivo que solucione todas las preocupaciones de calidad [8]. La labor principal de un arquitecto es definir la organización de un conjunto de elementos arquitecturales de acuerdo a las necesidades de calidad requeridas [8]. Además, debe hacer concesiones sobre las decisiones de diseño que realiza, para implementar el software de tal forma que optimice la mejor combinación de atributos de calidad [31]. Sin embargo, la consecución se los atributos de calidad deben ser considerados como parte del diseño, si no también, en la implementación y en el desarrollo en general. Los atributos de calidad no deben estar obligatoriamente relacionados en el diseño de la arquitectura, por ejemplo, la usabilidad considera aspectos que miden la facilidad de uso de acuerdo a la subjetividad de los usuarios, estas decisiones en la parte gráfica de la interfaz no tiene implicaciones en el diseño arquitectural, sin embargo, acciones como cancelar operaciones o reutilizar datos previamente ingresados requieren de la organización de componentes arquitecturales que habiliten su funcionamiento [25].

## Tácticas y patrones de arquitectura software

Las decisiones de diseño que se toman en un proyecto de desarrollo software tienen un alto impacto en las fases de diseño, mantenimiento y evolución de la arquitectura, estas decisiones frecuentemente se relacionan con la aplicación de tácticas y patrones arquitecturales [31]. Los patrones son estructuras arquitecturales comunes, las cuales están bien definidas, documentadas y bien entendidas para su reutilización [31] [32] [33]. La arquitectura define qué es lo que el sistema puede hacer mediante los patrones de arquitectura seleccionados [31]. Las tácticas son decisiones de diseño con un nivel mayor de abstracción, permitiendo dar solución a preocupaciones a nivel no funcional, buscando cumplir con ciertos atributos de calidad [25]. Una táctica es una decisión de diseño la cual tiene como objetivo mejorar una preocupación de diseño específica relacionada con a un atributo de calidad [25] [31]. Las tácticas se definen con el objetivo de controlar respuestas a estímulos relacionados con atributos de calidad.

Un patrón de arquitectura puede implementar diferentes tácticas, cada una relacionada con diferentes atributos de calidad, además, la implementación de ciertos patrones puede reflejarse en las decisiones que se realizan respecto a la selección de las tácticas. L. Bass et al. [25] definen una

colección de tácticas como una *estrategia arquitectural*, estas estrategias arquitecturales necesitan como insumo restricciones de tiempo, presupuesto, los objetivos y las reglas del negocio, con el objetivo de mejorar algunos aspectos de calidad, generando por cada estrategia unos beneficios y sus costos relacionados.

## Métodos de arquitectura software

En la actualidad existen diferentes metodologías, técnicas y herramientas que soportan el establecimiento de requisitos de arquitectura, el análisis, el diseño, la evaluación y su posible evolución. A continuación, se describen algunos de los métodos de análisis y diseño de arquitecturas software:

### 1. *Software Architecture Analysis Method (SAAM)*

Es un método para describir y analizar arquitecturas de software propuesto por Kazman et al. [34]. Este método tiene como objetivo definir tres perspectivas para el entendimiento y la descripción de la arquitectura en términos de funcionalidad, estructura y localización, además de, proveer un lenguaje simple para describir la perspectiva estructural, el cual permite que diferentes arquitecturas sean descritas consecuentemente, con el objetivo de tener un nivel común de entendimiento para comparar diferentes arquitecturas. Este método consta de las siguientes actividades: caracterización de una partición funcional para el dominio, mapear el particionado funcional dentro de la descomposición estructural de la arquitectura, escoger un conjunto de atributos de calidad con los cuales evaluar la arquitectura, escoger un conjunto de tareas concretas las cuales prueban los atributos de calidad deseados y finalmente evaluar el grado en el cual cada arquitectura provee soporte para cada una de las tareas. Este método sigue una técnica de evaluación mediante escenarios, estos representan las bases para manifestar las propiedades de una arquitectura software. Los escenarios muestran las actividades que pueden realizar ciertos actores en el sistema y también permiten observar cambios anticipados en algunas actividades sobre el sistema. La característica más notable de este método es que los escenarios siempre están formados de acuerdo con ciertos atributos de calidad, además, es necesaria la participación intensiva de los stakeholders [23].

### 2. *Architecture Tradeoff Analysis Method (ATAM)*

Es una técnica estructurada para el entendimiento de las concesiones realizadas por los arquitectos de software en sistemas de software intensivos propuesta por Kazman et al. [35]. ATAM es un modelo de evaluación del diseño arquitectónico en espiral, en el que en cada iteración se obtiene mayor entendimiento del sistema, se reducen los riesgos y se manipula el diseño. Este método consta de 4 grupos de actividades: recopilación de escenarios y requisitos, vistas arquitectónicas y realización de escenarios, construcción y análisis de modelos, y concesiones (Tradeoffs), las cuales dan seguimiento a los siguientes pasos del método: recopilación de escenarios, recopilación de requerimientos/restricciones/ambiente, descripción de las vistas arquitecturales, análisis de atributos específicos, identificación de puntos sensibles y finalmente identificación de concesiones.

### 3. *Quality Attribute Workshops (QAWs)*

Es un método simplificado, para intervenir en etapas tempranas del desarrollo de software; usado para generar, priorizar y refinar escenarios de atributos de calidad antes de que el diseño de la arquitectura de software esté terminado [36]. Este método es desarrollado por el Instituto de

Ingeniería de Software de la Universidad Carnegie Mellon SEI (Por sus siglas en inglés) [37] y se enfoca en las preocupaciones a nivel de sistema y depende de una gran participación por parte de los stakeholders, estos deben responder y realizar preguntas en cualquier momento del taller con el objetivo de resolver dudas o conflictos de intereses que se puedan presentar. En muchas ocasiones hay un moderador encargado de recortar la discusión generada por las preguntas y respuestas realizadas por los participantes.

QAW se define mediante los siguientes pasos que deben realizar los participantes del taller:

1. Presentación e introducción del método QAW
2. Presentación del negocio y la misión
3. Presentación del plan arquitectural
4. Identificación de manejadores arquitecturales
5. Lluvia de ideas de escenarios
6. Consolidación de escenarios
7. Priorización de escenarios
8. Refinamiento de escenarios

### 4. *Attribute-Driven Design (ADD)*

Este método de manera similar a los métodos anteriormente mencionados, también tiene como base en el proceso de diseño de la arquitectura software, los requerimientos a nivel no funcional o atributos de calidad de un software. Fue desarrollado por el SEI [37] con el objetivo de definir la arquitectura software mediante un proceso de diseño recursivo que descompone un sistema o elementos de un sistema con la intención de aplicar tácticas y patrones [25] que dirijan el funcionamiento del sistema de acuerdo a un conjunto de atributos de calidad requeridos. ADD sigue un ciclo de tres acciones a nivel general “Planear”, “Hacer” y “Probar”, estas acciones se repiten hasta que se cubra con todos los requerimientos arquitecturales significantes. A continuación, se presentan las actividades que se realizan a nivel específico en las anteriores acciones mencionadas:

1. Confirmar si hay suficiente información de requisitos.
2. Escoger un elemento del sistema para descomponer.
3. Identificar candidatos de requerimientos funcionales, restricciones de diseño o atributos de calidad que tengan impacto en la arquitectura.
4. Seleccionar una vista previa de la arquitectura que describa el mayor tipo de elementos y sus relaciones.
5. Instanciar elementos arquitecturales y definir responsabilidades.
6. Definir interfaces para los elementos instanciados.
7. Verificar, refinar y convertir los requisitos en restricciones para los elementos instanciados.
8. Repetir los pasos del 2 al 7 hasta terminar con todos los elementos del sistema.

### *5. Views and Beyond Method*

La documentación de la arquitectura es una de las tareas más importantes y que requiere más atención por parte de los stakeholders del sistema, la debida documentación de la arquitectura puede marcar la diferencia entre un sistema extensible y un sistema frágil y estático [38]. Esta puede ser un escrito formal tanto en forma física como digital que tiene como objetivos: servir como medio

de comunicación entre las partes involucradas durante y después del proceso de desarrollo, de repositorio de información para la persona encargada de evolucionar y mantener el sistema y como guía para los usuarios del software [12]. El software sin una correcta documentación tiende a fragmentarse y colisionar puesto que no son explícitos los atributos de calidad requeridos por las partes interesadas, y tampoco las preocupaciones de los diseñadores y desarrolladores. El SEI propone una colección de técnicas para documentar la arquitectura software denominada Views and Beyond, con el objetivo de crear un documento de arquitectura que ayude a las personas a realizar su trabajo de acuerdo con un rol establecido. Una vista no es más que una representación de un conjunto de elementos que están relacionados entre sí para cumplir con una determinada responsabilidad. Sin embargo, no siempre es suficiente con tener la vista de los elementos, en muchas ocasiones es necesario gestionar información adicional que va más allá de las vistas arquitecturales, como por ejemplo las decisiones involucradas en un proyecto de desarrollo de software. Esta colección de técnicas permite a los stakeholders tener un entendimiento más claro sobre la arquitectura de software, permitiendo que se puedan entender las preocupaciones y las funcionalidades con mayor facilidad.

## Rationale Arquitectónico

El Rationale es la justificación o la razón de ser de las decisiones o acciones; en ingeniería de software, éste es capturado y gestionado para mejorar el entendimiento de un sistema por parte de los implicados en el desarrollo [39]. El término Rationale frecuentemente hace referencia a las razones por las cuales se crean o se usan determinados artefactos [40]. En el proceso de desarrollo de software existen diferentes usos del Rationale, estos usos van desde el levantamiento de requisitos por parte de los stakeholders hasta la implementación de las pruebas del sistema. En la Figura 1 se muestra la relación que existe en cada una de las etapas con los roles involucrados y el uso del Rationale en cada fase del proceso. En este trabajo estudiamos el Rationale del diseño de la arquitectura, este puede ser definido como una expresión de las relaciones entre un diseño de la arquitectura, su propósito, la conceptualización del arquitecto y las reglas contextuales en la realización del propósito [41] [42]. Este Rationale representa conocimiento implícito, el cual responde a preguntas específicas sobre la escogencia de una decisión de diseño en particular o el proceso seguido para tomar dicha decisión [43].

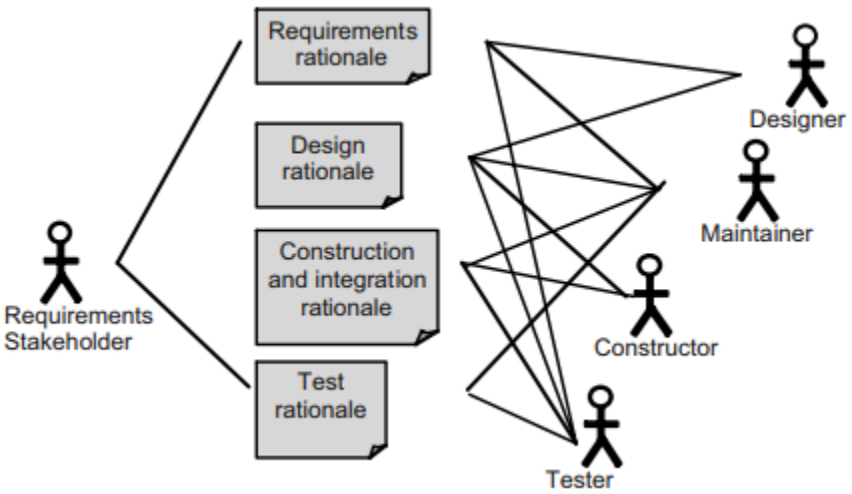

Figura 1, Usos del Rationale en Ingeniería de Software, tomado de A. Dutoit *[38]*

## Anotaciones de código fuente

Las anotaciones de código son abstracciones de alto nivel que complementan el código fuente, personalizando la funcionalidad del sistema. Una anotación permite ocultar una implementación para un dominio específico, con el fin de hacer que el desarrollo le sea mucho más ligero y efectivo al desarrollador [44]. Una ventaja de las anotaciones es que permiten la reutilización de código, lo cual posibilita a que los desarrolladores se enfoquen en la implementación principal de la lógica del negocio. Las anotaciones expresan el estado de un programa y hacen parte de un lenguaje que permite la colocación de anotaciones en algunos componentes del programa, llamados objetivos de anotación, estos objetivos pueden ser variables globales, parámetros de funciones, valores de retorno y tipos definidos por el usuario [45]. Las anotaciones de código fuente son la base de un paradigma orientado a atributos, el consiste en una técnica de marcado de elementos de código fuente con información relacionada a sus posibles atributos [46] [47]. Las anotaciones de código generalmente cuentan con numerosos usos relacionados con:

1. Brindar información del código fuente al compilador para la detección de errores o advertencias.
2. Obtener información del programa en tiempo de compilación y despliegue para generar código, archivos XML u otros.
3. Mecanismos de reflexión para obtener información de las anotaciones en tiempo de ejecución.

Generalmente la implementación de las anotaciones tiene dos enfoques: 1. Como un software complementario para obtener información del código fuente en tiempo de compilación. 2. Como un mecanismo de reflexión para obtener información del programa en tiempo de ejecución. En el primer enfoque es necesaria la existencia de una fábrica de procesadores para manejar cada anotación declarada y en el segundo enfoque es necesario tener librerías que permitan la captura de metadatos en el código fuente [48]. JSR-175 introdujo las anotaciones de código fuente en el año 2004 en la versión 5 de Java, el objetivo principal era autogenerar código repetitivo Java EE a partir de metadatos. Anteriormente la generación de EJBs se lograba mediante xdoclet [49] o herramientas personalizadas como ejbgen (v2.0) [50], otro objetivo era mejorar la comunicación del desarrollador al compilador, para la optimización y la comprobación de errores. En la Figura 2 se puede observar la declaración de una anotación Java de código fuente, la cual representa la información de contacto con tres atributos miembro: El nombre, el número de teléfono y el correo electrónico del autor. La declaración es similar a la de una interface común, con la diferencia de que la palabra reservada *interface* va antecedida del carácter '@' y los atributos son declarados como métodos públicos abstractos; algunos de estos atributos miembros pueden tener valores por defecto fijados mediante la palabra reservada *default* y seguido del valor por defecto que se quiere establecer. Antes de la declaración de la anotación se pueden utilizar meta anotaciones que permiten modificar el comportamiento de la anotación, entre las más importantes se encuentran: @Documented la cual permite registrar los elementos marcados con las anotaciones en el documento generado por javadoc [51], @Retention(RUNTIME) retiene el archivo de clases en tiempo de ejecución para obtener el valor de las anotaciones por medio de técnicas de reflexión, @Target({METHOD,PACKAGE,TYPE}) define los componentes que pueden ser anotados, en este caso los objetivos de anotación son métodos, clases y paquetes.

```java
@Documented
@Retention(RUNTIME)
@Target({METHOD,PACKAGE,TYPE})
public @interface ContactInfo {
    String name();
    String tel();
    String email() default "";
}
```

Figura 2, Declaración de anotaciones de código

En la Figura 3 se presentan las maneras de uso de la anotación anteriormente declarada en los diferentes objetivos de anotación. En la parte izquierda de la Figura 3, en la sección 1 está marcada una clase Java nombrada package-info, en la cual se describe toda la información relacionada con un paquete, en la sección 2 en el centro de la imagen se marca un elemento de tipo clase y finalmente en la parte derecha de la Figura en la sección 3 se marca un elemento de tipo método con la anotación.

```java
@ContactInfo(
    name = "Pepito",
    tel = "23546815",
    email = "pepito@example.com"
)
package examplePackage;
```

1

```java
@ContactInfo(
    name = "Pepito",
    tel = "23546815",
    email = "pepito@example.co
)
public class Target {

}
```

2

```java
public class Target {
    @ContactInfo(
        name = "Pepito",
        tel = "23546815",
        email = "pepito@example.com"
    )
    public void methodExample(){

    }
}
```

3

Figura 3, Uso de anotaciones de código

Las anotaciones de código son como los comentarios javadoc pero con sintaxis y tipos fuertes, los cuales permiten que se realice cualquier tipo de programación con el modelo definido y la información marcada en los elementos, estas anotaciones son mucho más legibles tanto por humanos como por las máquinas. Algunos casos de uso comunes se pueden observar en frameworks Java como Spring [52] para realizar diferentes configuraciones en el comportamiento del framework. También se pueden encontrar anotaciones de código en herramientas de pruebas unitarias como JUnit [53], este corredor de pruebas encuentra las clases anotadas, las crea, ejecuta los métodos y contrasta los resultados. Finalmente, otro caso de uso de uso de las anotaciones de código lo podemos observar en el framework de mapeo objeto relacional Hibernate [54], cuando se requiere realizar validaciones de restricciones en el servidor de aplicaciones.

## Revisión de la literatura

En esta sección se describen los pasos que se siguieron para la realización de la revisión de la literatura. El objetivo de esta revisión es resumir la evidencia existente relacionada con la documentación del Rationale Arquitectónico y el uso de anotaciones de código fuente para su documentación, mediante las tres actividades de la revisión: planeación, ejecución y reporte de resultados. A continuación, se muestra cada una de las actividades realizadas.

# 1. Planeando la revisión

En esta sección, se especifica la necesidad de crear una revisión de la literatura, así como también cual es la pregunta de investigación que motiva este trabajo investigativo.

## *1.1. Identificar la necesidad de una revisión*

La definición del Rationale Arquitectónico o Rationale de diseño es de vital importancia cuando se requiere hacer mantenimiento o evolución de una arquitectura, pues este describe la justificación de un conjunto de decisiones de diseño que han sido tomadas en la fase de diseño en el proceso de desarrollo de software [9]. Sin embargo, según L. Bratthall et al. [55] este Rationale Arquitectónico frecuentemente está implícito o no se encuentra redactado en un documento estructurado, lo cual produce que se requiera de mucho más tiempo crearlo y mantenerlo. Es por esto que se debe realizar una búsqueda y un análisis de propuestas, soluciones, modelos, herramientas y demás trabajos relacionados con la documentación del Rationale arquitectónico, a fin de determinar el impacto del uso de las anotaciones de código fuente como herramienta para documentar decisiones implícitas en el proceso de diseño.

## *1.2. Especificar la pregunta de investigación*

El objetivo de esta revisión fue analizar la documentación del Rationale Arquitectónico, con el propósito de determinar los trabajos relacionados respecto a la utilidad de las anotaciones de código como herramienta para documentar el Rationale Arquitectónico con respecto a la mantenibilidad de la arquitectura en términos de eficiencia, efectividad y comprensión de la Arquitectura y su Rationale, durante cambios relevantes arquitectónicamente desde el punto de vista del arquitecto y los desarrolladores, en el contexto del mantenimiento de un sistema intensivo en software. Para dirigir los resultados de la revisión es necesario plantear las siguientes preguntas de investigación:

***¿Qué valor han tenido las anotaciones de código como herramienta para la documentación de decisiones durante el ciclo de vida del desarrollo de software?***

***¿Cómo y dónde es expresado el Rationale arquitectónico?***

# 2. Conduciendo la revisión

## *2.1. Identificación de la investigación*

El proceso de revisión de la literatura se lleva a cabo mediante búsquedas de estudios primarios preliminares a través de bases de datos bibliográficas, fuentes manuales, reportes, consultas con expertos en el tema tratado y revisiones de resultados de investigación. Para la búsqueda de trabajos relacionados en bases bibliográficas e internet se listan los términos clave, sus acrónimos y sus sinónimos, con el objetivo de formar cadenas de búsqueda unidas mediante conectores booleanos “AND” y “OR”. En la Tabla 1 se muestran las palabras clave utilizadas en esta revisión, junto con sus acrónimos y/o sinónimos, y en la Tabla 2 se muestran las cadenas de búsqueda generadas a partir de las palabras claves definidas en la Tabla 1.

Tabla 1, Palabras clave, sinónimos y/o acrónimos

| Palabras clave | Acrónimos o sinónimos |
|---|---|
| Software | Program, system, operating system, computer program |
| Documentation | Archive, form, written communication |

| | |
|---|---|
| Software Architecture | Design of software |
| Design | Architecture, composition, construction |
| Decisions | Conclusions, choices, agreement |
| Source Code | Code |
| Annotations | Tags, notes |
| Source code annotations | Code annotations |
| Architectural Rationale | Design Rationale |
| Attribute Oriented Programming | @OP |

Tabla 2, Cadenas de búsqueda

| Palabras clave | Cadena de búsqueda |
|---|---|
| Software Architecture, Documentation, Source Code Annotations | (Software Architecture AND Documentation AND Source Code Annotations) OR (Design of software AND Documentation AND Code Annotations) |
| Software, Documentation, Design, Decisions | (Software AND Documentation AND Design AND decisions) |
| Software Architecture, Annotations, Source Code Annotations | (Software Architecture AND Annotations) OR (Software Architecture AND Source Code Annotations) OR (Source Code annotations AND Software design) |
| Software, Documentation, Source Code Annotations, Attribute Oriented Programming | (Software AND Documentation AND Source Code Annotations) OR (Documentation AND Software AND Attribute Oriented Programming) OR (System AND Documentation AND Code Annotations AND Attribute Oriented Programming) |
| Architectural Rationale, Documentation, Software, Source Code Annotations | (Architectural Rationale AND Documentation AND Software AND Source Code Annotations) OR (Design Rationale AND Documentation AND (Software OR system) AND (Source Code Annotations OR Code Annotations)) |
| Design, Decisions, Documentation, Attribute Oriented Programming, Software | (Design AND Decisions AND Software AND @OP) OR (Architecture AND Choices AND Documentation AND Software AND Attribute Oriented Programming) |

Para la búsqueda de trabajos relacionados con las cadenas de búsqueda anteriormente descritas se hace uso de las bases bibliográficas mostradas en la Tabla 3, así como también, se realiza una búsqueda en los resultados orgánicos en el motor de búsqueda de Google y otras fuentes de información relevante.

Tabla 3, Fuentes de información

| Tipo | Fuente de información |
|---|---|
| Base de datos electrónica | Springer Link |
| Base de datos electrónica | ScienceDirect |

| | |
|---|---|
| Base de datos electrónica | IEEE Xplore Digital Library |
| Base de datos electrónica | SEI Digital Library |
| Base de datos electrónica | ACM Digital Library |
| Base de datos electrónica | Scopus Preview |
| Base de datos electrónica | ResearchGate |
| Otros recursos | Google Search Engine |
| Otros recursos | Google Scholar |

### *2.2. Selección de estudios*

Después de que se obtuvieron los estudios primarios potencialmente relevantes, éstos necesitaron ser evaluados para determinar su importancia actual mediante criterios de inclusión y exclusión previamente definidos.

#### Criterios de inclusión y exclusión

Se hace una revisión de artículos que hayan sido publicados entre los años 2004 hasta la actualidad, teniendo en cuenta que, para la inclusión de los estudios, éstos deben estar estrictamente relacionados con:

- Técnicas, métodos o herramientas relacionadas con la documentación del Rationale Arquitectónico mediante anotaciones de código fuente.
- Rationale en el diseño o el mantenimiento de la arquitectura software.

Fueron excluidos 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. También 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.

### *2.3. Síntesis de datos*

Los artículos recopilados se sintetizan de acuerdo a trabajos relacionados con los siguientes tres temas:

1. Fases en el ciclo de vida del desarrollo de software en la cual se menciona el Rationale.
2. Enfoques de documentación del Rationale arquitectónico.
3. Propuestas, métodos, técnicas y/o artefactos utilizados en la documentación.

## 3. Resultados

En esta sección se describen los resultados encontrados en la revisión sistemática de la literatura, en ésta se pueden evidenciar los trabajos relacionados con la definición, la estructura y los casos de uso relacionados con el Rationale. Además, se puede apreciar que el Rationale tiene un alto impacto en la fase de diseño, en la cual se toman un conjunto de decisiones, para las cuales no se registran las razones que motivan dicha selección de un grupo de opciones, debido a esto, la revisión se enfocó en la búsqueda de trabajos relacionados con el Rationale Arquitectónico y el uso de las anotaciones de código fuente como herramienta para documentar el conocimiento que frecuentemente se evapora con el paso del tiempo. De todos los estudios encontrados se hace un análisis para determinar cuáles de estos tienen mayor relevancia para la investigación en curso y

cuales pueden dar una respuesta a las preguntas de investigación planteadas en la revisión sistemática de la literatura.

