VNIVERSIDAD NACIONAL
AVTNOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

ARQUITECTURA DE SOFTWARE PARA EL ENTORNO COMPUTACIONAL DE KUALI-BEH

T E S I S
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA (COMPUTACIÓN)

P R E S E N T A:
ALBERTO IVÁN TAPIA DURÁN

DIRECTORA DE TESIS:
M. EN C. MARÍA GUADALUPE ELENA IBARGÜENGOITIA GONZÁLEZ
FACULTAD DE CIENCIAS - UNAM

MÉXICO, D. F. ABRIL 2014

DEDICATORIAS

A Dios.

Por derramar sus bendiciones sobre mí y darme fuerzas para seguir adelante y no rendirme en los problemas que se me presentaron, enseñándome a encarar las adversidades sin perder nunca la dignidad ni desfallecer en el intento.

A mi madre Regina.

Por todo tu esfuerzo y sacrificio, por brindarme todo tu amor, comprensión y apoyo incondicional, además de depositar en mí, toda tu confianza en cada momento de mi vida. Siempre encuentro en tu voz, las palabras de aliento que necesito para seguir adelante. Nunca terminaré de agradecerte que tu vida entera la haz dedicado a mí, siendo la luz que guía mi camino, y el mejor ejemplo de fortaleza y tenacidad que un hijo pueda tener. Te amo mamá.

Al amor de mi vida.

Por estar siempre a mi lado y por creer en mí. A tu paciencia y comprensión, por sacrificarlo todo para que yo pueda cumplir mis sueños y mis metas. Por tu cariño y amor, me haz inspirado a ser mejor para tí, ahora puedo decir que esta tesis lleva mucho de tí, gracias por estar siempre a mi lado, mi niña.

A mis adorados hijos, Jesús y Axel.

Por darme su amor incondicional, y por ser el motor de mi vida. Por llenarme de abrazos y besos cuando más lo necesite. Mis niños, siempre estaré para ustedes.

A mis abuelitos.

Que siempre me trataron como a uno más de sus hijos y estuvieron ahí cuando más los necesitaba, a pesar de que no están físicamente en estos momentos conmigo, sé que sus almas sí lo están, además de que siempre los llevo en mi corazón.

AGRADECIMIENTOS

Un agradecimiento especial a la ***Maestra Lupita Ibargüengoitia*** *que, como directora de esta tesis, me ha orientado, apoyado y corregido en mi labor, con un interés y una entrega que han sobrepasado, con mucho, todas las expectativas que, como alumno, deposité en su persona. Muchas Gracias !*

También le agradezco a la ***Dra. Hanna Oktaba*** *por sus consejos y apoyo en mis estudios de maestría y en este trabajo de tesis.*

A mis sinodales y maestros, por su apoyo así como por la sabiduría que me transmitieron en el desarrollo de mi formación como maestro.

A mi amigo ***Miguel Morales****, por ayudarme y apoyarme siempre. Por el tiempo compartido y por impulsar lo mejor de mi.*

A la Universidad Nacional Autónoma de México y en especial al Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, por permitirme ser parte de una generación de triunfadores y personas productivas.

Definición de Hijo

"Hijo es un ser que Dios nos prestó para hacer un curso intensivo de cómo amar a alguien más que a nosotros mismos, de cómo cambiar nuestros peores defectos para darles los mejores ejemplos y, de nosotros, aprender a tener coraje. Sí ¡Eso es! Ser padre o madre es el mayor acto de coraje que alguien pueda tener, porque es exponerse a todo tipo de dolor, principalmente de la incertidumbre de estar actuando correctamente y del miedo a perder algo tan amado.
¿Perder? ¿Cómo? ¿No es nuestro? Fue apenas un préstamo . . .
El más preciado y maravilloso préstamo ya que son nuestros sólo mientras no pueden valerse por sí mismos, luego le pertenece a la vida, al destino y a sus propias familias. Dios bendiga siempre a nuestros hijos pues a nosotros ya nos bendijo con ellos".

José Saramago

CONTENIDO

Introducción

La Arquitectura de Software se refiere a la estructura o estructuras del sistema, que integra los elementos de software, las propiedades visibles de esos elementos y las relaciones entre ellos [SEI2013]; esta estructura debe realizarse antes de que los desarrolladores inicien su labor, pues representa el diseño de alto nivel del sistema y sirve como guía en el desarrollo de software.

El arquitecto de software es la persona que diseña estas estructuras, cuidando siempre que se satisfagan los requerimientos descritos por los interesados en el proyecto, estos interesados son todas aquellas personas que tengan un interés en el desarrollo del sistema.

Existen atributos de calidad que se desprenden de los requerimientos de los usuarios, y junto con las restricciones y los requerimientos funcionales, sirven de guía al arquitecto en el diseño de las estructuras de la arquitectura.

El arquitecto hace uso de patrones, los cuales son soluciones conceptuales a problemas comunes en el diseño, aunado a los patrones el arquitecto emplea tácticas, las cuales son decisiones de diseño que impactan en la forma en que se controlan las respuestas para los diferentes atributos de calidad; las tácticas buscan controlar las respuestas del sistema a los estímulos recibidos por el usuario o agentes externos.

El Software Engineering Institute (SEI) propone cuatro métodos, a través de los cuales se realiza la captura de los requerimientos que servirán de insumos para realizar el diseño y evaluación de la arquitectura de software.

El primero de estos métodos lleva por nombre *Taller de Atributos de Calidad* (“QAW” por sus siglas en inglés), y tiene el propósito de capturar y priorizar los atributos de calidad, restricciones y los casos de uso, descritos por los interesados en el sistema (stakeholders).

El segundo método es el *Diseño Guiado por Atributo*s (“ADD” por sus siglas en inglés), este método toma como insumos las salidas del método QAW y guía al arquitecto en el diseño de la arquitectura de software, realizando un proceso iterativo e incremental.

El tercero de los métodos sirve para documentar los diagramas hechos por el arquitecto, que le servirán para plasmar sus ideas y el diseño de la arquitectura, este método lleva por nombre *Vistas y más allá* (“V&B” por sus siglas en inglés).

Finalmente, el cuarto método propuesto por el SEI, ayudará al arquitecto y a los stakeholders a realizar las pruebas necesarias al diseño, a fin de identificar que tan bien satisface a los requerimientos descritos por los stakeholders y a los atributos de calidad detectados en el sistema. Además de descubrir y avisar de los riesgos y puntos sensibles detectados en la arquitectura.

En resumen, es importante destacar que, con estos antecedentes, las ideas clave con relación con este trabajo de tesis son las siguientes:

1. La arquitectura de un sistema de software define las estructuras de sus componentes, sus relaciones o comportamiento externamente visible y los principios que gobiernan su evolución a lo largo del tiempo.
2. No es posible abordar la construcción adecuada de un sistema de software de cierto tamaño y complejidad sin un diseño cuidadoso de su arquitectura.
3. El diseño de una arquitectura de software requiere de notaciones formales, de metodologías y de herramientas que permitan automatizar el proceso de diseño y simular comportamientos.
4. La arquitectura de software debe ser evaluada, ya que las decisiones que hace patentes son precisamente las que más van a condicionar el desarrollo del sistema.
5. Los atributos de calidad sólo tienen sentido dentro del contexto definido por las especificaciones del sistema de software y no como *entes* abstractos.
6. Los atributos de calidad no son independientes, sino que interaccionan entre sí a través de las relaciones estructurales impuestas por la arquitectura. La mejora de unos frecuentemente solo puede lograrse a costa de empeorar otros. Por eso en

necesario llegar a compromisos de diseño en los que también hay que tener en cuenta factores sociales y de negocio.

7. El estudio y la clasificación de las arquitecturas de software existentes, permite identificar partes comunes, con el fin de obtener una *arquitectura de referencia* que sirva para generar una nueva arquitectura de software, adecuada a un problema en específico.

KUALI-BEH [Oktaba2012] ha sido desarrollada en respuesta al Llamado a la Acción (RFP) *Fundamentos para la creación y ejecución ágil de métodos de Ingeniería de Software* lanzado por el Object Management Group (OMG) [OMG2012]. Esta propuesta propone un *"buen camino"* para definir las prácticas y métodos que los practicantes de ingeniería de software emplean en sus proyectos. Estas prácticas y métodos para el desarrollo de software, son capturadas, adaptadas y mejoradas por las propias organizaciones en función de su experiencia y madurez. KUALI-BEH es una propuesta mexicana desarrollada con base en el conocimiento de fuentes reconocidas y la experiencia en el desarrollo de estándares y modelos de referencia de procesos de desarrollo de software.

El *Entorno Computacional de KUALI-BEH* es un proyecto llevado a cabo en el Posgrado en Ciencias e Ingeniería de la Computación de la UNAM (PCIC), que tiene como objetivo construir una herramienta computacional basada en KUALI-BEH, para apoyar a las organizaciones de desarrollo de software en:

1. Expresar sus métodos y prácticas empíricas de manera estructurada.
2. Resguardar esos métodos y prácticas en un repositorio de la organización.
3. Seleccionar y adaptar al contexto de un proyecto de desarrollo de software, algún método y sus prácticas extrayéndolo del repositorio de la organización.
4. Aplicar el método y sus prácticas durante la ejecución de un proyecto, dándole seguimiento a través de tableros de control.
5. Enriquecer y mejorar el repositorio de la organización a partir de la experiencia en los proyectos y el reconocimiento adquirido de fuentes externas.

A fin de construir este *Entorno Computacional* se dividió el trabajo en cuatro proyectos de tesis cuyos objetivos son los siguientes:

1. Arquitectura de software para el *Entorno Computacional de KUALI-BEH*. Cuyo objetivo es definir la arquitectura de software del *"Entorno Computacional de KUALI-BEH"* aplicando los métodos propuestos por el SEI.

2. Repositorio de métodos y prácticas de proyectos de software para KUALI-BEH [Barrera2014]. Su objetivo es crear un repositorio que permita administrar métodos y prácticas expresadas por practicantes de organizaciones de desarrollo de software de acuerdo a KUALI-BEH.

3. Selección y adaptación de métodos en proyectos de software aplicando KUALI-BEH [Ruíz2014]. Sus objetivos son:

- Registrar la información relevante para conformar un proyecto de software.
- Desarrollar el entorno que permita seleccionar y adaptar un método para el contexto de un proyecto de software específico.
- Finalmente realizar la recolección de información para el cierre del proyecto de software.

4. Tableros de control para el seguimiento de proyectos de software aplicando KUALI-BEH [Urrutia2014]. Los objetivos de este punto son:

- Desarrollar tableros de control para el seguimiento de un proyecto, que permita:
- Instanciar un método para un proyecto de software.
- Reflejar en los tableros de control las operaciones de adaptación sobre el método y sus prácticas.
- Reflejar en los tableros los cambios de estado de las instancias de prácticas y del método durante la ejecución del proyecto de software.

Objetivo de la Tesis

El objetivo de esta tesis es definir la arquitectura de software del *Entorno Computacional de KUALI-BEH* aplicando los métodos propuestos por el SEI enfocados en la realización del diseño de la arquitectura de software.

Estructura y Contenidos de la Tesis

La presente tesis está dividida en 6 capítulos y 3 apéndices. Los primeros 4 capítulos describen el marco teórico de este trabajo de tesis: el concepto de arquitectura de software; arquitectura y atributos de calidad, y una introducción a los métodos del SEI para definir la arquitectura de software.

El capítulo 5 describe la propuesta KUALI-BEH y los conceptos relacionados a la misma, así como también se detalla el *Entorno Computacional* basado en KUALI-BEH.

El capítulo 6 corresponde a la aplicación de los métodos del SEI a un problema particular, definir la arquitectura para el *Entorno Computacional de KUALI-BEH*. En él se describen los requisitos de partida, el proceso de análisis, desarrollo, construcción y evaluación de la arquitectura, así como los resultados de los mismos.

Posteriormente se resumen las conclusiones, aportaciones y trabajos futuros.

Como ya se menciono, adicionalmente a este trabajo de tesis se incluyen 3 apéndices, cuyo objetivo es separar la información que por su grado de detalle puede dificultar la lectura de la tesis, pero que debe incluirse en la misma. En el apéndice I se detallan los casos de uso del *Entorno Computacional de KUALI-BEH*. El apéndice II muestra los detalles de los escenarios de atributos de calidad construidos al ejecutar el método QAW. En el apéndice III se pueden observar los demás diagramas desarrollados a lo largo de la ejecución del método V&B.

Capítulo 1. Arquitectura de Software

Introducción

En los últimos años ha tomado gran fuerza el término Arquitectura de Software, y aunque no se trata de un término nuevo, se utiliza en gran medida dentro del ámbito de la ingeniería de software, esto se debe a que en los desarrollos de software es fundamental prestarle la debida atención a la fase de diseño del software a crear, es decir, al estudio de la estructura del software. Hablando de los orígenes de la arquitectura de software, tenemos que sus raíces datan del año de 1968, en el cual Dijkstra [Dijkstra1968] hablaba del estudio de la estructura del software, centrándose principalmente en la importancia de establecer una estructura correcta de los sistemas de software. Esta estructuración debía realizarse antes de que los programadores iniciaran su labor, toda la estructuración de los sistemas de software debía realizarse con la intención y finalidad de crear un resultado correcto [ClementsNorthrop1996].

El Software Engineering Institute (SEI) es un instituto el cuál se dedica a la investigación y desarrollo de modelos de evaluación y mejora de procesos en el desarrollo de software, este instituto es financiado por el Departamento de Defensa de los Estados Unidos y Administrado por la Universidad Carnegie Mellon [SEI2013][WSEI2013]. Y es precisamente el SEI quien publica en el año de 1990 el libro de Thomas G. Lane (Tom Lane), “Studying Software Architecture Through Design Spaces and Rules” [Lane1990], en él se aborda el tema de Arquitecturas de Software y se expone una definición, la cual está basada en la definición dada por Mary Shaw un año antes, Shaw dio a conocer su definición a través de su libro “Larger Scale Systems Require Higher-Level Abstractions”, La definición de arquitectura de software dada por Mary Shaw es “Arquitectura de Software es el estudio de la estructura a gran escala y el rendimiento de los sistemas de software. Los aspectos importantes de la arquitectura de un sistema de software incluyen la división de funciones entre los módulos del sistema, los medios de comunicación entre esos módulos, y la representación de la información compartida [Shaw1989].”

Años después, en el año 1994 Mary Shaw y David Garlan exponen que hay necesidad de crear a la Arquitectura de Software, como una disciplina de carácter científico. Los

autores hablan que, como consecuencia del aumento tanto en la complejidad como en el tamaño de los productos de software, las empresas enfrentan un gran problema y que este problema no radica en la complejidad que pudieran tener los algoritmos utilizados, ni tampoco en las estructuras de datos que pudieran tener los sistemas de software, mas bien el problema principal radica en cómo se realiza la organización de los componentes que conforman el software.

1.1 Definiciones de Arquitectura de Software

Existe una variedad de definiciones acerca de lo que es la Arquitectura de Software, cada autor define a la Arquitectura de Software centrándose principalmente en los elementos que desea resaltar, pero en general casi todos los autores hablan acerca de que debe existir una organización de elementos o componentes que conformen una estructura o estructuras de un sistema, el cómo es que todos estos elementos se relacionan entre sí y participan en lograr un objetivo común. Las más sobresalientes son las siguientes:

1. El SEI define a la Arquitectura de Software de un programa o de un sistema de cómputo, como “La estructura o estructuras de un sistema, que comprende a los elementos del software, las propiedades visibles de forma externa de estos elementos y las relaciones entre ellos” [SEI2013].
2. Mary Shaw y David Garlan definen a la Arquitectura de Software como “La descripción de los elementos que comprenden un sistema, la interacción y patrones de estos elementos, los principios que guían su composición, y las restricciones de estos elementos” [ShawnGarlan1995].
3. El estándar 1471 de la IEEE en el año 2000 define a la Arquitectura de Software como “La organización fundamental de un sistema encargado de sus componentes, las relaciones entre ellos y el ambiente o entorno y los principios que orientan su diseño y evolución” [IEEE 1471].
4. Len Bass, Paul Clements y Rick Kazman escribieron: “La Arquitectura de Software es a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de estos componentes según son percibidos desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para

alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones" [Bass2012].

La Real Academia de la lengua española [RAE2013] define la palabra "estructura" como la distribución y orden de las partes con que está compuesta una obra. Así mismo define a la palabra "sistema" como un conjunto de reglas, principios o cosas que relacionadas entre sí ordenadamente contribuyen a un determinado objetivo.

La definición dada por el SEI acerca de lo que es la Arquitectura de Software será la adoptada para este trabajo de tesis.

La arquitectura de software de un sistema está constituida por *componentes* y *conectores*, este ordenamiento de los componentes y conectores en una forma particular para un problema específico es conocido como "*configuración arquitectónica*". En el libro "*Software Architecture Foundations, Theory, and Practice*" escrito por Richard N. Taylor, Nenad Medvidovid y Eric M. Dashofy [Taylor2010], se da una definición acerca de la configuración arquitectónica, la cual es la siguiente:

- "Una *configuración arquitectónica* es un conjunto de asociaciones específicas entre los componentes y conectores de software de la arquitectura del sistema."
- *Componente*: "Un componente de software es una entidad arquitectónica que encapsula un conjunto de funcionalidades de un sistema o dato, restringe el acceso a través de una interfaz definida explícitamente, y tiene definidas las dependencias que son necesarias en el contexto de su ejecución".
- *Conector*: "Un Conector une a los elementos (componentes) de la arquitectura, indicando al mismo tiempo que un componente recibe o envía un mensaje con información fundamental para el funcionamiento de otro componente". Los componentes son clasificados según los atributos que ellos contengan, por ejemplo, si tienen un indicador, la forma en que éstos se sincronizan, el tipo de implementación en que se ocupen, la información que a través de ellos es transmitida, el tiempo en que éstos están activos, etcétera.

1.2 Importancia de la Arquitectura de Software

La arquitectura de software es un conjunto de decisiones de diseño, y es el instrumento clave que le permitirá al arquitecto razonar y gestionar los cambios que el sistema pueda sufrir al evolucionar, esto es, tener una perspectiva de crecimiento del producto, tomando en cuenta que las decisiones tomadas, afectarán directamente los atributos de calidad y cualidades del sistema. Además la arquitectura, le permitirá tanto al arquitecto como al administrador del proyecto el tomar decisiones acerca de los costos, tiempos y organización de los equipos de trabajo y desarrollo de la programación del sistema a crear.

La arquitectura de software también es de gran importancia para mejorar la comunicación con los stakeholders, queriendo decir con esto, que la arquitectura deberá estar documentada, y que dicha documentación deberá ser a nivel de comprensión de las diferentes partes interesadas. Entonces vemos que la documentación de la arquitectura del sistema no se basa solamente en un solo tipo de diagramas, sino mas bien, el traducir desde diferentes perspectivas las ideas plasmadas en el diseño. Cada parte interesada en un sistema de software, es decir, el cliente, el administrador del proyecto, el probador (tester), el programador y hasta el usuario final, se preocupan y ocupan de diferentes características del sistema que se ven afectadas por la arquitectura, ejemplo de lo anterior tenemos:

- Al usuario final le preocupa que el sistema sea fiable, rápido y sobre todo que esté disponible en cualquier momento.
- Al director de la empresa u organización, además de preocuparse por los tiempos y costos, le preocupa que la arquitectura permita a los equipos trabajar de una manera muy independiente, pero al mismo tiempo teniendo una interacción de una forma disciplinada y controlada.
- Al cliente le preocupa que la arquitectura se pueda implementar adecuándose al tiempo estimado y que no se salga del presupuesto acordado o destinado a este fin.
- Y finalmente, al arquitecto le preocupa la elección correcta de las estrategias, patrones y tácticas de diseño para lograr alcanzar los objetivos del proyecto.

Además de los razones anteriores, se listan a continuación nueve de los argumentos más importantes, del por qué considerar ampliamente a la arquitectura de software.

1. Una arquitectura inhibirá o habilitar los atributos de calidad que guían a un sistema.
2. El análisis de una arquitectura permite la predicción temprana de las cualidades de un sistema.
3. La arquitectura es la portadora de las primeras y por lo tanto más importantes decisiones de diseño, además de que estas decisiones de diseño son las más difíciles para realizarles cambios.
4. Una arquitectura define un conjunto de restricciones sobre su posterior implementación.
5. La arquitectura dicta la estructura de una organización, o viceversa.
6. Una arquitectura puede proporcionar la base para la creación de modelos y prototipos, transferibles y reutilizables; haciendo con esto, que la arquitectura se convierta en el corazón de una línea de sistemas similares.
7. El desarrollo basado en la arquitectura, centra su atención en el montaje de los componentes, en lugar de simplemente crearlos.
8. Al restringir las alternativas de diseño, también se restringe a la Arquitectura, y por ende la creatividad de los desarrolladores, lo que reduce el diseño y la complejidad del sistema.
9. La documentación de una arquitectura puede servir de base para la formación e introducción de un nuevo miembro a un equipo de trabajo.

Estos nueve puntos sumados a las razones descritas en los párrafos anteriores, podemos verlos como las maneras útiles para emplear la arquitectura de software en un proyecto.

1.3 El Arquitecto de Software

Como hemos visto es de vital importancia la Arquitectura de Software dentro del desarrollo de los sistemas de software, ya que esta arquitectura es un puente entre los objetivos del negocio y el sistema final. El diseño de los sistemas de software recae sobre los hombros de un actor importantísimo "el *arquitecto de software*", esta persona debe poseer grandes conocimientos y un carácter firme, que le ayuden a desenvolverse dentro de este difícil ámbito del desarrollo de sistemas de software. Un arquitecto de software no nace de la noche a la mañana, más bien es el resultado del esfuerzo continuo por el

mejoramiento y diseño del software, todo esto con el fin de alcanzar una mejor organización de los componentes y conectores que tendrán una interacción en el software a crear, si bien el camino de crear arquitecturas de software es complejo, este software es en esencia las “ideas”, el “pensamiento” y el “razonamiento” de cómo vislumbra el arquitecto el software, la solución a un problema determinado, haciendo uso de análisis, diseños, documentación e implementación de técnicas que apoyarán al logro de los objetivos del negocio, tomando en cuenta a todos los involucrados (*stakeholders*) pero principalmente las necesidades de cada uno de ellos, estas necesidades se convierten en las directrices y funcionalidades que deberá entregar el software final.

Cuando se habla acerca de que el arquitecto de software hace uso de tácticas que le ayudarán en la creación de nuevas estructuras de software, se refiere a que el arquitecto hace uso de patrones arquitectónicos ya probados exitosamente en problemas recurrentes, además de que es válido el reciclaje de arquitecturas, es decir, tomar arquitecturas como base para crear nuevos sistemas, esto no sólo ayuda al arquitecto en su trabajo, sino que también ayuda económicamente a las empresas ya que les da la oportunidad de tener una reducción en los tiempos y los costos del desarrollo de sus sistemas, además de entregar software de mejor calidad ya que se hace uso de arquitecturas exitosas ya probadas con anterioridad.

El arquitecto de software plasma en la arquitectura sus ideas de la solución de un problema determinado, en sí, el arquitecto realiza una abstracción de un sistema, los elementos que interactúan entre sí por medio de interfaces que detallan la participación sobre un elemento. El realizar una abstracción arquitectónica nos permite ver a un sistema en términos de sus componentes, el cómo se organiza, relacionan y se componen estos componentes, además de dar a conocer cuáles son sus propiedades que apoyen el pensamiento del arquitecto. Es entonces que decimos que todo sistema de software tiene una arquitectura de software. Es ahí donde podemos enfrentarnos a un gran problema cuando hablamos de sistemas ya creados y que por una u otra razón se tienen nuevas necesidades que cubrir, y que las personas involucradas en la creación y diseño de esta arquitectura ya no se encuentran dentro de la organización o empresa, además de que la documentación de la arquitectura no existe porque el arquitecto no hizo uso de un método

para dejar claro el funcionamiento del sistema, el código fuente de este sistema a sufrido cambios y modificaciones con el paso del tiempo, y lo único que se tiene es código binario en ejecución. Todo esto revela que la arquitectura puede existir independientemente de su descripción o especificación, y con ello la importancia de la documentación de dicha arquitectura hecha por el arquitecto.

1.4 Los Involucrados (Stakeholders)

Existen muchas personas y organizaciones interesadas en el desarrollo de un software, a estas entidades se les llama “Stakeholders”. Es así, que un stakeholder es cualquier persona que tenga interés en el éxito del sistema, por ejemplo: el cliente, los usuarios finales, los desarrolladores, el director del proyecto, e incluso los vendedores que comercializarán el sistema. Pero a pesar de todo, los stakeholders tienen una participación compartida en el éxito del sistema, es decir, cada stakeholder le da a conocer al arquitecto de software las distintas necesidades y funciones que espera cumpla el sistema. La figura 1.1 muestra como el arquitecto recibe de los stakeholders unas útiles “sugerencias”.

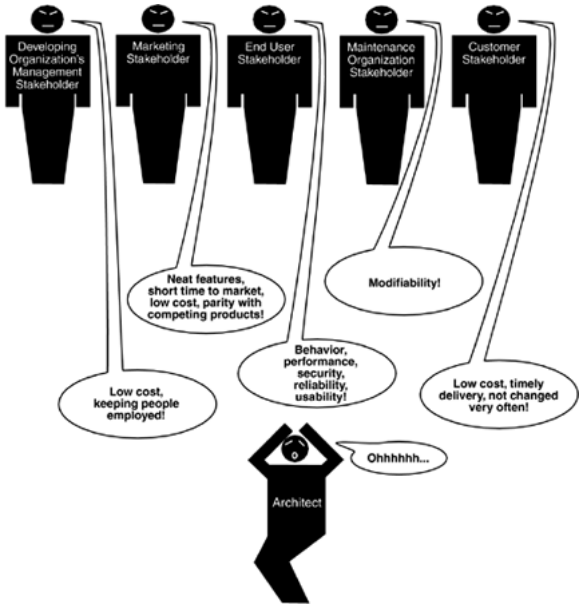


Figura 1.1: El arquitecto y los distintos stakeholders (Imagen obtenida de [ET2013]).

Para tener un sistema aceptable, implica varias cosas, entre ellas: un adecuado desempeño, una gestión de memoria y de red, fiabilidad, disponibilidad, seguridad,

modificabilidad, facilidad de uso, compatibilidad e interoperabilidad con otros sistemas, etcétera.

El problema radica en que cada uno de los diferentes stakeholders tiene distintas preocupaciones y objetivos, algunos de los cuales son contradictorios unos con otros, es entonces que el arquitecto debe realizar una muy buena captura de los distintos requerimientos, a fin de que salgan a flote los distintos requerimientos de calidad que necesita el sistema y balancear los conflictos que surjan a partir de estos requerimientos.

Para lograr ese balance, los arquitectos deben conocer personalmente a los diferentes stakeholders, hablar con ellos, escucharlos y hacerlos participar, además de ponerse en sus zapatos para ver lo que desean desde diferentes puntos de vista.

1.5 Los múltiples contextos de la Arquitectura de Software

La arquitectura de software es sostenida e informada por varias actividades críticas en los diversos contextos en los que juega un papel importante. Estos contextos, son los siguientes:

1. Técnica. ¿Qué rol juega la técnica de la arquitectura de software en el sistema o sistemas de los cuales forma parte?
2. Ciclo de vida del proyecto. ¿Cómo una arquitectura de software se relacionan con las otras fases del ciclo de vida de desarrollo de software?
3. Negocios. ¿Cómo afecta la presencia de una arquitectura de software el entorno empresarial de una organización?
4. Profesional. ¿Cuál es el papel de un arquitecto de software en una organización o un proyecto de desarrollo?

   Un reto para el arquitecto es el de imaginar lo que podría cambiar y adoptar mecanismos para proteger al sistema, así como también su desarrollo, si los cambios previstos sucedieran.

1.5.1 La Arquitectura en un contexto técnico

Las Arquitecturas inhiben o permiten el logro de los atributos de calidad, uno de los usos de una arquitectura es apoyar el razonamiento del arquitecto acerca de las

consecuencias de realizar un cambio o cambios, particularmente en los atributos de calidad importantes para un sistema en sus inicios.

Las arquitecturas inhiben o permiten alcanzar los Atributos de Calidad

En el apartado 1.3 se mencionaron las razones por las que la "Arquitectura de Software" es importante y merece ser estudiada. Las primeras razones se refieren a las incidencias técnicas de una arquitectura en todos los sistemas que la utilizan:

1. Una arquitectura inhibirá o permitirá el logro de los atributos de calidad de un sistema.
2. Es posible predecir muchos aspectos de las cualidades de un sistema mediante el estudio de su arquitectura.
3. Una arquitectura hace que sea más fácil el razonar y gestionar un cambio.

Todas estas razones son acerca del efecto que tiene la arquitectura sobre los atributos de calidad de un sistema, aunque el primero de ellos lo afirma de una manera más explícita. Si bien todas las razones mencionadas en el apartado 1.3 son declaraciones válidas acerca de la contribución de la arquitectura a un sistema, probablemente la razón más importante es su efecto crítico sobre los atributos de calidad.

1.5.2 La Arquitectura en el contexto del ciclo de vida de un proyecto

Los procesos de desarrollo de software son los enfoques estándar para el desarrollo de sistemas de software. Estos procesos imponen una disciplina a los ingenieros de software y, aún más importante, a los equipos de ingenieros de software. Le dicen a los miembros del equipo qué es lo que deben de hacer antes y después. Existen cuatro procesos dominantes en el desarrollo de software, los cuales se describen de manera cronológica en cuanto a su aparición:

Modelo en Cascada: Durante muchos años el modelo en cascada dominó el campo del desarrollo de software. Este modelo organizó el ciclo de la vida en una serie de actividades secuenciales conectadas, cada una con las condiciones de entrada y salida y una relación formal con las etapas tanto de arriba como de abajo. El proceso inicia con la especificación

de requerimientos, seguido del diseño, la implementación, la integración, las pruebas, la instalación y finalmente el mantenimiento.

Iterativo e Incremental: Con el tiempo la retroalimentación del modelo en cascada llegó a ser tan pronunciada, de tal manera, que fue evidente que era mejor el pensar en el desarrollo de software como una serie de ciclos cortos, en los cuales algunos requerimientos conducirían a un cierto diseño, que puede ser implementado y probado, mientras que para el próximo ciclo, las necesidades están siendo capturadas y diseñadas. Estos ciclos fueron llamados "iteraciones", en el sentido de que cada una de las iteraciones nos acerca más a la solución definitiva de un problema de software dado, es decir, cada una de las iteraciones deberá de entregar un adelanto del software final y este adelanto deberá ser útil para el cliente. Un proceso iterativo especialmente conocido es el "Proceso Unificado" (originalmente era llamado "Proceso Unificado de Rational") RUP por sus siglas en inglés; fue la empresa Rational quien desarrolló este proceso. En él se definen cuatro fases para cada iteración: Inicio, Elaboración, Construcción y Transición. Un conjunto de casos de uso definen los objetivos de cada iteración, y las iteraciones se ordenan de tal manera que se traten primero los riesgos más grandes.

Ágil: El término "Desarrollo de Software Ágil" se refiere a un grupo de metodologías de desarrollo de software, entre las metodologías ágiles más conocidas se encuentran: Scrum, Kanban, eXtreme Programming, Lean y Crystal Clear. Todas estas metodologías son iterativas e incrementales, por tal motivo, se pueden considerar algunos métodos iterativos como ágiles. Lo que distingue a las prácticas ágiles es la entrega temprana y frecuente de partes del software final, además de una estrecha colaboración entre los desarrolladores y el cliente, existen equipos auto-organizados y un enfoque de adaptación a los cambios.

Desarrollo dirigido por Modelos: El "Model-Driven Development" está basado en la idea que se puede no escribir código en lenguajes de programación, sino que se deben de crear modelos de domino, de los cuales se generará automáticamente el código. En el desarrollo dirigido por modelos, se crea un modelo independiente de la plataforma (PIM), el cual es combinado con un modelo de definición de plataforma (PDM) para generar un código ejecutable. De esta manera, el PIM es una realización pura de los requerimientos

funcionales, mientras que el PDM dirige las especificaciones de la plataforma y los atributos de calidad.

Todos estos procesos incluyen el diseño entre sus obligaciones, y sabiendo que la arquitectura es un tipo especial de diseño, la arquitectura de software está presente en cada uno de estos procesos. El cambio de un proceso de desarrollo a otro dentro de un proyecto, requiere la presencia de un arquitecto para guardar toda la información que sea útil y que le ayude a determinar la forma en que se integrará en el nuevo proceso.

Sin importar que proceso de desarrollo de software o modelo de ciclo de vida se este usando, existen una serie de actividades que están involucradas en la creación de una arquitectura de software, usando la arquitectura para realizar un diseño completo, y después implementar o gestionar la evolución de un objetivo del sistema. Dependiendo el proceso que se utilice determinará con qué frecuencia y cuándo se deberán revisar y elaborar cada una de éstas actividades. Estas actividades incluyen:

1. Elaborar casos de uso para el sistema.
2. Comprender los requerimientos significativos para la arquitectura.
3. Crear o seleccionar la arquitectura.
4. Documentar y comunicar la arquitectura a los distintos stakeholders.
5. Analizar y evaluar la arquitectura.
6. Implementar y probar el sistema basado en la arquitectura.
7. Asegurarse que la implementación del sistema sea conforme a la arquitectura.

1.5.3 La Arquitectura en el contexto de negocios

Las arquitecturas de software y los sistemas no se construyen de manera frívola y superficial, se toman en cuenta algunos si no es que todos los propósitos y objetivos que tienen los negocios y las organizaciones, sabiendo que esos propósitos pueden cambiar con el tiempo.

Las arquitecturas y los objetivos de las empresas

Los sistemas son creados para satisfacer los objetivos del negocio de una o más organizaciones. Las organizaciones de desarrollo desean obtener un beneficio, o capturar a

una parte del mercado, o permanecer dentro de cierto negocio o giro, o también ayudar a sus clientes a hacer mejor su trabajo, o mantener a sus empleados remunerados, o hacer a sus accionistas felices, o un poco de cada una.

Las arquitecturas y la organización en el desarrollo

Una organización en el desarrollo contribuye a muchos de los objetivos del negocio, los cuales influyen en la arquitectura. En términos más generales, una organización que a menudo invierte en activos, es decir, en arquitecturas existentes y en los productos basados en ella. La arquitectura de software puede ser la base para un proyecto de desarrollo, y dentro de ese proyecto se contemplan sistemas similares, ayudando con esto a la estimación de los costos, pues se asume que se tendrá un alto grado de reutilización, además de las habilidades y productividad de los programadores.

Adicionalmente, una organización puede hacer una inversión a largo plazo en una infraestructura para conseguir objetivos estratégicos, y de esta forma ver a un sistema como un medio de financiamiento para ampliar su infraestructura.

Es entonces, que la estructura de una organización puede dar forma a la arquitectura de software y viceversa. A menudo las organizaciones están organizadas en torno a la tecnología y a las aplicaciones, por ejemplo, un grupo de bases de datos, un grupo de redes, un equipo para las reglas del negocio, o un grupo para las interfaces de usuario, etcétera. Así, la identificación explícita de subsistemas distintos en la arquitectura, con frecuencia dará lugar a la creación de un grupo con el mismo nombre que tiene el subsistema dentro de la arquitectura.

1.5.4 La Arquitectura en el contexto profesional

Se sabe que los arquitectos de software necesitan algo más que habilidades técnicas. Los arquitectos tienen que explicar a los distintos stakeholders las prioridades elegidas así como sus propiedades de diferentes maneras, además de aclarar por qué algunos stakeholders en particular no están teniendo todas sus expectativas cumplidas. Entonces, para ser un arquitecto de software eficaz, se necesitan tener las habilidades de negociación y comunicación, conjuntas en una forma diplomática.

Capítulo 2. Arquitectura de Software y Atributos de Calidad

Introducción

En este capítulo se presentan los atributos de calidad más importantes, profundizando sobre sus relaciones con la arquitectura de software.

2.1 Atributos de Calidad

Un Atributo de Calidad "QA" (por sus siglas en inglés "Quality Attribute"), es una propiedad medible y comparable de un sistema, la cual se utiliza para indicar qué tan *bien* el sistema satisface las necesidades de los diferentes stakeholders, o cual es el grado de satisfacción de los usuarios y los desarrolladores con respecto al sistema. Los atributos de calidad influencian a la arquitectura de software; por tal motivo, se puede pensar en un atributo de calidad como la medición de qué tan bueno o qué tan bien está realizado un producto a lo largo de alguna medición (dimensión) de interés para uno o varios de los stakeholders [Bass2012]. El SEI en [Bass2012] propone los siguientes atributos de calidad:

- **Modificabilidad** (modifiability). Se refiere al costo de realzar un cambio, tomando en cuenta qué se va a cambiar, cuándo y quién va a realizar el cambio.
- **Disponibilidad** (availability). Tiene que ver con las fallas del sistema y las consecuencias asociadas a estas fallas.
- **Desempeño** (performance). Se refiere al tiempo de respuesta del sistema cuando ocurre un evento como interrupciones, mensajes, solicitudes del usuario, etcétera.
- **Usabilidad** (usability). Concierne a la facilidad de usar el sistema, por ejemplo la facilidad de aprender las características, aprender a usar el sistema de una forma eficiente, minimizar el impacto de errores en su manejo, qué tan fácil se adapta el sistema a las necesidades del usuario y la satisfacción que tiene el usuario para con el sistema.
- **Seguridad** (security). Es la habilidad que tiene el sistema de resistir usos no autorizados o ataques, sin dejar de proveer sus servicios a usuarios legítimos.

- **Facilidad de Pruebas** (testability). Este atributo se refiere a la facilidad de mostrar las fallas del sistema a través de pruebas y de esta forma poder solucionarlas.
- **Interoperabilidad** (interoperability). Este atributo mide la capacidad que tiene los componentes de un sistema para trabajar con otros componentes de otro sistema.

La inhibición o habilitación de Atributos de Calidad del Sistema

Si un sistema será capaz de exhibir sus requerimientos en sus atributos de calidad, estos atributos de calidad son substancialmente determinantes para su arquitectura.

La frase anterior es substancialmente importante, los ejemplos siguientes, nos ayudarán a asimilarla y comprenderla mejor.

1. Si el sistema requiere un alto rendimiento, entonces es necesario prestar atención a la gestión del comportamiento de los elementos basándose en el tiempo, del uso de recursos compartidos, de la frecuencia y el volumen de la comunicación entre elementos.
2. Si la modificabilidad es importante, entonces es necesario prestar atención en la asignación de responsabilidades a los elementos, de tal manera que la mayoría de los cambios en el sistema afecte a un pequeño número de esos elementos (Lo ideal sería que cada cambio afecta a un solo elemento).
3. Si el sistema debe ser altamente seguro, entonces es necesario gestionar y proteger la comunicación, controlando el acceso a la información entre elementos; también puede ser necesario introducir elementos especializados (tales como un mecanismo de autorización) en la arquitectura.
4. Si se cree que la escalabilidad será importante para el éxito del sistema, entonces es necesario localizar cuidadosamente el uso de los recursos para facilitar la sustitución de elementos de mayor capacidad.
5. Si el proyecto necesita tener la habilidad de desarrollar un sistema de forma iterativa e incremental, se debe manejar cuidadosamente el uso entre los componentes.
6. Si se desea que los elementos que conforman el sistema puedan ser reutilizables en otros sistemas, entonces es necesario el restringir el acoplamiento entre los elementos,

de modo que al extraer un elemento, éste no salga con demasiados elementos adjuntos de su entorno actual, para ser útil en otros sistemas.

Las estrategias para estos y otros atributos de calidad son sumamente arquitectónicas. Pero una arquitectura por sí sola no puede garantizar la funcionalidad o la calidad requerida de un sistema. Un diseño pobre o implementaciones deficientes pueden debilitar e incluso destruir un diseño arquitectónico adecuado. Las decisiones en todas las etapas del ciclo de vida, comenzando desde el diseño arquitectónico, pasando por la codificación hasta llegar a la implementación, afectan la calidad del sistema. Por lo tanto, la calidad de un sistema no es completamente una función de un diseño arquitectónico.

Una buena arquitectura es necesaria, pero no suficiente, para asegurar la calidad. Para lograr el alcance de los atributos de calidad, éstos deben ser considerados en todo el diseño, implementación y despliegue. Ningún atributo de calidad es totalmente dependiente del diseño, ni tampoco es totalmente dependiente de la implementación o del despliegue. Los resultados satisfactorios son una cuestión de conseguir el panorama completo (arquitectura), así como los detalles (implementación) correctos.

Por ejemplo, la modificabilidad es determinada por cómo la funcionalidad es dividida y acoplada (arquitectura) y por las técnicas de codificación dentro de un modulo o elemento. Por lo tanto, un sistema es típicamente modificable si los cambios implican el menor número posible de elementos distintos. Sin embargo, a pesar de tener la arquitectura ideal, siempre existe la posibilidad de encontrarnos con un sistema difícil de modificar.

2.2 Calidad

La calidad aparece como una necesidad, pues permite competir con mayores posibilidades de éxito. Específicamente para la calidad de software aparecen varias definiciones, las cuales son:

- Pressman [Pressman 2009] dice que “es la concordancia con los requisitos funcionales y de rendimiento establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado de forma profesional”.

- La ISO/IEC (International Standart Organitation), dice la calidad de software es “la totalidad de rasgos y atributos de un producto de software que le apoyan en la capacidad de satisfacer sus necesidades explícitas o implícitas” [ISO/IEC 25060].
- Barbacci y colegas [Barbacci 2003] establecen que “el desarrollo de formas sistemáticas para relacionar los atributos de calidad de un sistema a su arquitectura provee una base fundamental para la toma de decisiones objetivas sobre acuerdos de diseño y permite a los arquitectos de software realizar predicciones sobre los atributos del sistema que son libres de prejuicios, obligaciones y responsabilidades no triviales”.

2.3 Arquitectura y Requerimientos

Los requerimientos para un sistema pueden darse en una gran variedad de formas, como por ejemplo: maquetas, sistemas existentes, casos de uso, requerimientos textuales, historias de usuarios, etcétera.

No importa la fuente, todos los requerimientos abarcan las siguientes categorías:

1. ***Requerimientos Funcionales.*** Estos requerimientos establecen lo que el sistema deberá hacer, y cómo deberá comportarse o reaccionar a los diferentes estímulos en tiempo de ejecución.
2. ***Requerimientos de Atributos de Calidad.*** Estos requerimientos son las calificaciones de los requerimientos funcionales o del sistema en general. Una calificación de un requerimiento funcional es un elemento, por ejemplo, la rapidez con que una función es realizada por el sistema, o qué tan resistente es el sistema a una entrada errónea. Ahora, una calificación de un producto global, también es un elemento, pero este elemento deberá abarcar a todo el sistema, por ejemplo, el tiempo de implementación del producto o una limitación de los costos operativos.
3. ***Restricciones.*** Una restricción es una decisión de diseño con cero grados de libertad. Es decir, una decisión de diseño que ya ha sido tomada desde antes de iniciar con la arquitectura. Ejemplos de éstos pueden ser: el requerimiento de utilizar un determinado lenguaje de programación, o el de (re)utilizar un determinado módulo ya existente, etcétera. Estas restricciones o elecciones,

deberían de ser tomadas exclusivamente por el arquitecto, pero factores externos han llevado a diferentes stakeholders a dictar estas restricciones de diseño.

¿Pero cómo dar una respuesta arquitectónica a cada uno de éstos tipos de requerimientos?

Los requerimientos funcionales se satisfacen en el diseño, mediante la asignación de una secuencia apropiada de las responsabilidades. La asignación de responsabilidades a cada uno de los elementos dentro de la arquitectura es una decisión fundamental de diseño arquitectónico.

Los requerimientos de los atributos de calidad son satisfechos por las distintas estructuras diseñadas en la arquitectura, además de los comportamientos y funciones de los elementos que pueblan esas estructuras.

Finalmente las restricciones son satisfechas por la aceptación de las decisiones de diseño ya tomadas y conciliándolas con las demás decisiones de diseño que pudieran verse afectadas.

Los atributos de calidad deben considerarse tanto en las fases de diseño como en las fases de implementación del sistema, sin embargo no todas las cualidades se manifiestan ni se consiguen de igual manera en cada una de estas fases. La arquitectura es crítica para la obtención de la mayoría de estos atributos, que por tanto deben ser evaluados a este nivel, mientras que otros dependen más de los algoritmos utilizados y del estilo y calidad de la implementación. Por supuesto, muchos de los atributos de calidad dependen de ambas cosas.

2.4 Consideraciones de los atributos de calidad

Las funciones de un sistema están estrechamente ligas con los atributos de calidad, es decir, las funciones que realice un sistema son sostenidas por los atributos de calidad.

Los atributos de calidad han sido de gran interés por la comunidad de la ingeniería de software desde hace ya varios años, desde el punto de vista del arquitecto hay tres problemas con los atributos de calidad de un sistema:

1. Las definiciones dadas para un atributo no son comprobables. No tiene sentido decir que un sistema sea “modificable” , pues cada sistema puede ser modificable con respecto a un conjunto de cambios y no modificable con respecto a otros. Los atributos de calidad son similares a este respecto: un sistema puede ser robusto con respecto a algunos fallos y frágil con respecto a otros. Y así sucesivamente.
2. La discusión a menudo se centra sobre un solo atributo de calidad. Un arquitecto debe entender que un sistema es la conjunción de varios atributos de calidad y debe crear soluciones arquitectónicas para el manejo de esos atributos de calidad.
3. Para cada atributo de calidad, existe una comunidad que ha desarrollado su propio vocabulario. Cuando existe algún problema ya sea de performance, o de seguridad, o de cualquier otro, hay diferentes comunidades o grupos de personas especialistas en un atributo en particular, pero que han decidido llamarlos de diferente forma, cuando en realidad se puede estar hablando de lo mismo, pero haciendo uso de diferentes términos.

Una solución para los primeros dos problemas es el uso de escenarios de atributos de calidad como un medio para la caracterización de éstos atributos de calidad. Y una solución para el tercer problema sería el realizar una discusión conjunta sobre cada atributo y centrarse en las preocupaciones de cada uno de los grupos participantes, para así poder ilustrar los conceptos fundamentales.

Aunque existen diferentes formas para catalogar a los atributos de calidad, Clements, Bass y Kazman en [Bass2012] dividen a los atributos de calidad en dos grupos, en el primer grupo se encuentran los atributos de calidad que describen alguna propiedad del sistema en tiempo de ejecución, tales como la disponibilidad, el rendimiento o la aplicabilidad. En el segundo grupo están los atributos de calidad que describen alguna propiedad del desarrollo del sistema, tales como la modificabilidad o la capacidad de prueba.