Con las cadenas de búsqueda definidas anteriormente, se encuentran un total aproximado de 60 estudios relacionados con los temas en cuestión. De estos, se encuentran un total de 17 estudios relacionados con los criterios de inclusión y exclusión definidos anteriormente, de los cuales 13 tratan sobre la gestión de las decisiones de diseño y el Rationale en la ingeniería de software, su definición, sus casos de uso más frecuentes, sus barreras y los beneficios de documentarlo y gestionarlo. Los otros 4 artículos tratan temas relacionados con la programación orientada a atributos a través de anotaciones de código fuente, los usos más comunes y algunos casos de estudio que mencionan el uso de anotaciones de código como herramienta para documentar conocimiento implícito.

En la Tabla 4 se muestra la clasificación de los trabajos relacionados con la gestión, definición, captura y uso del Rationale Arquitectónico. Seguido de esto, se puede apreciar en la Tabla 5 los trabajos relacionados con la definición y el uso de las anotaciones de código como herramienta para documentar conocimiento tácito.

Tabla 4, Trabajos relacionados con el Rationale

| Trabajo N° | Autores | Fundamentos | | |
|---|---|---|---|---|
| | | Representación | Captura | Uso |
| 1 | A.H. Dutoit et al. [41] | X | X | X |
| 2 | J. Horner et al. [56] | X | | |
| 3 | K. Schneider et al [57] | X | X | X |
| 4 | S.J. Buckingham et al [58] | X | X | X |
| 5 | Janet E. Burge et al [40] | X | X | X |
| 6 | M.A. Babar et al [59] | X | X | X |
| 7 | L. Bass et al [60] | X | X | X |
| 8 | J.R Burge et al [61] | | X | X |
| 9 | J. S. Van der Ven et al [62] | | X | X |
| 10 | L. Bratthall et al [55] | | X | X |
| 11 | M. Shanin et al [63] | | X | X |
| 12 | J. Adrich et al [64] | X | X | X |
| 13 | T. M. Jesse et al. [65] | X | X | X |

Tabla 5, Trabajos relacionados con anotaciones de código Fuente

| Trabajo N° | Autores | Representación | Uso |
|---|---|---|---|
| 1 | Milan Nosál et al [47] | X | |
| 2 | Ghulam Rasool et al [66] | | X |
| 3 | Peter Kajsa et al [67] | | X |
| 4 | Matús Sulír et al [68] | | X |

## Estudios relacionados con el Rationale arquitectónico

En esta sección se detalla cada uno de los aportes que han realizado los trabajos anteriormente mencionados cuyo contenido está dirigido a temas relacionados con el Rationale arquitectónico.

### 1- Rationale Management in Software Engineering: Concepts and Techniques.

Allen Dutoit et al [41] nos brindan una idea general sobre el estado actual de la literatura respecto a los enfoques en la gestión del Rationale. Describen diferentes enfoques como los basados en procesos y los estructurales, cada uno de ellos con esquemas de argumentación que permiten capturar el Rationale, entre ellos se encuentran: IBIS (Issue Based Information System), el cual representa el Rationale como una secuencia de pasos en la toma de decisiones capturando la información cuando ésta se genera, QOC (Questions, Options, Criteria) representa el Rationale como un espacio de alternativas y evaluación de criterios que permite reconstruir el Rationale cuando se toma una decisión. DRL (Decision Representation Language) es una extensión de IBIS, la cual relaciona varios aspectos de QOC para llegar a un nivel de granularidad más alto en la especificación y captura del Rationale. Duitoit et al. muestran las diferentes ventajas y desventajas entre cada uno de los esquemas de argumentación del Rationale, haciendo una comparación entre los componentes de cada uno de ellos. En el trabajo de esta investigación se puede catalogar como un lenguaje de representación de decisión, enfocado en la documentación del Rationale Arquitectónico más que en la representación de la decisión en cuestión.

### 2- Effective Design Rationale: Understanding the Barriers.

En este trabajo J. Horner et al. [69], analizan sistemas de Rationale de diseño en desarrollo de software que capturan, registran y comunican la argumentación y el razonamiento detrás de un proceso de diseño. El objetivo principal del análisis es brindar un marco teórico sobre los temas actuales en términos de perspectivas actuales en la teoría del diseño, para definir ciertas restricciones y barreras que se encuentran tanto en la investigación del diseño, como en la descripción del Rationale en términos de elementos como la auto inspección, la comunicación y el análisis de los procesos de diseño. En este trabajo también realizan una clasificación las barreras que disminuyen la efectividad en la descripción del Rationale del diseño, entre ellas se encuentran: limitaciones cognitivas, limitaciones de captura, limitaciones de recuperación y limitaciones de uso. En nuestro trabajo se logra identificar algunas limitaciones respecto al uso del modelo de Rationale propuesto, debido a que es necesaria cierta capacitación en la utilización de las anotaciones con información de las razones arquitecturales. Para tratar de satisfacer esta limitación se realiza un manual usuario con sugerencias de uso de la herramienta, donde se definen roles y actividades para la especificación del Rationale con las anotaciones de código.

### 3- Rationale as a By-Product.

K Shneider [70] considera el Rationale como un activo en la ingeniería del software que es frecuentemente olvidado y difícil de documentar, debido a que este se genera en muchas de las actividades del proyecto como el diseño y el prototipado. En este trabajo se define el Rationale como un enfoque por producto el cual es representado mediante siete principios, los cuales definen mediante la construcción de dos aplicaciones a la medida. Estas aplicaciones se hacen con el objetivo de soportar la captura del Rationale a través de dos estrategias para atrapar conocimiento implícito del diseño, mientras se realiza el flujo normal de actividades del proyecto. Los principios

del Rationale basado por producto son: 1. Enfóquese en una tarea del proyecto en la que está emergiendo la lógica; 2. Capture el Rationale en esa actividad, no en una actividad diferente; 3. Ponga la menor carga adicional posible en el portador del Rationale, pero puede delegar este conocimiento a otras personas 4. Concéntrese 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 una computadora 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 primer caso se realiza mediante la estrategia FOCUS, la cual se utiliza frecuentemente para documentar conocimiento en las actividades de prototipado. Para el segundo caso se realiza un análisis de riesgo el cual se origina desde alguna situación en la que sea necesario tomar una decisión. En ambos casos se aplica el enfoque de Rationale basado por producto a través de la relación entre los principios definidos anteriormente con las siguientes preguntas: ¿Dónde ocurre el Rationale?, ¿cómo desviar el esfuerzo extra de los expertos?, ¿Que se puede capturar durante qué tarea?, ¿Quiénes van a ser los beneficiados?, ¿Es necesario análisis adicional de grabaciones de computadora? Finalmente, los autores concluyen que siguiendo estos 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 nuestro trabajo se trata de definir una herramienta que pueda responder las siguientes preguntas: ¿Dónde ocurre el Rationale?, ¿Que se puede capturar durante qué tarea?

## 4- Hypermedia Support for Argumentation-Based Rationale: 15 Years on from gIBIS and QOC

S.J. Buckingham [58] definen un enfoque de argumentación simple basado en anteriores enfoques tales como IBIS y QOC. El centro de este enfoque se encuentra en capturar en tiempo real, la mediación de opiniones que se produce en las reuniones con todas las partes interesadas en el desarrollo del proyecto. Para representar el enfoque propuesto desarrollan un entorno distribuido basado en hipermedia denominado Compendium, este 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. Aunque el enfoque permite obtener gran cantidad de información obstruye la fluidez en el desarrollo de los proyectos, lo que finalmente se convierte en una actividad intrusiva, la cual genera quejas entre los participantes. Nuestro trabajo no se enfoca en la recolección de documentación, en cambio, trata de abordar aspectos de documentación del Rationale Arquitectónico desde el código fuente.

## 5- Rationale-Based Software Engineering - chapter 12 (Rationale and Software Design)

En este capítulo del libro Rationale-Based Software Engineering, Janet E. Burge et al. [40] abordan la complejidad que tiene integrar las actividades de captura del rationale con el proceso de diseño, además analizan una variedad de enfoques donde se resalta el valor del rationale de diseño para los mantenedores, usuarios, desarrolladores, etc. Para entender estas actividades complejas en este

estudio utilizan dos métodos: el primero de ellos es un análisis teórico de varios tipos y los diferentes roles en el Rationale de diseño y la forma en que éstos afectan el diseño del software. El segundo método fue una revisión al estado del arte en la cual investigan los diferentes enfoques que han planteado otros autores, en la relación entre el diseño del Rationale y el diseño de la arquitectura de software. En el primer análisis se centran en las posibles dificultades y beneficios en la captura y uso del diseño del Rationale, en la segunda investigación se enfocan en las modificaciones a anteriores enfoques las cuales han brindado aportes positivos en el tema.

## 6- A Framework for Supporting Architecture Knowledge and Rationale Management.

M.A. Babar et al [71] proponen un framework que proporciona mecanismos de soporte para capturar y gestionar el conocimiento de diseño de la arquitectura (ADK: por sus siglas en inglés). Para establecer este framework, se analizan diferentes métodos y enfoques para capturar el conocimiento de diseño tácito e implícito, describen también un proceso de extracción de ADK a través de patrones y definen una forma para documentarlo. Finalmente describen un modelo de datos que se puede adaptar a cualquier implementación de repositorio de ADK. En la siguiente Figura se muestra la forma en la que documentan el conocimiento de diseño de arquitectura y un ejemplo de uso.

| **Pattern Name:** *Name of the pattern* | | **Pattern Type:** *Architecture, design, or style* | |
|---|---|---|---|
| **Description** | *A brief description of the pattern.* | | |
| **Context** | *The situation for which the pattern is recommended.* | | |
| **Problem** | *What types of problem the pattern is supposed to address?* | | |
| **Suggested** | *What is the solution suggested by the pattern to address the problem?* | | |
| **Forces** | *Factors affecting the problem and solution and pattern's justification.* | | |
| **Tactics** | *What tactics are used by the pattern to implement the solution?* | | |
| **Affected Attributes** | **Positively** | **Negatively** | |
| | *Attributes supported* | *Attributes hindered* | |
| **Abstract scenarios** | S | *A textual, system independent specification of a quality attribute.* | |
| | S | | |
| **Example** | *Some known examples of the usage of the pattern to solve the problems.* | | |

| | |
|---|---|
| Generic quality attribute | Flexibility/Scalability (ASR entity) |
| Abstract scenario | Application shall instantly notify changes to the interested clients (Scenario entity). |
| Abstract scenario | Application shall be able to handle simultaneous notification requests from increased number of clients (Scenario entity). |
| Architecture Decision | Event notification (Architecture Decision entity). |
| Design option 1 | Publish scribe (Alternative entity). |
| Design option 2 | Java RMI (Alternative entity). |
| Design Pattern | Publish on demand (Pattern entity). |

Figura 4, Framework para gestionar el Rationale

En este trabajo se tomaron algunos elementos de este Framework como son: los atributos de calidad, las tácticas, las decisiones que se toman y los patrones de arquitectura. Sin embargo, nuestro enfoque no se centra en los patrones de arquitectura, si no en las razones que hay detrás de la selección de una determinada táctica o patrón.

## 7- Capturing and Using Rationale for a Software Architecture.

En este trabajo Len Bass et al [72] analizan el papel fundamental que tiene la captura del Rationale de diseño en la evolución de la arquitectura. Una de las actividades principales de este trabajo es la definición de dos grafos para estructurar el rationale, facilitando su recuperación posiblemente de manera automatizada, con el objetivo de servir como recurso para responder a preguntas futuras de carácter arquitectural. El primer grafo que permite documentar el rationale consta de una secuencia de decisiones que conforman el diseño. Este grafo es utilizable para cualquier tipo de decisión, no solamente para procesos de desarrollo software. Es un grafo causal abstracto basado en una cadena de decisiones. Cada nodo del grafo debería definir alguno de los siguientes niveles de información: 1. Únicamente la decisión, 2. La decisión con su contexto, 3. La decisión junto con la información que sea adecuada.

En la raíz del segundo grafo se establece un atributo de calidad el cual generalmente impacta fuertemente en el diseño de la arquitectura. En este grafo se definen algunos patrones de arquitectura de los cuales se desprenden ciertas tácticas arquitecturales que se relacionan con cada patrón. Las decisiones de diseño quedan explícitas en la selección de los patrones y las tácticas a lo cual se le conoce como estrategia arquitectural. La justificación de las decisiones de diseño que conforman la estrategia de arquitectura, están definidas en un catálogo de tácticas y patrones donde se describen las características, ventajas y desventajas de cada patrón y de cada táctica. Cada estrategia tiene como objetivo responder a un determinado estímulo que genera la necesidad de calidad y retornar métricas que mejoren o alcancen dichas necesidades de calidad. A continuación, se muestra en la Figura 5 el grafo estructurado basado en patrones, tácticas estímulos y resultados.

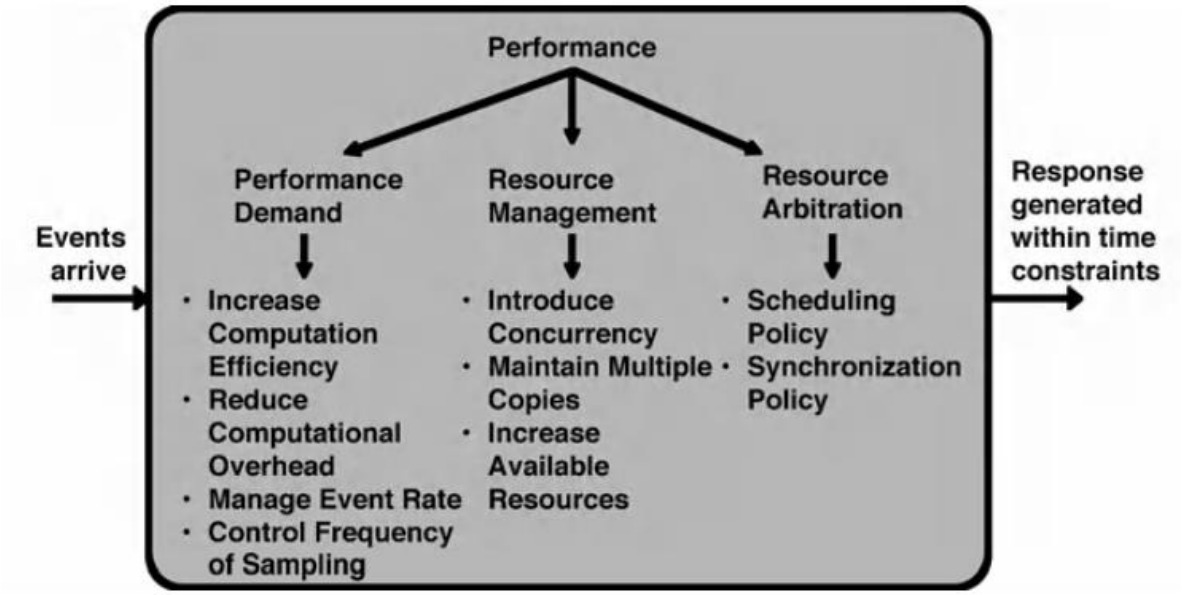


Figura 5, Grafo de tácticas y patrones como soluciones arquitecturales

Finalmente, los autores brindan una guía que permite a los arquitectos capturar el Rationale a través del siguiente formulario de preguntas:

a. ¿Qué es?
b. ¿Por qué fue hecho?
c. ¿Dónde se manifiesta en el diseño?
d. ¿Cuál fue el contexto de la decisión?
e. ¿Cuáles son las implicaciones de dicha decisión?

En este trabajo se desarrolla un modelo de Rationale Arquitectónico que pueda responder a las preguntas de los ítems a, b, c y e, a través de atributos de anotaciones de código fuente.

## 8- Rationale-Based Support for Software Maintenance.

J.E Burge et al [73] hablan sobre las dificultades a la hora de realizar el mantenimiento de un sistema, debido a que se debe analizar cuidadosamente el impacto de los cambios en otros módulos funcionales del software. Para estos autores es de suma importancia tener consistencia entre la documentación del software con la implementación de éste. Para ello desarrollan un sistema denominado SERAUT, el cual admite la entrada y la visualización de los motivos, así como también las inferencias sobre el Rationale. Este sistema tiene como principal objetivo ayudar a garantizar que el Rationale proporcionado para realizar modificaciones durante el mantenimiento a un sistema sea coherente con la intención inicial del diseñador. Este sistema se desarrolla como un complemento estrechamente ligado al entorno de desarrollo eclipse y maneja persistencia mediante tablas en una base de datos relacional de MySql. Para representar el Rationale los autores de esta herramienta definen RATSpeak, una representación de Rationale con los siguientes elementos: Requerimientos, problemas de decisión, preguntas, alternativas, argumentos, pretensiones, supuestos, ontología de argumentos y finalmente conocimiento de trasfondo. Para evaluar la utilidad de este sistema los autores realizan un experimento con 20 participantes entre ellos estudiantes graduados y desarrolladores de la industria, en el cual se realizaron tres tareas de mantenimiento: adaptativo, correctivo y de mejora. El objetivo del experimento era comparar el rendimiento del sujeto con y sin acceso al Rationale registrado con SERAUT. En este caso la variable de control dependía únicamente del tiempo debido a la baja complejidad de las tareas, este tiempo se captura en dos ocasiones, la primera cuando el participante logra encontrar el componente de código que debe cambiar, y la otra captura del tiempo se realiza cuando el participante termina la actividad. Los resultados del experimento no muestran alguna significancia estadística respecto al tiempo involucrado para realizar estas modificaciones, pero arroja información relevante en la actuación de las personas con más y menos experiencia en desarrollo, además, el promedio en los resultados del grupo que utilizó SERAUT fue mejor que el grupo sin el uso de este sistema. Por lo cual concluyen que se deben realizar más experimentos para determinar su utilidad de manera completa. En este trabajo se busca no sólo medir la velocidad con la que hacen modificaciones, además de esto se busca determinar que tan bien las hacen y cuál es el nivel de comprensión de las razones de las decisiones arquitecturales antes y después de desarrollar los cambios.

## 9- Design Decisions: The Bridge between Rationale and Architecture.

J.S van der Ven et al [62] definen la arquitectura de software como un proceso de toma de decisiones de diseño en donde se realiza la selección de las decisiones adecuadas dentro de un conjunto de posibilidades. Generalmente este conocimiento se encuentra implícito en diferentes artefactos del software, lo cual causa que quede en las mentes de los diseñadores y finalmente ese conocimiento se pierda con facilidad a través del tiempo. Es por eso que en este trabajo los autores utilizan el

modelado explícito de decisiones de diseño como alternativa para aumentar la consistencia entre el Rationale y la arquitectura de software. Para ello, desarrollan una extensión de Java denominada ARCHIUM y la cual consiste de un compilador en una plataforma en tiempo de ejecución. Esta extensión se conforma principalmente de tres elementos: 1. Modelo arquitectural, el cual define formalmente la arquitectura de software usando conceptos de lenguajes de descripción de arquitecturas. 2. Modelo de decisión, el cual representa las decisiones de diseño junto con su Rationale. 3. Modelo de composición, el cual permite la integración de los diferentes conceptos entre los modelos anteriormente mencionados. El modelo de Rationale de decisiones de diseño se muestra en la Figura 6.

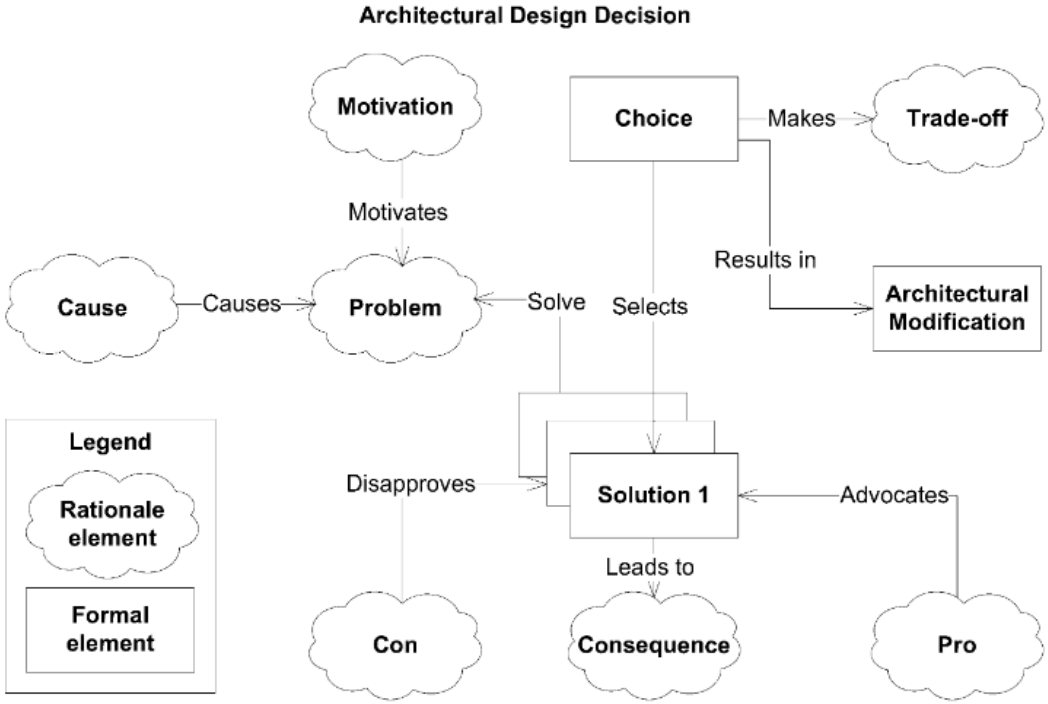


Figura 6, Modelo de decisiones de diseño

## 10- Is a Design Rationale Vital when Predicting Change Impact? – A Controlled Experiment on Software Architecture Evolution.

Lars Bratthall et al [55] desarrollan un experimento controlado con el objetivo de determinar el valor de tener acceso al Rationale cuando se realizan modificaciones con un impacto nivel de evolución arquitectural. En este trabajo evalúan el valor de tener el Rationale tanto cualitativa como cuantitativamente, para ello ponen a realizar tareas de modificación de la arquitectura a 17 participantes entre ellos estudiantes de último semestre y personas de la industria, los participantes deben modificar dos sistemas de información en tiempo real, con el objetivo de determinar si los participantes que tienen acceso al Rationale realizan en menos tiempo y de mejor manera los cambios solicitados. El resultado del estudio cualitativo arroja una significancia estadística en la corrección y en la velocidad de los participantes para realizar los cambios, a diferencia del estudio cuantitativo, donde no se encuentra diferencia alguna en la realización de los cambios con o sin acceso al Rationale. En nuestro estudio no se busca medir si los participantes realizan las tareas en menos tiempo, en cambio se busca determinar a través de un experimento si los participantes que

tienen las anotaciones de código fuente comprenden mejor las razones detrás de las decisiones de arquitectura antes y después de realizar cambios con un impacto arquitectural.

## 11. Do architectural design decisions improve the understanding of software architecture? two controlled experiments

M. Shahin et al. [63] hacen dos experimentos controlados para analizar como las decisiones de diseño arquitecturales (Architectural Design Decisions ADD: por sus siglas en inglés) y su rationale influyen en el entendimiento de la arquitectura software. Para poder analizar esta suposición es necesario cuantificar el entendimiento que alguien tiene de una decisión arquitectural, por lo cual formulan 2 preguntas de investigación y 4 hipótesis sobre dicha suposición, las cuales se describen a continuación: pregunta 1-¿El uso de ADD y su justificación reduce el tiempo que se necesita para completar una tarea de diseño de arquitectura?, pregunta 2-¿El uso de ADD y su justificación aumentan la exactitud de los resultados de diseño de arquitectura?, hipótesis 1-El uso de ADD y su justificación no afecta el tiempo necesario para completar una tarea de diseño de arquitectura, hipótesis 2-El uso de ADD y su justificación no afecta la exactitud de los resultados del diseño de la arquitectura, hipótesis 3-El uso de ADD y su justificación aumenta el tiempo necesario para completar una tarea de diseño de arquitectura, hipótesis 4-El uso de ADD y su justificación mejora la exactitud de los resultados del diseño de la arquitectura. Lo primero que hacen es dividir el experimento en dos grupos, ambos grupos tienen una descripción relacionada con la arquitectura del sistema, la cual toma aproximadamente 45 minutos entender. A ambos grupos se les asigna un nuevo requerimiento, el cual deben integrar con la arquitectura anteriormente mostrada. A un grupo se les presenta un documento con información de ADD a diferencia del otro grupo, el cual debe realizar el diseño sin conocer la información de ADD. Por último, se guardan los tiempos y se evalúa la exactitud de las decisiones arquitecturales de ambos equipos, para posteriormente analizarlas y determinar cuáles hipótesis se cumplen. Finalmente se presentan los siguientes resultados: 1. El uso de ADD y su justificación no afecta el tiempo necesario para completar las tareas de diseño de la arquitectura. 2. El grupo que recibió información de ADD obtuvo resultados significativamente mejores a comparación del otro grupo. 3. Los participantes con más experiencia de diseño obtuvieron mejores resultados que los participantes menos experimentados. El principal aporte de este trabajo es la metodología utilizada para analizar si el uso de ADD en la documentación de las decisiones de diseño mejora la comprensión de la arquitectura software. En nuestro trabajo se pretende desarrollar un experimento controlado para analizar si el uso de anotaciones de código para gestionar decisiones arquitecturales, mejora la comprensión de la arquitectura.

## 12. ArchJava: Connecting Software Architecture to Implementation

J. Aldrich et al. [64] desarrollan una extensión para Java, con el fin de integrar la especificación de la arquitectura con la implementación en el código Java. ArchJava permite unir la estructura arquitectónica con la implementación en un solo lenguaje, garantizando la trazabilidad entre la arquitectura y el código fuente, lo cual nos da como primer beneficio el apoyo a la co-evolución de la arquitectura y la implementación. Otro de los beneficios que nos brinda este enfoque es llamado integridad de comunicación, es un concepto que consiste en que todos los componentes implementados en una arquitectura sólo se comuniquen con los componentes establecidos en el diseño. Para evaluar este enfoque J. Aldrich et al. aplican ArchJava al sistema Aphyds, el cual consiste en una aplicación pedagógica de diseño de circuitos. La arquitectura de este sistema se plasma en un documento no formal hecho a mano, el primer paso de la metodología consiste en hacer

reingeniería para recuperar la arquitectura del sistema actual, con la que se determinan 3 problemas concretos: Comprensión de la comunicación dentro del programa, refactorización del programa para limpiar su arquitectura y corrección de los defectos relacionados con las actualizaciones del sistema. El siguiente paso, consiste en aplicar la extensión para expresar las características de la estructura arquitectónica y luego unir la implementación del código fuente, lo cual da como resultado la integridad de comunicación entre componentes, con el objetivo de conseguir la correcta comprensión de la aplicación y una debida evolución hacia futuro. En este trabajo, vamos a determinar la integridad de comunicación entre las partes del código arquitecturalmente relevantes y las decisiones arquitecturales que se toman durante y después del ciclo de vida del software. También se pretende utilizar las anotaciones de código para documentar únicamente las decisiones arquitecturales tomadas en un sistema.

### 13. DecDoc: A Tool for Documenting Design Decisions Collaboratively and Incrementally

T. M. Hesse et al. [65] 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 código fuente, consiguiendo una trazabilidad entre el diseño y la implementación. T. M. Hesse et al. utilizan como ejemplo un sistema de gestión de ventas e inventario llamado CoCoME, suponiendo que 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 [19]. Las personas responsables de asumir estos cambios, utilizan la 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 en 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 se tomará únicamente el enfoque relacionado al código fuente, creando un nuevo modelo de anotaciones capaz de capturar y gestionar las decisiones arquitecturales de un sistema.

## Estudios relacionados con anotaciones de código fuente

Estos son los trabajos de investigación sobre anotaciones de código fuente, algunos de ellos describen las anotaciones como un lenguaje formal mientras que otros trabajos brindan ejemplos de cómo se utilizan las anotaciones para capturar información a partir de meta-datos.

### 1- Source Code Annotations as Formal Languages.

Milan N. et al [47] comprueban la hipótesis de que la programación orientada a atributos a través de anotaciones de código fuente, tiene una estrecha relación con los lenguajes formales convencionales, para ello realizan diferentes observaciones a las anotaciones de código a partir de aspectos puntuales en la definición del lenguaje como: sintaxis concreta, abstracta y semántica. El

análisis de anotaciones toma como ejemplo diferentes anotaciones conocidas como las presentadas por Java Persistence API (JPA), algunas anotaciones nativas de Java e incluso anotaciones propias en un contexto dado. Este artículo sirve de referencia en anotaciones de código debido al gran contenido explicativo sobre este paradigma orientado a atributos y sus herramientas, que para efectos de estudio toman las anotaciones de código en Java. En los resultados de este análisis se determina que las anotaciones de código discrepan de los lenguajes formales debido a las restricciones estructurales de las anotaciones Java, al no permitir tener referencias a otras anotaciones externas. Otro punto en los cuales las anotaciones difieren de los lenguajes formales, es que éstas se pueden implementar de acuerdo a dos enfoques: 1. En tiempo de compilación como una herramienta de procesamiento de anotaciones o 2. Como una herramienta de procesamiento de anotaciones en tiempo de ejecución. En otros aspectos relacionados con la sintaxis abstracta, la sintaxis concreta y la correspondencia semántica, las anotaciones de código no difieren en gran medida de los lenguajes formales.

## 2- Design pattern recovery based on annotations.

En este trabajo Ghulam Rasool et al. [66] presentan un enfoque basado en anotaciones de código fuente, expresiones regulares y consultas a bases de datos para recuperar patrones de diseño como un método de ingeniería inversa. Lo primero que hacen en este trabajo es definir las características que son variantes en un patrón de diseño, con el objetivo de aplicar reglas que sean coherentes con las características de los elementos del código fuente, de esta forma se puede hacer la extracción de los patrones de diseño en caso de presentarse en el código. Cada característica es traducida en consultas SQL o en una expresión regular de acuerdo a los artefactos disponibles en el modelo del código fuente. Finalmente se desarrolla un ejemplo comparativo con otras herramientas de recuperación de patrones de diseño como PINOT y FUJABA, los cuales manejan enfoques diferentes a las anotaciones, con lo cual se determina que la detección de patrones con el enfoque de anotaciones conlleva a una mejor detección de los patrones de diseño con un grado menor de falsos positivos y falsos negativos.

## 3- Design Pattern Support Based on the Source Code Annotations and Feature Models.

Peter Kajsa et al [67] realizan un trabajo similar al tratar de recuperar y documentar los patrones de diseño a través del código fuente. En la Figura 7 se describe los componentes de la anotación donde se modela un determinado grupo de patrones de diseño. Esta anotación se utiliza como base para definir los roles que especifican un determinado patrón, esto con el objetivo de cargar un modelo de características con sus clases y métodos a través de los atributos marcados en la anotación, tanto para la instanciación del patrón como para su evolución. Para la realización de los patrones de diseño a través de anotaciones de código se generan plantillas con el Framework de Oracle JET, el cual es encargado de capturar a través del modelado en UML, los elementos marcados con estereotipos nombrados de igual forma que las anotaciones disponibles. Esto, con el objetivo de crear cada elemento participante del patrón con las anotaciones de código correspondientes directamente marcadas en el código fuente.

```
public @interface DesignPattern{
    PatterNames patterName();
    String instanceAlias();
    RoleNames roleName();
    String variant() default "DEFAULT";

    enum PatterNames{ Observer; //...
    }
    enum RoleNames{Subject, attach, detach, notifyObservers,
        update, observers, Observer, ConcreteObserver; //...
    }
```

Figura 7, Anotación con modelo de patrón de diseño

La evaluación de esta herramienta la hacen mediante experimentos en los cuales se pide realizar la implementación de varias instancias del patrón observador, un grupo con la herramienta y otro grupo sin la herramienta tomando en cuenta el tiempo en el que realizan las tareas asignadas. Los resultados son positivos para la herramienta indicando una mejora significante en el promedio de los resultados para el grupo con anotaciones de código.

### 4- Recording concerns in source code using annotations.

M. Sulir et al. [68] presentan una técnica para registrar todas las intenciones de los desarrolladores que quedan implícitas en componentes de código, mediante el uso de anotaciones en lenguaje Java en términos de preocupaciones o "concerns" del software. Estos autores, realizan un estudio para determinar si los desarrolladores comparten o superponen decisiones de diseño en el desarrollo del código fuente mediante la creación individual de anotaciones por cada preocupación, además, realizan un segundo estudio con las anotaciones creadas anteriormente, con el fin de determinar el uso de estas por terceros. La segunda parte del trabajo, se enfoca en un experimento controlado realizado con estudiantes de maestría en Ciencias de la Computación, y en un experimento replica con desarrolladores con experiencia industrial, todos con conocimientos en lenguaje Java para evitar el ruido en la comprensión del programa. En ambos experimentos se dividen los grupos en dos: Al primer grupo se les asigna un proyecto con anotaciones de preocupación en el código fuente, y al segundo grupo se les asigna un proyecto sin anotaciones. Las tareas que debían realizar involucran mantenimiento en el código fuente, con el fin de confirmar la hipótesis de que las anotaciones de preocupación tienen un efecto positivo en la comprensión y el mantenimiento del sistema. Uno de los aportes de este trabajo es el estudio de los modelos mentales de los desarrolladores, en el cual concluyen que las intenciones de los programadores se superponen en el código fuente. Otro aporte son los experimentos realizados, los cuales concluyen que las anotaciones del código fuente tienen un aporte positivo en el tiempo requerido para la comprensión y el mantenimiento del sistema.

## Aportes

Los resultados de la revisión anterior permiten observar que existen esfuerzos comunes alrededor del mundo para aportar a la captura, documentación y gestión del Rationale Arquitectónico. El estado del arte nos muestra los diferentes enfoques con los que se trata de gestionar el conocimiento implícito relacionado con las decisiones arquitecturales, así como también, nos

permite observar los diferentes enfoques, modelos y herramientas que utilizan las anotaciones de código fuente, como una propuesta para manipular información que frecuentemente está implícita en diferentes artefactos generados en el proceso de desarrollo. Todos los trabajos anteriormente mencionados, recalcan la importancia de documentar las razones que se encuentran detrás de las decisiones arquitecturales que se toman durante el proceso de diseño del software, también nos indican los múltiples beneficios que conlleva documentar el Rationale Arquitectónico, así como también de las dificultades y las restricciones que ocurren cuando se quiere gestionar. Los resultados de las investigaciones nos permiten observar que hay avances positivos en la gestión del Rationale Arquitectónico, mediante diferentes enfoques, modelos y herramientas. Sin embargo, también podemos observar que todavía hay mucho por explorar en la gestión del conocimiento en el diseño de software.

De acuerdo a lo anterior y en concordancia con los objetivos de investigación expresados en este trabajo, a continuación, se resaltan los principales aportes de esta propuesta de investigación:

1. A nivel técnico esta herramienta permite gestionar información tácita e implícita en el código fuente a través de anotaciones de Java. Este enfoque no está ligado a un entorno de desarrollo específico y tiene total libertad para ser agregado como una dependencia a un proyecto realizado en Java.
2. El uso de las anotaciones de código hoy en día se ha incrementado mucho desde la aparición de frameworks de desarrollo, los cuales utilizan dichas anotaciones para extraer información desde el código fuente y generar archivos en tiempo de compilación o ejecución que faciliten la vida de los programadores. Las posibilidades de uso que tienen las anotaciones de código son muchas y solo dependen de las necesidades funcionales que se requieran. En este trabajo las anotaciones de código Java, se utilizan para documentar el Rationale Arquitectónico, además de generar en tiempo de ejecución reportes en pdf con la información marcada en el código fuente, los cuales sirven como apoyo para personas externas a la construcción y evolución del código fuente. En la Figura 8 se puede apreciar la información plasmada en el archivo .pdf generado por la herramienta.

| ARCHITECTURAL RATIONALE ANNOTATIONS TOOL | | |
|---|---|---|
| Universidad del Cauca | ORGANIZATION: | Universidad del Cauca |
| | DESCRIPTION: | .jar library to manage the Architectural Rationale through Java Source Code Annotations |
| | VERSION: | 1.0 |
| | AUTHOR: | Santiago Hyun Dorado |
| | CURRENT DATE: | 23 Aug 2018 02-42-27 GMT |

| RATIONALE: | 1 | ID: | 1.1 |
|---|---|---|---|
| TYPE: | Method | NAME: | main |
| PATH: | com.unicauca.arat.presentation.Main.main(java.lang.String[]) | | |
| QUALITY ATTRIBUTES: | | | |
| -PERFORMANCE | | | |
| CAUSES: | | | |
| -Causas que describen la necesidad de calidad | | | |
| TACTICS: | | | |
| -Definición de tácticas(opcional) | | | |
| PATTERNS: | | | |
| -Definición de patrones relacionados a las tácticas | | | |
| ALTERNATIVES: | | | |
| -Alternativas que se consideran para tomaruna decisión | | | |
| DECISIONS: | | | |
| -Registro de decisiones que se toman sobre el tiempo | | | |
| REASONS: | | | |
| -Razones por las cuales se toman las anteriores decisiones | | | |

*Figura 8, Reporte generado por la herramienta*

3. Para la industria de software es de gran importancia documentar adecuadamente la arquitectura de sus sistemas, esto significa que el software pueda ser más fácil de mantener y de modificar con el paso del tiempo. Las empresas utilizan diversas metodologías y procesos de desarrollo, nuestra propuesta no está ligada a ninguna metodología o proceso en particular, es aplicable a cualquier desarrollo en Java. Por su naturaleza ligera, esta herramienta se puede agregar a cualquier proyecto como una librería .jar con gran facilidad y puede ser utilizada con mucha simplicidad de acuerdo a las instrucciones de uso que se generan.
4. Para el estado del arte es un aporte significativo en la representación del Rationale Arquitectónico, ya que en este trabajo se define un modelo que busca representar el dicho Rationale. Este relaciona atributos de calidad, decisiones, las causas y las razones de las decisiones, las diferentes alternativas para cada decisión, las tácticas y patrones arquitecturales y finalmente links a otras fuentes de información.
5. Otro aporte al estado del arte es la forma en la que se utilizan las anotaciones de código como una herramienta sencilla y poderosa para documentar y gestionar conocimiento implícito directamente en el código fuente. Además, se define el diseño de un experimento que se puede reutilizar y replicar con los artefactos desarrollados en este trabajo de investigación.
6. Este trabajo tiene un impacto positivo también en la academia, debido a que los temas relacionados con el Rationale Arquitectónico no se mencionan frecuentemente y este trabajo permite identificar este concepto junto con sus problemas y la importancia de documentarlo, además brinda una forma práctica y sencilla de capturarlo y gestionarlo. Los resultados de este trabajo son un inicio en la documentación del Rationale Arquitectónico

a través de anotaciones de código, este tema puede seguir siendo estudiado como una línea propia de investigación de la cual pueden salir muchos resultados relevantes.

7. Para el grupo de investigación representa una nueva línea de investigación sobre el uso de anotaciones de código en la documentación del diseño y el código, la cual puede ser ampliamente estudiada y de la cual pueden surgir más trabajos de investigación. Además, se aporta una publicación científica en la revista indexada Springer (Ver Anexo N° *8: Carta aceptación de publicación en Springer).*
8. Para el semillero de investigación IRIS es un aporte en la visualización del semillero representando a la Universidad del Cauca en el 13 Congreso Colombiano de Computación (Ver Anexo N° *7: Certificado de participación en 13CCC)*, así como también en el primer encuentro interno de semilleros. Otro aporte al semillero son las presentaciones realizadas a estudiantes de semestres inferiores con el objetivo de incentivar y promover la investigación sobre el Rationale Arquitectónico.

En la Tabla 6 se muestran una clasificación de los principales trabajos que relacionan las anotaciones de código y el Rationale Arquitectónico.

Tabla 6, Comparación de aportes del estado del arte

| Autores | Año | Alcance | Artefactos | Enfoque |
|---|---|---|---|---|
| A.H. Dutoit et al. | 2006 | Diseño | Documentos de texto | Esquemas de argumentación |
| J. Horner et al. | 2006 | Diseño | Documentos de texto | Esquemas de argumentación |
| K. Schneider et al. | 2006 | Diseño | Documentos de texto | Como producto |
| S.J. Buckingham et al. | 2006 | Arquitectura | Hypermedia | Entorno distribuido de información |
| Janet E. Burge et al. | 2008 | Arquitectura | Documentos de texto | Esquemas de argumentación |
| M.A. Babar et al. | 2006 | Arquitectura | Documentos de texto | Esquemas de argumentación |
| L. Bass et al. | 2006 | Arquitectura | Documentos de texto y Diagramas | Esquemas de argumentación |
| J.R Burge et al. | 2006 | Diseño | Herramienta eclipse | Modelo de Rationale |
| J. S. Van der Ven et al. | 2007 | Diseño | Código fuente | Lenguaje de Descripción de Arquitectura |

| | | | | |
|---|---|---|---|---|
| L. Bratthall et al. | 2000 | Diseño | Documentos de texto | Esquemas de argumentación |
| M. Shahin et al. | 2014 | Diseño | Documentos de texto | Architectural Design Decisions |
| J. Aldrich et al. | 2002 | Arquitectura | Código fuente | Lenguaje de Descripción de Arquitectura |
| T. M. Hesse et al. | 2016 | Diseño | Código fuente | Anotaciones |
| Milan N. et al. | 2015 | Diseño | Código fuente | Anotaciones |
| Ghulam Rasool et al. | 2010 | Diseño | Código fuente | Anotaciones |
| Peter Kajsa et al. | 2012 | Diseño | Código fuente | Anotaciones |
| M. Sulir et al. | 2016 | Diseño | Código fuente | Anotaciones |
| Hyun et al. | 2018 | Arquitectura | Código fuente | Anotaciones |

# Capítulo 3: Definición del modelo de Rationale Arquitectónico

## Principios fundamentales

Para comprender mejor la definición y la importancia del Rationale en este trabajo se aborda el tema mediante la metáfora del muro, esta metáfora explica con un ejemplo de la vida cotidiana, cómo las razones que soportan una decisión suelen ser olvidadas y cómo estas razones frecuentemente toman valor a través del tiempo.

Imagine que usted es el nuevo arquitecto de uno de los hoteles de su localidad, como primera tarea usted tiene que rediseñar la parte frontal del hotel teniendo en cuenta los planos establecidos por el arquitecto anterior. Revisando la fachada del edificio y los planos disponibles, se da cuenta que hay un muro que no aparece en los planos de la construcción, pero que en efecto se encuentra ubicado sin aparente funcionalidad alguna. Como primera medida usted opta por verificar la funcionalidad inmediata de este muro sin obtener resultado alguno diferente a sostener una cámara de seguridad, por lo cual decide prescindir de este y en su lugar ubicar un poste de concreto. Después del paso de los días, el invierno llega a la zona y las lluvias provocan grandes concentraciones de agua con lodo, la cual baja desde las montañas hasta las calles de la zona. Debido a que el hotel no cuenta con puertas capaces de aislar el agua con el lodo, se empieza a inundar y los huéspedes comienzan a sentir incomodidad e inconformidad con las instalaciones del hotel, causando un impacto negativo en su prestigio además de los costos extra que involucra remediar esta situación. El muro que anteriormente parecía no tener funcionalidad alguna servía para desviar el curso del agua con el lodo hacia un lugar a una distancia segura de la puerta del hotel. Si el arquitecto desde un comienzo hubiese conocido las razones por las cuales se estableció ese muro en dicha ubicación, se hubiesen evitado sobre costos y retrabajos en tareas que aparentemente estaban terminadas. Como se puede observar en el ejemplo anterior, las decisiones que se toman respecto a una determinada preocupación están motivadas por razones que justifican dichas decisiones, estas razones siempre están seguidas de beneficios, consecuencias, alternativas, restricciones, reglas y otros recursos que en conjunto soportan una decisión en particular. Estas razones o Rationale se genera en el momento en que se toma una decisión, por lo cual sería conveniente capturarlo y documentarlo en ese preciso momento. Sin embargo, frecuentemente se tienen preocupaciones de momento que aportan mayor valor al cliente a corto plazo, por lo cual capturar y documentar el Rationale no hace parte de una de las actividades que generan valor.

De manera similar, los reprocesos y los sobre costos en el mantenimiento de grandes sistemas software se deben en gran parte a la falta de comprensión de las razones por las cuales dicho sistema tiene sus componentes organizados de una forma o de otra. Haciendo una analogía con el ejemplo anterior podemos comparar el hotel con un sistema software, en el cual han participado diferentes personas en su construcción y del cual se quieren realizar cambios que conduzcan a la evolución del mismo. De igual forma al arquitecto tradicional, el arquitecto de software debe comprender la estructura y las razones que definen un sistema, con el objetivo de llevar a cabo la evolución de este con la mejor optimización de los recursos disponibles. En el caso del muro, este se puede comparar con un componente software que está susceptible a cambios por necesidades de los stakeholders o interesados del proyecto, en muchas ocasiones estas necesidades motivan

cambios que en un principio parecen sencillos y sin mucho impacto, sin embargo, cuando estos se efectúan se desprenden grandes inconvenientes, los cuales incrementan cuando se desconoce o se obvian las razones detrás de las decisiones. El invierno, el lodo y el cauce del agua representan las condiciones que motivan a un arquitecto a tomar una decisión respecto a dicha preocupación. Finalmente se encuentran los huéspedes, quienes personifican a los stakeholders de un proyecto, estos al momento de ver los retrocesos, los sobre costos y los daños causados por las circunstancias pierden motivación y credibilidad sobre el sistema haciendo que la inseguridad, la inconformidad y la incomodidad aumenten.

Generalmente en el proceso de creación de software, las personas que inicialmente hicieron parte de la construcción, no siempre se encuentran disponibles en el momento en que se quiere realizar un cambio a largo plazo, esto provoca una dependencia directa con las personas que fueron encargadas de realizar dicho componente, causando que el tiempo necesario para efectuar el cambio incremente, debido a que se debe dedicar gran parte de este tiempo a comprender la estructura definida por los autores y las decisiones que se efectuaron sobre los componentes. En muchas ocasiones las razones que explican las decisiones que se efectuaron no son explícitas y como tarea adicional el encargado de realizar un cambio debe tratar de comprender las decisiones que se tomaron de acuerdo a lo que se encuentra implementado y documentado. Sin embargo, estas razones generalmente se encuentran implícitas en muchos artefactos del desarrollo, complicando la labor de la persona encargada y causando que haya diferentes ambigüedades sobre los supuestos que debe hacer dicha persona sobre lo construido.

Debido a lo anterior, hoy en día se presta mucha atención al conocimiento implícito que existe en el desarrollo de software, con el objetivo de garantizar una mejor capacidad para mantener los sistemas legados y disminuir el tiempo de que se requiere para llegar a una completa comprensión de las estructuras definidas con anterioridad. En este trabajo se presenta una alternativa para capturar y documentar dicho conocimiento implícito a nivel de diseño de arquitectura, específicamente para gestionar las razones que están detrás de las decisiones de diseño que componen un sistema. Para ello, se define un modelo de Rationale Arquitectónico teniendo presente los esfuerzos que se han hecho con anterioridad para establecer una definición que represente este Rationale a través de diferentes enfoques y tecnologías, brindando como principal aporte la definición de un modelo propio a través de tecnologías de uso creciente como lo son las anotaciones de código fuente en Java. A continuación, se presenta la evolución del modelo del Rationale que se estableció en este trabajo.

# Modelo conceptual

Un modelo conceptual busca representar conceptos abstractos que tienen como objetivo explicar una situación o un sistema de la vida real. En este trabajo se define un modelo que representa los componentes más destacados del Rationale arquitectónico, realizando una evolución desde la definición básica del Rationale, hasta llegar a una definición completa que contemple aspectos importantes de arquitectura.