Dentro de un sistema complejo, los atributos de calidad nunca pueden ser alcanzados de manera aislada. El logro de cualquier atributo de calidad tendrá un efecto, a veces positivo y otras negativo, en el logro de los demás atributos de calidad que convivan en un mismo sistema.

El determinar un diseño que satisfaga todos los requerimientos de atributos de calidad es parcialmente una cuestión de hacer las negociaciones o valoraciones adecuadas, a modo de balancear los atributos más importantes para el sistema en cuestión.

2.5 Especificaciones de los requerimientos de los atributos de calidad

Todo requerimiento de atributo de calidad deberá ser inequívoco y comprobable, para esto se deberá utilizar una forma común para especificar todos los requerimientos de atributos de calidad. Esto tiene la ventaja de poner énfasis en los puntos en común entre todos los atributos de calidad, pero a su vez también tiene la desventaja de forzar algunos aspectos de los atributos de calidad.

Clements y colegas en [Bass2012] expresan un atributo de calidad con las siguientes partes:

1. *Estímulo*. El término "estímulo" es utilizado para describir un evento o condición que llegue al sistema y requiera una respuesta, es decir, un estímulo puede ser cualquier evento, por ejemplo, un ataque al sistema, una solicitud de funcionalidad del sistema que exija un alto performance, o simplemente cualquier operación de un usuario.
2. *Fuente de estímulo*. Un estímulo tiene que tener una fuente, es decir, debe venir de alguna parte, puede ser un ser humano o un sistema externo, el cual genere el estímulo. La fuente del estímulo puede afectar la forma en que es tratada por el sistema. Por ejemplo, una petición de un usuario de confianza no será sometida al mismo escrutinio que una petición que venga de un tipo de usuario con bajo perfil de confianza.
3. *Respuesta*. La forma en que el sistema debe responder a los estímulos también debe de ser especificada. La respuesta consiste en las responsabilidades que el sistema o los desarrolladores deben llevar a cabo en respuesta al estímulo, es decir, la respuesta es la actividad que se realiza como consecuencia de la llegada del estímulo. Por ejemplo, en un escenario de rendimiento, un evento llega (estímulo) y el sistema debe de procesar ese evento y generar una respuesta. En un escenario de modificabilidad, llega una solicitud de modificación (estímulo)

los desarrolladores deberían aplicar la modificación (sin efectos secundarios) y luego probar y desplegar la modificación.

4. *Medida de la respuesta*. La determinación de si una respuesta es satisfactoria o si el requerimiento se cumple, se habilita al proporcionar una medida de la respuesta. Cuando se produce una respuesta, debe ser medible de alguna manera para que el requerimiento pueda ser probado.
5. *Medio ambiente*. El medio ambiente se refiere al entorno de un requerimiento, es el conjunto de circunstancias en las que el escenario se lleva a cabo. Por lo tanto, podemos ver al "medio ambiente" como las condiciones bajo las cuales se encuentra el sistema en el momento exacto en que se produce un estímulo. Por ejemplo, un sistema pueda estar en una condición de sobrecarga o de funcionamiento normal, o en algún otro estado. El entorno actúa como calificador del estímulo.
6. *El artefacto*. Es la porción del sistema al que se aplica el requerimiento, con frecuencia se toma todo el sistema, pero algunas veces también son tomadas porciones específicas del sistema.

Se pueden distinguir escenarios generales de atributos de calidad, los que son independientes del sistema y pueden, potencialmente, pertenecer a cualquier otro sistema, hasta escenarios concretos de los atributos de calidad de un sistema, los cuales son específicos para un sistema en particular. También se pueden caracterizar los atributos de calidad como un conjunto de escenarios. Por supuesto, para traducir estas caracterizaciones de atributos genéricos en los requerimientos para un sistema en particular, los escenarios deberán modificarse para un sistema específico. La figura 2.1 ejemplifica las partes de un escenario general de un atributo de calidad.

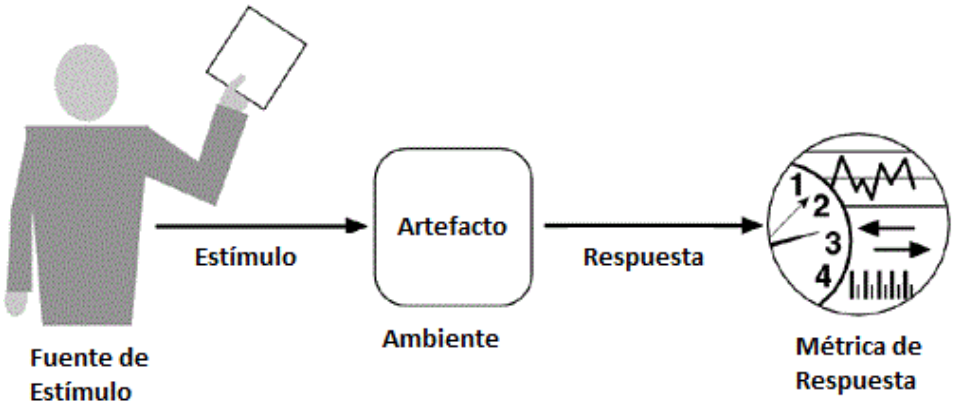


Figura 2.1 Las partes de un escenario de un atributo de calidad (Imagen obtenida de [Testing2012]).

2.6 La satisfacción de los atributos de calidad a través de las tácticas

Los requerimientos de atributos de calidad especifican las respuestas del sistema que, con un poco de suerte y una buena planificación, hacen visibles los objetivos de la empresa. Un arquitecto de software puede hacer uso de tácticas arquitectónicas para lograr el éxito de los atributos de calidad requeridos. Una táctica es una decisión de diseño que influye en el logro de una respuesta positiva a un atributo de calidad, la cual afecta directamente en la respuesta del sistema ante algún estímulo. Una táctica consiste en dar una sola respuesta a un atributo de calidad, dentro de una táctica no existe ninguna consideración para el balance o acuerdos entre los distintos atributos de calidad existentes en ese momento. El balance debe ser considerado y controlado por el arquitecto de una forma explícita.

Un diseño de una arquitectura de software de un sistema consiste en aplicar un conjunto de decisiones de diseño, algunas de estas decisiones ayudan a controlar las respuestas para un atributo de calidad, mientras que otras aseguran el logro de la funcionalidad del sistema.

2.7 Funcionalidad

La funcionalidad es la capacidad del sistema para realizar el trabajo para el que fue concebido. De entre todos los requerimientos existentes, la funcionalidad tiene una relación muy estrecha con la arquitectura. Pero también debe quedar claro que la funcionalidad no determina la arquitectura. Es decir, si la funcionalidad fuera lo único que nos importara, no se tendría que dividir el sistema en pedazos, con un solo modulo sin estructura interna

estaría perfecto. Pero en lugar de ello, se diseñan los sistemas como un conjunto estructurado de elementos arquitectónicos, en niveles, capas, servicios, etcétera; eso se realiza para que sean comprensibles y para apoyar a los atributos de calidad. Pero la funcionalidad es independiente de cualquier estructura particular, la funcionalidad se consigue asignando responsabilidades a los diferentes elementos arquitectónicos, lo que resulta en una estructura arquitectónica básica.

Aunque las responsabilidades se pueden asignar arbitrariamente a cualquiera de los módulos, la arquitectura de software restringe esta asignación cuando otros atributos de calidad son importantes. Por ejemplo, los sistemas son divididos para que varias personas o grupos puedan construirlos cooperativamente y/o paralelamente. El interés del arquitecto en la funcionalidad está en la forma en que interactúa o limita a otras cualidades.

Capítulo 3. Estilos y Patrones Arquitectónicos

Introducción

Los diferentes estilos de arquitectónicos, y los diferentes tipos patrones, ayudan a definir el diseño de la arquitectura, como se podrá ver a lo largo de este capítulo.

3.1 Estilos Arquitectónicos

A través de los años, se les ha llamado a los *estilos arquitectónicos* de diferentes maneras, tales como "clases de arquitectura", "arquetipos recurrentes", "tipos arquitectónicos" y "paradigmas topológicos". Los estilos arquitectónicos toman y aíslan las cualidades y propiedades comunes de diseños arquitectónicos similares, además de expresar a la arquitectura en un sentido teórico y formal, estableciendo de esa manera un tópico esencial en la disciplina de la Ingeniería de Software. Llamarlos "*estilos arquitectónicos*" subraya explícitamente la existencia de una taxonomía de las distintas alternativas de estilos. Llamarlos patrones arquitectónicos, por el contrario, establece su vinculación con otros patrones posibles pero de distinta clase: de diseño, de organización o proceso, de código, de interfaz de usuario, de prueba, de análisis, o de refactorización [Reynoso2004].

En los comienzos de la arquitectura de software se observó que como resultado o respuesta a distintos problemas semejantes entre sí, hablando en el sentido de diseño e implementación, ciertas configuraciones aparecían una y otra vez, aunque los problemas a resolver fueran distintos. A esas soluciones que eran recurrentes pronto se les llamó "estilos" haciendo una analogía con el uso de ese término en la arquitectura de edificios. Es así, que un estilo arquitectónico describe una clase de arquitectura, dicho de otra manera, es la identificación de piezas que se encuentran repetidamente en la práctica, es un conglomerado de restricciones, patrones y reglas ya probados en sistemas exitosos anteriormente, que dan la base para estructurar un sistema, en los dominios de una arquitectura de software, es decir, en términos de componentes y conectores.

Al igual que sucede con los patrones de diseño de arquitecturas de software, también los estilos arquitectónicos tienen un nombre: cliente-servidor, modelo-vista-controlador, tuberías-filtros, arquitectura en capas, etcétera.

Dentro de los estilos arquitectónicos se especifican las reglas y limitantes acerca de qué tipo de componentes se pueden utilizar, y también el tipo de conectores dentro de este estilo arquitectónico para que los componentes puedan tener comunicación e interacción. La definición dada por Richard N. Taylor y colegas en [Taylor2010] es la siguiente:

"Un estilo arquitectónico es una colección de decisiones de diseños arquitectónicos que son aplicables a un contexto de desarrollo dado, limitan las decisiones de diseño arquitectónico que son específicos de un determinado sistema dentro de un contexto para obtener cualidades benéficas en cada sistema resultante".

Otra definición de estilo arquitectónico es la dada por Roy Thomas Fielding en [Fielding2000], en la cual expresa lo siguiente:

"Un estilo arquitectónico es un conjunto coordinado de restricciones arquitectónicas que restringe los roles/rasgos de los elementos arquitectónicos y las relaciones permitidas entre esos elementos dentro de la arquitectura que se conforma a ese estilo".

Como podemos observar, dentro de los estilos arquitectónicos se hace uso de los patrones arquitectónicos.

Clasificación De Los Estilos Arquitectónicos

Mary Shaw en [Shaw1995] clasifica los estilos arquitectónicos de la siguiente forma [Reynoso2004]:

- Arquitecturas orientadas a objetos
- Arquitecturas basadas en estados
- Arquitecturas de flujo de datos: Arquitecturas de control de realimentación
- Arquitecturas de tiempo real
- Modelo de diseño de descomposición funcional
- Modelo de diseño orientado por eventos
- Modelo de diseño de control de procesos
- Modelo de diseño de tabla de decisión

- Modelo de diseño de estructura de datos

De la misma manera, pero esta vez conjuntamente con David Garlan, Mary Shaw expresa una nueva y diferente clasificación de estilos arquitectónicos en [ShawGarlan1995]. La cual es la siguiente [Casales2011]:

- Abstracción de Datos y Organización Orientada a Objetos
- Flujo de Datos
  - Secuencial en Lotes
  - Red de Flujo de Datos (Tuberías y Filtros)
  - Control de Procesos
- Orientadas en Datos
  - Repositorios
  - Pizarra
- Jerárquicas
  - Principal – Subrutina (Main - Subrutine)
  - Maestro – Esclavo (Master - Slave)
  - Capas (Layers)
  - Máquina virtual (Vírtual Machine)
- Comunicación Implícita Asíncrona
  - Invocación Implícita Basada en Eventos sin Búfer
  - Invocación Implícita Basada en Mensajes con Búfer
- Orientadas a Interacción
  - Modelo – Vista – Controlador “MVC” (Model-View-Controller)
  - Presentación – Abstracción – Control “PAC” (Presentation-Abstraction-Control)
- Distribuidas
  - Cliente – Servidor
  - Multiniveles
  - Broker
  - Orientadas a Servicios “SOA”
- Basadas en Componentes
- Heterogéneas

3.2 Patrones Arquitectónicos

Frank Buschmann [Buschmann1996] define a un *patrón arquitectónico* como una descripción de un problema particular recurrente que surge en contextos específicos de diseño, y presenta un esquema genérico de eficiencia probada para su solución. El esquema

de solución es especificado describiendo los componentes de los que se construye, sus responsabilidades y relaciones, y la forma en que colaboran entre ellos [Buschmann1996][Casales2011].

Tomando la definición de Buschmann, podemos darnos cuenta que dentro de un patrón se encuentran documentadas las experiencias de diseño ya probadas, identificadas y especificadas respecto a los componentes de ese patrón, lo mejor de hacer uso de un patrón es que dicho patrón describe a los componentes, pero desde una perspectiva de clases u objetos, además de detallar sus relaciones y responsabilidades con otros componentes, por tanto, podemos ver que la unión de los componentes de un patrón dan solución a una cierta clase de problemas, he ahí que los patrones dan soluciones a problemas recurrentes en el diseño.

Los patrones arquitectónicos establecen una relación entre:

1. *Un contexto.* Una situación común que se repite en el mundo y que da lugar a un problema.
2. *Un problema.* Un problema generalizado que se encuentra en determinado contexto. La descripción de un patrón incluye la descripción de un problema y sus variantes, además de describir las “fuerzas” en pro y en contra. Las descripciones de los problemas a menudo incluyen los atributos de calidad que se deben cumplir.
3. *Una solución.* Una solución arquitectónica exitosa para un problema determinado, deberá estar apropiadamente resumida y, deberá contener la descripción de las estructuras arquitectónicas que resuelven el problema, incluyendo la forma en que se equilibran las cosas, además de una descripción de las responsabilidades y las relaciones estáticas entre los elementos, o la descripción del comportamiento en tiempo de ejecución y la interacción entre los elementos. La descripción también deberá dejar claro que atributos de calidad son proporcionados para las configuraciones estáticas de los elementos y en tiempo de ejecución. La solución en un patrón se determina y se describe por:
    - Un conjunto de tipos de elementos
    - Un conjunto de conectores o mecanismos de interacción.
    - Un diseño topológico de los componentes.

- Un conjunto de restricciones semánticas que abarque la topología anterior, el comportamiento de los elementos y los mecanismos de interacción.

Los sistemas complejos utilizan varios patrones a la vez, estos sistemas emplean un patrón arquitectónico, y dentro de este patrón también utilizan otros patrones y dentro de los patrones, tácticas; esto con el fin de modularizar el sistema y hacerlo más comprensible.

Listando algunas de las funcionalidades que tienen los patrones, tenemos:

- El uso de los patrones favorece la comunicación entre el arquitecto o arquitectos y los programadores del sistema, haciendo con esto que sea más fácil el debatir acerca de las mejores soluciones (patrones) a determinados problemas.
- Al hacer uso de los patrones, es que se puede comenzar a realizar la documentación de la arquitectura, tomando como base la descripción de los patrones utilizados en la arquitectura.
- Los patrones ayudan al arquitecto a reutilizar buenos diseños al basar los nuevos diseños en la experiencia previa, es decir, un arquitecto de software familiarizado con los patrones puede aplicarlos inmediatamente en los problemas de diseño sin tener que redescubrirlos.
- Los patrones de diseño ayudan al arquitecto a elegir las alternativas de diseño que hacen que un sistema sea reutilizable, y a evitar aquéllas que dificultan dicha reutilización.
- Mejoran la documentación y el mantenimiento de los sistemas existentes al proporcionar una especificación explícita de las interacciones entre clases y objetos y de cuál es su intención en el diseño. En definitiva, los patrones ayudan al arquitecto al lograr un buen diseño más rápidamente y a construir arquitecturas de software complejas.
- Un patrón nomina, abstrae e identifica los aspectos clave de una estructura de diseño común, lo que los hace útiles para crear un diseño arquitectónico reutilizable.
- Los patrones de diseño identifican las clases e instancias participantes, sus roles y colaboraciones, y la distribución de responsabilidades.

- Cada patrón se centra en un problema concreto, describiendo cuándo aplicarlo y si tiene sentido hacerlo tomando en cuenta otras restricciones de diseño, así como las consecuencias y las ventajas e inconvenientes de su uso.

Categoría de patrones

Buschmann en [Buschmann1996] agrupa los patrones en tres categorías [Casales2011]:

- ***Patrón arquitectónico*** el cual expresa un esquema de organización de manera estructural y que es fundamental para un sistema de software. Además incluye un cúmulo predefinido de subsistemas, en cada subsistema se especifican sus responsabilidades, y se incluyen sus reglas y directrices que ayuden a la organización de las relaciones entre ellos. Este tipo de patrones son un modelo para arquitecturas de software concretas, en los cuales se especifican las propiedades estructurales de todo el sistema, y tienen un impacto en los subsistemas de la arquitectura. La selección de un patrón arquitectónico es por consiguiente, una decisión fundamental de diseño en el desarrollo de un sistema de software.
- ***Patrón de diseño*** proporciona un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Dentro de estos patrones se da una descripción de una estructura recurrente de los componentes de comunicación, en el que resuelve un problema de diseño general en un contexto particular. La aplicación de un patrón de diseño no tiene efecto en la estructura fundamental del sistema de software, pero puede tener una fuerte influencia en la arquitectura.
- ***Modismos*** se concentran en la implementación de cuestiones específicas de diseño. Un modismo es un patrón de bajo nivel específico para un lenguaje de programación, maneja aspectos de diseño e implementación. Describe cómo implementar aspectos particulares de componentes así como las relaciones entre ellos haciendo uso de las características de un lenguaje de programación determinado.

Por lo tanto, al comenzar con el diseño arquitectónico general del sistema podemos hacer uso de los *patrones arquitectónicos,* después saltan a escena los *patrones de diseño*

los cuales como su nombre lo indica, serán utilizados durante toda la fase de diseño de la arquitectura, y finalmente los *modismos* que se ocuparan durante toda la fase de implementación.

3.3 Patrón Arquitectónico Modelo-Vista-Controlador (MVC)

El patrón arquitectónico MVC separa la interfaz de usuario de la lógica de control y de los datos de la aplicación [Bass2012].

Contexto: En un software, la interfaz de usuario suele ser la parte que frecuentemente sufre mayores modificaciones, por esta razón, es muy importante separar en el software la interfaz de usuario del resto del sistema.

Problema: ¿Cómo se puede mantener separada la interfaz de usuario de la funcionalidad de la aplicación y aún así ser sensible a las entradas del usuario, y a los cambios en los datos de la aplicación?, ¿Cómo se pueden crear múltiples "vistas" de la interfaz de usuario, además de mantener y coordinar los cambios que sucedan en los datos de la aplicación?

Solución: El patrón MVC separa la funcionalidad de la aplicación en tres tipos de componentes:

- Un "Modelo" el cual contiene una representación de los datos y la lógica de la aplicación. Es aquí donde se manipulan los datos y las reglas del negocio asociadas al sistema.
- Una "Vista" que muestra una representación de una parte de los datos e interactúa con el usuario. Aquí viven las distintas interfaces de usuario que ayudan al sistema a comunicarse con el exterior y a que el usuario manipule los datos.
- Un "Controlador" que sirve de mediador entre el Modelo y la Vista, además de gestionar las acciones del usuario y las modificaciones de los cambios de estado del sistema.

Es posible que existan muchas "vistas" y "controladores" asociados con un "modelo". Por ejemplo, que se quieran representar diferentes vistas de un mismo conjunto de datos, y éstas vistas se podrán actualizar de manera dinámica a medida que se modifica el modelo. O también un modelo puede ser actualizado por diferentes controladores, por ejemplo, los

movimientos y clics del ratón, el texto o números introducidos a través del teclado, o comandos de voz; cada una de estas diferentes formas de entrada deberá ser gestionada por un controlador diferente.

Debido a que estos tres componentes (Modelo, Vista y Controlador) están debidamente acoplados, es relativamente fácil el desarrollarlos y probarlos, además de dar un plus, el cual consiste en que si se llegara a tener que realizar algún cambio el impacto es mínimo en los demás. La figura 3.1 muestra las relaciones entre los componentes del patrón MVC.

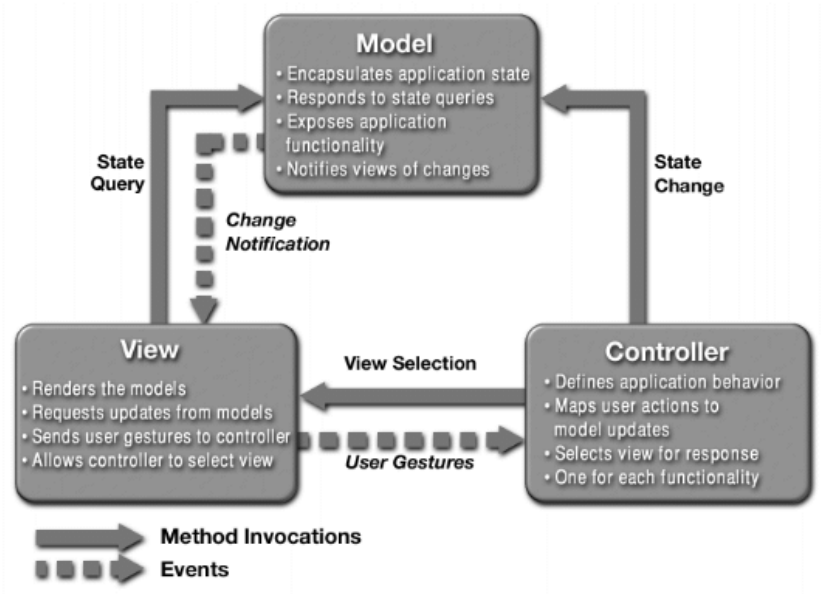


Figura 3.1 Patrón Modelo-Vista-Controlador (Imagen obtenida de [ObjSoft2013])

3.4 Patrón DAO (Data Access Object)

El patrón DAO se encarga de realizar las conexiones entre el sistema y la base de datos, es decir, es un patrón de diseño que enlaza el componente "Modelo" (tomando como ejemplo que se esté trabajando bajo un patrón arquitectónico MVC) con la base de datos. En este caso, el Modelo interactúa con el patrón DAO, el DAO juega el rol de una capa de persistencia de datos, que se encarga de la interacción entre el sistema y la base de datos.

Como podemos observar, al incluir el patrón DAO dentro del patrón arquitectónico MVC, le estamos sumando una capa más, la capa de persistencia, esta capa de persistencia es de gran utilidad, ya que hace más flexible el cambiar de manejador de base de datos

dentro de nuestro sistema.

¿Cómo crea el patrón DAO la capa de persistencia, o mejor dicho, cómo separa la lógica del negocio (Modelo) y la capa de persistencia?