El modelo inicial de Rationale parte de la definición básica descrita a continuación:

“Conjunto de razones o bases lógicas para un curso de acción o creencia” (Oxford)

“Las razones o intenciones para un conjunto particular de pensamientos o acciones” (Cambridge)

En esta definición se encuentran las causas o motivaciones que conducen a la toma de una decisión, así como también las razones por las cuales se selecciona una determinada opción de un espacio posible de decisiones. En la Figura 9 se muestra el primer modelo conceptual creado a partir de las anteriores definiciones. Con este simple modelo se pueden documentar importantes razones, que motivan las decisiones de diseño de cualquier sistema software. Incluso este modelo se podría implementar para registrar las razones de cualquier decisión, debido a que todas las decisiones poseen causas y razones que explican su existencia.

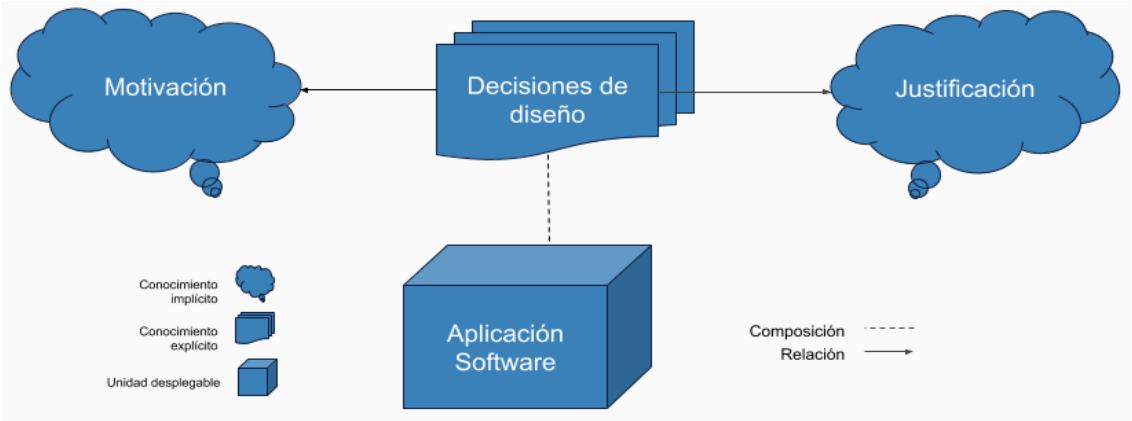


Figura 9, Primer modelo de Rationale arquitectónico

La representación del modelo de Rationale Arquitectónico que se define en primera instancia se muestra en la Figura 10, este primer acercamiento al modelo de anotaciones permite registrar además de las justificaciones y las causas que motivan las decisiones arquitecturales, un identificador que permite reconocer los componentes marcados en el reporte junto con el nombre del componente.

```
@Documented
@Retention(RUNTIME)
@Target({METHOD,PACKAGE,TYPE})
public @interface Rationale1 {
        String id();
        String componentName();
        String motivation() default "";
        String justification();
}
```

Figura 10, Primera implementación del modelo de Rationale arquitectónico

Esta declaración permite anotar componentes a nivel de paquete, clase y método. Además, permite la captura de información de los componentes anotados en tiempo de ejecución y la agregación de estos en el documento generado por la herramienta javadoc. A continuación, se muestra en la Figura 11 cómo se ve la anotación en uso.

```
@Rationale1(
        id = "1",
        componentName = "com.aratevolution.example",
        motivation = "Motivación de la construcción del componente(opcional)",
        justification = "Justificación de las decisiones"
)
package com.aratevolution.example;
```

Figura 11, Uso del primer modelo de Rationale arquitectónico

Este modelo contempla conocimiento explícito, como son las decisiones de diseño que se toman sobre un proyecto. Estas decisiones quedan explícitas en artefactos de diseño e incluso en la misma estructura del sistema. Sin embargo, la justificación y la motivación de estas decisiones frecuentemente quedan en la mente de las personas o inmersas en diferentes artefactos que se generan en el proceso. Para introducir al modelo de Rationale se debe especificar de manera más detallada la composición de las decisiones, para ello se toma por referencia el modelo de decisiones presentado por J.S Van der Ven et al [62], el cual se muestra en la Figura 12.

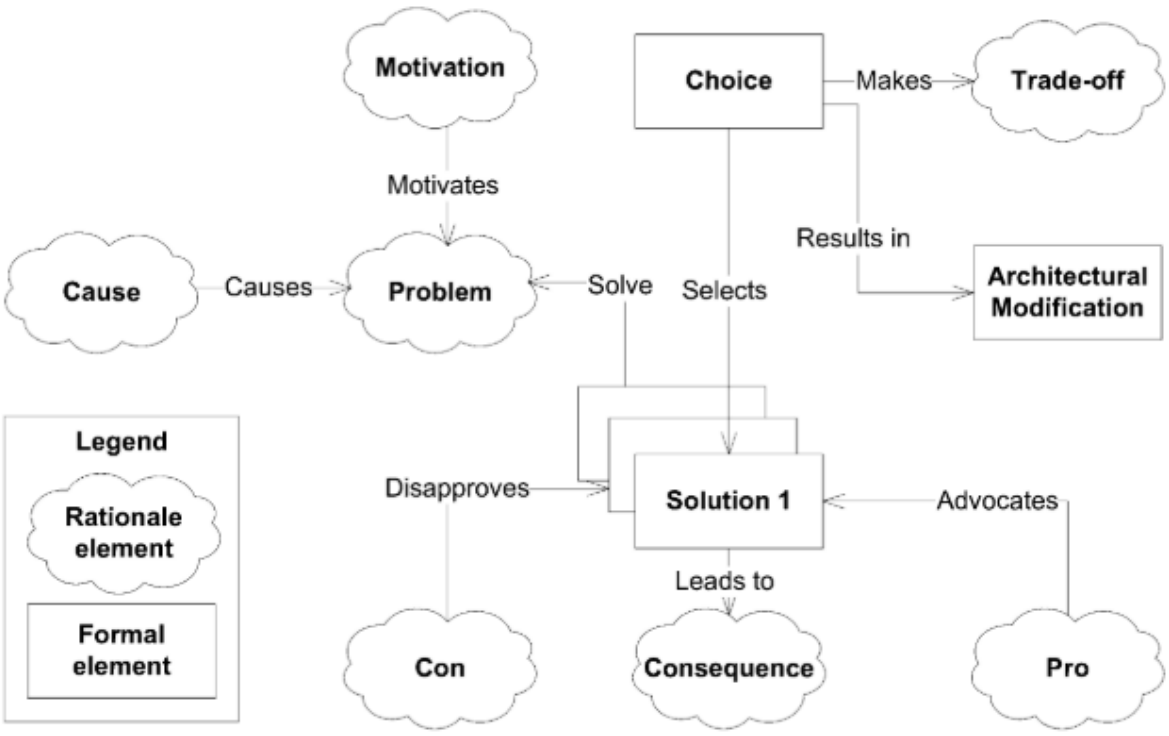


Figura 12, Modelo de decisiones de diseño

En este modelo J.S Van der Ven et al [62] expresan una decisión a través de diferentes elementos formales como son las posibles soluciones con las que se cuentan, la elección de una de esas soluciones y la modificación arquitectural resultante de la selección de una solución. De igual forma, el modelo representa elementos de Rationale como son, el problema, las causas y los motivos que realizan el problema, los pros y los contras y las consecuencias de una determinada solución y las concesiones que se hacen sobre una determinada selección de la solución seleccionada. Sin embargo, este modelo no contempla en ningún elemento las necesidades de calidad que conllevan

a tomar una decisión, de igual forma, el modelo tampoco es explícito sobre las posibles soluciones que se tienen sobre un determinado problema, generalmente estas soluciones conllevan tácticas y patrones implícitos o muchas veces desconocidos por los arquitectos y los desarrolladores, es por eso que se decide adaptar dichos elementos al nuevo modelo propuesto en este trabajo. En la Figura 13 se muestra el nuevo modelo que se crea a partir del presentado por J.S Van der Ven et al [62].

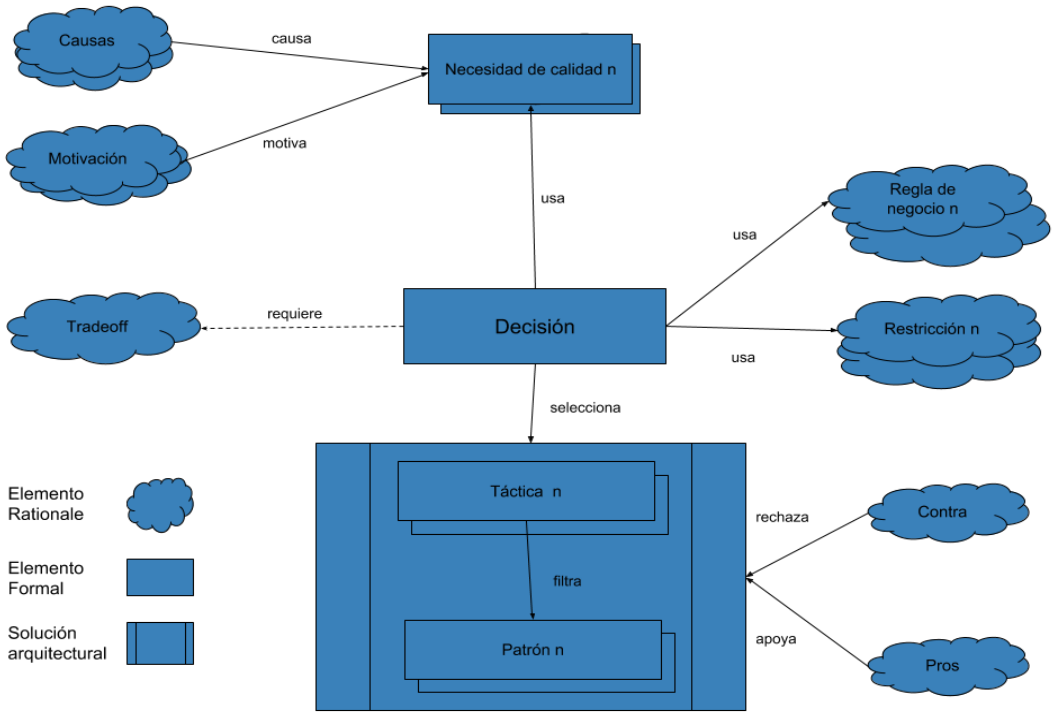


Figura 13, Segundo modelo de Rationale arquitectónico

En esta nueva versión del modelo de Rationale Arquitectónico se tiene en cuenta las necesidades de calidad que promueven una decisión, así como también las tácticas y patrones que conforman una posible solución arquitectural que satisfaga las necesidades de calidad establecidas. Este modelo también contempla las posibles restricciones y reglas de negocio que limitan el alcance de una decisión en específico.

Para la representación de este modelo también se tiene en cuenta la definición de Rationale Arquitectónico brindada por la ISO/IEC/IEEE 42010 [26], la cual especifica el Rationale como la explicación, justificación o el raciocinio de las decisiones que se han tomado, además de incluir los fundamentos de una decisión, las reglas, los límites, las alternativas y concesiones consideradas, las consecuencias potenciales de una decisión y citas a otros recursos con información adicional. También se toma como referencia uno de los trabajos relacionados, en el cual M.A. Babar [71] definen un framework a manera de plantilla para documentar el Rationale arquitectónico, en la Figura 14 se muestran los elementos que componen dicha plantilla. En este framework se toma en cuenta los atributos de calidad, así como también las tácticas y patrones de una determinada solución. Además, brinda información del contexto, del problema, sugerencias y factores que se deben contemplar en la aplicación de los patrones arquitecturales seleccionados.

| **Pattern Name:** *Name of the pattern* | | **Pattern Type:** *Architecture, design, or style* | |
|---|---|---|---|
| **Description** | | *A brief description of the pattern.* | |
| **Context** | | *The situation for which the pattern is recommended.* | |
| **Problem** | | *What types of problem the pattern is supposed to address?* | |
| **Suggested** | | *What is the solution suggested by the pattern to address the problem?* | |
| **Forces** | | *Factors affecting the problem and solution and pattern's justification.* | |
| **Tactics** | | *What tactics are used by the pattern to implement the solution?* | |
| **Affected Attributes** | | **Positively** | **Negatively** |
| | | *Attributes supported* | *Attributes hindered* |
| **Abstract scenarios** | S | *A textual, system independent specification of a quality attribute.* | |
| | S | | |
| **Example** | *Some known examples of the usage of the pattern to solve the problems.* | | |

Figura 14, Framework de gestión de Rationale

La implementación de este modelo en anotaciones de código fuente, se realiza mediante sub anotaciones que representan las decisiones de diseño de igual manera que las alternativas. Similarmente, las causas y los motivos están relacionados a las preocupaciones de calidad, las cuales se definen como anotaciones con valores de tipo enumerado que contienen los atributos de calidad. En la Figura 15 se muestra la implementación en anotaciones de código de esta nueva versión del modelo de anotaciones.

```
@Documented
@Retention(value = RetentionPolicy.RUNTIME)
@Target({METHOD,PACKAGE,TYPE})
public @interface Rationale2 {
    public @interface Decision {
        public String version();
        public String description();
    }
    public @interface Quality_concern {
        public Quality_attribute qualityAttribute();
        public String causes();
        public String reasons();
    }
    public enum Quality_attribute {
        ADECUACION_FUNCIONAL, EFICIENCIA_DESEMPENIO,
        COMPATIBILIDAD, USABILIDAD, FIABILIDAD,
        SEGURIDAD, MANTENIBILIDAD, PORTABILIDAD
    }
    public Quality_concern[] quality_concerns();
    public Decision[] decisions_record();
    public Decision[] alternatives() default {};
    public String id() default "";
}
```

Figura 15, Segunda implementación del modelo de Rationale arquitectónico

En la definición de este modelo ya no se define el nombre del componente, puesto que esta información se puede extraer por métodos reflexivos y le agregan carga al modelo de anotaciones. A continuación, en la Figura 16 se muestra un ejemplo real de este modelo en uso directamente en el código fuente.

```
@Rationale2(id = "1(opcional)",
        quality_concerns = {
            @Rationale2.Quality_concern(
                qualityAttribute = Rationale2.Quality_attribute.FIABILIDAD,
                causes = "Causas de la necesidad de calidad",
                reasons = "razones por las que se aborda la necesidad de calidad"
            )
        },
        alternatives = {
            @Rationale2.Decision(
                version = "1.0",
                description = "Descripción de la decision"
            )
        },
        decisions_record = {
            @Rationale2.Decision(
                version = "1.0",
                description = "Descripción de la decisión"
            )
        }
)
public class PruebaRationale2 {
```

Figura 16, Uso del segundo modelo de Rationale arquitectónico

Sin embargo, según la paradoja del Rationale mencionada por K. Schneider [70] “Cuanto más Rationale se crea, las posibilidades de capturarlo disminuye”, este modelo se vuelve complejo al tratar de aplicarlo cuando existen diferentes decisiones, alternativas y preocupaciones de calidad para un mismo componente, produciendo que no se utilice de manera eficiente y sencilla, interrumpiendo de esta forma las actividades normales del desarrollo, para poder desarrollar una actividad extra de documentación del Rationale arquitectónico, la cual frecuentemente no es bien tomada por los arquitectos y los desarrolladores.

Por lo anterior se define un modelo más sencillo a partir de las observaciones presentadas mediante la paradoja del Rationale, este modelo va enfocado a las razones de las decisiones más que a las decisiones mismas, en la Figura 16 se puede observar el modelo de Rationale Arquitectónico refinado. Este modelo provee un mecanismo simple para documentar las decisiones de diseño que se toman sobre un sistema, así como también las necesidades de calidad y las causas y las razones que componen dichas decisiones. Este modelo también permite registrar las diferentes alternativas que se consideran para la misma preocupación y la estrategia arquitectural junto con sus patrones y tácticas que la conforman. En la Figura 17, se muestra el modelo final presentado en este trabajo con todos sus componentes y descripciones.

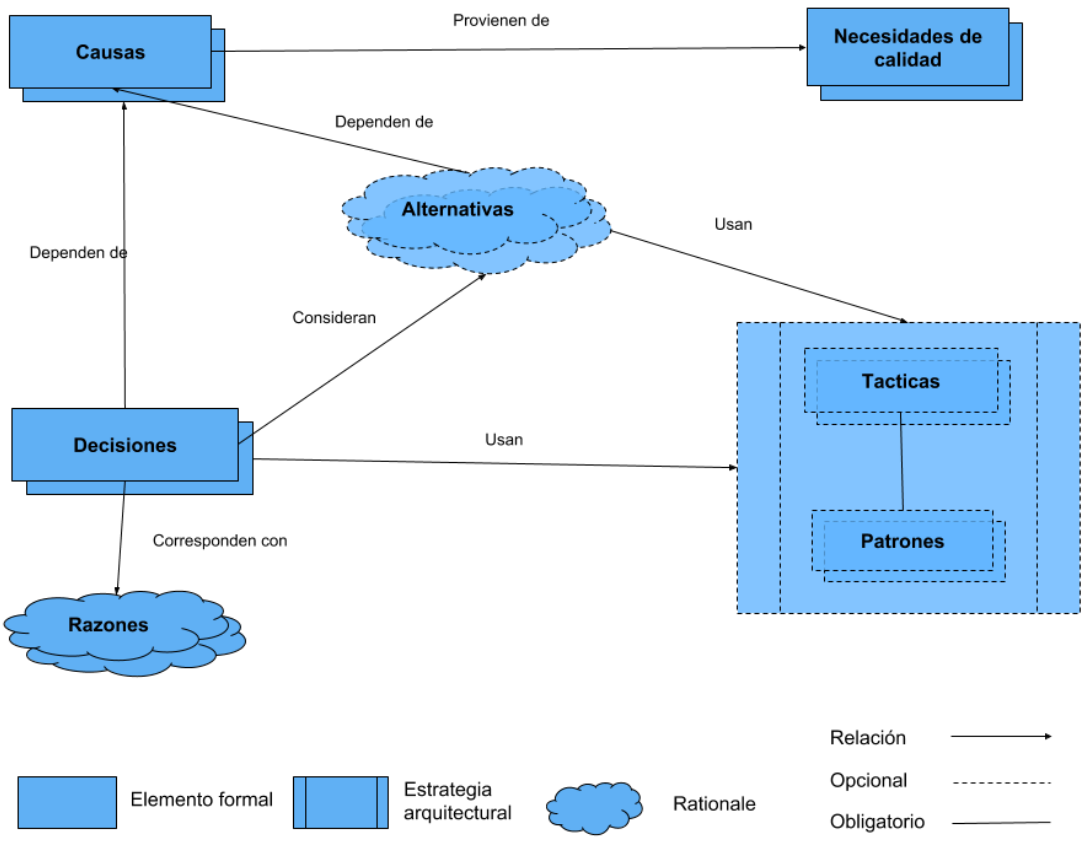


Figura 17, Tercer modelo de Rationale arquitectónico

- Las necesidades de calidad corresponden con escenarios donde es necesaria la definición y la evolución de la arquitectura respecto a atributos de calidad que se hacen evidentes e indispensables en el desarrollo o mantención de un sistema.
- Las causas son las motivaciones que conducen a la toma de un conjunto de decisiones.
- Las decisiones describen las acciones que se realizan para solucionar una determinada preocupación respecto a la funcionalidad o a la calidad del producto software.
- Las razones son la justificación que está detrás de cada decisión de diseño que se toma respecto a la evolución de cualquier sistema.
- Las alternativas representan las opciones adicionales a las decisiones que se toman en el transcurso del tiempo. Generalmente estas alternativas no se documentan y se pierden a través del tiempo, sin embargo, en algún momento en la evolución de la arquitectura estas deben considerarse de nuevo.
- Las tácticas y los patrones arquitecturales dan forma a lo que se denomina una estrategia arquitectural, la cual está construida con el objetivo de dar solución a esas necesidades a nivel funcional como a nivel de calidad.

Finalmente, se establece un modelo de anotaciones sin sub anotaciones, de tal forma que se puedan registrar diferentes decisiones, alternativas, preocupaciones de calidad, tácticas, etc, sin que se vea tan cargada la anotación y sin que aumente la complejidad en el uso de éstas. En la Figura 18 se muestra la declaración final del modelo de anotaciones que permite registrar los elementos del Rationale arquitectónico.

```java
@Documented
@Retention(RUNTIME)
@Target({METHOD, PACKAGE, TYPE})
public @interface Rationale3 {
    enum QualityAtribute {
        FUNCTIONAL_ADECUATION, PERFORMANCE, COMPATIBILITY, USABILITY,
        RELIABILITY, SECURITY, MAINTENANCE, PORTABILITY
    }
    String id() default "";
    boolean hiden() default false;
    String[] links() default {};
    QualityAtribute[] quality_attributes();
    String[] causes();
    String[] tactics() default {};
    String[] patterns() default {};
    String[] alternatives() default {};
    String[] decisions_record();
    String[] reasons();
}
```

Figura 18, Tercera implementación del modelo de Rationale arquitectónico

Esta declaración además de tener los elementos del Rationale Arquitectónico también tiene otros atributos de configuración que permiten ocultar los elementos marcados en el código en la generación de los reportes, además también tiene un elemento que permite agregar links a otras fuentes de información de Rationale arquitectónico. A continuación, se especifica cada uno de los atributos de la anotación. Nótese que los elementos de dividen en atributos de configuración y de información. Los atributos de información son un conjunto de listas, a diferencia de los atributos de configuración.

1. Atributos de configuración (opcionales):
    - id: String que permite identificar una anotación.
    - hiden: boolean que permite ocultar la información de una anotación en la generación del reporte. Por defecto viene con valor false
    - links: Lista de enlaces a otras fuentes de información
2. Atributos de información:
    - quality_atributes: Lista de atributos de calidad que quieren considerar.
    - causes: Lista de causas por las cuales es necesario cumplir con los atributos de calidad.
    - tactics (opcional/Recomendado): Lista de tácticas arquitecturales planteadas para lograr la consecución de los atributos de calidad.
    - patterns (opcional/Recomendado): Lista de patrones que complementan las tácticas arquitecturales.
    - alternatives (opcional/Recomendado): Lista de alternativas que se consideran en el momento de tomar decisiones.
    - decisions_record: Lista de decisiones que se toman en un determinado momento, para cumplir con un grupo de atributos de calidad específicos.

- reasons: Lista de razones que justifican o expresan el porqué de un grupo de decisiones.

En la Figura 19 se muestra cómo se visualiza la anotación en el código fuente, en el momento en que se usa.

```
@Rationale(
        id = "1",
        hiden = false,
        quality_attributes = Rationale.QualityAtribute.PERFORMANCE,
        causes = {"El sistema es un aplicativo web de comercio electrónico muy concurrido "
                + "en el cual se registran más de 1000 peticiones por minuto.",
                  "Cada petición al servidor representa una posible compra, es por esto que "
                + "el sistema debe responder de la manera adecuada y permitir al usuario "
                + "terminar con su transacción de manera satisfactoria y en el menor tiempo posible"},
        alternatives = {"Poner un servidor espejo el cual me responda a las peticiones en caso de "
                        + "que el servidor principal colapse por la cantidad de solicitudes de usuarios.",
                        "Sacar los componentes que se ven más afectados por el volumen de petición de "
                        + "los usuarios y definirlos en un servicio aparte",
                        "Introducir métodos que permitan la concurrencia a través de balanceo de carga.",
                        "Aumentar la capacidad de recursos físicos"},
        patterns = {"Ptrón arquitectural Multinivel"},
        tactics = {"Separación de responsabilidades","abstracción de servicios comunes"},
        decisions_record = {"Se realizan las consideraciones teniendo en cuenta las ventajas y "
                        + "desventajas de cada alternativa, a lo cual se llega que es mejor separar "
                        + "los componentes más afectados en un componente aparte."},
        reasons = {"Poner un servidor espejo no sería de utilidad ya que en caso de que el servidor "
                + "espejo falle se perderían las peticiones del usuario al sistema.",
                   "Respecto a los métodos de balanceo de carga y aumento de los recursos físicos "
                + "no son posibles debido a que no se cuenta con más recursos físicos disponibles."}
)
```

Figura 19, Uso del modelo final de Rationale arquitectónico

# Implementación del Plugin

El desarrollo del plugin se realiza siguiendo las guías del proceso descrito en ICONIX. Las actividades de este proceso están dirigidas a la correcta especificación de los requisitos con el objetivo de evitar ambigüedades desde la recolección de requerimientos hasta la implementación del código fuente. Este proceso consiste de las siguientes actividades:

1. Identificar los objetos de dominio del mundo real a través del modelado de dominio.
2. Definir el comportamiento de los requerimientos a través de diagramas de casos de uso.
3. Desarrollar un análisis de robustez para desambiguar los casos de uso e identificar brechas en el modelo de dominio.
4. Definir el comportamiento de los objetos a través de diagramas de secuencia.
5. Finalizar el modelo estático a través de diagramas de clase.
6. Escribir/Generar el código fuente.
7. Realizar pruebas de aceptación del usuario y de sistema.