DAO se encarga exclusivamente del acceso a la base de datos, por lo tanto, cuando el Modelo requiere interactuar con la base de datos, interactúa con el patrón DAO, y este a su vez con una API (Aplication Programming Interface). La API del DAO se encarga de guardar, leer, actualizar y borrar datos en la base de datos.

Hablando en términos de programación, el DAO consiste básicamente en una clase que es la que interactúa con la base de datos. Los métodos de esta clase dependen de la aplicación y de lo que se requiera hacer. Pero generalmente se implementan los métodos CRUD (Create, Read, Update, Delete) para realizar las "4 operaciones básicas" de una base de datos [JavaM2013].

La forma en que el patrón DAO realiza el transporte de información entre la base de datos y el Modelo y viceversa, es a través de un objeto llamado DTO (Data Transfer Object) o también llamado VO (Value Object).

Las ventajas de este patrón son:

- Facilitar la migración
- Organizar todos los accesos a datos en una capa separada
- Centralizar el control de acceso a datos
- Transparencia en el acceso a datos

La estructura del patrón DAO se divide en cuatro objetos, como se muestran en la figura 3.2. La descripción de la estructura del patrón DAO es la siguiente [Oracle2013]:

- **BusinessObject**: Representa los datos del cliente. Es el objeto que quiere acceder a la fuente de datos para poder almacenar o consultar datos.
- **DataAccessObject**: Es el objeto principal de este patrón. Abstrae al BusinessObject de los detalles del acceso a la fuente de datos.
- **DataSourse**: Representa una aplicación de origen de datos, es decir, la implementación de la fuente de datos en sí.

- **TransferObject (ValueObject)**: Representa una transferencia de objetos utilizados como soporte de datos. Es un objeto intermedio entre el BusinessObject y el DataAcessObject.

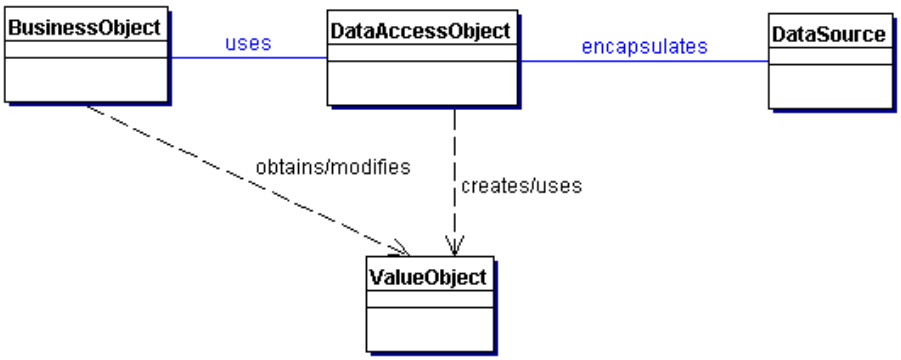


Figura 3.2: Estructura del patrón DAO (Imagen obtenida de [Oracle2013]).

3.5 Frameworks

Según Johnson [Johnson1990] un *framework* es una aplicación semi-completa y reutilizable que puede especializarse, aplicando ciertos mecanismos de extensión o de configuración, para producir aplicaciones específicas, y que incluye a menudo una infraestructura de ejecución. Un *framework* es un marco de trabajo en un sistema (subsistema) informático parcialmente completo que pretende ser una instancia. Define la estructura para una familia de sistemas (subsistemas) y provee los bloques de construcción básicos para crearlas.

Los principales beneficios de los frameworks son [Pastor2002][CSE2013]:

- **Modularidad**: Los frameworks favorecen la modularidad mediante el encapsulamiento de los detalles de implementación detrás de las interfaces estables.
- **Reusabilidad**: Mediante la definición de componentes genéricos que pueden ser utilizados para crear nuevas aplicaciones. La reutilización de los componentes de un framework puede mejorar sustancialmente la productividad y los atributos de calidad del software.
- **Extensibilidad**: Proporcionando mecanismo explícitos que permitan a las aplicaciones extender sus interfaces estables (*hook methods*). La extensibilidad del

framework es esencial para asegurar el rápido desarrollo de nuevas aplicaciones, características y servicios.

- **Inversión de control**: La arquitectura run-time de un framework se caracteriza por una inversión de control. Cuando ocurre un evento, el *dispatcher* del framework reacciona invocando *hook methods* de manejadores pre-registrados, que realizan un procesamiento del evento específico de la aplicación. La inversión de control permite al framework (en lugar de las aplicaciones) determinar el conjunto de métodos específicos de la aplicación que deben invocarse cuando ocurre un evento.

Desventajas del uso de frameworks

Los frameworks son el máximo exponente de la reutilización en las tecnologías orientadas a objetos. Sin embargo, presentan también una serie de problemas o inconvenientes [Pastor2020]:

- Un framework debe ser lo suficientemente sencillo para ser aprendido y utilizado, y al mismo tiempo debe proporcionar las suficientes características y prestaciones para ser usado eficientemente.
- Como los frameworks inevitablemente evolucionan, las aplicaciones que los usan deben evolucionar junto con ellos.
- La validación y la corrección de defectos se hace más difícil. Los componentes genéricos son difíciles de evaluar en abstracto.
- Por último, los distintos frameworks suelen ser incompatibles entre sí, de forma que optar por uno significa ligarse a un cierto tipo de tecnología.

Capítulo 4. Captura de Requerimientos y Diseño de la Arquitectura de Software

Introducción

La oportuna captura de los requerimientos, proporcionan la base la comenzar a vislumbrar cómo será el diseño de la arquitectura de software, además de delimitar el alcance de la arquitectura y del sistema en sí.

4.1 Requerimientos de Software

De las muchas definiciones que existen para requerimiento, a continuación se presentan algunas [Casales2011]:

- La IEEE menciona que un "Requerimiento" es una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo.
- Un "Requerimiento" es una condición o capacidad que debe estar presente en un sistema o componentes de un sistema para satisfacer un contrato, estándar, especificación u otro documento formal.
- Un "Requerimiento" es una representación documentada de una condición o capacidad.

Los requerimientos son la pieza fundamental en un proyecto de desarrollo de software, estos requerimientos son capturados en la fase más temprana del desarrollo, son el fundamento del ciclo de vida del proyecto. La documentación de los requerimientos, es el mejor camino para comenzar a realizar la documentación de un sistema. De ahí su importancia que deban de ser definidos y manejados de la forma más adecuada posible.

4.2 Características de los Requerimientos de Software

Las características de un requerimiento son sus propiedades principales. Un conjunto de requerimientos en estado de madurez, deben presentar una serie de características tanto individualmente como en grupo.

A continuación se presentan las más importantes [Casales2011][Galeon2014]:

- **Necesario:** Un requerimiento es necesario si su omisión provoca una deficiencia en el sistema a construir, y además su capacidad, características físicas o factor de calidad no pueden ser reemplazados por otras capacidades del producto o del proceso.
- **Conciso:** Un requerimiento es conciso si es fácil de leer y entender. Su redacción debe de ser simple y clara para aquellos que vayan a consultarlo en el futuro.
- **Completo:** Un requerimiento está completo si no necesita ampliar detalles en su redacción, es decir, si se proporciona la información suficiente para su comprensión.
- **Consistente:** Un requerimiento es consistente si no es contradictorio con otro requerimiento.
- **No ambiguo:** Un requerimiento no es ambiguo cuando tiene una sola interpretación.
- **Verificable:** Un requerimiento es verificable cuando puede ser cuantificado de manera que permita hacer uso de los siguientes métodos de verificación: inspección, análisis, demostración o pruebas.

Tradicionalmente, los atributos o requerimientos técnicos de un sistema se han clasificado en dos categorías: requerimientos funcionales y requerimientos no funcionales.

Requerimientos Funcionales. Atributos del sistema observables en tiempo de ejecución.

Los Requerimientos Funcionales definen las funciones y el comportamiento básico que el sistema será capaz de realizar, es decir, soportan las metas, tareas y actividades que requiere el usuario. Describen las transformaciones que el sistema realiza sobre las entradas para producir salidas.

Dentro de los requerimientos funcionales existe un sub-conjunto que se considera arquitectónicamente significativo o que influye de alguna forma en la arquitectura, a este grupo se le llama **Requerimientos Funcionales Primarios.** Estos requerimientos son seleccionados por ser esenciales para el funcionamiento del sistema, por su importancia para el negocio y porque afectan de alguna manera la arquitectura del sistema que se va a desarrollar (interactúan con muchos elementos arquitectónicos), dejando de lado todos aquellos que no la afectan.

Requerimientos No Funcionales. Atributos del sistema no observables en tiempo de ejecución.

Los *Requerimientos no funcionales* son aquellos que no se refieren directamente a las funciones específicas que entrega el sistema, es decir, no describen un comportamiento del sistema, sino las propiedades emergentes o limitaciones de éste. La gran mayoría de este tipo de requerimientos, hacen referencia al sistema como si se tratará de un todo, más no a rasgos particulares del mismo. Mientras que el incumplimiento de los requerimientos funcionales degradará el sistema, una falla en un requerimiento no funcional del sistema lo inutilizará.

Drivers Arquitectónicos

Se les llama "Drivers Arquitectónicos" al conjunto compuesto por los requerimientos funcionales (Casos de uso), los atributos de calidad y las restricciones. La identificación de estos Drivers Arquitectónicos ayuda al arquitecto para comenzar a realizar el diseño de la arquitectura de software.

4.3 El Método QAW (Quality Attribute Workshops, Taller de Atributos de Calidad)

Entrevistar a los stakeholders más relevantes es la forma más fácil y segura de aprender lo que saben y necesitan, es necesario un proyecto para capturar esta información crítica para el sistema, de una manera sistemática, clara y repetible. La recopilación de esta información por parte de los stakeholders se puede lograr por muchos métodos, uno de ellos es el "*Taller de Atributos de Calidad*" (QAW, Quality Attribute Workshops).

Los resultados de las entrevistas realizadas a los diferentes stakeholders incluye una lista priorizada de los Drivers Arquitectónicos y un conjunto de los escenarios de los atributos de calidad descubiertos por el arquitecto o dichos explícitamente por los stakeholders. Toda esta información puede ser utilizada para realizar lo siguiente:

- Refinar los requerimientos y el sistema de software a desarrollar.
- Comprender y aclarar los Drivers Arquitectónicos del sistema.
- Proporcionar las justificaciones necesarias del por qué el arquitecto tomará ciertas decisiones de diseño.

- Guiar el desarrollo de los prototipos y las simulaciones.
- Influir en el orden en el que se desarrollará la arquitectura.

El QAW es un método facilitador entre los stakeholders y el arquitecto, para generar, priorizar y refinar los escenarios de los atributos de calidad antes de que se comience con el diseño de la arquitectura de software. Este método se centra en las preocupaciones por parte de los stakeholders y más específicamente en las diferentes funciones que esperan que realice el sistema, por ende, el QAW depende profundamente de la participación de los distintos stakeholders [Bass2012].

El QAW tiene los siguientes pasos:

Paso 1: Realizar una presentación e introducción del método QAW.

Los facilitadores del QAW describen la motivación de realizarlo y explican cada paso del método. En este paso, cada participante del QAW deberá describirse a sí mismo, indicando brevemente sus antecedentes, su rol dentro de la organización y la relación que tiene con el sistema que está siendo construido.

Paso 2: Presentación de la misión y objetivos del negocio de la empresa.

Los stakeholders presentaran las inquietudes y preocupaciones del negocio, es decir, el contexto empresarial, que estará detrás de la construcción del sistema. Así como también los requisitos funcionales generales, las limitaciones y los requerimientos de atributos de calidad. Los atributos de calidad expuestos en este paso se derivan en gran parte de las necesidades y misión del negocio, éstos se perfeccionarán en los pasos posteriores.

Paso 3: Presentación del Plan arquitectónico.

Si bien un detallado de la arquitectura de software del sistema podría no existir, es posible una descripción general del sistema, la cual puede contener texto y dibujos con el fin de describir los diferentes artefactos técnicos del sistema. En este punto del QAW, el arquitecto presentará sus diagramas de la arquitectura del sistema tal y como se encuentren

en ese momento, esto permitirá a los stakeholders conocer el actual pensamiento arquitectónico.

Paso 4: Identificación de los Drivers Arquitectónicos.

Los facilitadores compartirán su lista de Drivers Arquitectónicos clave reunidos en los pasos 2 y 3, con la finalidad de que los stakeholders los califiquen, los prioricen, agreguen o eliminen Drivers a la lista, o hasta se realicen correcciones a los ya presentados. La idea es llegar a un consenso sobre una lista refinada de los Drivers Arquitectónicos, es decir, que incluya los requerimientos generales, las restricciones y los atributos de calidad.

Paso 5: Lluvia de ideas de posibles escenarios.

Cada stakeholder expresará un escenario que represente sus inquietudes con respecto al sistema. Los facilitadores deberán asegurarse que cada escenario tenga un estímulo y una respuesta explícita. Además, los facilitadores deberán verificar que exista al menos un escenario representativo para cada uno de los Drivers Arquitectónicos que aparecen en la lista del paso 4.

Paso 6: Consolidación de los escenarios.

Después de la lluvia de ideas para cada escenario, se consolidarán los escenarios similares, haciéndolo de una manera razonable. Para esto, los facilitadores piden ayuda a los stakeholders, para identificar a los escenario que sean muy similares en su contenido. Los escenarios que sean similares en su contenido se fusionarán, siempre y cuando los stakeholders que los propusieron estén de acuerdo y crean que sus escenarios no se diluirán en el proceso. La consolidación de escenarios ayuda a prevenir que los stakeholders vayan a distribuir sus votos en varios escenarios que expresen algo similar.

Paso 7: Priorización de escenarios.

La priorización de los escenarios se logra mediante la asignación de los votos que dé cada stakeholder, con un 30% de votos del total del número de escenarios generados después de la consolidación. Los stakeholders pueden dar cualquier número de sus votos a

cualquier escenario o combinación de escenarios. Finalmente, se cuentan los votos y los escenarios son priorizados.

Paso 8: Refinamiento de los escenarios.

Después de la priorización, los escenarios son refinados y elaborados. Los facilitadores ayudan a los stakeholders a plasmar los escenarios en unas plantillas que contienen las seis partes necesarias para crear un escenario (Estímulo, Fuente del estímulo, Respuesta, Medida de la respuesta, Medio ambiente y El artefacto).

4.4 Diseñando una Arquitectura

En los párrafos anteriores ya se ha hablado de los pasos a seguir para capturar los requerimientos de software más significativos, el capturar los atributos de calidad, así como elegir a los principales que servirán de guía en el diseño. Ahora hablaremos acerca de cómo utilizar esos Drivers Arquitectónicos para diseñar una arquitectura de software. El SEI propone el uso del método ADD (por sus siglas en inglés, Atributte-Driven Design), o traducido al español sería, Diseño Guiado por Atributos.

Estrategia de Diseño

Clements y colegas en [Bass2012] presentan una idea esencial para el diseño de la arquitectura, la "descomposición de los requerimientos arquitectónicos", esta descomposición ayudara al arquitecto a generar y probar los bosquejos de la arquitectura que se generen, lo cual es muy útil para el método ADD.

Descomposición de los requerimientos arquitectónicos

Los atributos de calidad determinan las propiedades que contendrá la arquitectura de software de un sistema. Es posible ver a los atributos de calidad de un sistema como un todo, de este modo, si se desea diseñar pensando en el éxito de los atributos de calidad, es necesario comenzar tomando al sistema en su conjunto. Como en el diseño es posible descomponer en partes ese conjunto, los requerimientos de atributos de calidad también se pueden descomponer asignándolos a los elementos de nuestra descomposición.

El hecho de tener una estrategia de descomposición no significa que el realizar un diseño sea fácil, o que no hay restricciones de utilizar determinados componentes desarrollados previamente. El arquitecto debe tener siempre presentes las restricciones que se tengan, de tal manera que se ordene la descomposición, para que ésta se adapte a las restricciones. Debemos de entender entonces, que el objetivo principal de la actividad del diseño es el de generar un diseño que se adapte a las limitaciones y al mismo tiempo lograr la calidad y los objetivos del negocio para el sistema.

Una descomposición habitual de una arquitectura, es una descomposición de un componente en componentes y conectores (C&C). Por ejemplo, si se implementa el patrón MVC, se descompone la arquitectura en un número de componentes para el "modelo", un número más de componentes para las "vistas", y otras más para el "controlador".

Diseño según los requerimientos arquitectónicos importantes.

Los requerimientos arquitectónicos importantes "ASR" por sus siglas en inglés (Architecturally Significant Requirements), son los requerimientos que impulsan el diseño arquitectónico, es por eso que son importantes y significativos; el hecho de que conduzcan el diseño arquitectónico significa que estos requerimientos tienen un profundo efecto en la arquitectura, en otras palabras se debe diseñar para satisfacer estos requerimientos [Bass2012].

4.5 El Método ADD (Attribute-Driven Design, Diseño Guiado por Atributos)

El método ADD es un método iterativo que envuelve a las estrategias. Y en cada iteración ayuda al arquitecto a hacer lo siguiente:

- Elegir una parte del sistema a diseñar.
- Elegir los ASR´s y Drivers Arquitectónicos importantes para la parte del sistema seleccionada para diseñar.
- Crear y probar el diseño para esa parte.

Las primeras salidas de las iteraciones de método ADD no entregan una arquitectura completa para cada uno de sus detalles, el método entrega una arquitectura completa en la

N iteración, y se sabrá que es la última iteración del método en el momento en que se ven satisfechos por completo todos y cada uno de los Drivers Arquitectónicos contenidos en la lista que se creó en el método QAW. Este método provee de una temprana y rápida división del trabajo que puede repartirse a los diferentes equipos dentro del proyecto para que puedan comenzar su trabajo, mientras el arquitecto sigue con su labor realizando las demás iteraciones del método y refinando la arquitectura.

4.5.1 Entradas del método ADD

Antes de realizar un proceso de diseño, los requerimientos funcionales, los atributos de calidad y las restricciones deben de ser conocidos. Por lo tanto, el método ADD puede iniciar después de concretado el método QAW, o dicho de diferente forma, puede comenzar cuando se conoce un conjunto importante de requerimientos arquitectónicos. Esto aumenta la importancia de contar con un correcto conjunto de ASR´s y Drivers Arquitectónicos. Si éste conjunto cambia después de iniciado el diseño, entonces el diseño puede necesitar ser reelaborado. Aunque todos los ASR´s no pueden ser conocidos a priori, los requerimientos de atributos de calidad son un excelente comienzo. Además de los ASR´s, las entradas para el ADD deben incluir una descripción, esta descripción proporciona dos piezas vitales para el diseñador:

1. *Conocer los límites del sistema que está siendo diseñado.* Tanto lo que está fuera como dentro del sistema que está siendo diseñado debe ser conocido, con el fin de limitar el problema y establecer el alcance de la arquitectura que se está realizando.
2. *Conocer los sistemas externos, los dispositivos, los usuarios y las condiciones del medio donde se desenvolverá el sistema, con los cuales tendrá interacción*. Es posible que no se conozcan todos los agentes externos al sistema con los que tendrá interacción, pero la descripción deberá citar algunos supuestos que pueden suceder, aunque sus detalles aún no se conozcan.

4.5.2 Salidas del método ADD

La salida del ADD es un conjunto de dibujos y diagramas de diferentes vistas arquitectónicas, la conjunción de las vistas permitirá identificar una colección de elementos arquitectónicos y de sus relaciones o interacciones. Cada una de las vistas será una vista de

una descomposición de un modulo, y cada vista contendrá elementos con responsabilidades.

Las interacciones de los elementos se describen en términos de la información que pasa entre los elementos, esta descripción bosqueja pero no detalla las especificaciones de las interfaces, ni da los parámetros de los tipos de información o les da nombre a los "métodos" ocupados al momento de programar.

Cuando al fin el método llegue a su final, se tendrá una arquitectura plena y estará documentada a través de un conjunto de vistas. Después se podrán refinar las vistas, quizás fusionando algunas de ellas, en la medida que el proyecto lo requiera.

4.5.3 Pasos del método ADD

El método ADD tiene cinco pasos [Bass2012]:

Paso 1: Elegir un elemento del sistema para diseñar

El ADD comienza con una parte del sistema que aún no ha sido diseñada y se empieza a trabajar en ella. Para un diseño totalmente nuevo, es decir que aún no se ha diseñado nada, el "elemento" para empezar es siempre todo el sistema. La primera iteración del método ADD entregará un diseño amplio y poco profundo que contendrá un conjunto de nuevos elementos arquitectónicos identificados y sus interacciones. Estos elementos muy probablemente requerirán de nuevas decisiones de diseño para profundizar en lo qué hacen y cómo lo hacen a fin de satisfacer los ASR´s y/o Drivers Arquitectónicos que se les asignen, y durante las siguientes iteraciones esos elementos de la primera iteración se convertirán en "candidatos" para ser trabajados, y descompuestos en elementos cada vez más detallados, y así sucesivamente.

Paso 2: Identificar los ASR´s y los Drivers Arquitectónicos para el elemento seleccionado

Un método bastante bueno para identificar los ASR´s y los Drivers Arquitectónicos asociados al elemento a diseñar es el construir un "árbol de utilidad", este árbol de utilidad sirve de guía para los stakeholders en la priorización de los requerimientos de atributos de

calidad. Los dos factores que se utilizan para la priorización son el valor del negocio y el impacto que tendrá sobre la arquitectura, lo bueno es que es muy poco probable que el valor del negocio, es decir las metas de la empresa, cambien a lo largo del proceso de diseño, esto quiere decir que el arquitecto ya no tendrá que preocuparse por reconsiderarlos más adelante.

Paso 3: Generar una solución de diseño para el elemento seleccionado

Este paso es el corazón del método ADD, pues en él se lleva a cabo la generación y prueba de los bosquejos arquitectónicos. Cuando se comienza este paso, ya se tiene un elemento elegido para diseñar y una lista de ASR´s y Drivers Arquitectónicos que se le pueden aplicar a ese elemento. Para cada ASR y Driver Arquitectónico se desarrollara una solución de diseño. Es muy probable que se realicen diseños que abarquen a varios ASR´s y también a varios Drivers Arquitectónicos, esto se debe a que se esta construyendo en la medida de los problemas a resolver, por tanto si se encuentra con una diseño que abarque y satisfaga a varios ASR´s y Drivers Arquitectónicos, será mucho mejor para la arquitectura que se está creando. Las decisiones de diseño tomadas en este paso se convertirán en restricciones para los futuros pasos e iteraciones del método.

Paso 4: Verificar y refinar los requerimientos restantes y, generar y seleccionar los que servirán de entrada para la siguiente iteración del método

Es posible que la solución de diseño generada en el paso 3, no se ajuste a todos los ASR´s y Drivers Arquitectónicos, es por eso que en este paso 4 se aplican más pruebas al diseño del elemento elegido en el paso 1, y entonces nos encontraremos con la posibilidad de tener que retroceder en nuestro diseño, porqué un Driver Arquitectónico o un ASR no se vio satisfecho por completo; en este caso será necesario rediseñar.

Paso 5: Repetir los pasos 1 – 4 hasta que todos los Drivers Arquitectónicos y los ASR´s hayan sido satisfechos

Después de realizar los cuatro pasos anteriores, nos encontramos con que cada elemento diseñado tiene un conjunto de responsabilidades, además de que también se les han

asignado un conjunto de requerimientos de atributos de calidad y de restricciones. Está claro que si todo lo anterior se cumple, es decir, si se ven totalmente satisfechos todos los ASR´s y Drivers Arquitectónicos, entonces habrá terminado el método ADD.