También se realiza el manual de usuario del plugin, donde se especifica además de cómo instalar la herramienta, cómo se pueden utilizar los atributos para documentar el Rationale arquitectónico (Ver Anexo N° 3: *Manual de Usuario).*

## Modelo de dominio

En la identificación del modelo de dominio es necesario establecer cuáles son los requisitos con los cuales se pretende desarrollar este plugin, para ello, se toma como referencia las observaciones establecidas por K. Schneider [70], las cuales se presentan a continuación:

- El Rationale se crea cuando se toma una decisión: Las razones que justifican las decisiones arquitecturales se crean al inicio del diseño y en el transcurso del desarrollo. Generalmente estas no se documentan y se pierden con el paso del tiempo.
- Durante el proceso de toma de decisiones los participantes son muy activos: Esto causa que diferentes personas generen Rationale Arquitectónico para decisiones en particular. En muchas ocasiones, éste se sobre escribe o se omiten las opiniones de algunos participantes.
- El Rationale se considera importante y evidente en el momento en el que se crea. Sin embargo, con el paso del tiempo, éste generalmente suele ser olvidado
- Generalmente las decisiones futuras están basadas en decisiones antiguas, provocando que las decisiones se sobrescriban en el desarrollo del proyecto.
- Cuando se combinan el conocimiento y la experiencia se crea un estado de flujo en el cual a las personas les parece fluir una situación que tiene una alta demanda de trabajo. Durante ese estado de flujo las personas no están dispuestas a cambiar de actividades y ser precavidos con el Rationale [74].
- Existe la necesidad de que se exprese el conocimiento tácito involucrado en el desarrollo de las actividades por parte de los participantes, sin embargo, interrumpir el flujo de actividades frecuentemente provoca la pérdida de motivación y puede ralentizar el trabajo. Por ello, los participantes deciden enfocarse en las actividades "principales" y posponer la documentación del Rationale [75] [76].

Según las observaciones anteriores se establecen los siguientes requisitos funcionales:

- RF01: El arquitecto debe tener la capacidad de registrar las razones de las decisiones arquitecturales (Rationale arquitectónico).
- RF02: El desarrollador debe poder ver el Rationale Arquitectónico por el cual los componentes se organizan de una determinada manera.
- RF03: El desarrollador/arquitecto debe poder agregar o modificar el Rationale Arquitectónico desde el código fuente.
- RF04: El arquitecto debe poder agregar enlaces a otros artefactos de documentación.
- RF05: El desarrollador/arquitecto puede generar un reporte con el Rationale Arquitectónico especificado en el código fuente.

Para definir la calidad del producto se entregan los siguientes requisitos NO funcionales:

- RNF01- Compatibilidad: El sistema debe permitir su integración en diferentes sistemas de software desarrollados con el lenguaje de programación Java.
- RNF02- Rendimiento: El sistema debe tener la capacidad de generar los reportes en menos de 1 minuto.
- RNF03- Mantenibilidad: El sistema debe estar bien modularizado con el objetivo de detectar los errores con mayor facilidad.
- RNF04- Mantenibilidad: El sistema debe permitir agregar funcionalidad adicional sin modificar el código ya existente.
- RNF05- Portabilidad: El sistema debe poder instalarse mediante cualquier entorno de desarrollo.
- RNF06- Usabilidad: La herramienta debe permitir su utilización de manera rápida y sencilla

En la Figura 20 se muestra el diagrama del modelo de dominio con los objetos del mundo real que representa la herramienta. Todo parte desde un cliente (en este caso es el arquitecto o el

desarrollador) el cual necesita generar reportes por cada componente marcado, para ello utiliza un tipo de escáner que se encarga de revisar el código fuente y capturar las anotaciones de los diferentes tipos de componentes marcados.

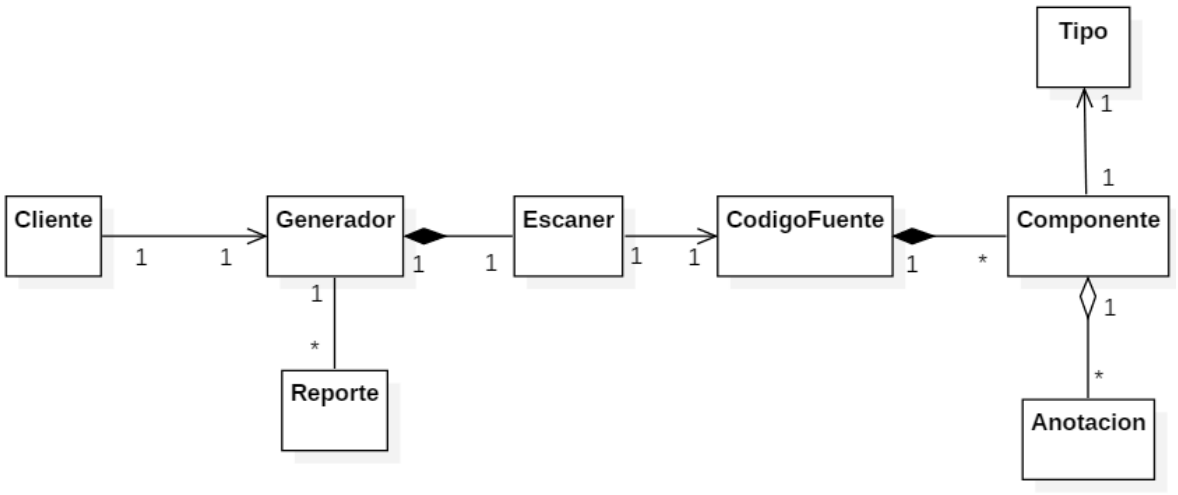


Figura 20, Modelo de dominio de la herramienta ARAT

## Casos de uso

Los casos de uso están estrechamente relacionados con los requisitos funcionales expresados en el ítem anterior. Estos diagramas tienen como función principal mostrar de manera detallada los roles y las funcionalidades a las que tienen acceso los usuarios del sistema. En la Figura 21 se puede detallar todas las funciones descritas anteriormente.

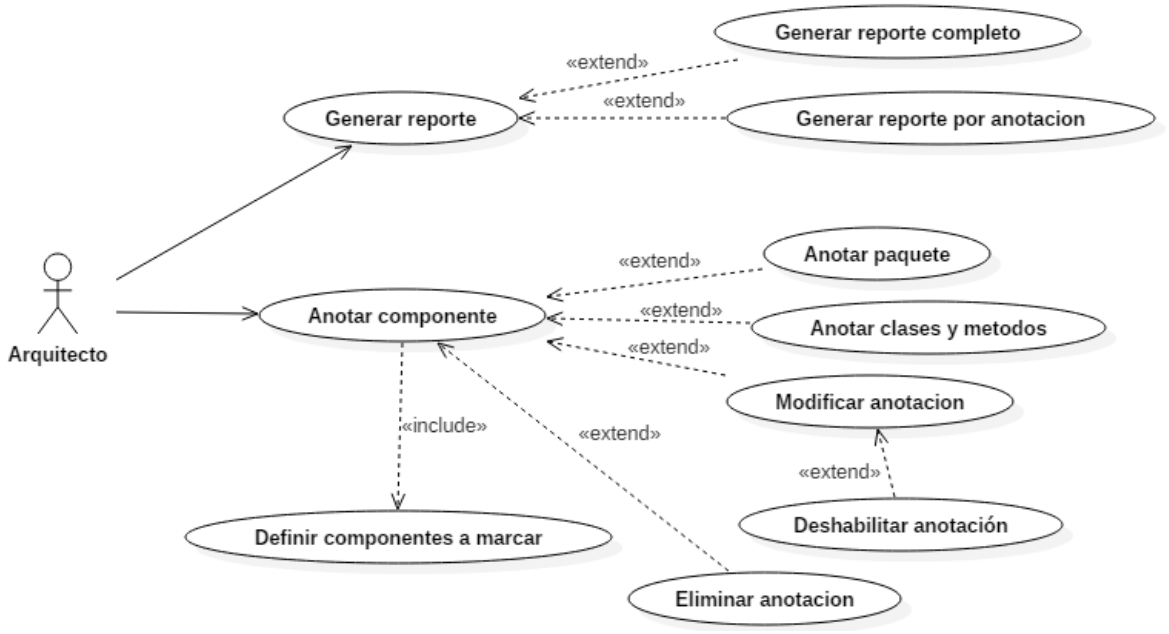


Figura 21, Diagrama de Casos de uso de la herramienta ARAT

- CU01- Anotar componente: El arquitecto tiene la capacidad de anotar componentes de software con el modelo de Rationale Arquitectónico definido a través de anotaciones de

código. Los componentes que el arquitecto puede marcar se derivan en tres niveles diferentes de abstracción desde el más alto hasta el más bajo: Paquete, clase y método.

- CU01.1- Anotar paquete: El arquitecto puede anotar un paquete creando un package-info y estableciendo la anotación en el encabezado de este archivo.
- CU01.2- Anotar clases y métodos: El arquitecto puede marcar la declaración de una o varias clases, así como también puede anotar uno o más métodos de una clase. Las anotaciones no son excluyentes, lo que quiere decir que se puede anotar clases y también los métodos de dichas clases.
- CU01.3- Modificar anotación: El arquitecto puede modificar la información escrita con anterioridad en las anotaciones de código sin ninguna restricción.
  - CU01.3.1- Deshabilitar anotación: El arquitecto puede deshabilitar una anotación para que no aparezca en el reporte, sin embargo, si se puede visualizar en el código fuente.
- CU01.4- Eliminar anotación: El arquitecto puede eliminar una anotación directamente en el código fuente.

- CU02- Generar reporte: El arquitecto tiene la capacidad de generar reportes para documentar cada uno de los componentes marcados con el modelo de Rationale Arquitectónico.
  - CU02.1- Generar reporte individual: El arquitecto puede generar reportes por cada componente marcado en archivos diferentes.
  - CU02.2- Generar reporte general: El arquitecto puede generar un solo reporte con toda la información de los componentes marcados en un solo archivo.
- CU03- Definir componentes: El arquitecto puede establecer los componentes que se van a marcar, como una actividad previa a la generación del código fuente que implementa la funcionalidad del sistema que se pretende documentar.

## Diagrama de robustez

En la Figura 22 se puede observar el diagrama de robustez planteado para esta herramienta. Este diagrama permite observar el flujo, las acciones, los límites, los controladores y las entidades involucradas en la interacción entre el arquitecto y el sistema.

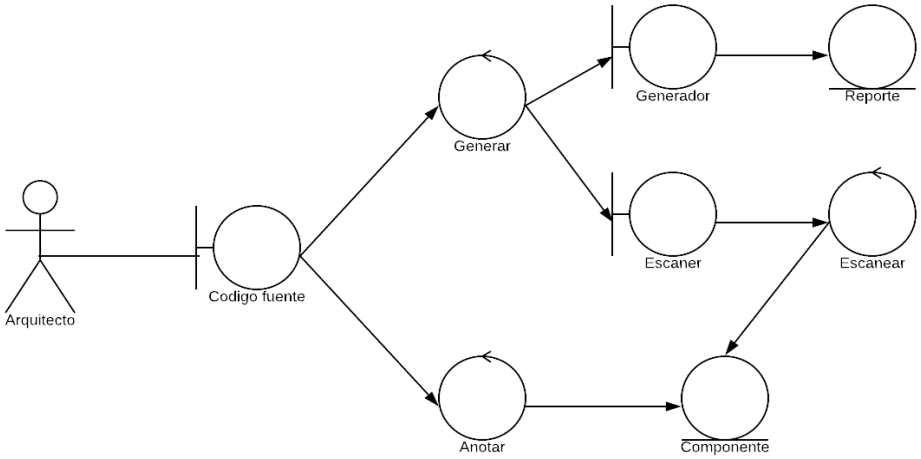


Figura 22, Diagrama de Robustez de la herramienta ARAT

### Diagrama de secuencia

Este diagrama representa el flujo de actividades que realiza el arquitecto en la documentación del Rationale Arquitectónico a través del modelo presentado anteriormente. En la Figura 23, se expresa la comunicación entre los componentes que estructuran la herramienta basada en anotaciones de código.

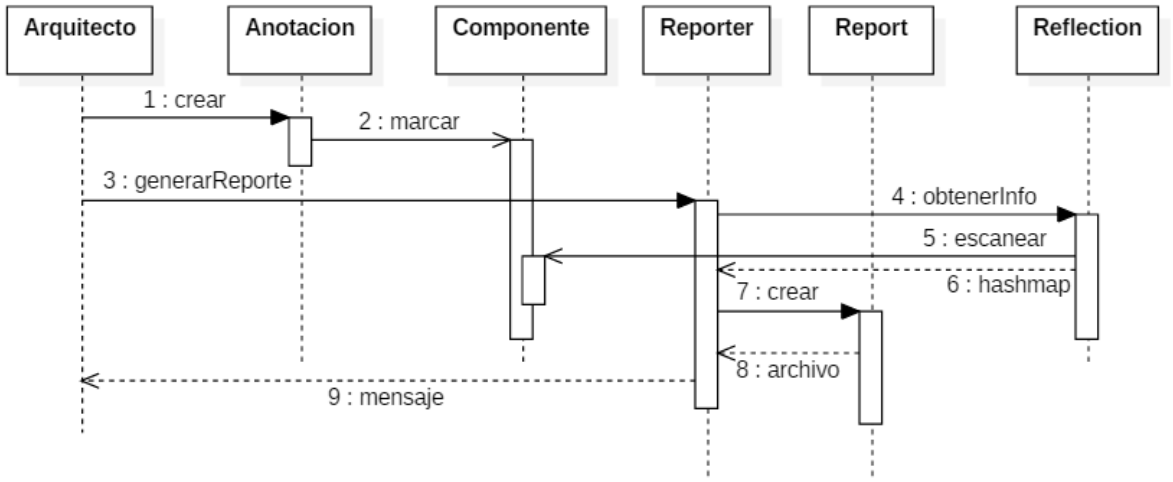


Figura 23, Diagrama de secuencia de la herramienta ARAT

## Diagrama de clase

La organización de las entidades que permiten la interacción entre el arquitecto y las anotaciones de código fuente con información del Rationale Arquitectónico se describe en el diagrama de clases descrito en la Figura 24, en este diagrama se puede ver organización entre las clases principales. En la Figura 25 se puede observar la interacción entre los paquetes definidos en la herramienta.

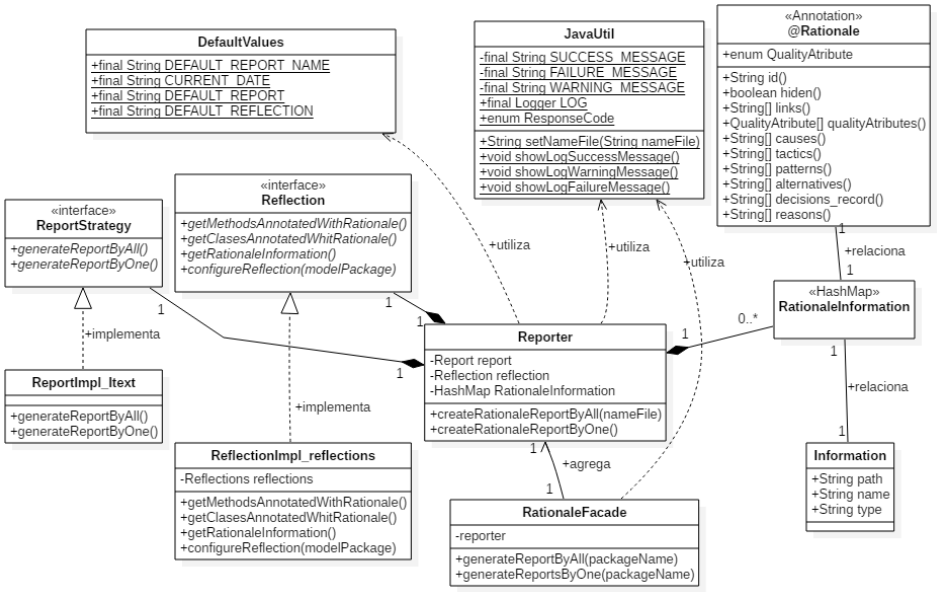


Figura 24, Diagrama de Clases de la herramienta ARAT

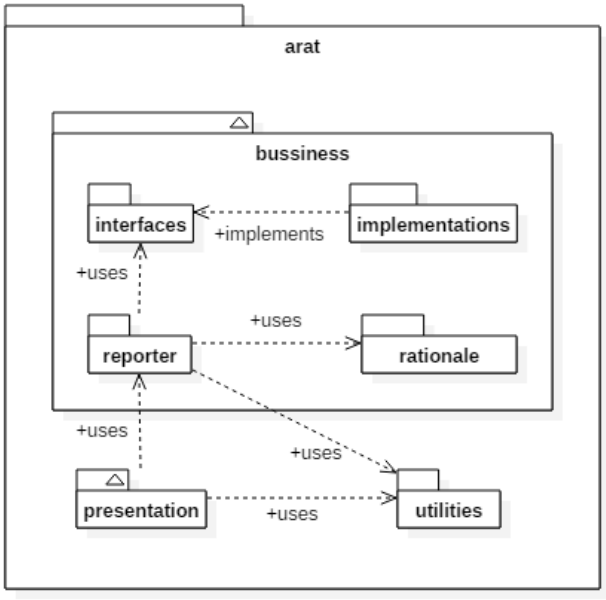


Figura 25, Diagrama de Paquetes de la herramienta ARAT

ReflectionStrategy: encargado de capturar la información marcada en el código fuente a través del modelo de anotaciones

ReportStrategy: es la clase que se encarga de construir el reporte con la información capturada por la estrategia de reflexión.

Reporter: es la clase que permite la orquestación entre la estrategia de reflexión y la estrategia de reporte.

RationaleFacade: expone los servicios de la generación de los reportes al arquitecto mediante métodos estáticos.

## Creación del código fuente

El código fuente se encuentra escrito en lenguaje Java y subido en el siguiente repositorio en GitHub (https://github.com/zahydo/arat-V1.0.git). Este código fuente cuenta con un manual de usuario con las definiciones, los conceptos y los detalles de instalación de la herramienta. Este repositorio cuenta también con su sitio web (https://zahydo.github.io/arat-V1.0/) donde se describe el proyecto, los fundamentos, la importancia del Rationale arquitectónico, los medios de instalación, la forma y las licencias de uso. Además de las características del proyecto, el manual de usuario brinda una guía que potencia la utilidad de la herramienta, así como también un ejemplo de cómo se puede utilizar, la nomenclatura y la jerarquía que gobierna los atributos que describen el modelo.

El código fuente se encuentra marcado con el modelo de Rationale Arquitectónico en anotaciones de código fuente, explicando las razones por las cuales se organizan los componentes en la forma en la que se encuentran definidos. Estas anotaciones se encuentran ocultas para que no salgan en los reportes que realicen desde otros sistemas a los que se integre esta herramienta. Sin embargo, en el código fuente si se visualizan y las personas que en un futuro pueden hacer mantenimiento de este código las pueden encontrar.

## Diagrama de despliegue

La herramienta se empaqueta en un archivo .jar (Ver Anexo N° 4: arat), 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 26 se puede observar cómo la herramienta se integra a un sistema y captura los componentes marcados con las anotaciones de código fuente con información del Rationale arquitectónico, con el objetivo de documentar y generar un reporte con la información estructurada.

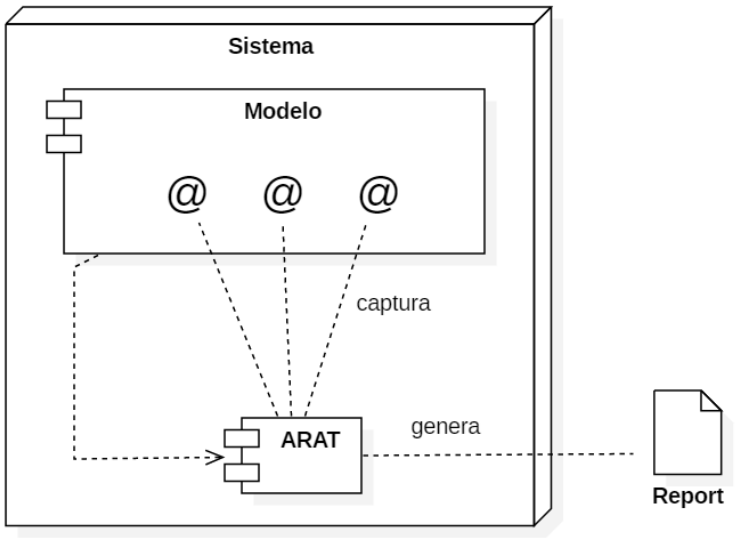


Figura 26, Diagrama de despliegue de la herramienta ARAT

# Capítulo 4: Evaluación del modelo

En este capítulo se aborda el proceso de evaluación de la herramienta, este proceso comienza con un estudio exploratorio, el cual tiene como objetivo tener un acercamiento a la problemática real en la falta de documentación del Rationale arquitectónico. Para ello se realiza un quasi-experimento con ingenieros de software de la región, este estudio exploratorio permite obtener resultados cualitativos como cuantitativos los cuales son alentadores respecto a la utilidad de la herramienta. En el segundo estudio se realiza un experimento controlado con un número mayor de ingenieros, además de implementar otro problema con un contexto diferente, para que la evaluación de la herramienta no dependa de un problema en específico y tenga mayor aleatoriedad en la muestra y en la evaluación de la herramienta.

## Estudio exploratorio

### Planeación del quasi-experimento

El objetivo de este estudio es analizar la documentación del Rationale Arquitectónico a través de anotaciones de código, con el propósito de determinar el valor de las anotaciones de código como herramienta para documentar el Rationale Arquitectónico con respecto a la mantenibilidad de la arquitectura en términos de eficiencia, efectividad y comprensión durante cambios relevantes arquitectónicamente desde el punto de vista del arquitecto y los desarrolladores, en el contexto del mantenimiento de un sistema en una pequeña organización. Para la elaboración de este estudio se hace seguimiento de la guía para reportar experimentos en ingeniería de software propuesta por Andreas J. et al [17].

#### *Hipótesis*

La hipótesis de este experimento es que la existencia de anotaciones de código fuente con información del Rationale Arquitectónico mejora la mantenibilidad de la arquitectura con respecto a la eficiencia, efectividad y comprensión de las decisiones, al realizar cambios arquitecturales. Para confirmar esta hipótesis se formula la hipótesis nula y alternativa para cada aspecto de interés:

**H1:** La efectividad al realizar un cambio arquitectural en un sistema con la documentación del Rationale Arquitectónico en anotaciones de código:

- **Nula:** es menor o igual a la efectividad de realizar el cambio sin anotaciones de código.
- **Alternativa:** es mayor a la efectividad de realizar el cambio sin anotaciones de código.

**H2:** La eficiencia al realizar un cambio arquitectural en un sistema con la documentación del Rationale Arquitectónico en anotaciones de código:

- **Nula:** es menor o igual a la eficiencia de realizar el cambio sin anotaciones de código.
- **Alternativa:** es mayor a la eficiencia de realizar el cambio sin anotaciones de código.

**H3:** La comprensión de las decisiones de la arquitectura y su Rationale al realizar un cambio arquitectural en un sistema con la documentación del Rationale Arquitectónico en anotaciones de código:

- **Nula:** es menor o igual a la comprensión al realizar el cambio sin anotaciones de código.
- **Alternativa:** es mayor a la comprensión al realizar el cambio sin anotaciones de código.

### *Variables*

### Variables independientes

En este trabajo la única variable independiente es la existencia o no de anotaciones de código fuente con información del Rationale Arquitectónico. Sus únicos valores posibles son “Si” o “No”, esta variable es categórica puesto que representa si un participante debe realizar los cambios a un código con anotaciones de código o sin ellas.

### Variables dependientes

Para el cálculo de las variables dependientes eficiencia y esfuerzo, es necesario medir otras variables como el tiempo y el nivel de correctitud en la realización de las tareas. Además de definir los valores estimados, con el fin de obtener los resultados de las variables eficiencia y efectividad en términos de porcentajes. Otra variable dependiente es el nivel de comprensión de la Arquitectura y su Rationale, esta se comprende en valores entre 0 y 100. Los participantes deben realizar 3 tareas a las cuales se les califica según el nivel de correctitud de cada tarea, para la primera, segunda y tercera tarea, el nivel de correctitud se describe en las Tablas 7, 8 y 9 respectivamente.

Tabla 7, Nivel de Correctitud de la tarea 1

| **Nivel Correctitud Tarea 1 (NC1)** | |
|---|---|
| **Valor** | **Descripción** |
| 100 | No realizó el cambio. |
| 200 | Realizó cambios pero no logró terminar la tarea. |
| 300 | Completó la tarea pero no mantuvo el diseño establecido. |
| 400 | Completó la tarea manteniendo el diseño, pero tardó más de lo establecido. |
| 500 | Completó la tarea manteniendo el diseño y en el tiempo establecido. |

Tabla 8, Nivel de Correctitud de la tarea 2

| **Nivel Correctitud tarea 2 (NC2)** | |
|---|---|
| **Valor** | **Descripción** |
| 100 | No realizó el cambio. |
| 200 | Realizó modificaciones pero no logró cambios funcionales. |
| 300 | Hizo la tarea cambiando el diseño y manteniéndose dentro de la arquitectura. |
| 400 | Hizo la tarea cambiando el diseño y la arquitectura. |
| 500 | Hizo la tarea manteniendo el diseño y cambiando la arquitectura. |

Tabla 9, Nivel de Correctitud de la tarea 3

| Nivel Correctitud tarea 3 (NC3) | |
|---|---|
| **Valor** | **Descripción** |
| 100 | No escribió la documentación. |
| 200 | Documentó los cambios a nivel funcional. |
| 300 | Documentó los cambios a nivel de diseño. |
| 400 | Documentó los cambios a nivel arquitectural sin su Rationale. |
| 500 | Documentó los cambios arquitectónicos junto con su Rationale. |

La suma de los niveles de correctitud de todas las tareas se define como **'NCT'** y está expresada de la siguiente manera:

$$\boldsymbol{NCT} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{NCi}$$

(*n*: cantidad total de tareas)

El tiempo en que un participante realiza una tarea se denota por la letra **'t'**. El tiempo en que tarda en realizar todas las tareas **'T'** se define como la sumatoria de cada uno de los tiempos tomados en cada tarea, tal como se expresa en la siguiente ecuación:

$$\boldsymbol{T} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{ti}$$

Los valores estimados que se definen para el tiempo total y la correctitud total se muestran a continuación: el tiempo estimado para la primera, segunda y tercera tarea es de 1 hora, 50 min y 30min respectivamente, sumando un tiempo total estimado **'Te'** para realizar todas las tareas de 140 min. En este tiempo se estima que una persona con las características de inclusión de la muestra poblacional terminaría las actividades.

$$\boldsymbol{Te = 140}$$

Nivel de correctitud total estimado **'NCTe'** en la realización de todas las tareas asignadas.

$$\boldsymbol{NC1 = 500}, \quad \boldsymbol{NC2 = 500}, \quad \boldsymbol{NC3 = 500}$$

$$\boldsymbol{NCTe = 500 + 500 + 500 = 1500}$$

**Eficiencia**

Valor medido respecto a la correctitud y el tiempo total obtenidos en la realización de las tareas, la eficiencia **'E'** se define a continuación como:

$$\boldsymbol{E = \frac{NCT}{T}}$$

Para dar un valor de la eficiencia en términos de porcentaje es necesario establecer un punto de referencia en la realización de las tareas, el cual se define como:

$$\boldsymbol{ER = \frac{NCTr}{Te}}$$

Donde el nivel de correctitud total referente **'NCTr'** es:

$$\boldsymbol{NC1 = 400}, \qquad \boldsymbol{NC2 = 500}, \qquad \boldsymbol{NC3 = 500}$$

$$\boldsymbol{NCTr = 400 + 500 + 500 = 1400}$$

Con lo cual conseguimos el siguiente valor de eficiencia referente 'ER':

$$\boldsymbol{ER = \frac{1400}{140} = 10\ \frac{NCT}{min}}$$

Finalmente obtenemos el porcentaje de eficiencia por cada participante:

$$\%\,\boldsymbol{eficiencia = \frac{E}{ER} * 100}$$

**Efectividad**

Valor medido mediante la correctitud en la realización de todas las tareas dividido por la correctitud total esperada. En este trabajo se define la efectividad 'EF' como:

$$\boldsymbol{EF = \frac{NCT}{NCTe}}$$

Finalmente obtenemos el porcentaje de efectividad de cada participante mediante:

$$\%\,\boldsymbol{efectividad = EF * 100}$$

**Comprensión**

Es un valor cuantitativo evaluado mediante el cruce de información entre las respuestas de una encuesta y la documentación proporcionada en la tarea 3. Para calcular el valor de esta variable se necesita definir el nivel de documentación según el entendimiento expresado por los participan-tes en los diferentes artefactos del experimento.

Tabla 10, Nivel de Comprensión de la arquitectura y su Rationale

| **Participante** | **DC** | | | **DR** | | **C** |
|---|---|---|---|---|---|---|
| | **f** | **d** | **a** | **d** | **a** | |

| | | | | | | |
|---|---|---|---|---|---|---|
| N | 100 | 100 | 100 | 100 | 100 | 100 |

DC: Documentación de los cambios, DR: Documentación del Rationale, f: Nivel funcional, d: Nivel de diseño, a: Nivel de arquitectura.

En la Tabla 10 se muestra el diseño para calcular el nivel de comprensión de la arquitectura y su Rationale, mediante la evaluación del contenido en la documentación de las tareas realizadas y lo escrito en las respuestas de la encuesta. Estas variables auxiliares se encuentran en un rango de 0 a 100 sumando un máximo total de 500. El nivel de comprensión denotado como **'C'**, es el promedio de los valores de las variables auxiliares.

$$C = \frac{DFf + DFd + DFa + DRd + DRa}{5}$$

### *Participantes*

Se hace una convocatoria a través de correos electrónicos, posters y reuniones personales con el fin de obtener la participación de personas que cumplieran con las siguientes características:

1. Conocimientos básicos en Arquitecturas de software.
2. Tener experiencia en Java de por lo menos dos años.

Se realizaron 15 invitaciones a diferentes ingenieros que cumplen con el perfil descrito anteriormente, de cuales 8 de ellos aceptaron. Sin embargo, por diferentes inconvenientes, el día del experimento se presentaron 4 de ellos. Entre los participantes se tienen ingenieros de sistemas y de electrónica que tienen relación en la actualidad con la industria del software. Todos son hombres mayores de 25 años y cuentan con una experiencia entre 2 y 12 años en la industria desempeñando diferentes roles como: analista, ingeniero de producto, desarrollador y arquitecto de software. Además, dos participantes tienen una labor activa como docentes en una institución universitaria relacionada con la ingeniería de sistemas. A los participantes se les explicó el objetivo de la sesión y el proceso que se iba a seguir, a partir de esto se consiguió su consentimiento de forma escrita. Todas las personas seleccionadas tienen conocimientos en campos de la ciencia en computación y afines. Para el reclutamiento de los participantes se siguió la guía de Andrew J. et al [17].

### *Material experimental*

El código fuente experimental es una familia de juegos de 'n' en línea, el cual consiste en un juego en el que cada jugador debe colocar en un tablero n fichas consecutivas en línea de manera vertical, horizontal o diagonal. Este juego tiene muchas variantes, una de ellas se llama 'Cuatro en línea': esta variante es un juego sobre un tablero de siete columnas de ancho y seis filas de alto. Cada jugador usa fichas de un color particular y, alternando el turno, coloca las fichas en el tablero con el fin de lograr ser el primero en lograr que cuatro piezas de su color en una línea, bien sea horizontal, vertical o diagonalmente. Dado que el tablero es vertical, las fichas insertadas en una columna siempre bajarán hasta la posición disponible de la columna.

En la Figura 27 se puede observar la barra de opciones de la interfaz gráfica de usuario, esta tiene como parámetros de entrada el número de casillas contiguas para ganar y el tipo de juego seleccionado.

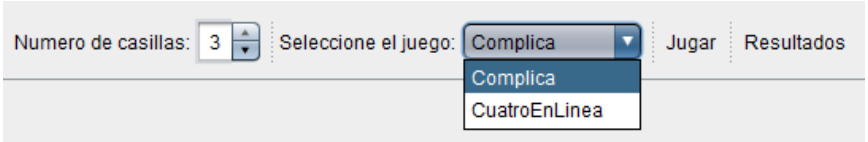


Figura 27, Barra de opciones Material experimental 1

Este código fuente se entrega con dos paquetes 'src' y 'lib'. En 'src' se encuentran todas las clases y paquetes que representan la arquitectura del juego y en 'lib' está la libre-ría 'reflections-0.9.9-RC1.jar' y sus dependencias, la cual permite obtener las clases que heredan de algún tipo. Los participantes que tienen las anotaciones de código con información del Rationale Arquitectónico declaradas en el código fuente tienen una librería de más en la carpeta 'lib' llamada 'ARAT.jar' (Architectural Rationale Annotations Tool: por sus siglas en inglés) la cual permite documentar las razones arquitecturales y mostrar un reporte con la ubicación y la información marcada en las anotaciones. Es necesario instalar el JDK de Java y algún entorno de desarrollo como Eclipse o NetBeans IDE. El código fuente ubicado en 'src' se puede copiar y reemplazar con la carpeta 'src' creada por Eclipse o NetBeans IDE cuando se inicia un nuevo proyecto. La carpeta 'lib' se copia junto a 'src' en el proyecto creado en el IDE y se deben agregar las librerías al path del proyecto.

Finalmente, todos los participantes cuentan con un Documento de Arquitectura de Software SAD (Software Architectural Document: por sus siglas en inglés), en donde se representa la arquitectura desde diferentes vistas: escenarios, lógica, de desarrollo, de proceso y la vista física, con diferentes modelos como: Diagramas de casos de uso, de clases, de secuencia, de paquetes, de componentes y de despliegue. Los cuales permiten representar la arquitectura para diferentes audiencias. Este documento se puede observar en el anexo N° 1: *Reporte estudio exploratorio/Materiales y recursos.*

### *Tareas*

El experimento consiste en la realización de tres tareas para cada participante:

1. Mantenimiento aditivo (Nuevo juego): Es una variante del juego N en línea llamada Complica, en donde el tablero está constituido por cuatro columnas y siete filas.
2. Mantenimiento correctivo (Cambio arquitectural)
3. Documentación del Rationale Arquitectónico (Documentación)

### Mantenimiento aditivo: Nuevo juego

Las fichas pueden ser colocadas en una columna llena, en este caso la ficha se inserta y todas las fichas descienden una posición, eliminando las fichas de abajo. Como consecuencia de esta regla, se presentan situaciones más complejas, por ejemplo, ambos jugadores pueden en una misma jugada lograr colocar n de sus fichas en línea, y entonces el juego deberá continuar. También el jugador que coloca una ficha podría perder el juego. Esto sucedería si la pieza es ubicada en una columna llena y consigue que su oponente quede con n de sus fichas en línea. A esta variante se le denomina Complica.

### Mantenimiento correctivo: Cambio arquitectural

El resultado de un juego se guarda localmente en un archivo ubicado en un directorio del sistema, este directorio puede ser accedido por cualquier persona que sepa dónde se ubican los archivos del proyecto. Se requiere que el sistema tenga la capacidad de delegar la responsabilidad de gestionar los archivos con la información de los resultados a un servidor que esté activo escuchando las

peticiones de las instancias del juego. Esto con el objetivo de mantener seguros los datos en un sistema aparte del juego para que no puedan ser modificados y estos puedan ser accedidos por varias instancias del juego al mismo tiempo. En la Figura 28 se puede apreciar un diagrama de despliegue con la arquitectura inicial del sistema, en la cual podemos ver que el componente de acceso a datos está en el nivel de aplicación.

En la Figura 29 se muestra cómo deben ubicarse los componentes, de tal forma que la responsabilidad de guardar los datos caiga sobre un nuevo nivel de datos.

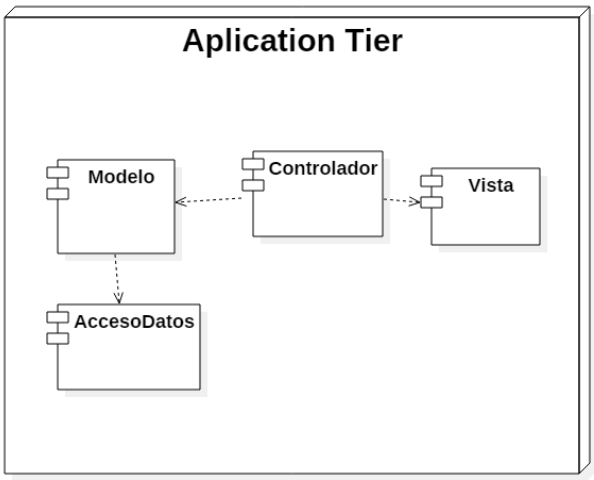


Figura 28, Diagrama de despliegue Material experimental 1

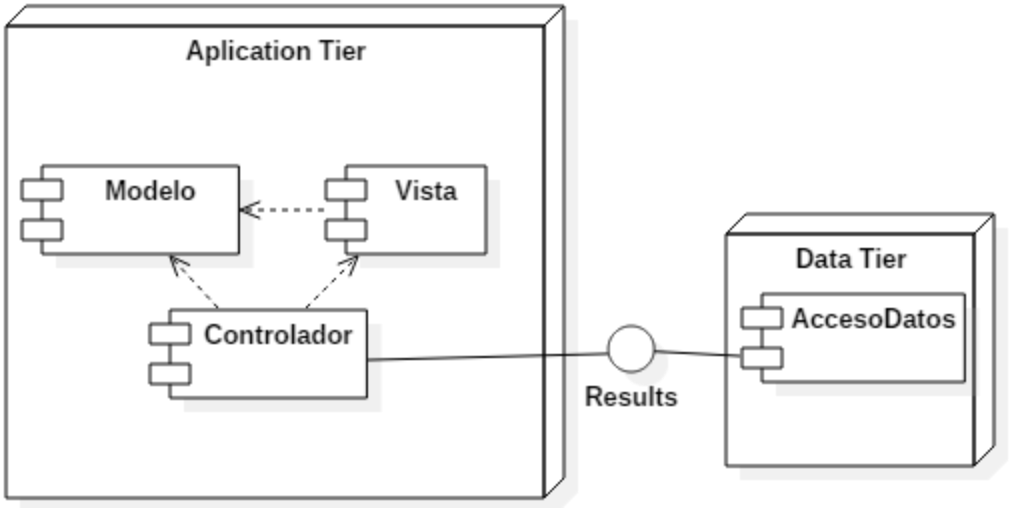


Figura 29, Diagrama de despliegue deseado Material experimental 1

## Documentación del Rationale

Con el objetivo de verificar la comprensión de la arquitectura y su Rationale, es necesario que los participantes documenten los cambios realizados en las tareas anteriormente mencionadas de acuerdo a la estrategia de documentación que le haya correspondido, documento de texto/word o con la primera versión del modelo de anotaciones ARAT. Los participantes también deben escribir cuál es la necesidad de calidad que se quiere alcanzar y cuales técnicas o estrategias se utilizan. Para

terminar esta tarea los participantes se pueden documentar en los diferentes artefactos del software como el Documento de Arquitectura Software SAD (Software Architecture Document: por sus siglas en inglés), el código fuente en bruto y las anotaciones de código si le corresponde.

### *Diseño del experimento*

Los participantes se agrupan inicialmente en 2 parejas, el procedimiento de agrupación se realiza utilizando una herramienta web [77], la cual permite crear grupos aleatorios adjuntando los nombres de los participantes separados por coma. En la Figura 30 se puede observar la distribución de los grupos.

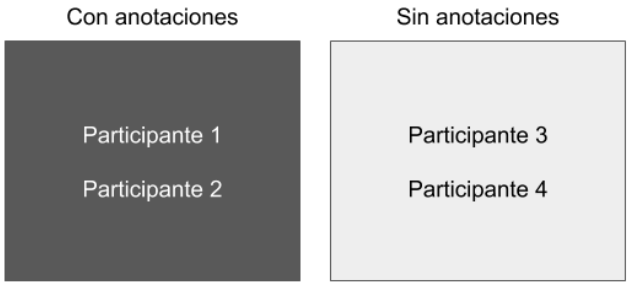


Figura 30, Distribución de los participantes Estudio exploratorio

### *Procedimiento*

El experimento inicia haciendo una introducción al Rationale Arquitectónico con el objetivo de familiarizar el término a los participantes, seguido de esto se hace la presentación de las anotaciones de código fuente en Java, realizando un ejemplo para comprender la declaración, el uso y algunas restricciones. Después, se muestra el modelo de anotaciones con información del Rationale Arquitectónico y se procede a explicar el sistema y las tareas que se deben realizar sobre este. Finalmente se hace la selección de los grupos de manera aleatoria, se entrega el material experimental adecuado y se empiezan a tomar tiempos en la terminación de cada tarea. Por último, después de realizar todas las tareas y entregar los resultados, cada participante debe completar una encuesta con las siguientes preguntas:

1. Describa las razones por las cuales el código fue organizado bajo la estructura sobre la cual trabajó.
2. Describa las razones por las cuales la estructura arquitectónica sobre la cual trabajó tuvo que ser alterada para satisfacer los cambios solicitados.
3. ¿La información del Rationale brindada en esta experiencia fue de utilidad para realizar las modificaciones?
4. ¿Por qué cree usted que es importante documentar el Rationale Arquitectónico en el desarrollo de software?
5. ¿Qué información considera es la más relevante para documentar el Rationale de la Arquitectura?

### *Procedimiento de análisis*

Luego de tomar los tiempos de cada participante se reciben los sistemas con las modificaciones con el objetivo de evaluar el nivel de correctitud en la realización de las tareas. Para determinar los resultados del experimento se realiza una prueba de hipótesis con la cual se busca confirmar que el

uso de las anotaciones de código con información del Rationale Arquitectónico mejora la mantenibilidad de la Arquitectura en términos de eficiencia, efectividad y comprensión de la Arquitectura y su Rationale, cuando se realizan cambios con un impacto a nivel arquitectural. La prueba de hipótesis que se utiliza es una prueba t student, esta permite determinar si existe una significancia estadística entre dos variables, esto quiere decir que la presencia de la variable independiente afecta positivamente los resultados de las variables dependientes. Para esta prueba es necesario tener dos muestras iguales y se debe suponer igual varianza entre las muestras de cada grupo. En este trabajo los grupos se dividen en: Grupo Con Anotaciones (GCA) y Grupo Sin Anotaciones (GSA).

## Ejecución del quasi-experimento

A los participantes se les da una introducción del término Rationale Arquitectónico en donde se mencionan algunos problemas que se causan no documentarlo. También se les da una capacitación sobre el uso de las anotaciones de código en Java, algunos casos de uso populares y un ejemplo en el cual se busca aclarar los conceptos de anotaciones de código. Después, se explica el modelo de anotación con el que se realiza el experimento. Seguido de esto, se les explica el funcionamiento del sistema y algunas partes de su arquitectura. Finalmente se explican las tareas que se deben realizar, cómo crear un nuevo proyecto en NetBeans IDE y cómo agregar las clases y paquetes del sistema al nuevo proyecto. El desarrollo del sistema se implementó en Eclipse IDE, por lo cual se pensaba en realizar los cambios del experimento en el mismo IDE. Sin embargo, se separan las clases y los paquetes del proyecto en una carpeta aparte 'src' junto con las librerías necesarias para su correcto funcionamiento 'lib'. Esto permitió desarrollar el experimento en el entorno de desarrollo NetBeans, el cual se encuentra preinstalado en las salas de la Universidad del Cauca. Después de capacitar a los participantes se les hace entrega del material experimental. Se hace la configuración del proyecto en el entorno de desarrollo NetBeans y se copian los archivos descargados del código fuente del material experimental. Los participantes empiezan a realizar las tareas y se capturan los tiempos de realización de cada tarea para cada participante. A medida que cada participante finaliza las tareas se le entrega la encuesta, con el objetivo de tener resultados cualitativos que aporten en la definición de la estructura final del modelo de anotaciones para el Rationale Arquitectónico.

A continuación, se muestran algunas de las respuestas más acertadas a la encuesta realizada a los participantes:

1. Describa las razones por las cuales el código fue organizado bajo la estructura sobre la cual trabajó.
   **RTA:** El sistema fue organizado así para facilitar la mantenibilidad y la extensibilidad de la arquitectura.

2. Describa las razones por las cuales la estructura arquitectónica sobre la cual trabajó tuvo que ser alterada para satisfacer los cambios solicitados.
   **RTA:** Porque se requirió una nueva funcionalidad en un nuevo componente donde se guardarán los datos por seguridad.

3. ¿La información del Rationale brindada en esta experiencia fue de utilidad para realizar las modificaciones?

**RTA:** En términos generales los participantes respondieron que si fue de utilidad la estrategia de documentación que les correspondió.

4. ¿Por qué cree usted que es importante documentar el Rationale Arquitectónico en el desarrollo de software?
   **RTA:** A nivel general, los participantes respondieron a esta pregunta qué es importante para facilitar la mantenibilidad de un sistema a personas subsiguientes a los que lo desarrollan.

5. ¿Qué información considera es la más relevante para documentar el Rationale de la Arquitectura?
   **RTA:** En esta pregunta los participantes nombraron los siguientes aspectos que les parecieron importantes: Tipo de requerimiento a satisfacer, prioridad del requerimiento, historial de modificaciones al Rationale.

En la Tabla 11 se muestran los tiempos capturados en cada tarea para cada uno de los participantes de los dos grupos. Seguido de esto, en la Tabla 12 se puede apreciar el nivel de correctitud evaluado en las tareas de cada participante. Por último, en la Tabla 13 se muestran los valores para el nivel de comprensión de la Arquitectura y su Rationale.

Tabla 11, Tiempos en la realización de las tareas Estudio exploratorio

| **Grupo Sin Anotaciones GSA** | | | | |
|---|---|---|---|---|
| **Participante** | **t1** | **t2** | **t3** | **T(min)** |
| 1 | 101 | 32 | 16 | 149 |
| 2 | 54 | 35 | 46 | 135 |
| **Grupo Con Anotaciones GCA** | | | | |
| 1 | 52 | 90 | 2 | 144 |
| 2 | 104 | 45 | 2 | 151 |

Tabla 12, Nivel de correctitud de las tareas Estudio exploratorio

| **Grupo Sin Anotaciones GSA** | | | | |
|---|---|---|---|---|
| **Participante** | **NC1** | **NC2** | **NC3** | **NCT** |
| 1 | 300 | 300 | 200 | 800 |
| 2 | 500 | 500 | 400 | 1400 |
| **Grupo Con Anotaciones GCA** | | | | |

| | | | | |
|---|---|---|---|---|
| 1 | 500 | 300 | 400 | 1200 |
| 2 | 400 | 400 | 500 | 1300 |

Tabla 13, Nivel de comprensión en la documentación Estudio exploratorio

| **GSA** | **DC** | | | **DR** | | **C** |
|---|---|---|---|---|---|---|
| | **f** | **d** | **a** | **d** | **A** | |
| 1 | 100 | 100 | 50 | 0 | 0 | 50 |
| 2 | 100 | 30 | 100 | 30 | 50 | 62 |
| **GCA** | | | | | | |
| 1 | 100 | 100 | 100 | 100 | 50 | 90 |
| 2 | 100 | 100 | 100 | 50 | 50 | 80 |

Finalmente se muestran en la Tabla 14, los resultados para la eficiencia, efectividad y comprensión de la Arquitectura y su Rationale en la realización de las tareas que involucran cambios en la Arquitectura del Software.