Además de lo anterior, otro punto de vista a considerar para detener las iteraciones del método ADD, es qué tanto se conoce al equipo de desarrollo y programación, ya que si se sabe que el equipo de desarrollo no necesita unos diseños demasiado detallados, se podrán tener bosquejos menos minuciosos, pero de lo contrario, si no se conoce al equipo de desarrollo o si se sabe que ellos necesitan el máximo detalle posible, entonces será necesario realizar diagramas que no dejen la menor duda de lo que se esta plasmando en ellos.

Podría darse el caso, de poner fin abruptamente al método ADD, cuando el presupuesto para la parte del diseño del proyecto se ha agotado.

Árbol de Utilidad (Utility Tree)

Un Árbol de Utilidad es un esquema en forma de árbol, que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno [Bass2012].

La intención del árbol de utilidad es la identificación de los atributos de calidad más importantes para un proyecto. No existe un conjunto preestablecido de atributos, sino que son definidos por los stakeholders en el desarrollo del sistema al momento de la construcción del árbol.

El árbol de utilidad contiene como nodo raíz la *utilidad general* del sistema. Los atributos de calidad asociados al mismo conforman el segundo nivel del árbol, los cuales se refinan hasta la obtención de un escenario lo suficientemente concreto para ser analizado y otorgarle prioridad a cada atributo considerado.

Cada atributo de calidad perteneciente al árbol contiene una serie de escenarios relacionados, y una escala de importancia y dificultad asociada, que será útil para efectos de la evaluación de la arquitectura.

4.6 Documentando la Arquitectura de Software

Incluso la mejor arquitectura, la más perfecta para un proyecto de desarrollo de software, será inútil si las personas que la deben utilizar no la comprenden, o no la logran entender lo suficiente como para poder usarla y construir el software, o hasta en el peor de los casos malinterpretar los diagramas e implementarlos de forma incorrecta. Y todo el esfuerzo por parte de los stakeholders y del arquitecto en los métodos anteriores habrá sido en vano y se tendrá muchísimo tiempo desperdiciado.

El diseño de la arquitectura debe ser comunicado de una manera adecuada y de distintas formas, acorde a cada tipo de stakeholder. La documentación es la portadora de las ideas y pensamiento del arquitecto, y por ende, la que habla por él en su ausencia, ya sea por que esté atendiendo otros proyectos o porque ya no se encuentre dentro de la organización.

Todo buen arquitecto de software debe producir una buena documentación no porque sea “necesario”, sino porque es esencial para tener un producto de alta calidad. Además de que la documentación es la representación física del software.

Usos para la documentación de la arquitectura de software

Las diferentes personas encargadas del desarrollo tienen la esperanza que la documentación de la arquitectura les ayude a realizar su trabajo. El que el arquitecto comprenda los diferentes usos que se le darán a la arquitectura, determinará que tipo de información estará plasmada en las diferentes vistas. Fundamentalmente, la documentación de la arquitectura tiene tres usos:

1. *La documentación de la arquitectura sirve como un medio de educación.* El uso educativo consiste en introducir a las personas en un sistema, estas personas pueden ser nuevos miembros que se integren al equipo, analistas externos, o incluso un nuevo arquitecto. Estos nuevos arquitectos están interesados en aprender cómo sus predecesores abordaron las cuestiones difíciles del sistema y por qué se tomaron ciertas decisiones. En muchas otras ocasiones, esa “nueva” persona es el cliente al que se le tiene que mostrar la solución del sistema, este cliente espera que la documentación le ayude a comprender el funcionamiento del software.

2. *La documentación de la arquitectura sirve como el vehículo principal para la comunicación entre los stakeholders.* La documentación arquitectónica es precisamente usada como el medio para realizar una comunicación exitosa. Quizás en muchos casos sea el mismo arquitecto el más interesado en la documentación de la arquitectura, pues le ayudará a recordar cuales fueron sus decisiones y por qué fueron tomadas, además de ayudarlo con los futuros cambios.
3. *La documentación sirve de base para analizar el sistema y su construcción.* La documentación arquitectónica le dice a los implementadores lo que deben de hacer. La documentación no sólo proporciona instrucciones acerca de lo que se tiene que realizar, sino que también ayuda a determinar con qué otros equipos de desarrollo habrá que comunicarse para completar las interfaces de los elementos que se encuentran en la arquitectura.

Notaciones para la documentación de la arquitectura

Las notaciones a usar en la documentación difieren considerablemente en el grado de formalidad que se requiera. En términos generales, existen tres categorías principales de notación:

- *Notación informal*. Las vistas se representan haciendo uso de diagramas de propósito general y herramientas de edición poco convencionales. La semántica de la descripción de este tipo de documentación se caracteriza por tener un lenguaje natural, y que no pude ser analizada formalmente.
- *Notación semi-formal*. Las vistas son expresadas en una notación estándar, con elementos gráficos y normas de construcción, que proveen un tratamiento semántico semi-completo. Ejemplo de este tipo de notación es el estándar UML.
- *Notación formal*. En este tipo de notación, las vistas se describen con una precisión semántica, por lo general tienen una base matemática. En esta notación es posible realizar un análisis sintáctico y semántico. Generalmente a estas notaciones se les conoce como ADLs o Lenguajes de Descripción Arquitectónica, que suelen ofrecer un vocabulario gráfico y una semántica reservada para la representación de la arquitectura. La utilidad de los ADLs radica principalmente en la capacidad que

tienen para apoyar la automatización a través de herramientas asociadas para la generación de código.

Para determinar qué tipo de notación se debe usar implica hacer varios acuerdos con los stakeholders. Típicamente entre más formales sean las vistas, más tiempo y esfuerzo se deberá dedicarles para crearlas y entenderlas. Independientemente del tipo de formalidad que se elija, el arquitecto siempre deberá recordar que las diferentes notaciones expresan diferentes tipos de información, y que la información que se necesite dependerá exclusivamente de la clase de stakeholders con que se este trabajando.

4.7 Vistas Arquitectónicas

En [Bass2012] Clements y colegas, señalan que el concepto más importante asociado con la documentación de un arquitectura es el de la "Vista". Una arquitectura de software es una entidad compleja que no se puede describir de una manera simple. Una "Vista" es una representación de un conjunto de elementos del sistema y de las relaciones entre ellos, no necesariamente una vista debe contener todos los elementos del sistema, sino que puede contener un tipo particular de esos elementos. El concepto de "Vista" proporciona un principio fundamental para la documentación arquitectónica, el cual es que una arquitectura es una cuestión de documentación de las diferentes "Vistas" o perspectivas de una misma arquitectura. El SEI propone el método "Views and Beyond, Vistas y más allá" para realizar éstas vistas arquitectónicas. Las vistas son conducidas por las necesidades de la documentación, además de que cada vista está compuesta por componentes y conectores (C&C) de acuerdo a la funcionalidad del sistema; sincronización y comunicación de procesos, distribución física, propiedades estáticas o dinámicas y propiedades de ejecución entre otras.

Bushmann [Buschmann1996] establece que una *vista arquitectónica* representa un aspecto parcial de una arquitectura de software, que muestra propiedades específicas del sistema. Por su parte, Kruchten define una vista arquitectónica como una descripción simplificada o abstracción de un sistema desde una perspectiva específica, que cubre intereses particulares y omite entidades no relevantes a esta perspectiva [Kruchten2004].

Para la definición de una vista, Kruchten propone la identificación de ciertos elementos, que se mencionan a continuación:

- Punto de vista: involucrados e intereses de los mismos.
- Elementos que serán capturados y representados en la vista y las relaciones entre estos.
- Principios para organizar la vista.
- Forma en que se relacionan los elementos de una vista con otras vistas.
- Proceso a ser utilizado para la creación de la vista.

Kruchten [Kruchten2004], Bass, Clements y Klein [Bass2012], Clements, Kazman y Klein [Clements2002] y Hofmeister [Hofmeister2000], proponen, en función de las características del sistema o del proceso de desarrollo del mismo, distintas vistas arquitectónicas. Es importante resaltar que las vistas propuestas no son independientes entre sí, puesto que son perspectivas distintas de un mismo sistema [Kruchten2004]. Por tal motivo, las vistas arquitectónicas deben estar coordinadas, de tal manera que al realizar cambios, estos se vean correctamente reflejados en las vistas afectadas, garantizando consistencia entre las mismas.

De acuerdo al nivel de responsabilidad dentro del desarrollo de un sistema y la relación que se establezca con el mismo, son muchas las partes involucradas e interesadas en la arquitectura de software [Kruchten2004], estas son:

- El analista del sistema, quien la utiliza para organizar y expresar claramente los requerimientos y entender las restricciones de tecnología y los riesgos.
- Usuarios finales y clientes, que necesitan conocer el sistema que están adquiriendo.
- El administrador del proyecto, que la utiliza para organizar el equipo y planificar el desarrollo.
- Los diseñadores, que lo utilizan para entender los principios subyacentes y localizar los límites de su propio diseño.
- Otras organizaciones desarrolladoras (si el sistema es abierto), que la utilizan para entender cómo interactuar con el sistema.

- Las compañías subcontratadas, que la utilizan para entender los límites de su sección de desarrollo.
- Los arquitectos, quienes velan por la evolución del sistema y la reutilización de componentes.

Todas estas personas deben comunicarse de manera efectiva para discutir y razonar acerca de la arquitectura, y así alcanzar las metas del desarrollo.

Una única representación de la arquitectura del sistema resultaría demasiado compleja y poco útil para todos los involucrados, pues contendría mucha información irrelevante para la mayoría de estos. Es por ello que se plantea la necesidad de representaciones distintas que contengan únicamente elementos que resulten de importancia para cierto grupo de personas involucradas en el sistema.

Combinando Vistas

El principio básico de la documentación de una arquitectura es tomarla como un conjunto de "vistas", donde cada una de estas vistas representa a la misma arquitectura o secciones de la misma arquitectura pero desde una perspectiva distinta, cuidando siempre que cada una de las vistas se asocie con las demás, para poder así entender el sistema en su conjunto.

En ocasiones, la forma más conveniente de mostrar una fuerte asociación entre dos vistas es encapsulándola en una sola vista combinada, esta vista combinada es una vista que contiene elementos y relaciones que surgen a partir de dos o más puntos de vista. La forma más fácil de combinar vistas es crear una "superposición", o dicho de otra manera, el poner una vista sobre la otra, de modo que se combine la información que de otro modo habría quedado en dos vistas separadas. Esto funciona si el acoplamiento entre las dos vistas es fuerte, es decir, que exista una estrecha relación entre los elementos de una vista y los elementos de la otra vista, el resultado de esta combinación nos entregará una sola vista que será más fácil de entender que si las dos vistas se vieran por separado.

4.8 El Método V&B (Views and Beyond, Vistas y más allá)

La arquitectura es una herramienta de comunicación que permite explicar a los interesados (stakeholders) las decisiones de diseño y las consecuencias de estas decisiones. En particular, la arquitectura es la guía para la implementación y tiene que ser comunicada efectivamente a los desarrolladores.

La documentación de la arquitectura es creada para facilitar la comunicación entre los stakeholders y suele planear las iteraciones, se vuelve especialmente importante cuando hay muchas personas envueltas en el desarrollo del sistema de software, cuando la persona que implementa el sistema no es la misma que creó la arquitectura, o cuando los equipos de desarrollo están distribuidos geográficamente.

Otro motivo importante para realizar la documentación es para realizar el mantenimiento del producto, generalmente al realizarse se involucran a personas nuevas, los desarrolladores originales se encuentran en otros proyectos o dejaron la empresa de desarrollo, etcétera. Al realizar la documentación se facilita enormemente el realizar el mantenimiento.

4.9 El Modelo 4 + 1 Vistas

El modelo 4+1 vistas (figura 4.1) de Kruchten define cuatro estructuras a considerar al momento de realizar la documentación de todo sistema de software [Kruchten1995] y, una quinta vista (+1) la cual tiene como propósito relacionar las cuatro vistas anteriores. Cada una de estas vistas muestra aspectos distintos de la arquitectura de software. A continuación, se explica qué documenta cada una de estas vistas [Pastor2002]:

- **Vista Física.** En esta vista se detallan las relaciones que existen entre el software y el hardware, además de la forma en que se encuentran distribuidos. Es decir, en esta vista se describe cómo el software se ha mapeado sobre el hardware del sistema.
- **Vista Lógica.** Describe los requerimientos funcionales del sistema. Pudiendo hacer uso de distintos diagramas, entre los cuales se encuentran los diagramas de comunicación y/o secuencia. Dicho de otra manera, dentro de esta vista se describen los servicios que deben proporcionar los componentes del sistema, ya sea a otros componentes, o bien a un usuario externo.

- **Vista de Procesos.** En esta vista de procesos, como su nombre lo indica, se describe el comportamiento dinámico del sistema y sus procesos. Cubre aspectos tales como el paralelismo, la concurrencia y la sincronización de tareas, entre otros.
- **Vista de Despliegue.** Describe la organización estática del software en su entorno de desarrollo. El software se agrupa en módulos o subsistemas. En esta vista se definen las relaciones de uso (importación y exportación de servicios) entre los diferentes módulos del sistema. Esta vista es muy importante para el Project Manager en la planificación del proyecto, ya que es la más usada para la asignación de trabajo a los diferentes equipos de desarrollo, en esta vista se hace uso de los diagramas de componentes y de paquetes.
- **Vista de escenarios.** Los escenarios, son la representación de los diferentes casos de uso más significativos del sistema a desarrollar y es entonces la vista que *une*a las otras diferentes vistas del sistema, ya que muestra al sistema como un todo. La vista de escenarios tiene dos propósitos:
  1. Servir de herramienta para descubrir elementos arquitectónicos (componentes y relaciones).
  2. Validar la arquitectura. Los mismos casos de uso que se han usado para el diseño, aunque más elaborados, sirven de trazas para validar el sistema en sus diferentes fases de desarrollo hasta llegar al producto final.

Figura 4.1: El modelo 4+1 vistas de la arquitectura de software de Kruchten (Imagen obtenida de [KruchtenJarr2013]).

Ahora que ya se ha visto lo que se documenta cada una de las vistas que propone Kruchten, podemos utilizar este modelo para realizar la documentación de la arquitectura propuesta en esta tesis. Y al mismo tiempo darle la documentación necesaria al método V&B que propone el SEI, ya que desde mi punto de vista, el método de Kruchten puede mapearse al método V&B, en el cual se trata de expresar a la arquitectura con distintos diagramas, enfocados para el entendimiento de los diferentes stakeholders.

4.10 Evaluación Arquitectónica

La evaluación arquitectónica es el proceso para determinar si una arquitectura es apta para el propósito para el cual está destinada. La arquitectura es fundamental para conseguir el éxito de un proyecto de ingeniería de software, por tal motivo, tiene mucho sentido el hacer una pausa y asegurarse de que el diseño de la arquitectura será capaz de ofrecer todo lo que se espera de ella, y es ese el rol que juega la evaluación.

Factores de Evaluación

Generalmente la evaluación arquitectónica toma una de las tres formas siguientes:

1. Evaluación realizada por el diseñador en el proceso de diseño

Cada vez que el diseñador toma unas decisiones de diseño o completa el diseño, las elecciones y las alternativas tomadas deben ser evaluadas, por ejemplo haciendo uso de los pasos del método ADD, descritos ya en los párrafos anteriores. En todos los análisis que se realicen, se debe tener presente que es una cuestión de costo-beneficio, es decir, no se debe gastar demasiado tiempo en la toma de decisiones a menos que sean las que conducirán la construcción de todo el diseño. Algunas consideraciones que debieran tomarse incluyen:

- *La importancia de la decisión.* Cuanto más importante es la decisión, más atención se le debe prestar para asegurar que la decisión está bien tomada.
- *El número de alternativas posibles*. Si se tienen demasiadas alternativas, más tiempo llevará el evaluar cada una de ellas. Se debe utilizar el sentido común y la experiencia del arquitecto para eliminar algunas alternativas rápidamente, de tal manera, que el número de posibles alternativas sea pequeño.

- *Alternativas bastante buenas como para estar en oposición.* Muchas veces, las posibles alternativas no difieren radicalmente en sus consecuencias, es este caso, es más importante el optar rápidamente por algunas de las opciones y seguir adelante con el proceso de diseño, en lugar de dedicarles demasiado tiempo en revisarlas.

2. Evaluación entre pares en el proceso de diseño

Los diseños arquitectónicos también pueden ser revisados entre pares. La revisión entre pares se puede llevar a cabo en cualquier punto del diseño de la arquitectura, o al menos en el momento en que exista una parte que sea coherente con lo que se está buscando. Debe haber un tiempo fijo, medido en horas, y que no sea más de la mitad de un día. La revisión entre pares tiene varios pasos:

1. **Los revisores determinan una serie de escenarios de atributos de calidad para guiar la revisión.** Muchas de las veces estos escenarios serán requerimientos arquitectónicos importantes. Estos escenarios pueden ser desarrollados por el equipo revisor o por los stakeholders.
2. **El arquitecto presenta una porción de la arquitectura para ser evaluada.** Puede que en este punto aún no se tenga una documentación completa. Los revisores deberán asegurar individualmente que comprenden la arquitectura, para poder evaluar la porción de la arquitectura a revisar.
3. **Para cada escenario, el diseñador servirá de guía a través de su arquitectura y explicará cómo el escenario es satisfecho.** Los revisores realizaran preguntas para determinar dos tipos de información. Primero, qué se busca para determinar que el escenario está satisfecho. Segundo, determinar si algunos de los otros escenarios que se están considerando pueden no ser satisfechos a causa de las decisiones tomadas en la parte de la arquitectura que se está revisando.
4. **Capturar problemas potenciales.** La lista de los problemas potenciales es la base para el seguimiento de la revisión. Si el problema potencial es un problema real, la decisión tomada por el arquitecto no debe ser siempre fija, por más explicita que ésta haya sido, siempre hay que estar atentos a su probabilidad de ocurrencia.

3. Evaluación y análisis por agentes externos, una vez que la arquitectura ha sido diseñada.

Los evaluadores externos pueden hacer una revisión imparcial y objetiva sobre una arquitectura. Externos es una palabra subjetiva, ya que se puede tratar de revisores externos al proyecto de desarrollo, pero que estén dentro de la misma organización. En la medida en que los evaluadores sean más "externos", será más probable que tengan menos *miedo*, por así decirlo, de plantear problemas sensibles, o de decir problemas que no son tan evidentes en los ojos de la organización.

A menudo, son elegidos por las organizaciones los revisores externos, porque poseen conocimientos o una larga experiencia en la evaluación de arquitecturas exitosas.

4.11 El Método ATAM (Método de Análisis y Acuerdos Arquitectónicos)

El Método de Análisis y Acuerdos Arquitectónicos "ATAM" por sus siglas en inglés (The Architecture Tradeoff Analysis Method) se ha utilizado por más de una década en ámbitos que van desde la automotivación hasta diseños que son del ejercito y financieros [Bass2012]. El método ATAM está diseñado para que los evaluadores no necesiten estar familiarizados con la arquitectura o con los objetivos del negocio, además de que el sistema no necesita estar todavía construido, y puede haber un gran número de stakeholders.

4.11.1 Participantes del método ATAM

El método ATAM necesita la participación y mutua cooperación de tres grupos:

1. *El equipo evaluador.* Este grupo es externo al proyecto, cuya arquitectura está siendo evaluada. Por lo general se compone de tres o cinco personas, a cada una de ellas se le asigna un número de funciones o roles específicos para jugar durante la evaluación (La Tabla 4.1 presenta una descripción de éstos roles, junto con un conjunto de características deseables de cada uno de ellos). Una sola persona puede adoptar varios roles en la ejecución del ATAM.
2. *Las personas que toman las decisiones del proyecto.* Estas personas tienen la facultad y el poder de hablar en nombre del proyecto de desarrollo y por ende, pueden exigir cambios en él. Dentro de este grupo de personas, es común que se encuentre el administrador del proyecto, directivos de la empresa y los

patrocinadores del proyecto. El arquitecto siempre se incluye, es una regla fundamental de la evaluación que el arquitecto debe participar "voluntariamente".

3. *Los demás stakeholders de la arquitectura.* Los stakeholders tienen un interés personal en la realización de la arquitectura y del software en sí. En este grupo se incluyen a los desarrolladores, probadores (testers), integradores, ingenieros de software, usuarios y cualquier otra persona que interactúe con el sistema. Su trabajo durante la evaluación es relacionar los objetivos específicos con los atributos de calidad que la arquitectura deberá cumplir para que el sistema pueda considerarse de éxito. A diferencia de los otros dos grupos, este conjunto de personas no participan en toda la ejecución del método.

Rol	Responsabilidades
Líder del Equipo	Estructurar la evaluación Coordina la evaluación con el cliente Hace que se vean realizadas las necesidades del cliente Establece el contrato de evaluación Integra al equipo de evaluación Se encarga de que se produzca un informe al finalizar la evaluación
Líder de Evaluación	Ejecuta las evaluaciones Facilita la obtención de los escenarios Administra los proceso de selección de escenarios (priorización) Facilita la evaluación de los escenarios contra la arquitectura Facilita el análisis on-site
Escritor de escenarios	Escribe los escenario durante la obtención de los mismos Realiza una redacción acorde con los escenarios Detiene la discusión hasta que tiene una redacción exacta del escenario
Escritor de procedimientos	Captura los procedimientos en formato electrónico Captura la esencia y la solución esperada de los escenarios, que a menudo se pierde en la formulación y construcción de los mismos
Interlocutor / Interrogador	Plantea cuestionamientos de interés arquitectónico y los relaciona con los atributos de calidad

Tabla 4.1. Roles en la ejecución del método ATAM

4.11.2 Salidas del método ATAM

Al igual que en cualquier proceso de prueba, un gran beneficio se deriva de su preparación. Una parte de esta preparación la ejercen *los tomadores de decisiones*, quienes deben de planificar lo siguiente:

1. *Una preparación concisa de la arquitectura.* Uno de los requisitos del método ATAM es que la arquitectura sea presentada en más o menos 60 minutos, por lo que la presentación deberá ser concisa y comprensible.
2. *La estructuración de los objetivos del negocio.* Los objetivos del negocio que se presentan en el ATAM son vistos por primera vez por algunos de los participantes de la reunión. Esta descripción de los objetivos del negocio sobreviven a la evaluación y pasan a formar parte del legado del proyecto. El ATAM utiliza una priorización de los escenarios de los atributos de calidad como base para realizar la evaluación arquitectónica, estos escenario son los mismos que se realizaron en el método QAW.
3. *Priorización de los requerimientos de atributos de calidad expresados como escenarios de atributos de calidad.* Estos escenarios también sobreviven más allá de la evaluación y se pueden usar para guiar la evolución de la arquitectura.
4. *Un conjunto de riegos y no-riesgos.* En el método ATAM, se define un riesgo como una decisión arquitectónica que puede conducir a consecuencias no deseadas de los requerimientos de atributos de calidad. Del mismo modo, un no-riesgo es una decisión arquitectónica que, en el análisis, se considera seguro. Los riesgos identificados servirán como insumos para un plan de mitigación de riesgos.
5. *Un conjunto de temas de riesgo.* Cuando el análisis se ha completado, el equipo de evaluación examina el conjunto completo de riesgos descubiertos, para buscar cuales son los temas generales que podrían identificar deficiencias en la arquitectura o en el proceso de desarrollo de la misma, o inclusive en el equipo. Estos temas de riesgo identificados, amenazan los objetivos del negocio del proyecto.
6. *Mapeo de las decisiones arquitectónicas a los requerimientos de calidad.* Las decisiones arquitectónicas se pueden interpretar en términos de las cualidades que apoyen u obstaculicen los requerimientos de calidad. Para cada escenario de atributo

de calidad examinado durante el ATAM, las decisiones arquitectónicas son determinadas y capturadas, a su vez, estas acciones pueden servirle al arquitecto como una declaración que justifique sus decisiones.

7. *Un conjunto de los puntos sensibles y de acuerdos.* Estas son decisiones arquitectónicas que tienen un marcado efecto en uno o más de los atributos de calidad.

Las salidas del método ATAM son utilizadas para construir un informe final que resuma el método, se resumen las actuaciones, se capturan los escenarios así como sus análisis, y se catalogan los hallazgos. Al término de la evaluación existe una relación palpable entre todos los stakeholders, y una mejor comprensión global de la arquitectura, así como de sus fortalezas y debilidades.

4.11.3 Fases del método ATAM

Las actividades en la evaluación están repartidas en cuatro fases:

Fase 0: Coordinación y Preparación.

El equipo de evaluación y los tomadores de decisiones se reúnen de manera informal para trabajar en los detalles de la ejecución del método. Los representantes del proyecto hacen una breve evaluación acerca de qué personas deberán estar presentes en la ejecución del método. También se ponen de acuerdo acerca de la hora y lugar de la reunión principal, así como de las personas que recibirán el informe final.

El equipo de evaluación examina la documentación de la arquitectura, así como el diseño de los principales enfoques arquitectónicos. Por último, el líder del equipo de evaluación explica qué información se espera que presenten tanto el administrador del proyecto como el arquitecto, además de ayudar en la construcción de las presentaciones.

Fase 1 y 2: Son las fases de evaluación.

En estas fases todo el mundo se da a la tarea de analizar la documentación de la arquitectura, comprendiendo todos los enfoques tomados y los atributos de calidad.

Durante la fase 1, el equipo de evaluación se reúne con los responsables de las decisiones del proyecto, para comenzar la recopilación y análisis de la información.

En la fase 2, los stakeholders reúnen los procedimientos realizados durante la fase de diseño de la arquitectura y continúan con el análisis.

Fase 3: Seguimiento

En esta fase, el equipo de evaluación elabora y entrega un informe final por escrito. En primer lugar, se distribuye a los principales stakeholders, asegurándose de que el informe no contenga elementos ambiguos que dificulten su comprensión. Finalmente este informe de evaluación se entrega a la persona encargada de haber realizado la evaluación del sistema (en caso de que sean agentes totalmente externos a la organización).

La tabla 4.2 presenta las cuatro fases del ATAM, qué participantes existen en cada fase y un tiempo estimado para cada fase.

Fase	Actividad	Participantes	Duración Aproximada
0	Coordinación y Preparación	El equipo de evaluación y los tomadores de decisiones del proyecto	Las reuniones informales requieran tal vez más de un par de semanas
1	Evaluación	El equipo de evaluación y los tomadores de decisiones del proyecto	1-2 días seguidos de una pausa de 1-2 semanas
2	Evaluación (continuación)	El equipo de evaluación, los tomadores de decisiones del proyecto y los stakeholders	2 días
3	Seguimiento	El equipo de evaluación y el cliente	1 semana

Tabla 4.2: Fases del método ATAM

4.11.4 Pasos de las fases del método ATAM

Las fases de análisis del método ATAM (fases 1 y 2) consisten en nueve pasos. Los primeros seis pasos son llevados a cabo en la fase 1, con el equipo de evaluación y los tomadores de decisiones del proyecto. En la fase 2, con todos los stakeholders presentes, los pasos 1-6 son resumidos y los pasos 7-9 se llevan a cabo.

Paso 1: Presentación del ATAM

En este primer paso, el líder de evaluación realiza una breve presentación a los representantes del proyecto acerca de los pasos del ATAM y de los resultados que entregará. Explicará el proceso que todos seguirán, se explicará el contexto y las expectativas de cada una de las actividades a realizar, y finalizará respondiendo a preguntas que surjan en ese momento.

Paso 2: Presentación de los objetivos del negocio

Los representantes del proyecto y los miembros del equipo de evaluación, necesitan conocer el contexto del sistema, además de los principales objetivos del negocio que motivaron su desarrollo. Por tal motivo, el director del proyecto o la persona que representa al cliente del sistema, realiza una presentación, la cual contiene una perspectiva general del sistema en el contexto empresarial. La presentación deberá describir lo siguiente:

- Las funciones más importantes del sistema
- Todas las restricciones importantes (técnicas, económicas, políticas o de gestión)
- Los objetivos del negocio y cómo se relacionan con el proyecto
- A los principales stakeholders
- Los requerimientos de gran importancia que se esperan cumpla la arquitectura

Paso 3: Presentación de la arquitectura

Aquí, el arquitecto realiza una presentación de la arquitectura con un "nivel apropiado" de detalle, este nivel apropiado dependerá del tiempo disponible y del grado de detalle requerido por los stakeholders para la presentación. La presentación deberá cubrir las restricciones técnicas y los enfoques arquitectónicos utilizados por él para la creación de la arquitectura.

El arquitecto deberá tener la astucia para llevar la presentación de un modo que se transmita la esencia de la arquitectura sin desviarse o profundizar demasiado en unos pocos detalles, pero nombrando explícitamente cuáles han sido los patrones y las tácticas utilizadas.

Paso 4: Identificar los enfoques arquitectónicos

El ATAM se centra en el análisis de la arquitectura y el entendimiento de los enfoques arquitectónicos, y hasta este momento todos los presentes tienen ya una buena idea de qué patrones y tácticas ha utilizado el arquitecto para el diseño del sistema.

En este paso, el equipo de evaluación cataloga los patrones y las tácticas que han sido identificadas. Cualquiera de los personas que fungen como "escribas" captura en una lista esos patrones y tácticas, para así hacerlas públicas, además de servir de base para su posterior análisis.

Paso 5: Generación del Árbol de Utilidad con los atributos de calidad

En este paso, los objetivos de los atributos de calidad y los escenarios de estos atributos sirven de insumos para la creación de un "Árbol de Utilidad", en el cual se organizan, priorizan y se refinan aún más estos escenarios. También es posible el retomar el árbol de utilidad generado en el paso 5 del método QAW, y realizarle las modificaciones necesarias para que contenga los nuevos escenarios de atributos de calidad propuestos por los stakeholders.

Paso 6: Análisis de los enfoques arquitectónicos

En esta parte, el equipo de evaluación se encarga de analizar uno a uno los escenarios, iniciando con los más prioritarios, tal cual como aparecen en el árbol de utilidad. Se le pide al arquitecto que explique como la arquitectura soporta o da solución a cada uno de los escenarios. A lo largo de este paso, el equipo de evaluación identifica y documenta las decisiones arquitectónicas más relevantes, cataloga los riesgos, los no-riesgos, los puntos sensibles y los acuerdos arquitectónicos. El objetivo de este paso, es que el equipo de evaluación sea convencido de que los módulos y los elementos de la arquitectura tienen un enfoque adecuado y cumplen con todos los requerimientos y con los atributos de calidad. Con este paso, la fase 1 llega a su fin.

Inicio de la fase 2

El equipo de evaluación resume todo lo que se ha visto y aprendido acerca de la arquitectura, e interactúa y se comunica con el arquitecto de una manera “informal” (videoconferencia o por teléfono) durante un par de semanas. Algunos escenarios más podrían ser analizados durante este tiempo, e incluso el arquitecto puede resolver algunas preguntas que surjan hasta este momento.

En esta fase dos, el primer paso es recordarles a los stakeholders el método ATAM y los roles de cada uno dentro de la ejecución del método, el líder de evaluación resume los resultados de los pasos 2-6 y comparte la lista de las decisiones arquitectónicas, los riesgos, los no-riesgos, los puntos sensibles y los acuerdos arquitectónicos.

Paso 7: Lluvia de ideas y priorización de escenarios

En este paso, es posible tomar los escenarios realizados en el paso 5 del método QAW, refinarlos y priorizarlos, además todos los presentes pueden generar nuevos escenarios si se cree que son necesarios, e incluirlos en el árbol de utilidad. O bien, es ocupado el árbol de utilidad generado en el paso 5 de éste método, para entender cómo percibe el arquitecto los atributos de calidad dentro de la arquitectura. El propósito de la lluvia de ideas, es entender lo que significa que el sistema sea exitoso para cada uno de los stakeholders.

Paso 8: Nuevo Análisis de los enfoques arquitectónicos

En el paso 6 de éste método, ya se ha realizado un análisis de lo enfoques arquitectónicos; pero ahora se realiza de nuevo después de que los escenarios del paso 7 ya han sido colectados y priorizados, realizando las mismas actividades del paso 6. En este paso el equipo de evaluación guía al arquitecto en el proceso de analizar los escenarios de más alto rango. El arquitecto explica la relevancia de las decisiones arquitectónicas que contribuyen a la realización de los requerimientos y de los atributos de calidad. Lo ideal es que esta actividad de análisis estuviera dominada por las explicaciones del arquitecto, acerca de los escenarios en términos de los enfoques arquitectónicos.

Paso 9: Presentación de los resultados

En este paso, la información obtenida a través de los 8 pasos anteriores es resumida y presentada a los stakeholders. El líder de evaluación recapitula los pasos del ATAM y lo obtenido en cada uno de ellos, incluido el contexto del negocio, los requerimientos, las restricciones y los Drivers de la arquitectura. Por lo tanto, la presentación de los resultados incluye lo siguiente:

1. Documentación de los enfoques arquitectónicos
2. Un conjunto de escenarios ya priorizados en base a la lluvia de ideas
3. El árbol de utilidad
4. La documentación de los riesgos y no-riesgos descubiertos
5. Los puntos de sensibilidad encontrados y los acuerdos obtenidos
6. Los temas de riesgo basados en los dos puntos anteriores, y cómo éstos amenazan a los objetivos del negocio

La tabla 4.3 muestra en resumen el método ATAM, sus fases y sus pasos, así como una breve descripción de cada uno.

Como se puede observar, para realizar todos los pasos del método ATAM es necesario invertirle una gran cantidad de tiempo, que sólo tiene sentido en un proyecto grande y costoso, donde el riesgo de cometer un error importante en la arquitectura es inaceptable.

Por ese motivo, el SEI propone una manera “ligera”, por así decirlo, de ejecutar el método ATAM, la cual puede ser factible para realizarse en proyectos más pequeños. Esta evaluación ligera del método ATAM puede ser llevada a cabo en un solo día o incluso en medio día. Se ejecuta en su totalidad por miembros internos de la organización, la contra es que este análisis de la arquitectura no se realizará tan profundamente, pero ésta es una cuestión de costo/beneficio. En [Bass2012] se da una propuesta de calendario para las fases 1 y 2, la cual se muestra en la tabla 4.4.

En la evaluación ligera, no existe ningún informe final como pasa en la ejecución “normal” del ATAM, pero en este tipo de evaluación también existe un “escriba” responsable de la captura de los resultados, que luego podrán ser distribuidos y servirán de base para remediar los riesgos. Los resultados obtenidos dependerán de lo bien que el

equipo reunido haya entendido los objetivos del método, las técnicas del método y el sistema mismo. Al ser interno el equipo de evaluación le resta un poco de objetividad e imparcialidad al método, y esto puede poner en peligro el valor de los resultados. Pero como ya se dijo, esta versión ligera, es barata, fácil de convocar y relativamente informal, por lo que se puede hacer de una forma rápida cada vez que se requiera una garantía mínima de que la arquitectura es "buena".

Fases	Participantes	Tiempo Estimado	Pasos	Descripción
Fase 0: Coordinación y preparación	El equipo de evaluación y los tomadores de decisiones del proyecto	2 semanas	Paso 1: Presentación del ATAM	El líder de evaluación realiza una breve presentación a los representantes del proyecto acerca de los pasos del ATAM y de los resultados que entregará.
Fase 1: Evaluación	El equipo de evaluación y los tomadores de decisiones del proyecto	1-2 días, seguidos de una pausa de 1-2 semanas	Paso 2: Presentación de los objetivos del negocio	El director del proyecto o la persona que representa al cliente, realizan una presentación, la cual contiene una perspectiva general del sistema en el contexto empresarial.
			Paso 3: Presentación de la arquitectura	El arquitecto realiza una presentación de la arquitectura, utilizando los diagramas y vistas realizadas.
			Paso 4: Identificar los enfoques arquitectónicos	El equipo de evaluación cataloga los patrones y las tácticas que han sido identificadas.
			Paso 5: Generación del Árbol de Utilidad con los atributos de calidad	Se organizan, priorizan y refinan los atributos de calidad y los escenarios de estos atributos para generar el árbol de utilidad.
			Paso 6: Análisis de los enfoques arquitectónicos	El equipo de evaluación analiza los escenarios, mientras que el arquitecto explica cómo la arquitectura da solución a cada uno de ellos. El equipo de evaluación identifica y documenta las decisiones arquitectónicas, incluyendo los riesgos, los no-riesgos, los puntos sensibles y los acuerdos arquitectónicos.
Fase 2: Evaluación (Continuación)	El equipo de evaluación, los tomadores de	2 días	Paso 7: Lluvia de ideas y priorización de	Se pide a los stakeholders que expresen sus ideas acerca de los escenarios que podrían suceder para probar que la arquitectura

	decisiones del proyecto y los stakeholders		escenarios	cumple con lo esperado.
			Paso 8: Nuevo análisis de los enfoques arquitectónicos	El arquitecto explica la relevancia de las decisiones arquitectónicas que contribuyen a la realización de los requerimientos y de los atributos de calidad.
Fase 3: Seguimiento	El equipo de evaluación y el cliente	1 semana	Paso 9: Presentación de los resultados	La información obtenida a través de los 8 pasos anteriores es resumida y presentada a los stakeholders. El líder de evaluación recapitula los pasos del ATAM y lo obtenido en cada uno de ellos, incluido el contexto del negocio, los requerimientos, las restricciones y los Drivers de la arquitectura.

Tabla 4.3: Resumen Método ATAM

Número de Paso	Tiempo asignado	Descripción
1: Presentación del método ATAM	0 horas	Los participantes deben estar familiarizados con el proceso de este método, por tal motivo éste paso se puede omitir.
2: Presentación de los objetivos del negocio	25 minutos	Se espera que los participantes entiendan y comprendan el sistema, los objetivos del negocio y sus prioridades.
3: Presentación de la Arquitectura	30 minutos	Se da una breve introducción de la arquitectura, se utilizan al menos dos “vistas” y uno o dos escenarios de atributos de calidad que abarquen esas vistas.
4: Identificación de los Enfoques Arquitectónicos	25 minutos	Son identificados en la arquitectura los enfoques arquitectónicos y los atributos de calidad concernientes a estos. Este paso puede estar embebido en el paso 3.
5: Generación del Árbol de Utilidad	Variable 0.5-1.5 horas	Ya debe existir un árbol de utilidad, el equipo lo revisará y si es necesario lo actualizará con nuevos escenarios, nuevas respuestas o nuevas prioridades.
6: Análisis de los Enfoques Arquitectónicos	2-3 horas	En este paso, se mapean los escenarios prioritarios con la arquitectura.
7: Lluvia de ideas y Priorización de escenarios	0 horas	Este paso puede ser omitido, pues se espera que las personas que integran esta reunión ya se hayan organizado con anterioridad, además de haber expresado sus necesidades en el paso 5.
8: Nuevo Análisis de los Enfoques Arquitectónicos	0 horas	Este paso también se puede omitir, ya que todos los análisis se realizaron en los pasos anteriores.
9: Presentación de los Resultados	0.5 horas	Al final de la evaluación, el equipo revisa los riesgos existentes y los recién descubiertos, los no-riesgos, los puntos sensibles de la arquitectura y los acuerdos arquitectónicos.
Total	4-6 horas	

Tabla 4.4: Agenda para una evaluación *ligera* ejecutando el método ATAM [Bass2012]

Capítulo 5. KUALI-BEH

5.1 Antecedentes

KUALI-BEH [Oktaba2012] ha sido desarrollada en respuesta al Llamado a la Acción (RFP) *Fundamentos para la creación y ejecución ágil de métodos de Ingeniería de Software* lanzado por el Object Management Group (OMG) [OMG2012]. KUALI-BEH es la fusión de dos palabras, KUALI que en náhuatl significa *"bueno"* y BEH que en maya significa *"camino"*. KUALI-BEH propone un *"buen camino"* para definir las prácticas y métodos para el desarrollo de software, que las propias organizaciones capturen, adapten y mejoren en función de su experiencia y madurez. KUALI-BEH es una propuesta mexicana desarrollada con base en el conocimiento de fuentes reconocidas y la experiencia en el desarrollo de estándares y modelos de referencia de procesos de desarrollo de software [Oktaba2012][OMG2012][IEEE1471].

KUALI-BEH define conceptos comunes en proyectos de software. Su objetivo es facilitar a los desarrolladores de software a representar sus métodos y prácticas. KUALI-BEH está compuesta de dos vistas: la *estática* y la *operacional*; estas dos vistas conforman el *Núcleo de conceptos comunes* en proyectos de software (figura 5.1).

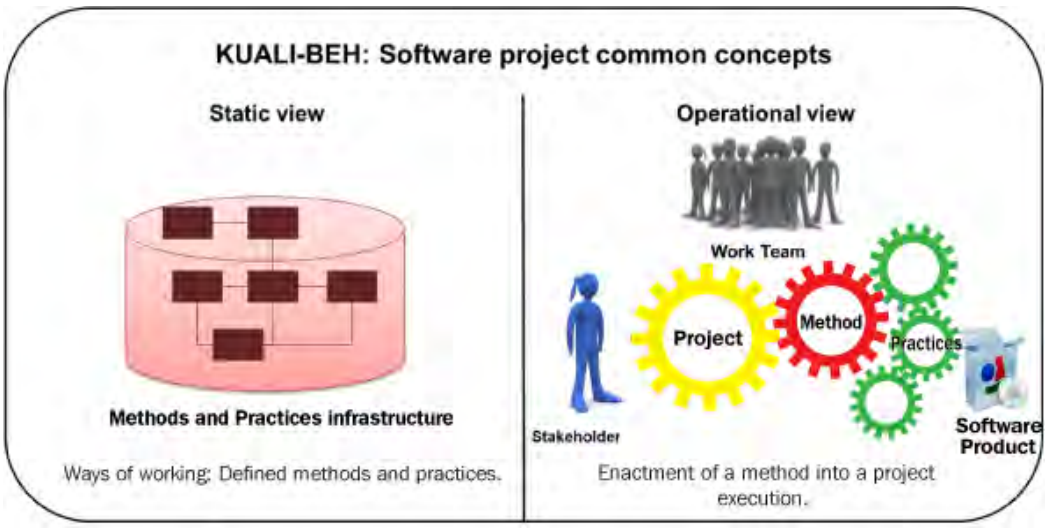


Figura 5.1: KUALI-BEH (Imagen obtenida de [Oktaba2012])

5.1.1 Vista Estática

La vista estática presenta un marco de trabajo para la definición de las distintas maneras de trabajo de los practicantes de Ingeniería de Software. Estas maneras de trabajo están organizadas como métodos compuestos por prácticas. Al conjuntar el conocimiento de cada organización, se conforma su infraestructura de métodos y prácticas que puede ser aplicado por los demás practicantes de la organización [Oktaba2012].

5.1.2 Vista Operacional

La vista operacional está relacionada con la realización de proyectos de software; provee a los equipos de trabajo de la organización, de mecanismos para ejecutar un método y adaptar las prácticas que lo conforman, a sus contextos y necesidades específicas [Oktaba2012].

5.2 Entorno Computacional de KUALI-BEH

El *Entorno Computacional de KUALI-BEH* es un proyecto llevado a cabo en el curso de Ingeniería de Software Orientada a Objetos del Posgrado en Ciencias e Ingeniería de la Computación de la UNAM (PCIC), que tiene como objetivo construir una herramienta computacional que se base en KUALI-BEH, para apoyar a las organizaciones de desarrollo de software en:

1. Expresar sus métodos y prácticas empíricas de manera estructurada.
2. Resguardar esos métodos y prácticas en un repositorio de la organización.
3. Seleccionar y adaptar al contexto de un proyecto de desarrollo de software, algún método y sus prácticas extrayéndolo del repositorio de la organización.
4. Aplicar el método y sus prácticas durante la ejecución de un proyecto, dándole seguimiento a través de tableros de control.
5. Enriquecer y mejorar el repositorio de la organización a partir de la experiencia en los proyectos y el reconocimiento adquirido de fuentes externas.

A fin de construir este *Entorno Computacional* se dividió el trabajo en cuatro proyectos de tesis cuyos objetivos son los siguientes:

1. Arquitectura de software para el *Entorno Computacional de KUALI-BEH*. Definir la arquitectura de software del *"Entorno Computacional de KUALI-BEH"* aplicando los métodos propuestos por el SEI.

2. Repositorio de métodos y prácticas de proyectos de software para KUALI-BEH [Barrera2014]. Crear un repositorio que permita administrar métodos y prácticas expresadas por practicantes de organizaciones de desarrollo de software de acuerdo a KUALI-BEH.

3. Selección y adaptación de métodos en proyectos de software aplicando KUALI-BEH [Ruíz2014]. Objetivos:

- Registrar la información relevante para conformar un proyecto de software.
- Desarrollar el entorno que permita seleccionar y adaptar un método para el contexto de un proyecto de software específico.
- Finalmente realizar la recolección de información para el cierre del proyecto de software.

4. Tableros de control para el seguimiento de proyectos de software aplicando KUALI-BEH [Urrutia2014]:

- Desarrollar tableros de control para el seguimiento de un proyecto, que permita:
- Instanciar un método para un proyecto de software.
- Reflejar en los tableros de control las operaciones de adaptación sobre el método y sus prácticas.
- Reflejar en los tableros los cambios de estado de las instancias de prácticas y del método durante la ejecución del proyecto de software.

Este trabajo de tesis trata del primer proyecto, cuyo objetivo es seguir los métodos del SEI para la construcción del diseño de la arquitectura así como también de la evaluación de la misma.

5.3 Casos de Uso del Entorno Computacional de KUALI-BEH

Para la construcción del *Entorno* se inició con la Especificación de Requerimientos. Se obtuvo el listado de los casos de uso que el *Entorno Computacional* deberá atender, los cuales son:

- Administrar Proyecto, Método, Práctica, Guía y Herramientas.
- Administrar Practicantes y Stakeholders
- Mostrar Vista Textual y Grafica de las Prácticas y Métodos
- Asociar Método y adaptarlo gráficamente
- Componer Método, Instanciar Métodos y Prácticas
- Ejecutar, Cancelar y Finalizar el Método
- Publicación de Tablero de Control de un Proyecto
- Manejo de Versiones de: Proyectos, Métodos, Prácticas, Guías y Herramientas

A partir del listado anterior de casos de uso, se realizó el modelado en el diagrama general de casos de uso (figura 5.2).