Tabla 14, Resultados del Estudio exploratorio

| **% Eficiencia** | | |
|---|---|---|
| **Participante** | **GCA** | **GSA** |
| 1 | 83.3 | 53.7 |
| 2 | 86.1 | 103.7 |
| **% Efectividad** | | |
| 1 | 80.0 | 53.3 |
| 2 | 86.7 | 93.3 |
| **% Comprensión** | | |
| 1 | 90.0 | 50.0 |
| 2 | 80.0 | 62.0 |

## Análisis

### *Estadística descriptiva*

Cada una de las hipótesis considera una variable independiente en una escala nominal con dos niveles (con anotaciones, sin anotaciones) y una variable dependiente (efectividad, eficiencia o comprensión de la Arquitectura y su Rationale). Para confirmar las hipótesis se utiliza una prueba de t student suponiendo una distribución normal, varianzas y tamaños de muestras iguales, para dos medidas independientes. Debido a que nos interesa saber si los resultados están por encima de la distribución normal se hace uso de una prueba t student de una cola.

### *Preparación del conjunto de datos*

Para determinar el valor de las variables dependientes es necesario transformar las horas de terminación de las tareas en minutos. Posteriormente, se evalúa el nivel de correctitud para cada tarea según los criterios mostrados en las Tablas 7, 8 y 9 respectivamente. Para el cálculo del nivel de comprensión de la Arquitectura y su Rationale es necesario hacer una revisión manual de los resultados de la tarea 3 de cada participante, además, es necesario evaluar las respuestas de la encuesta que se entrega a cada participante, las cuales confirman el entendimiento de la Arquitectura y el Rationale implicado, además de dar un valor crítico al modelo de anotaciones.

### *Prueba de hipótesis*

Una vez calculadas las variables dependientes se hace el contraste de hipótesis mediante la prueba de t student con un nivel de significancia igual a 0.05.

### Eficiencia

El porcentaje de eficiencia promedio para el grupo con anotaciones de código fue del 84.7% a diferencia del grupo sin anotaciones de código el cual tiene un porcentaje promedio de eficiencia del 78,7%. Sin embargo, el p-valor es de 0.4913 el cual es mucho mayor que alfa (0.05), por lo tanto, se acepta la hipótesis nula y rechazamos la hipótesis alternativa. Esto significa que el uso de anotaciones de código con información del Rationale Arquitectónico, no mejora la eficiencia en la realización de cambios con un impacto arquitectural en este experimento.

### Efectividad

Los resultados muestran que el promedio de efectividad para el grupo con anotaciones de código es de 83,3% y el porcentaje promedio para el grupo sin anotaciones de código es del 73,3%. Sin embargo, con el p-valor de 0.4369 mayor a 0.05 se debe rechazar la hipótesis alternativa y aceptar la hipótesis nula, esto significa que el uso de las anotaciones de código con información del Rationale Arquitectónico no tiene una significancia estadística sobre la efectividad al realizar cambios con un impacto arquitectural en este experimento.

#### Comprensión

El promedio de comprensión para el grupo con las anotaciones de código es de 85%, a diferencia del grupo sin anotaciones el cual es de 56%. Cuando se calcula el p-valor da como resultado 0.0327, el cual es menor a nuestro alfa 0.05, por lo tanto, aceptamos la hipótesis alternativa y rechazamos la hipótesis nula. Esto significa que el uso de anotaciones de código tiene una significación estadística sobre la comprensión de la arquitectura y su Rationale al realizar cambios con un impacto arquitectural en este experimento.

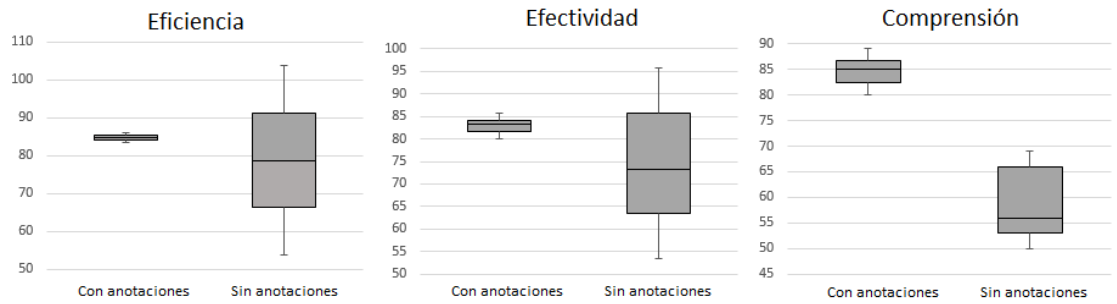


Figura 31, Resultados gráficos prueba t-student Estudio exploratorio

Como podemos observar en los resultados anteriores la documentación del Rationale Arquitectónico mediante el uso de anotaciones de código, tiene un efecto positivo en la comprensión de la arquitectura y su Rationale. Sin embargo, no garantiza eficiencia o efectividad a la hora de realizar cambios con un impacto arquitectural, lo que nos indica que la eficiencia y la efectividad al realizar cambios con un impacto arquitectural en este cuasi-experimento no dependieron de la especificación del Rationale, sino que hay otros factores tales como la habilidad para desarrollar, la experiencia, el entorno de desarrollo, etc. Los resultados demuestran que algunos participantes se desempeñan mejor a nivel funcional que otros, esto se debe a que algunos de ellos trabajan actualmente en desarrollo de software y los demás llevan tiempo sin desarrollar software, estos últimos se dedican a otras labores de ingeniería, como el análisis de requisitos y el diseño de sistemas. Los participantes que leyeron la información de las anotaciones expresan de mejor medida las razones por las cuales se construye el sistema de dicha forma y las razones de fondo de las nuevas modificaciones. El cuasi experimento anterior además de probar la utilidad de las anotaciones de código en la comprensión de la arquitectura y las decisiones que hay detrás de ella, permitió capturar comentarios y consejos que aportan en la definición y mejora del modelo de anotaciones, a través de una encuesta en la que los participantes expresan libremente sus impresiones del Rationale Arquitectónico, por ejemplo, varios de ellos sostuvieron que harían uso del modelo de anotaciones en la industria, ya que esto facilita la transmisión del conocimiento a través del tiempo.

# Experimento controlado

El estudio exploratorio anterior permitió dar una idea general sobre la utilidad de la herramienta basada en anotaciones de código al documentar razones arquitecturales cuando se produce la necesidad de una evolución de la arquitectura. Este estudio se basa en la versión final del Rationale Arquitectónico que se define en el capítulo 3, la primera versión permitió tener un acercamiento a las necesidades de los arquitectos y desarrolladores para documentar las razones que están detrás de las decisiones relacionadas con la arquitectura, así como también permitió obtener opiniones favorables sobre la herramienta y la investigación en curso. Sin embargo, el estudio exploratorio permitió observar que los arquitectos necesitan una estructura definida para documentar el Rationale Arquitectónico de manera correcta, puesto que algunos de los participantes no documentaron las razones, en cambio escribieron las decisiones que tomaron respecto a los cambios solicitados. Es por esto que se decide realizar otro estudio formal sobre la utilidad de la herramienta, en esta ocasión con la última versión del modelo de Rationale Arquitectónico propuesto en este trabajo, una muestra significativa de ingenieros y otro problema de dominio para

variar la complejidad, el planteamiento del problema, las decisiones de diseño y el Rationale Arquitectónico involucrado.

La planeación del experimento y el diseño es similar al estudio anterior puesto que este brinda las guías para replicar las actividades de la investigación.

## Planeación del experimento

Este experimento tiene el mismo objetivo que el estudio exploratorio anterior, pretende determinar el valor de las anotaciones de código fuente como una alternativa para documentar el Rationale Arquitectónico. A diferencia del estudio anterior en este experimento se cuenta con la última versión del modelo de Rationale Arquitectónico definido en este trabajo. La planeación de este experimento es una réplica del estudio exploratorio, razón por la cual varios de los ítems van a tener cierta similitud.

### *Hipótesis*

La hipótesis de este experimento es que la existencia de anotaciones de código fuente con información del Rationale Arquitectónico mejora la mantenibilidad de la arquitectura con respecto a la eficiencia, efectividad y comprensión al realizar cambios arquitecturales. Para confirmar esta hipótesis se formula la hipótesis nula y alternativa para cada aspecto de interés:

**H1:** La efectividad al realizar un cambio arquitectural en un sistema con la documentación del Rationale Arquitectónico en anotaciones de código:

- **Nula:** es menor o igual a la efectividad de realizar el cambio sin anotaciones de código.
- **Alternativa:** es mayor a la efectividad de realizar el cambio sin anotaciones de código.

**H2:** La eficiencia al realizar un cambio arquitectural en un sistema con la documentación del Rationale Arquitectónico en anotaciones de código:

- **Nula:** es menor o igual a la eficiencia de realizar el cambio sin anotaciones de código.
- **Alternativa:** es mayor a la eficiencia de realizar el cambio sin anotaciones de código.

**H3:** La comprensión de la Arquitectura y su Rationale al realizar un cambio arquitectural en un sistema con la documentación del Rationale Arquitectónico en anotaciones de código:

- **Nula:** es menor o igual a la comprensión al realizar el cambio sin anotaciones de código.
- **Alternativa:** es mayor a la comprensión al realizar el cambio sin anotaciones de código.

### *Variables*

#### Variables independientes

En este trabajo la única variable independiente es la existencia o no de anotaciones de código fuente con información del Rationale Arquitectónico, en este experimento se utiliza el modelo de la última versión de Rationale Arquitectónico definido en el capítulo 3. Sus únicos valores posibles son “Si” o “No”, esta variable es categórica puesto que representa si un participante debe realizar los cambios a un código con anotaciones de código o sin ellas.

## Variables dependientes

**Eficiencia**

Valor medido respecto a la correctitud y el tiempo total obtenidos en la realización de las tareas, la eficiencia **'E'** se define a continuación como:

$$\boldsymbol{E = \frac{NCT}{T}}$$

Para dar un valor de la eficiencia en términos de porcentaje es necesario establecer un punto de referencia en la realización de las tareas, el cual se define como:

$$\boldsymbol{ER = \frac{NCTr}{Te}}$$

Donde el nivel de correctitud total referente **'NCTr'** es:

$$\boldsymbol{NC1 = 400}, \qquad \boldsymbol{NC2 = 500}, \qquad \boldsymbol{NC3 = 500}$$

$$\boldsymbol{NCTr = 400 + 500 + 500 = 1400}$$

Con lo cual conseguimos el siguiente valor de eficiencia referente 'ER':

$$\boldsymbol{ER = \frac{1400}{140} = 10\ \frac{NCT}{min}}$$

Finalmente obtenemos el porcentaje de eficiencia por cada participante:

$$\%\,\boldsymbol{eficiencia = \frac{E}{ER} * 100}$$

**Efectividad**

Valor medido mediante la correctitud en la realización de todas las tareas dividido por la correctitud total esperada. En este trabajo se define la efectividad 'EF' como:

$$\boldsymbol{EF = \frac{NCT}{NCTe}}$$

Finalmente obtenemos el porcentaje de efectividad de cada participante mediante:

$$\%\,\boldsymbol{efectividad = EF * 100}$$

**Comprensión**

Es necesario refinar la variable que mide el nivel de comprensión de los participantes al realizar cada una de las tareas. Para ello se establecen nuevos criterios que permiten definir el nivel de entendimiento del Rationale Arquitectónico cuando los participantes realizan las tareas. Este nivel de comprensión se evalúa mediante un cuestionario mostrado en el anexo N° 2: *Reporte experimento controlado/Materiales y recursos*. El formulario contiene las siguientes preguntas:

1. Indique las razones que justifiquen la organización del código en los paquetes especificados.

2. ¿Cuál es(son) el(los) atributo(s) de calidad que se busca conseguir con la arquitectura originalmente planteada (antes de considerar los cambios solicitados)? Justifique su respuesta.
3. Explique los componentes que creó.
4. ¿Dónde posicionó los nuevos componentes y explique porque los puso ahí?
5. ¿Cuál es(son) el(los) atributo(s) de calidad que se quiere lograr después de realizar los cambios? Justifique su respuesta.

Cada pregunta está dirigida a especificar las razones de las decisiones que se tomaron en el proceso de realización de las actividades. Cada pregunta tiene una calificación de 100 puntos para un total de 500 puntos, este valor se divide entre el número total de preguntas, dando como resultado máximo un nivel de comprensión **C = 100**. La puntuación de las preguntas se verifica de acuerdo a la definición de la respuesta de acuerdo con la correspondencia en la definición del Rationale brindada por la ISO/IEC/IEEE 42010.

Con lo cual el nivel de comprensión del Rationale Arquitectónico en la realización de las tareas se define como:

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

Donde n es igual al número de preguntas y P es el total de puntos de cada pregunta.

### *Participantes*

El grupo poblacional consta de 21 ingenieros de sistemas y electrónica, en este grupo se encuentran profesionales actualmente trabajando en la industria de desarrollo de software y tesistas de grado de la Universidad del Cauca. La mayoría de personas son hombres de aproximadamente 22 a 35 años de edad, con experiencia de por lo menos 2 años en desarrollo de software en el lenguaje de programación Java. Todas las personas habitan en la ciudad de Popayán en áreas cercanas a sitio del experimento. Se realiza un muestreo aleatorio asignando a cada participante un número consecutivo, la selección de cada participante se realiza generando un número del 1 al 21 y se contacta directamente a la persona por algún medio disponible (Redes sociales, Correo electrónico, Llamada telefónico o mensaje de Whatsapp), si esta persona por cualquier motivo no puede asistir al evento se contacta directamente a la persona que corresponda con el siguiente número consecutivo, hasta completar la muestra de 8 participantes.

Finalmente, al día del experimento llegan 7 personas de las cuales 1 de ellas llega tarde y se le presentan inconvenientes con la instalación y el despliegue de las herramientas necesarias para la realización de este experimento, por lo cual se decide extraer de la muestra poblacional y trabajar con una muestra de 6 participantes.

### *Material experimental*

Para este experimento se realiza un sistema para la gestión de los productos en pequeñas organizaciones que permite la gestión de productos, ingredientes, tiendas, pedidos, usuarios, clientes y pagos. Las tecnologías utilizadas para este sistema son Java EE, Hibernate, Oracle 11g, EJB y Primefaces. Este sistema permite la gestión de los productos para pequeñas empresas dedicadas a la venta de productos alimenticios. Además de gestionar la información relacionada con los

productos y sus ingredientes, este sistema permite configurar tiendas, categorías de productos, los usuarios que hacen uso del sistema junto con sus respectivos roles, la información de los clientes y los pedidos que estos realizan. El código fuente junto con su documentación se puede observar en el anexo N° 2: *Reporte experimento controlado/Materiales y recursos.* Este código fuente se entrega con la documentación del sistema en un documento .docx en la carpeta reports y con un pdf con las actividades que deben realizar; para los participantes que tienen las anotaciones de código incorporadas con información del Rationale arquitectónico cuentan con los reportes generados por cada anotación marcada directamente en el código.

### *Tareas*

Las actividades que debe realizar cada participante se describen a continuación. De igual forma estas están descritas en el anexo N° 2: *Reporte experimento controlado/Planificación.*

1. Se debe crear un módulo de software que permita gestionar los pagos relacionados a un pedido. Este módulo se va a construir de manera iterativa e incremental. Sin embargo, se puede definir la estructura inicial y la organización de los componentes, de tal manera que se puedan implementar las funcionalidades más prioritarias. La primera implementación que se debe realizar consta de verificar si un pedido se puede pagar, realizando una validación con el saldo del método de pago. Esta implementación debe seguir con los siguientes parámetros de entrada y salida:
    a. Entrada: Número que identifica el método de pago idMetodoPago (Long), tenga en cuenta que MetodoPago no se define todavía en base de datos, pero se puede trabajar con objetos quemados en memoria.
    b. Salida: Long que representa el saldo de la cuenta.
2. Suponga que ha pasado tiempo y que se empieza a ver afectado el rendimiento del sistema debido a la gran cantidad de peticiones que realizan los usuarios sobre el sistema. Con un software de pruebas de rendimiento y un monitor de control de peticiones, se logra determinar que el componente más afectado es el componente de pagos que usted creó. Además de lo anteriormente mencionado, este componente utiliza servicios externos para consultar datos de los medios de pagos, es por eso que es necesario extraerlo del nivel de la aplicación principal y pasarlo a un entorno independiente, donde tenga recursos propios y pueda responder de manera más eficiente a la cantidad de peticiones de los usuarios. Se quiere que este módulo sea totalmente independiente del resto de la aplicación, fácil de desplegar, de mantener y que permita el incremento de peticiones de los usuarios de manera controlada.
3. Documente los cambios realizados en las tareas anteriores según los artefactos que disponga para la documentación de la arquitectura. Si el tiempo no le alcanza para terminar con las tareas, por favor indique los cambios que planearía implementar lo más detallado posible (diferenciándolos con la palabra clave TODO).
4. Finalmente responda las preguntas del formulario, que será entregado una vez indique que finalizó las modificaciones o el tiempo asignado para ellas haya terminado.

### *Diseño del experimento*

Al igual que en el estudio exploratorio los participantes se dividen en dos grupos, uno con anotaciones de código con información del Rationale Arquitectónico y otro sin estas anotaciones, a diferencia de que, en este, se utiliza la versión final de la herramienta. La agrupación de los

participantes se realiza teniendo en cuenta la experticia de los participantes en las tecnologías involucradas en el desarrollo del sistema, con el objetivo de formar grupos similares respecto a las habilidades de programación de cada participante.

### *Procedimiento*

Los participantes deben recibir una capacitación sobre los conceptos fundamentales relacionados con el Rationale Arquitectónico y las anotaciones de código, de igual forma que en el primer experimento. Después de esto, a los participantes se les entrega el código fuente de acuerdo con el grupo al que se encuentre asignado, se les da una introducción de los elementos del sistema y la organización actual del mismo. Una vez terminada la introducción se hace entrega de las actividades que deben realizar y empieza a contar el tiempo para la realización de las tareas. A medida que los participantes terminan se retiran los artefactos del experimento y se capturan los tiempos de cada actividad por participante. Finalmente se hace entrega de los formularios con las preguntas especificadas en el anexo N° 2: *Reporte experimento controlado/Materiales y recursos.*

### *Procedimiento de análisis*

Al igual que en el estudio exploratorio se realiza una prueba de t student para determinar si el uso de las anotaciones de código fuente con información del Rationale Arquitectónico tiene una influencia significativa sobre la eficiencia, la efectividad o la comprensión de la arquitectura y el Rationale involucrado luego de realizar cambios con un impacto arquitectural que conlleven a una evolución de la arquitectura debido a una necesidad de calidad.

## Ejecución del experimento

El procedimiento establecido en el ítem anterior se lleva a cabo sin ningún contratiempo, los participantes reciben la inducción de los conceptos relevantes, se les entrega el código fuente respectivo, se capturan los tiempos a medida que terminan las actividades y se les entrega el formulario con las preguntas relacionadas con la actividad y el Rationale Arquitectónico generado de esta. Los resultados de las preguntas definidas en el formulario de actividades del experimento se encuentran al final del anexo N°2: *Reporte experimento controlado/Materiales y recursos.*

Los resultados de acuerdo al tiempo y al nivel de correctitud en la realización de las tareas se muestra en las tablas 15 y 16 respectivamente.

Tabla 15, Tiempos en la realización de las tareas Experimento controlado

| **Grupo Sin Anotaciones GSA** | | | | |
|---|---|---|---|---|
| **Participante** | **t1** | **t2** | **t3** | **T(min)** |
| Javier A. | 60 | 60 | 20 | 140 |
| Eduar T. | 60 | 60 | 20 | 140 |
| Santiago G. | 60 | 60 | 20 | 140 |
| **Grupo Con Anotaciones GCA** | | | | |

| | | | | |
|---|---|---|---|---|
| Edwin M. | 60 | 60 | 20 | 140 |
| Santiago P. | 60 | 60 | 20 | 140 |
| Carlos M. | 60 | 60 | 20 | 140 |

Como se puede observar, el tiempo es igual para todos los participantes debido a que no fue suficiente para terminar completamente las actividades y se debió cortar en el límite para continuar con las demás actividades hasta finalizar con el tiempo establecido para el experimento. A diferencia del estudio exploratorio anterior, los participantes contaban con el tiempo justo para realizar este experimento y no se podía alargar más de lo pactado, ya que requerían realizar otras actividades personales en el transcurso del día.

Tabla 16, Nivel de correctitud de las tareas Experimento controlado

| **Grupo Sin Anotaciones GSA** | | | | |
|---|---|---|---|---|
| **Participante** | **NC1** | **NC2** | **NC3** | **NCT** |
| Javier A. | 400 | 300 | 300 | 1000 |
| Eduar T. | 400 | 400 | 400 | 1200 |
| Santiago G. | 300 | 200 | 300 | 800 |
| **Grupo Con Anotaciones GCA** | | | | |
| Edwin M. | 450 | 200 | 300 | 950 |
| Santiago P. | 200 | 400 | 300 | 900 |
| Carlos M. | 350 | 300 | 400 | 1050 |

El nivel de comprensión de la arquitectura y el Rationale detrás de las decisiones de diseño se mide a través de las respuestas del formulario presentado durante la actividad, los resultados de esta evaluación se muestran en la Tabla 17.

Tabla 17, Nivel de Comprensión Experimento controlado

| **Grupo Sin Anotaciones GSA** | | | | | | |
|---|---|---|---|---|---|---|
| **Participante** | **P1** | **P2** | **P3** | **P4** | **P5** | **C** |
| Javier A. | 60 | 50 | 100 | 70 | 50 | 330 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Eduar T. | 30 | 40 | 70 | 90 | 50 | 280 |
| Santiago G. | 70 | 80 | 70 | 70 | 40 | 330 |
| **Grupo Con Anotaciones GCA** | | | | | | |
| Edwin M. | 100 | 70 | 100 | 90 | 60 | 420 |
| Santiago P. | 100 | 100 | 100 | 85 | 90 | 475 |
| Carlos M. | 100 | 80 | 100 | 100 | 100 | 480 |

Los resultados finales de eficiencia, efectividad y comprensión en la realización de las actividades de este experimento controlado se muestran en la Tabla 18.

Tabla 18, Resultados del Experimento controlado

| **% Eficiencia** | | **% Efectividad** | | **% Comprensión** | |
|---|---|---|---|---|---|
| **GCA** | **GSA** | **GCA** | **GSA** | **GCA** | **GSA** |
| 67.86 | 71.43 | 63.33 | 66.67 | 84 | 66 |
| 64.29 | 85.71 | 60.00 | 80.00 | 95 | 56 |
| 75.00 | 57.14 | 70.00 | 53.33 | 96 | 66 |

## Análisis

El proceso de análisis se realiza de manera similar al estudio exploratorio utilizando estadística descriptiva, haciendo una preparación del conjunto de datos y realizando una prueba de hipótesis por cada variable dependiente definida en este experimento controlado.

### *Prueba de hipótesis*

El valor p de significancia utilizado en esta prueba es de 0.05 los resultados de cada variable tienen que ser menores a este valor para aceptar la hipótesis alternativa, en caso contrario se acepta la hipótesis nula.

**Eficiencia**

Cuando se realiza la prueba estadística para la eficiencia con un p-valor de 0.05 nos da como resultado 0.400539 el cual es un valor mucho más alto que 0.05, por lo tanto, se acepta la hipótesis nula y se rechaza la hipótesis alternativa.

**Efectividad**

Cuando se realiza la prueba estadística para la efectividad con un p-valor de 0.05 nos da como resultado 0.400337 el cual es un valor mucho más alto que 0.05, por to tanto se acepta la hipótesis nula y se rechaza la hipótesis alternativa.

**Comprensión**

En el momento de realizar la prueba estadística para la comprensión con un p-valor igual al de efectividad y eficiencia nos da como resultado 0.002342 el cual es mucho menor que 0.05, por lo cual se acepta la hipótesis alternativa y se rechaza la hipótesis nula.

Para esta prueba los resultados fueron muy interesantes, primero que todo para las variables eficiencia y efectividad no se encontró alguna significancia estadística respecto al uso o no uso de anotaciones de código fuente con información del Rationale arquitectónico, en la Figura 32 se puede observar que incluso para estas dos variables el resultado promedio fue mejor para el grupo con el código sin las anotaciones. Como en el estudio exploratorio anterior esto se debe a diferentes factores como la experticia en determinadas tecnologías, entornos de desarrollo, habilidad para desarrollar, etc. Por lo cual podemos confirmar con seguridad que la eficiencia y la efectividad no mejoran cuando se cuenta con la información del Rationale arquitectónico desde el código fuente. Sin embargo, al igual que en el estudio exploratorio la comprensión de la arquitectura y las razones que anteceden las decisiones de diseño si mejoran cuando se cuenta con la información del Rationale arquitectónico desde el código fuente.

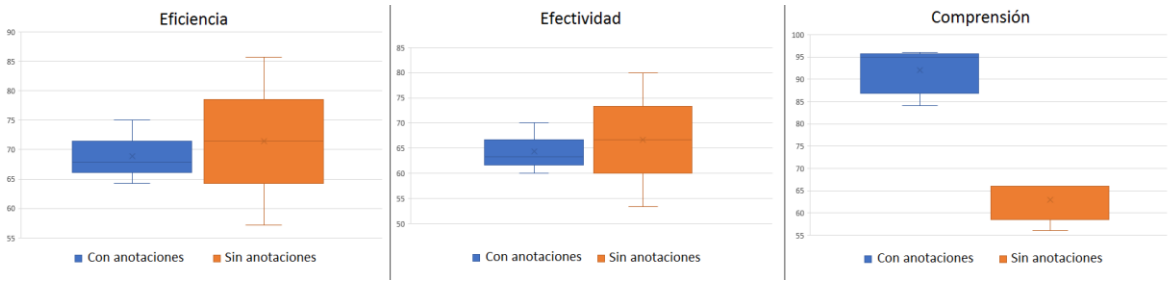


Figura 32, Resultados gráficos prueba t-student Experimento controlado

# Capítulo 5: Conclusiones, limitaciones y trabajos futuros

## Conclusiones

A partir de este estudio se pueden generar algunas conclusiones relevantes. El diseño y la ejecución de experimentos debe ser cuidadosamente realizadas, puesto que se busca evitar que existan variables que afecten su desarrollo y que les resten validez a los resultados. El cuasi-experimento trató de comprobar si las anotaciones de código con información del Rationale Arquitectónico tenían un impacto positivo sobre la eficiencia, la efectividad y la comprensión de la arquitectura y su Rationale en la realización de cambios arquitecturales. Para ello, a raíz de la variable dependiente, se debió desarrollar una solución inicial para la documentación del código fuente. El diseño del experimento tuvo una segunda parte ardua, que consistió en definir, modelar e implementar un sistema tipo que permitiera construir un historial de desarrollo arquitectónico. En el caso del cuasi-experimento, el sistema de prueba constó de la implementación de varios tipos de juegos y el cual los participantes deben modificar con el ánimo de documentar las razones del porqué está construido de esa forma y cuáles son las necesidades que obligan a cambiar su estructura. La definición de su variable y evaluar su significancia, fue otro reto a enfrentar, en particular en ingeniería de software donde conseguir un número de muestras significativo es bastante difícil. En este caso, la dificultad en conseguir desarrolladores expertos con alguna experiencia en el diseño o evaluación arquitectónica. Con los cambios logrados por los participantes y las observaciones sobre esta actividad, podemos observar que las tendencias de medida central como el promedio y la mediana favorecen el modelo de anotaciones con respecto a la eficiencia, la efectividad y la comprensión de la Arquitectura y su Rationale. Sin embargo, son las pruebas de confirmación de hipótesis las que permitieron definir cuales resultados serían válidos, dándonos como resultado en el cuasi-experimento que el uso de las anotaciones de código con información del Rationale Arquitectónico beneficia la comprensión de la Arquitectura y su Rationale y no afecta la eficiencia y efectividad para lograr los cambios. Esto quiere decir que la eficiencia y la efectividad al realizar cambios con un impacto arquitectural no dependieron en el experimento de haber especificado el Rationale a través del código fuente, sino que hay otros factores tales como la habilidad para desarrollar, la experiencia, el entorno de desarrollo y otros factores no controlados en el cuasi-experimento que podrían estar afectando estas variables dependientes. Para tener una información sobre la percepción de los participantes, una encuesta permitió obtener la opinión de los participantes respecto a la documentación del Rationale y su importancia, a lo cual los participantes respondieron a nivel general, que facilita las labores de mantenibilidad y la comprensión de la arquitectura de un sistema para las personas que llegan después de que se realiza e implementa el diseño.

El experimento pretendía comprobar si 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. Para ello, se desarrolla un sistema que permite la gestión de productos para pequeñas organizaciones de comercio y el cual los participantes deben modificar con el ánimo de documentar las razones que explican el porqué está construido de esa forma y cuáles son las necesidades que obligan a cambiar su estructura. Después de que los participantes realizan los cambios podemos observar que las tendencias de medida central como el promedio y la mediana ya no favorecen el modelo de anotaciones con respecto a la eficiencia y la efectividad. Sin embargo, la comprensión de la Arquitectura y su

Rationale sigue teniendo valores superiores para el grupo con la versión final del modelo de Rationale Arquitectónico definido en este trabajo. Sin embargo, cuando se realizan las pruebas de confirmación de hipótesis nos da como resultado que el uso de las anotaciones de código con información del Rationale Arquitectónico solo beneficia la comprensión de la Arquitectura y su Rationale. Esto quiere decir que claramente la eficiencia y la efectividad al realizar cambios con un impacto arquitectural no dependen de la especificación del Rationale, sino que hay otros factores tales como la habilidad para desarrollar, la experiencia, el entorno de desarrollo y otros factores.

En la realización tanto del estudio exploratorio como del experimento controlado se puede establecer la siguiente tesis:

Los dos estudios empíricos han permitido observar que la especificación del Rationale a través del código fuente versus solamente una especificación del Rationale en el documento de la arquitectura mejora la comprensión de las decisiones arquitecturales sin afectar la eficiencia y efectividad de las tareas de mantenimiento.

Además, se pudieron capturar varias observaciones, las cuales se muestran a continuación:

- Las personas necesitan apoyo constante del director del experimento para el despliegue y la ejecución de los sistemas de prueba. Esto requiere de tiempo adicional y debe ser considerado en la planificación del experimento.
- Los participantes no leen detenidamente la documentación del material experimental, algunos de ellos sustentan que el documento de arquitectura es extenso y no cuentan con el tiempo necesario para entenderlo.
- Algunos de los participantes se enfocan en la resolución de las actividades, más que en el entendimiento general del sistema.
- Cuando los participantes se conocen entre sí aumenta la distracción y en algunas ocasiones se comparten el conocimiento.
- Para responder las preguntas del formulario el grupo con anotaciones de código fuente utilizan directamente los reportes generados por la herramienta, en vez de utilizar el documento de especificación de arquitectura.
- Es importante tener a los participantes motivados con el experimento, esto permite que estén mucho más activos en el desarrollo de las tareas y tengan interés pleno en la realización del experimento.
- Es importante tener a los participantes motivados durante los estudios empíricos, esto permite que estén mucho más activos en el desarrollo de las tareas y tengan interés pleno en la realización del experimento. La capacitación sobre el Rationale Arquitectónico se debe abordar con más tiempo, puesto que es un aspecto que no se suele enseñar en la academia ni en la industria, pero que, según los participantes se debería abordar para evitar problemas a futuro en el mantenimiento de un sistema software. El uso de las anotaciones de código también es un tema que requiere de tiempo de capacitación y ejemplificación, esta tecnología se utiliza con frecuencia en muchos casos de uso conocidos, sin embargo, la declaración y el uso de anotaciones de código personalizadas es poco conocido y requiere de un esfuerzo adicional para generar valor como herramienta complementaria.

## Limitaciones

Una de las principales limitaciones relacionada con la implementación de la herramienta es el tiempo disponible (6 meses aproximadamente) para comprobar la utilidad de este plugin. Debido a que las condiciones para que suceda una definición de arquitectura o una evolución arquitectural, toman mucho tiempo y frecuentemente no se suele identificar estas actividades con claridad en las organizaciones. Por esto, se debió simular estas condiciones de cambio a través de experimentos controlados que permitieran validar la utilidad de esta herramienta.

Otra limitación se originó con la participación de las personas en el experimento, debido a que las actividades del experimento tomaban más de dos horas y el tiempo con el que cuentan las personas es muy poco, debido a diferentes factores a nivel laboral o personal, por lo cual las muestras para ambos experimentos fueron pequeñas, sin embargo, estas personas nos brindaron no solo resultados cuantitativos, si no también resultados cualitativos como opiniones y críticas que nos permitieron enriquecer la herramienta propuesta en este trabajo.

La naturaleza controlada de los experimentos también se convirtió en una limitación puesto que el material experimental como lo son los códigos fuentes, los documentos de arquitectura, los formularios de preguntas y todos los artefactos para desarrollarlos fueron diseñados e implementados por nosotros mismos. Aunque se tratan de simular aspectos reales del desarrollo de software hay situaciones que son complejas de simular en un periodo de dos horas aproximadamente.

## Trabajos futuros

La investigación sobre la documentación del Rationale Arquitectónico no es un tema reciente, sin embargo, actualmente se están concentrando esfuerzos en buscar alternativas de hacerlo sin interferir en las actividades normales que se llevan a cabo en el flujo normal de desarrollo de software. Como primer trabajo futuro se pretende utilizar todas las características de las anotaciones de código en Java, no solo en tiempo de ejecución, sino también en tiempo de compilación y codificación, con el objetivo de mostrar errores o advertencias mientras los desarrolladores realizan su código fuente y afectan un componente marcado con una anotación de Rationale Arquitectónico definido en este trabajo.

Otro trabajo futuro es validar la utilidad de esta herramienta mediante el desarrollo de un estudio de caso en diferentes organizaciones de desarrollo de software, con las cuales se puedan presentar las condiciones para que la herramienta se pueda utilizar sin ningún contratiempo y con la mayor naturalidad posible.

Finalmente, un trabajo futuro a largo plazo es establecer un meta modelo que permita diseñar componentes de software y anotarlos mediante un modelador de software como lo es StarUML y que, a través de éste, se pueda generar las anotaciones de código sin necesidad de escribirlas directamente en el código y sin necesidad de especificar algún lenguaje de desarrollo en especial.

# Bibliografía


[1] T. Dingsoyr and H. van Vliet, “Introduction to Software Architecture and Knowledge Management,” in *Software Architecture Knowledge Management: Theory and Practice*, M. Ali Babar, T. Dingsøyr, P. Lago, and H. van Vliet, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–17.

[2] P. Kruchten, H. Obbink, and J. Stafford, “The Past, Present, and Future for Software Architecture,” *IEEE Softw.*, vol. 23, no. 2, pp. 22–30, 2006.

[3] A. Jansen and J. Bosch, “Software Architecture as a Set of Architectural Design Decisions,” *5th Work. IEEE/IFIP Conf. Softw. Archit.*, pp. 109–120, 2005.

[4] J. D. Reese and N. G. Leveson, “Software deviation analysis,” *Proc. 19th Int. Conf. Softw. Eng.*, pp. 250–260, 1997.

[5] D. E. Perry and A. L. Wolf, “Foundations for the study of software architecture,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 17, no. 4, pp. 40–52, 1992.

[6] J. Tyree and A. Akerman, “Architecture decisions: Demystifying architecture,” *IEEE Softw.*, vol. 22, no. 2, pp. 19–27, 2005.

[7] N. Eftekhari, M. P. Rad, and H. Alinejad-Rokny, “Evaluation and classifying software architecture styles due to quality attributes,” *Aust. J. Basic Appl. Sci.*, vol. 5, no. 11, pp. 1251–1256, 2011.

[8] A. E. Sabry, “Decision Model for Software Architectural Tactics Selection Based on Quality Attributes Requirements,” *Procedia Comput. Sci.*, vol. 65, no. Icc, pp. 422–431, 2015.

[9] A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, “Rationale Management in Software Engineering: Concepts and Techniques,” in *Rationale Management in Software Engineering*, SPI Publisher Services, Pondicherry, 2006.

[10] F. P. Brooks, B. Evans, and A. C. Hill, *The.Mythical.Man.Month.F.Brooks*. .

[11] G. Fischer, A. C. Lemke, R. McCall, and A. I. Morch, “Making Argumentation Serve Design,” *Humani¿½Computer Interact.*, vol. 6, no. 3–4, pp. 393–419, 1991.

[12] W. Ding, P. Liang, A. Tang, and H. Van Vliet, “Knowledge-based approaches in software documentation: A systematic literature review,” *Information and Software Technology*, vol. 56, no. 6. pp. 545–567, 2014.

[13] N. Juristo, “The Role of Scientific Method in Software Development.” p. 58, 2011.

[14] B. Kitchenham and S. Charters, “Guidelines for performing Systematic Literature Reviews in Software Engineering,” *Engineering*, vol. 2, p. 1051, 2007.

[15] A. Wach-Kakolewicz, “Constructivist Approach in Teaching in Higher Education.” 2016.

[16] I. ICONIX Software Engineering, “ICONIX:: Better Agile Methodology and Project Management,” 2016. [Online]. Available: http://www.iconixsw.com/index.shtml.

[17] A. Jedlitschka, M. Ciolkowski, and D. Pfahl, “This is a preliminary version of a chapter in Reporting Experiments in Software Engineering,” 2007.

[18] K. Wiegers and J. Beatty, *Software Requirements*, Third. Redmond, Washington: Microsoft Press, 2013.

[19] I. Sommerville, *Software Engineering*, 9th ed. Boston, Massachusetts: Addison Wesley, 2010.

[20] M. Barbacci and Others, “Quality Attributes,” *IEEE Software,* vol. 18, no. CMU/SEI-95-TR-021, 1995.

[21] IEEE, “IEEE Standard for a Software Quality Metrics Methodology,” *IEEE Std 1061-1998*, vol. 1998, p. i, 1998.

[22] “ISO/IEC 9126 Information technology-Software product evaluation-Quality characteristics and guidelines for their use,” 1992.

[23] L. Dobrica and E. Niemela, “A survey on software architecture analysis methods,” *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 638–653, 2002.
[24] ISO/IEC, “ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models,” ISO/IEC, 2010.
[25] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Second. Addison Wesley, 2003.
[26] IEEE and ISO/IEC, “Systems and software engineering - Recommended practice for architectural description of software-intensive systems,” *ISO/IEC 42010 IEEE Std 1471-2000 First Ed. 2007-07-15*, pp. c1--24, 2007.
[27] H. van Vliet and A. Tang, “Decision making in software architecture,” *J. Syst. Softw.*, vol. 117, pp. 638–644, 2016.
[28] P. Kruntchen, “Architectural blueprints–the” 4+ 1” view model of software architecture,” *IEEE Softw.*, vol. 12, no. November, pp. 42–50, 1995.
[29] Object Management Group, “Unified Modeling Language UML.” [Online]. Available: http://www.uml.org/.
[30] P. Abrahamsson, M. A. Babar, P. Kruchten, uhammad A. Babar, and P. Kruchten, “Agility and Architecture: Can They Coexist?,” *IEEE Softw.*, vol. 27, no. 2, pp. 16–22, 2010.
[31] N. B. Harrison and P. Avgeriou, “How do architecture patterns and tactics interact? A model and annotation,” *J. Syst. Softw.*, vol. 83, no. 10, pp. 1735–1758, 2010.
[32] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software architecture*, 1st ed. Germany, 1996.
[33] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*, John Wiley., vol. 2. West Sussex PO19 1UD, England: John Wiley & Sons, Ltd, 2000.
[34] R. Kazman and M. Webb, “SAAM : A Method for Analyzing the Properties of Software Architectures,” 1994.
[35] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, “The architecture tradeoff analysis method,” *Proceedings. Fourth IEEE Int. Conf. Eng. Complex Comput. Syst. (Cat. No.98EX193)*, pp. 68–78.
[36] M. R. Barbacci, R. Ellison, A. J. Lattanze, J. a. Stafford, C. B. Weinstock, and W. G. Wood, “Quality Attribute Workshops, Third Edition,” *Quality*, no. August, p. 38, 2003.
[37] M. C. University, “Software Engineering Institute.” [Online]. Available: http://www.sei.cmu.edu/architecture/tools/index.cfm.
[38] K. A. De Graaf, P. Liang, A. Tang, and H. Van Vliet, “How organisation of architecture documentation affects architectural knowledge retrieval,” *Sci. Comput. Program.*, vol. 121, pp. 75–99, 2016.
[39] M. L. Roldan, S. Gonnet, and H. Leone, “Operation-based approach for documenting software architecture knowledge,” *Expert Syst.*, vol. 33, no. 4, pp. 313–348, 2016.
[40] J. E. Burge, J. M. Carroll, R. McCall, and I. Mistrík, *Rationale-Based Software Engineering*, Springer. Springer-Verlag Berlin Heidelberg, 2008.
[41] A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, “Rationale Management in Software Engineering: Concepts and Techniques,” in *Rationale Management in Software Engineering*, A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–48.
[42] H. Obbink *et al.*, “Report on Software Architecture Review and Assessment (SARA).” p. 58, 2018.
[43] T. R. Gruber and D. Russell, “Design Knowledge and Design Rationale: A Framework for

Representation, Capture, and Use," 1994.
[44] S. Z. Guyer and C. Lin, "An annotation language for optimizing software libraries," *ACM SIGPLAN Not.*, vol. 35, no. 1, pp. 39–52, 1999.
[45] M. Das *et al.*, "Source Code Annotation Language," vol. 2, no. 12, 2009.
[46] C. Noguera and R. Pawlak, "AVal: An extensible attribute-oriented programming validator for Java," *J. Softw. Maint. Evol.*, vol. 19, no. 4, pp. 253–275, 2007.
[47] M. Nosáľ, M. Sulír, and J. Juhár, "Source Code Annotations as Formal Languages," *Comput. Sci. Inf. Syst. (FedCSIS), 2015 Fed. Conf.*, vol. 5, no. 1, pp. 953–964, 2015.
[48] M. Vatkina, "JSR 345," *JSR 345: Enterprise JavaBeansTM 3.2*, 2013. [Online]. Available: https://jcp.org/en/jsr/detail?id=345.
[49] X. Team, "Xdoclet," *Attribute oriented programming*, 2003. [Online]. Available: http://xdoclet.sourceforge.net/xdoclet/index.html.
[50] Oracle, "EJBgen," *Programming WebLogic Enterprise JavaBeans*. [Online]. Available: https://docs.oracle.com/cd/E13222_01/wls/docs90/ejb/EJBGen_reference.html.
[51] Oracle, "Javadoc," *Java Platform, Standard Edition*. [Online]. Available: https://docs.oracle.com/javase/10/javadoc/title.htm.
[52] Pivotal, "Spring Framework," *Spring: the source for modern java*. [Online]. Available: https://spring.io/.
[53] T. Ju. Team, "JUnit," *The new major version of the programmer-friendly testing framework for Java 8 and beyond*. [Online]. Available: https://junit.org/junit5/.
[54] Redhat, "Hibernate," *MORE THAN AN ORM, DISCOVER THE HIBERNATE GALAXY.* [Online]. Available: http://hibernate.org/.
[55] L. Bratthall, E. Johansson, and B. Regnell, "Is a Design Rationale Vital when Predicting Change Impact? – A Controlled Experiment on Software," *2nd Int. Conf. Prod. Focus. Softw. Process Improv. (PROFES 2000)*, pp. 126–139, 2000.
[56] J. Horner and M. E. Atwood, "Effective Design Rationale: Understanding the Barriers," in *Rationale Management in Software Engineering*, A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 73–90.
[57] K. Schneider, "Rationale as a By-Product," in *Rationale Management in Software Engineering*, A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 91–109.
[58] S. J. B. Shum, A. M. Selvin, M. Sierhuis, J. Conklin, C. B. Haley, and B. Nuseibeh, "Hypermedia support for argumentation-based rationale," *Ration. Manag. Softw. Eng.*, no. January, pp. 111–132, 2006.
[59] M. A. Babar, I. Gorton, and B. Kitchenham, "A Framework for Supporting Architecture Knowledge and Rationale Management," in *Rationale Management in Software Engineering*, A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 237–254.
[60] L. Bass, P. Clements, R. L. Nord, and J. A. Stafford, "Capturing and Using Rationale for a Software Architecture," in *Rationale Management in Software Engineering*, A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 255–272.
[61] J. E. Burge and D. C. Brown, "Rationale-Based Support for Software Maintenance," in *Rationale Management in Software Engineering*, A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 273–296.
[62] J. S. van der Ven, A. G. J. Jansen, J. A. G. Nijhuis, and J. Bosch, "Design Decisions: The Bridge between Rationale and Architecture," in *Rationale Management in Software Engineering*, 2006.

[63] M. Shahin, P. Liang, and Z. Li, “Do architectural design decisions improve the understanding of software architecture? two controlled experiments,” *Proc. 22nd Int. Conf. Progr. Compr. - ICPC 2014*, pp. 3–13, 2014.
[64] J. Aldrich, C. Chambers, and D. Notkin, “ArchJava : Connecting Software Architecture to Implementation,” *Proc. 24th Int. Conf. Softw. Eng. - ICSE ’02*, p. 187, 2002.
[65] T. M. Hesse, A. Kuehlwein, and T. Roehm, “DecDoc: A Tool for Documenting Design Decisions Collaboratively and Incrementally,” *Proc. - 2016 1st Int. Work. Decis. Mak. Softw. Archit. MARCH 2016*, no. June, pp. 30–37, 2016.
[66] G. Rasool, I. Philippow, and P. M??der, “Design pattern recovery based on annotations,” *Adv. Eng. Softw.*, vol. 41, no. 4, pp. 519–526, 2010.
[67] P. Kajsa and P. Návrat, “Design Pattern Support Based on the Source Code Annotations and Feature Models,” in *SOFSEM 2012: Theory and Practice of Computer Science*, 2012, pp. 467–478.
[68] M. Sulir, M. Nosal, and J. Poruban, “Recording concerns in source code using annotations,” *Comput. Lang. Syst. Struct.*, vol. 46, pp. 44–65, 2016.
[69] J. Horner and M. E. Atwood, “Effective Design Rationale: Understanding the Barriers,” in *Rationale Management in Software Engineering*, A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 73–90.
[70] K. Schneider, “Rationale as a By-Product,” in *Rationale Management in Software Engineering*, A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 91–109.
[71] M. A. Babar, I. Gorton, and B. Kitchenham, “A Framework for Supporting Architecture Knowledge and Rationale Management,” in *Rationale Management in Software Engineering*, A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 237–254.
[72] L. Bass, P. Clements, R. L. Nord, and J. A. Stafford, “Capturing and Using Rationale for a Software Architecture,” in *Rationale Management in Software Engineering*, A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 255–272.
[73] J. E. Burge and D. C. Brown, “Rationale-Based Support for Software Maintenance,” in *Rationale Management in Software Engineering*, A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 273–296.
[74] M. Csikszentmihalyi, *Flow The Psychology of Optimal Experience*. New York, NY: Harper Perennial, 1991.
[75] R. J. Bogumil, “The reflective practitioner: How professionals think in action,” *Proc. IEEE*, vol. 73, no. 4, pp. 845–846, 1985.
[76] G. Fischer, “Turning breakdowns into opportunities for creativity,” *Knowledge-Based Syst.*, vol. 7, no. 4, pp. 221–232, 1994.
[77] “Echalo a la suerte.” [Online]. Available: https://echaloasuerte.com/draw/new/groups/.