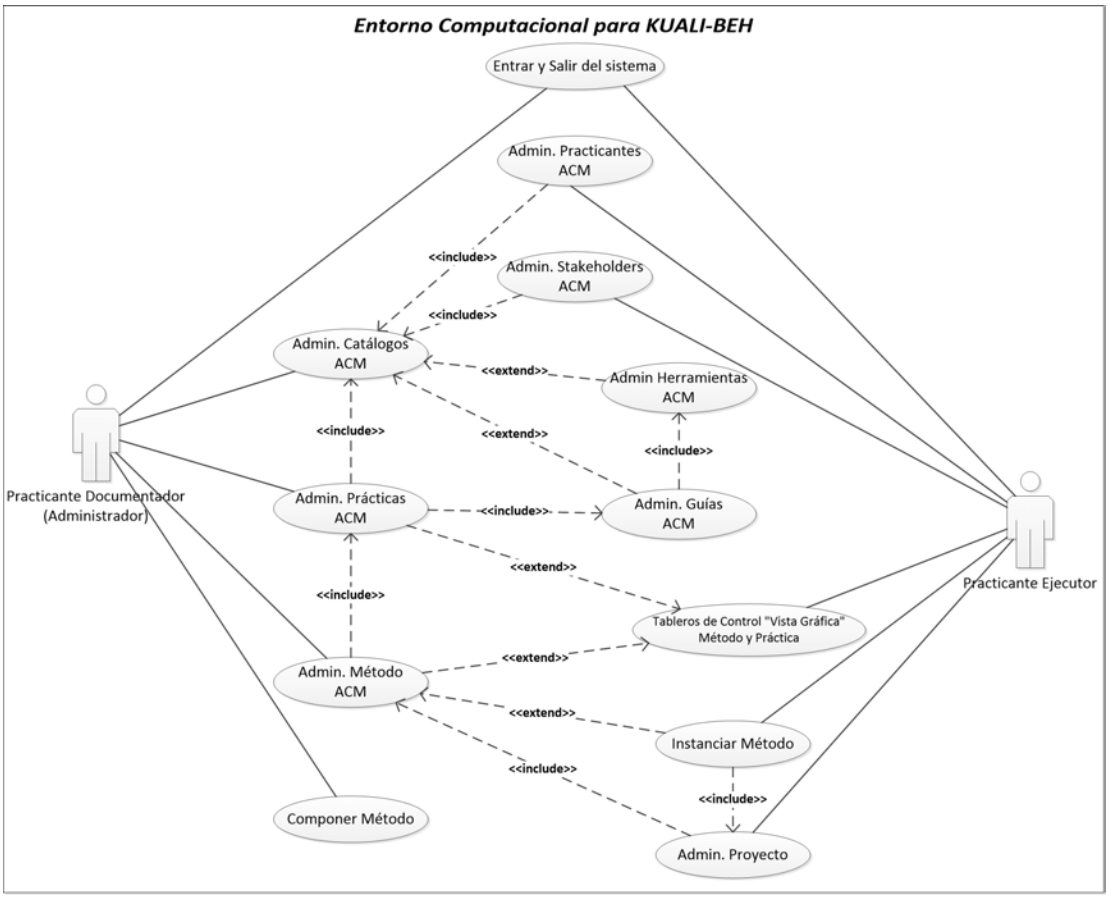


Figura 5.2. Diagrama General de Casos de Uso

Del diagrama anterior, y a modo de explicación se detalla el caso de uso "Administrar Prácticas". Para este caso de uso podemos destacar que debe ser capaz de realizar el Alta, Consulta y Modificación (ACM) de la Práctica, así como también el ACM del Nombre, Identificador, Versión, Objeto, Condiciones, Entradas/Resultados y Productos de Trabajo. Además que el administrar las prácticas, lleva consigo administrar las Guías asociadas a la Práctica, lo cual involucra a su vez el ACM de Actividades, Criterios de Verificación, Herramientas y los Conocimientos y Habilidades.

Para no dificultar la lectura del capítulo, en el apéndice I se podrá encontrar la especificación de los casos de uso mostrados en la figura 5.2.

5.4 Restricciones y Requerimientos No Funcionales

A continuación se muestran las Restricciones y los Requerimientos No Funcionales, identificados en la Fase de Especificación de Requerimientos, los cuales están asociados al *Entorno Computacional de KUALI-BEH*.

Restricciones:

- *Todas las herramientas deberán ser software libre:* Las herramientas de Software utilizadas en la construcción del *Entorno Computacional de KUALI-BEH*, deberan ser software libre.
- *Interfaz de Usuario WEB (Mozilla y Chrome):* La interfaz del sistema deberá ser soportada por los navegadores web “Mozilla” y “Chrome” en sus últimas versiones del año 2012.
- *Plataforma de Desarrollo:* La plataforma con la cual se realice la programación del *Entorno* deberá ser NetBeans IDE versión 7.3 y el lenguaje de programación a utilizar deberá ser con tecnologías basadas en JAVA.
- *Operación (SMBD, Browser, Web Server):* Esta restricción está de la mano de la primera restricción, es la cual se indica que se deberá usar sólo software libre, por tal motivo esta restricción de operación nos indica que el Sistema Manejador de Bases de Datos (SMBD) deberá ser libre (Se propone PostgreSQL), y que los navegadores soportados por el *Entorno Computacional* serán Mozilla Fire Fox y Google Chrome.

Requerimientos No funcionales:

- *Concurrente:* El *Entorno Computacional* deberá soportar a un equipo de trabajo (4 personas máximo) trabajando simultaneamente.
- *Seguridad de acceso al Entorno.* Se requiere que para poder acceder al *Entorno*, cada persona que interactua con él, deberá tener un nombre de usuario y una constraseña.
- *Interoperabilidad con otros sistemas.* Es necesario que el *Entorno* pueda tener interacción otros sistemas existentes, es decir, que se pueda obtener la información

contenida en el repositorio del *Entorno* (texto plano), para poder ser tratada por otros sistemas.

- *Facilidad de instalación.* La instalación del *Entorno Computacional* deberá ser a través de una interfaz gráfica y "amigable" para el usuario.
- *Facilidad de crecimiento (Mantenibilidad).* La arquitectura de software del *Entorno* deberá proveer cambios en el crecimiento del sistema, sin que estos cambios afecten a todos los elementos de la arquitectura.
- *Disponibilidad.* El *Entorno Computacional* deberá estar disponible los 365 días del año, y en caso de falla, que el tiempo de recuperación sea mínimo (minutos).
- *Facilidad de uso*. La interfaz de usuario del *Entorno* deberá ser intuitiva para el usuario y sin que contenga demasiados elementos que dificulten su uso.
- *Multilenguaje (español e inglés).* Se requiere que el *Entorno* se pueda instalar en cualquiera de los dos idiomas (español o inglés).

Al final del curso "Ingeniería de Software Orientada a Objetos" que forma parte del área de Ingeniería de Software del Posgrado en Ciencia e Ingeniería de la Computación (PCIC), se obtuvo un prototipo que incluía la primera iteración de la definición de la arquitectura según los métodos del SEI.

Este prototipo se realizó utilizando el patrón arquitectónico MVC, de tal manera, que se tomará como una *restricción* para la construcción del diseño de la arquitectura al momento de ejecutar los métodos propuestos por el SEI. El hecho de que se tome como una restricción, es una decisión tomada por el arquitecto de este *Entorno Computacional* y los Stakeholders, a fin de darle continuidad al trabajo ya realizado.

Capítulo 6. Definición de la Arquitectura del Entorno Computacional de KUALI-BEH

A continuación se describen los pasos seguidos para la definición de la arquitectura del *"Entorno Computacional de KUALI-BEH"* aplicando los métodos propuestos por el SEI. Adelantándonos un poco, se realizaron 2 iteraciones, las cuales se describen a continuación, describiendo cómo se aplicaron los métodos antes mencionados en cada una de las iteraciones.

6.1 Primera Iteración

En esta primera iteración sólo se ejecutaron los tres primeros métodos, es decir, los métodos QAW, ADD y V&B. La decisión de utilizar sólo estos tres primeros métodos, es porqué al finalizar la primera iteración del método ADD, aún no se tenían satisfechos todos los ASR´s y Drivers arquitectónicos, pues faltaba detallar aún más el contenido de cada uno de los paquetes realizados e identificados para el *Entorno Computacional*. Como se verá en los párrafos siguientes, en la primera iteración del método ADD se realizaron y diseñaron una serie de paquetes, y a cada paquete se le asigno una responsabilidad o varias (Casos de uso). Por tal motivo, desde la perspectiva del arquitecto, aún no era posible el realizar y ejecutar el método ATAM.

6.1.1 Ejecución del Método QAW

Recordemos que el QAW es el primero de los métodos que propone el SEI, y que en éste método es importantísima la participación de todos los stakeholders. El QAW guía al arquitecto en la obtención, priorización y refinamiento de los atributos de calidad, restricciones y casos de uso, además de que en éste método también se construyen escenarios de los atributos de calidad que fueron descubiertos por el arquitecto y manifestados por los stakeholders. Listando los pasos del método y recordando que el detalle de los mismos se encuentra en la sección 4.3 de este trabajo de tesis, se tienen:

1. Presentación e introducción del método QAW
2. Presentación de la misión y objetivos del Negocio

3. Presentación del Plan Arquitectónico
4. Identificación de los *Drivers Arquitectónicos*
5. Lluvia de ideas de los posibles escenarios
6. Consolidación de los escenarios
7. Priorización de los escenarios
8. Refinamiento de los escenarios

Es así que ejecutando los pasos listados anteriormente podemos darnos cuenta que algunos pasos ya han sido realizados, es decir, el paso 1 (Presentación e introducción del método QAW) ya se realizó en el sección 4.3, el paso 2 (Presentación de la misión y objetivos del Negocio) se encuentra en el capítulo 5 de este trabajo de tesis. Para el paso 3 (Presentación del Plan Arquitectónico) se presenta el diagrama de casos de uso (figura 5.2) con lo cual se empieza a dar una idea general de cómo se irán identificando las partes de la arquitectura, aunado a lo anterior y tomando en cuenta que uno de los objetivos del negocio es que el *Entorno Computacional de KUALI-BEH* sea una herramienta de tipo web, se toma la decisión entre el arquitecto y los desarrolladores de emplear el patrón arquitectónico Modelo-Vista-Controlador, ya que este tipo de patrones es muy utilizado en la creación de sistemas de tipo web, además de que ya se contaba con experiencia en su uso.

Para el paso 4 (Identificación de los Drivers Arquitectónicos) se toma como insumo lo obtenido en la fase de requerimientos, es decir, el listado de los casos de uso generales, las restricciones y los atributos de calidad identificados, obteniendo los *Drivers Arquitectónicos* de la arquitectura a diseñar y construir. Estos *Drivers Arquitectónicos* se muestran priorizados en la tabla 6.1.

La manera en que se realizó la priorización de los Drivers Arquitectónicos, fue por votación de los stakeholders y de las necesidades “prioritarias” por parte del cliente. Se discutieron las distintas necesidades y se llego un consenso.

Identi-ficador	Tipo de Driver	Descripción del Driver	Priori-dad
D1	Caso de Uso	Administración de Proyecto, Método, Prácticas, Guías y Herramientas	ALTA
D2	Caso de Uso	Mostar vista gráfica de la Práctica	ALTA
D3	Caso de Uso	Administrar Vista Gráfica del Método	ALTA
D4	Caso de Uso	Instanciar y Ejecutar Método	ALTA
D5	Caso de Uso	Cancelar o Finalizar Método	ALTA
D6	Caso de Uso	Publicación del Tablero de Control de un Proyecto	ALTA
D7	Restricción	Todas las herramientas deben ser Software Libre	ALTA
D8	Atributo de Calidad	Concurrente	ALTA
D9	Atributo de Calidad	Facilidad de uso	ALTA
D10	Restricción	Plataforma de desarrollo	ALTA
D11	Restricción	Operación (SMBD, Browser, Web Server)	ALTA
D12	Caso de Uso	Manejo de versiones de Prácticas, Métodos, Proyectos	ALTA
D13	Caso de Uso	Estadísticas del Proyecto	MEDIA
D14	Caso de Uso	Adaptar Método Gráficamente	MEDIA
D15	Restricción	Interfaz de usuario WEB (Mozilla y Chrome)	MEDIA
D16	Atributo de Calidad	Facilidad de crecimiento (Mantenibilidad)	MEDIA
D17	Atributo de Calidad	Disponibilidad	MEDIA
D18	Atributo de Calidad	Seguridad básica de acceso a la herramienta	BAJA
D19	Atributo de Calidad	Interoperabilidad con otros sistemas	BAJA
D20	Atributo de Calidad	Facilidad de instalación	BAJA
D21	Atributo de Calidad	Multilenguaje (Español e Inglés)	BAJA

Tabla 6.1: Drivers Arquitectónicos del *Entorno Computacional de KUALI-BEH*

En los pasos restantes del método (5, 6, 7 y 8) se realizó lo siguiente: todos los Stakeholders junto con el arquitecto dieron sus ideas acerca de cómo se imaginaban los escenarios del *Entorno* para cada uno de los *Drivers Arquitectónicos* obtenidos en el punto 4 de éste método. Tomando en cuenta todos los comentarios e ideas expresadas se consolidaron dichos escenarios, es decir, se fusionaron los escenarios que involucraban a los mismos *Drivers Arquitectónicos*, o que daban una idea de estar expresando un escenario

similar, con el fin de tener un número de escenarios que contemplará a todos los Drivers Arquitectónicos, pero que al mismo tiempo no se repitieran constantemente.

Se priorizaron los escenarios tomando la misma priorización de la tabla 6.1 de los *Drivers Arquitectónicos*. Finalmente se realizó un refinamiento de los escenarios, haciéndolos más claros y precisos en cuanto a: la fuente del estímulo, el o los artefactos involucrados, el entorno en el que se estaría llevando a cabo el escenario, así como también la respuesta esperada del sistema y la métrica utilizada para medir la respuesta. El escenario para el caso de uso "Agregar (Alta) una práctica", se encuentra en la tabla 6.2, los demás escenarios se encuentran en el apéndice II.

Escenario crudo:	*Agregar una práctica*
Objetivos de negocio correspondientes:	*El sistema soporta KUALI-BEH*
Atributos de calidad relevantes:	*Usabilidad, Funcionalidad*
Estímulo:	*Documentación de una forma de trabajo similar a google drive.* *Fase 1. Inserción individual* *Fase 2. Discusión de la información* *Fase 3. Consolidar documento final (colaborativa) y salvar*
Fuente de estímulo:	*Practicantes*
Entorno:	*Operación normal con 1 equipo (4 personas)*
Artefacto:	*Entorno Computacional de KUALI-BEH*
Respuesta:	*La forma de trabajo fue registrada (la inserción individual, la discusión y la consolidación) con la estructura de una práctica*
Medida de la respuesta:	*En dos pantallas y el tiempo de respuesta es menor a 5 segundos*

Tabla 6.2: Escenario "Agregar una práctica"

Este escenario podría fragmentarse en 2 escenarios, el primero (tabla 6.3) en el que se realice la inserción de la información en las plantillas correspondientes a una práctica, por parte de los integrantes del equipo de trabajo (inserción individual). En el segundo escenario (tabla 6.4), se realiza una discusión por parte de los integrantes del equipo acerca de lo escrito en las plantillas de la práctica y se llega a un consenso y consolidación de la información contenida en las platillas correspondientes a una práctica.

Escenario crudo:	*Agregar una práctica (Documentación individual de las plantillas)*
Objetivos de negocio correspondientes:	*El sistema soporta KUALI-BEH*
Atributos de calidad relevantes:	*Usabilidad, Funcionalidad*
Estímulo:	*Documentación de una forma de trabajo similar a google drive.* *Fase 1. Inserción individual*
Fuente de estímulo:	*Practicantes*
Entorno:	*Operación normal con 1 equipo (4 personas)*
Artefacto:	*Entorno Computacional de KUALI-BEH*
Respuesta:	*La forma de trabajo fue registrada (la inserción individual) con la estructura de una práctica*
Medida de la respuesta:	*En una pantalla para el usuario, y el tiempo de respuesta para el ingreso de la información y guardado de la misma en el repositorio del Entorno, es menor a 5 segundos*

Tabla 6.3: Escenario “Agregar una práctica (inserción individual)”

Escenario crudo:	*Agregar una práctica (Consolidación de la información de manera grupal)*
Objetivos de negocio correspondientes:	*El sistema soporta KUALI-BEH*
Atributos de calidad relevantes:	*Usabilidad, Funcionalidad*
Estímulo:	*Documentación de una forma de trabajo similar a google drive.* *Fase 1. Discusión de la información* *Fase 2. Consolidar documento final (colaborativa) y salvar*
Fuente de estímulo:	*Practicantes*
Entorno:	*Operación normal con 1 equipo (4 personas)*
Artefacto:	*Entorno Computacional de KUALI-BEH*
Respuesta:	*La forma de trabajo del equipo fue registrada con la estructura de una práctica según KUALI-BEH, después de la revisión, discusión y consolidación de la información.*
Medida de la respuesta:	*En una pantalla, en la que se muestre la información insertada por cada uno de los integrantes. En esta misma pantalla será posible guardar la información. Y el sistema deberá dar una respuesta de que se realizó exitosamente el guardado de la información. Dicha respuesta deberá ser menor a 5 segundos. (Nota: la discusión por parte del equipo acerca de la información contenida en la plantilla es variable y depende de cada equipo, por tal motivo, no se especifica el tiempo para realizar esta acción)*

Tabla 6.4: Escenario "Agregar una práctica (Consolidación de la información)"

6.1.2 Ejecución del Método ADD

Se utilizó el método *"Diseño Guiado por Atributos (ADD)"* el cual tomo como insumos los *Drivers Arquitectónicos* y los escenarios obtenidos en el método QAW. El método ADD consta de cinco pasos a través de los cuales se va construyendo el diseño de la arquitectura.

A continuación se describe lo obtenido al ejecutar cada uno de los pasos del método ADD.

Paso 1: Elegir un elemento del sistema para diseñar.

Para la primera iteración del método se consideró a todo el sistema como si se tratara de un solo elemento, para comenzar a trabajar en él y descomponerlo en fragmentos los cuales se revisarán y ocuparán en la siguiente iteración.

Paso 2: Identificar los ASR´s y *Drivers Arquitectónicos para el elemento seleccionado*.

Para este paso se tomaron en cuenta los casos de uso, las restricciones y los atributos de calidad que aparecen en la Tabla 6.1 "Drivers Arquitectónicos del *Entorno Computacional de KUALI-BEH*", y se realizó el árbol de utilidad (figura 6.9) plasmando ahí los escenarios de atributos de calidad con la finalidad de fragmentarlos en módulos o paquetes para cada uno de los Drivers Arquitectónicos y detallarlos en la siguiente iteración.

Paso 3: Generar una solución de diseño para el elemento seleccionado.

Se fragmentó el diseño del sistema en paquetes (módulos) como se muestra en la figura 6.1, en donde cada uno de los paquetes tiene una responsabilidad, esta responsabilidad o responsabilidades son asignadas tomando en cuenta los Drivers Arquitectónicos y los ASR´s del *Entorno Computacional*.

Se eligió usar el patrón arquitectónico Modelo-Vista-Controlador para comenzar a diseñar la arquitectura (figuras 6.1 a 6.4) puesto que el *Entorno* será una aplicación web, la decisión de utilizar este patrón fue que es muy socorrida para realizar aplicaciones web, además de que tanto el equipo de desarrollo como el arquitecto tienen experiencia en su uso, otra razón es que se acordó con los stakeholders que ésta sería la base de la arquitectura, tomando en cuenta los antecedentes ya mencionados. El que el *Entorno Computacional* sea una aplicación web, es una restricción, tal como aparece en la tabla 6.1 donde se listan los Drivers Arquitectónicos, el identificador para esta restricción es el número 15.

Aunque originalmente MVC fue desarrollado para aplicaciones de escritorio, ha sido ampliamente adaptado como arquitectura para diseñar e implementar aplicaciones web en los principales lenguajes de programación. Se han desarrollado multitud de frameworks que implementan este patrón; estos frameworks se diferencian básicamente en la interpretación de cómo las funciones MVC se dividen entre cliente y servidor. Se decidió que el framework ha utilizar para el desarrollo del *Entorno Computacional* sería Java Server Faces (JSF), puesto que el desarrollo se debería hacer con Java y este framework simplifica el desarrollo de interfaces de usuario.

Paso 4: Verificar y refinar los requerimientos restantes y, generar y seleccionar los que servirán de entrada para la siguiente iteración del método.

Ya que se está comenzando con las iteraciones del método, hasta este momento se ha tomado al sistema como un todo para comenzar a diseñar, se ha utilizado el patrón MVC para realizar este diseño y dentro de cada uno de los módulos (Modelo, Vista y Controlador) se ha fragmentado el sistema tomando como base los Drivers Arquitectónicos identificados.

Por tal motivo, los Drivers Arquitectónicos que servirán de insumo para la siguiente iteración, serán todos, ya que se pretende especificar y detallar el contenido de cada uno de los paquetes realizados, estos paquetes estas asociados a cada uno de los Drivers Arquitectónicos y para facilitar su identificación se les ha llamado de la misma manera que el Driver Arquitectónico al cual están asociados.

Paso 5: Repetir los pasos 1 a 4 hasta que todos los Drivers Arquitectónicos y los ASR´s hayan sido satisfechos.

Al finalizar la primera iteración del método ADD se obtuvo el primer diseño de la arquitectura, tomando como base la arquitectura Modelo-Vista-Controlador, se identificaron los principales paquetes para la interfaz de usuario, cada uno de estos paquetes tiene la responsabilidad de cumplir satisfactoriamente con el caso de uso correspondiente a ese *Driver Arquitectónico* asociado al paquete. Dado que estamos ejecutando la primera iteración del método, la forma que se utilizó para verificar que todos los Drivers

Arquitectónicos y los ASR´s hayan sido satisfechos, es revisar que cada uno de los paquetes realizados contemple uno o varios de los ASR´s y Drivers, a fin de que todos los paquetes en conjunto contemplen a todos los Drivers y ASR´s. En la siguiente iteración se continuará con el diseño y detalle del contenido de cada uno de los paquetes realizados.

A modo de conclusión para esta primera iteración del método ADD se puede ver que el diseño que se eligió para la arquitectura es patrón arquitectónico MVC y su descomposición en paquetes para cada caso de uso y cada componente del modelo, esto es, la Vista, el Modelo y el Controlador, ya que un mismo *Driver Arquitectónico* está involucrado en los tres paquetes (Modelo, Vista y Controlador), por la naturaleza del modelo arquitectónico que se está desarrollando.

6.1.3 Ejecución del Método V&B

Se utilizó el método V&B para realizar la documentación de la arquitectura de la primera iteración del método ADD, a través de diagramas de paquetes, comunicación y de distribución de UML.

En la figura 6.1 se muestra la vista de despliegue haciendo uso del diagrama de distribución para el *Entorno Computacional de KUALI-BEH*, en la figura 6.2 se muestra la vista lógica con el diagrama de paquetes de la “Vista”, en la 6.3 se muestra el diagrama de comunicación para el caso de uso “Agregar práctica”.

En el diseño arquitectónico se hace uso del patrón DAO (sección 3.4), a través del cual se realizan las conexiones del paquete del Modelo hacia la Base de Datos y viceversa. Los diagramas anteriores son vistas de los múltiples diagramas generados para este proyecto.

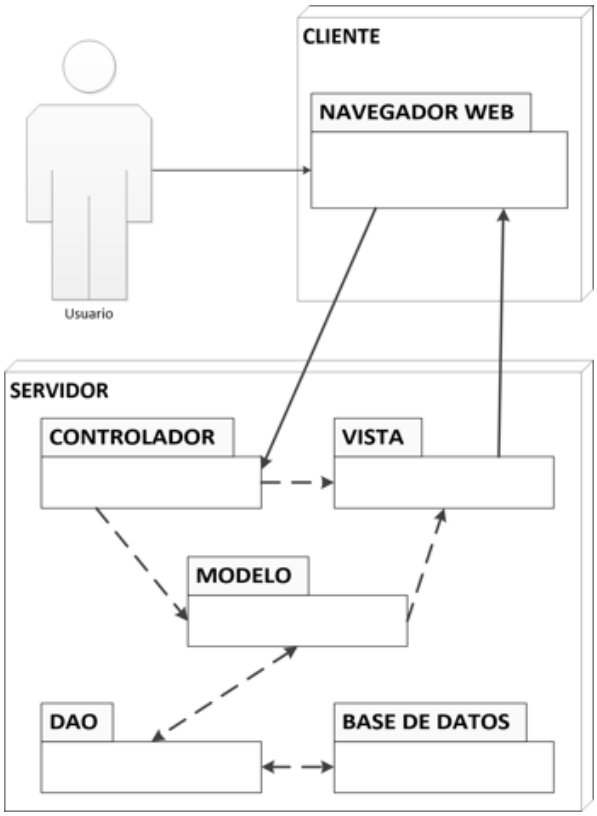


Figura 6.1. Diagrama de distribución del *Entorno Computacional*

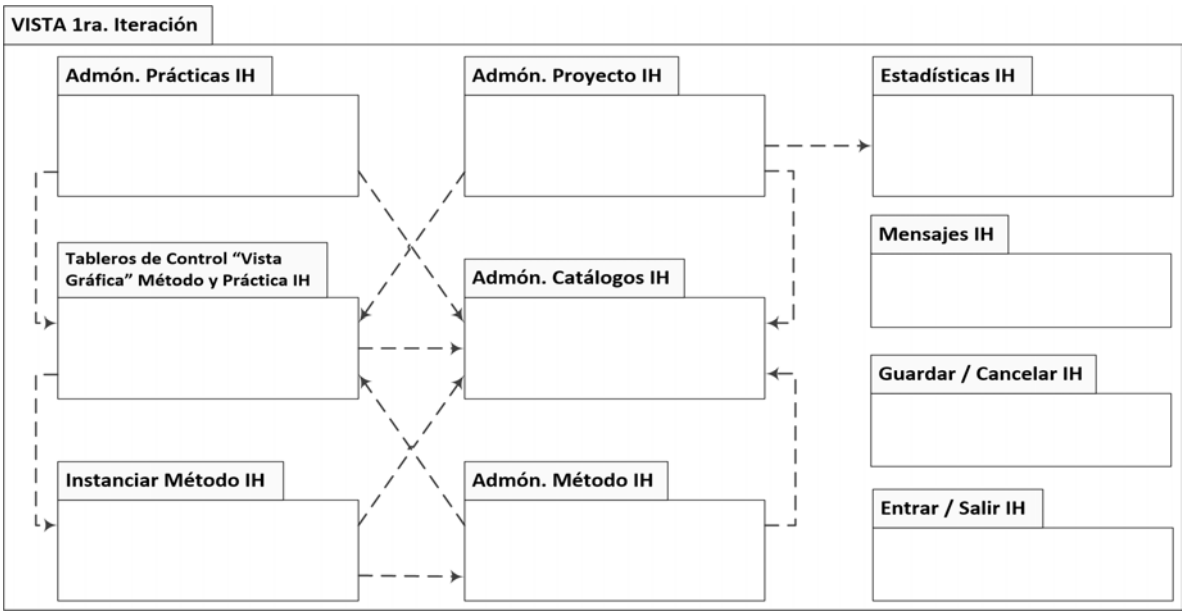


Figura 6.2. Diagrama del Paquete "Vista" en la primera iteración

Ya que se trata de una arquitectura MVC, el paquete anterior muestra los paquetes que contiene la "Vista", los paquetes correspondientes al "Modelo" y al "Controlador" contienen los mismos paquetes que el paquete de la "Vista", pero diferenciándolos con el sufijo "M" para el Modelo y "C" para el Controlador. Para comprobar que los paquetes contenidos en la "Vista", "Modelo" y "Controlador" cumplen con los *Drivers Arquitectónicos* en esta primera iteración, se realizó la tabla 6.5 la cual lista los *Drivers* y los paquetes asociados.

	Descripción del *driver*	Paquete Asociado
D1	Administración de Proyecto, Método, Prácticas, Guías y Herramientas	• Admón. Proyecto • Admón. Método • Admón. Prácticas
D2	Mostar vista gráfica de la Práctica	• Tableros de Control "Vista Gráfica" Método y Práctica
D3	Administrar Vista Gráfica del Método	• Tableros de Control "Vista Gráfica" Método y Práctica
D4	Instanciar y Ejecutar Método	• Instanciar Método
D5	Cancelar o Finalizar Método	• Instanciar Método
D6	Publicación del Tablero de Control de un Proyecto	• Tableros de Control "Vista Gráfica" Método y Práctica
D12	Manejo de versiones de Prácticas, Métodos, Proyectos	• Admón. Proyecto • Admón. Método • Admón. Prácticas
D13	Estadísticas del Proyecto	• Estadísticas
D14	Adaptar Método Gráficamente	• Tableros de Control "Vista Gráfica" Método y Práctica
D18	Seguridad básica de acceso a la herramienta	• Entrar / Salir

Tabla 6.5: Drivers Arquitectónicos vs Paquetes de la arquitectura

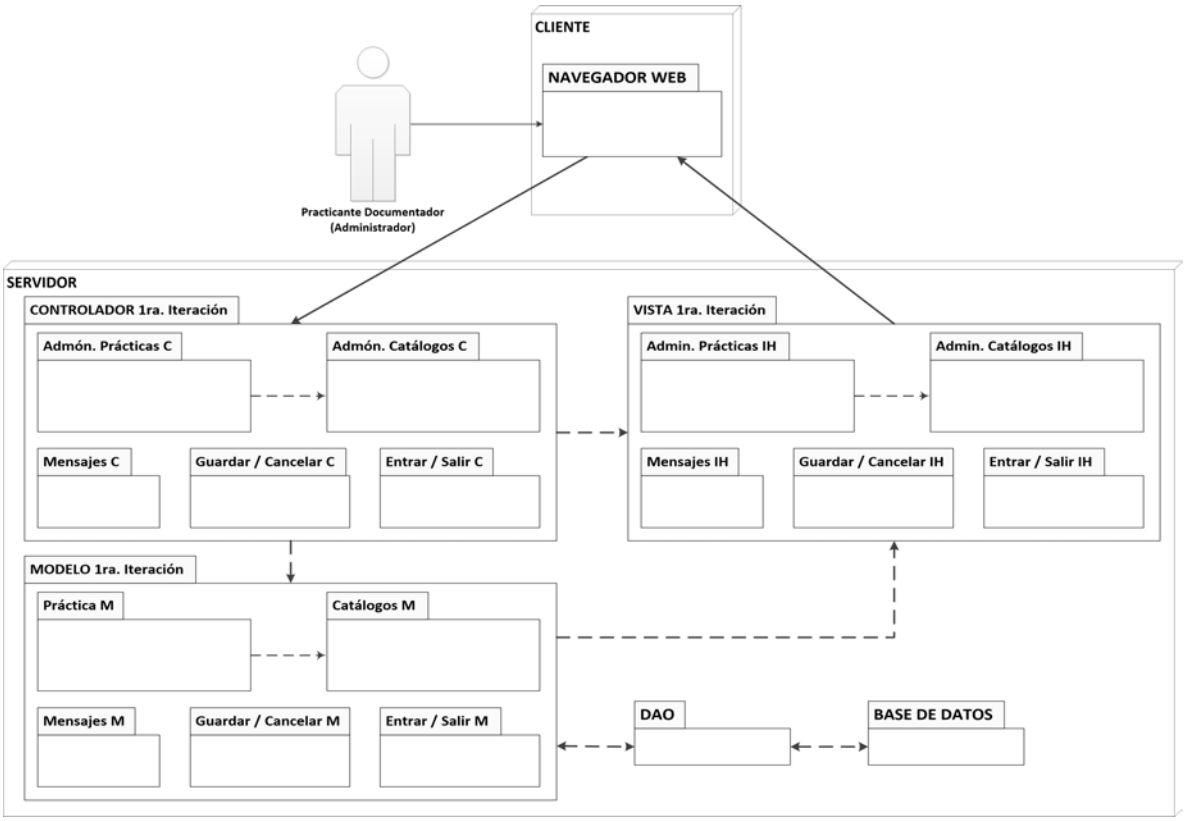


Figura 6.3. Diagrama de comunicación para el caso de uso “Agregar Práctica”.

En la figura 6.3 se ejemplifica con el diagrama de comunicación el caso de uso “Agregar Práctica”, en el cual se observa la intervención de los paquetes y sub-paquetes relacionados a este caso de uso. Describiendo el diagrama se observa que el actor “Practicante Documentador” es quien agrega una nueva práctica, esto lo realiza desde el navegador web, dicho navegador se conecta con el servidor, propiamente con el paquete Controlador, este paquete interactúa con el paquete Modelo, este a su vez con el paquete DAO el cual realiza la conexión a la base de datos, la BD regresa mensaje de éxito o error (según sea el caso) al paquete DAO, este último paquete le confirma la respuesta al paquete Modelo, el cual interactúa con el paquete Vista para mostrarle al usuario la respuesta del sistema a su petición. Cabe señalar que este diagrama es solo uno de los diferentes diagramas realizados, los cuales se encuentran en el apéndice III.

6.2 Segunda Iteración

Se elabora una segunda iteración, pero esta vez sólo se ejecutan los métodos ADD, V&B y ATAM, ya que la finalidad del método QAW es la captura y priorización de los *Drivers Arquitectónicos*, así como también la creación de escenarios, como estas acciones ya se han llevado a cabo no es necesario volver a ejecutar el método. Se lleva a cabo esta segunda iteración a fin de ampliar el detalle de los paquetes de la arquitectura que se está construyendo.

6.2.1 Ejecución del Método ADD

A continuación se describe lo que se obtuvo al ejecutar una segunda iteración de los pasos del método ADD.

Paso 1: Elegir un elemento del sistema para diseñar.

Para esta segunda iteración del método se tomaron los paquetes y sub-paquetes creados para el "Controlador", "Modelo" y "Vista" en la primera iteración, se irá haciendo el detalle y descomposición a la par de cada uno de los tres paquetes del modelo MVC.

Paso 2: Identificar los ASR´s y *Drivers Arquitectónicos para el elemento seleccionado*.

Se tomaron en cuenta los *Drivers Arquitectónicos* con prioridad alta de la Tabla 6.1, con el fin de hacer que se satisfagan en el detalle de la arquitectura. Después se tomaron en cuenta los *Drivers* con prioridad media y por último los *Drivers* con prioridad baja. Todo esto con el fin de que el diseño de la arquitectura contemple la totalidad de los *Drivers Arquitectónicos* obtenidos en el método QAW.

Paso 3: Generar una solución de diseño para el elemento seleccionado.

En esta segunda iteración se detalla el contenido de cada uno de los paquetes realizados en la iteración anterior, así como también las relaciones que existen entre cada uno de ellos, conservando como base el patrón arquitectónico MVC, además se incluye el patrón de diseño DAO, el cual ya fue descrito en la sección 3.4 de este trabajo de tesis.

La decisión de incluir al patrón DAO, es la dar mayor facilidad y libertad de realizar modificaciones en el futuro, en el tipo de manejador de base de datos que contenga el repositorio de los datos del *Entorno Computacional*.

Al mismo tiempo que se contempla la utilización del patrón DAO dentro del patrón arquitectónico MVC, se trabaja en darle solución y satisfacción a la "Mantenibilidad" requerida. Ya que al sumarle una "capa" más (capa de persistencia de datos) al patrón MVC se asegura que los cambios al modificar el Sistema Manejador de Base de Datos sean mínimos y no sea necesario el realizar cambios a los paquetes contenidos en el "Modelo".

Por otra parte, se diseño realizando la mayor modularidad posible, o por lo menos, esa es la perspectiva del arquitecto, al dividir y detallar el contenido de cada uno de los paquetes en subpaquetes y sus relaciones entre ellos, y por ende, también son divididas las responsabilidades que llevan consigo cada paquete. Este punto lleva un poco del mismo pensamiento que el empleado al pensar en separar la lógica del negocio (Modelo) y la capa de persistencia, en el sentido de que si fuera necesario el realizar modificaciones a los paquetes del diseño arquitectónico, esos cambios afecten lo menos posible al desempeño y funcionamiento de la arquitectura, ya que como se podrá imaginar, al tener divididas las responsabilidades del cumplimiento de los Drivers Arquitectónicos a través de los distintos paquetes y subpaquetes, los futuros cambios que llegase a tener el *Entorno Computacional* afectarán mínimamente a los demás paquetes, o por lo menos, se podrá tener más certeza de dónde está la falla después de realizar un cambio.

Paso 4: Verificar y refinar los requerimientos restantes y, generar y seleccionar los que servirán de entrada para la siguiente iteración del método.

En este paso, se verificó y mejoró el detalle de los paquetes realizados, contemplando todos los casos de uso, atributos de calidad y restricciones de los *Drivers Arquitectónicos*.

Dado que ya se han realizado los detalles en el contenido de los paquetes y subpaquetes de cada una de las partes del patrón MVC, se llega a la conclusión de que los paquetes existentes hasta este momento, satisfacen los Drivers Arquitectónicos.

Pero se deja claro, como vimos en los capítulos anteriores, que el grado de detalle que debe de tener el diseño de la arquitectura de software depende de las necesidades de los stakeholders y del equipo de desarrollo, pues se pretende que se entienda el funcionamiento del sistema descrito en el diseño arquitectónico, en otras palabras, ya no se realiza otra u otras iteraciones del método, pues el desarrollo del sistema para el *Entorno Computacional de KUALI-BEH* recae sobre los hombros de mis otros compañeros ([Barrera2014][Ruíz2014][Urrutia2014]) en sus trabajos de tesis, es decir, es decisión de cada uno de los desarrolladores del *Entorno* qué nombre llevarán las "clases", los "métodos", etcétera, que se encuentran dentro de cada uno de los paquetes y subpaquetes aquí desarrollados, no queriendo decir con esto, que el arquitecto se deslinde de cualquier obligación o preocupación, pues se resolverán las dudas que llegarán a surgir acerca del razonamiento plasmado en el diseño de la arquitectura, y de ser necesario se realizarán cambios en la misma.

6.2.2 Ejecución del Método V&B

Para esta segunda iteración se documenta la arquitectura resultante haciendo uso de los diagramas de paquetes con mayor detalle de cada uno de los paquetes.

En la figura 6.4 se muestra el nuevo diagrama de paquetes de la "Vista", así como también en la figura 6.5 se muestra el nuevo diagrama de paquetes del "Controlador", en la 6.6 se muestra el último diagrama de paquetes del "Modelo".

Además de los diagramas de paquetes, se realizó el diseño de la base de datos, haciendo uso del diagrama Entidad/Relación el cual se muestra en la figura 6.7. La transformación del diagrama Entidad/Relación al diagrama de Tablas, se muestra en la figura 6.8.

Al realizar y detallar los diagramas para la Base de Datos, se están describiendo las "tablas" que contendrá las base de datos, y la cardinalidad que contienen; lo cual ayudará a los desarrolladores a darles nombre a las clases, a los atributos de las clases y así como también a los métodos empleados, etcétera.

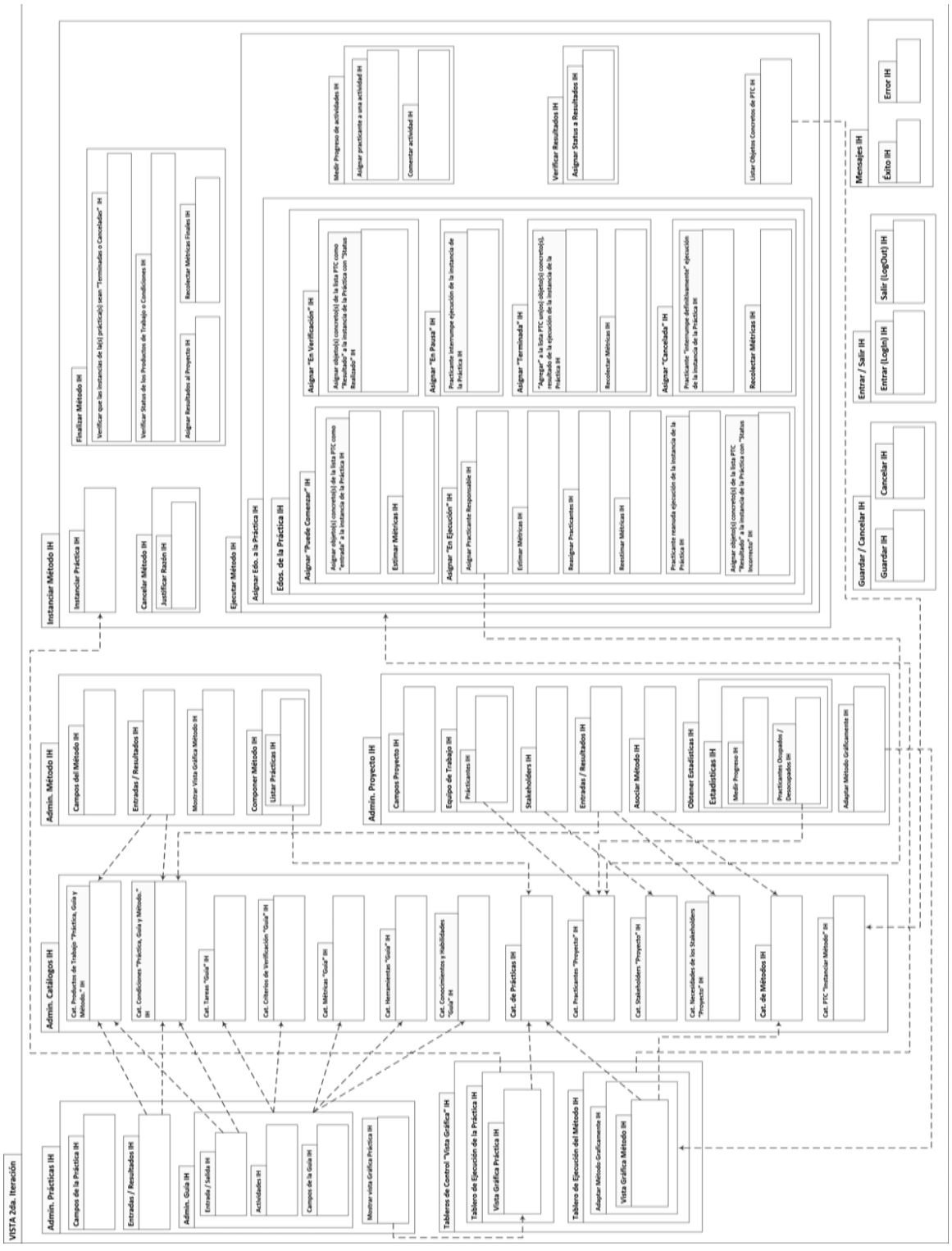


Figura 6.4: Diagrama del Paquete de la “Vista” en la segunda iteración.

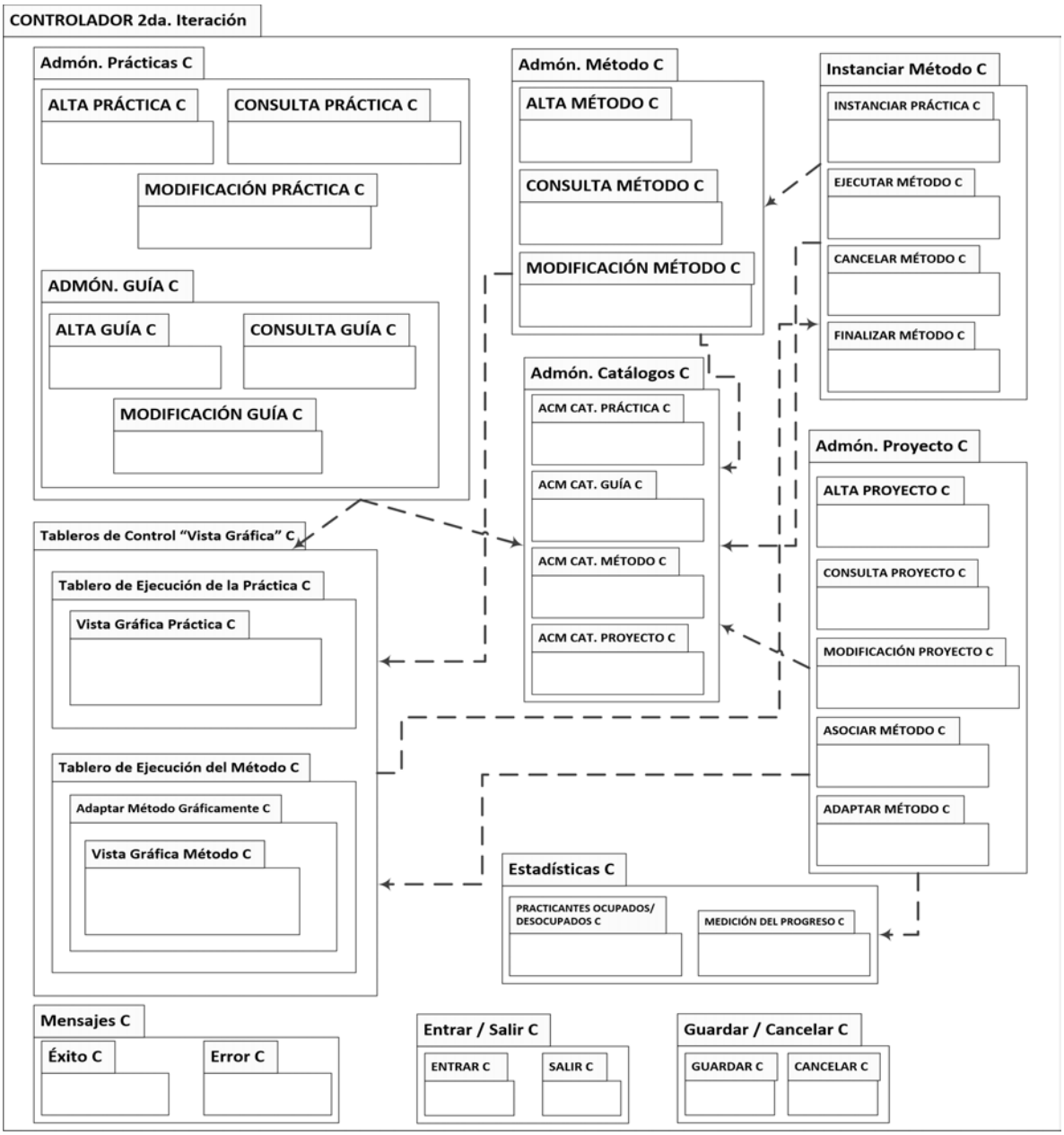


Figura 6.5: Diagrama del Paquete "Controlador" en la segunda iteración

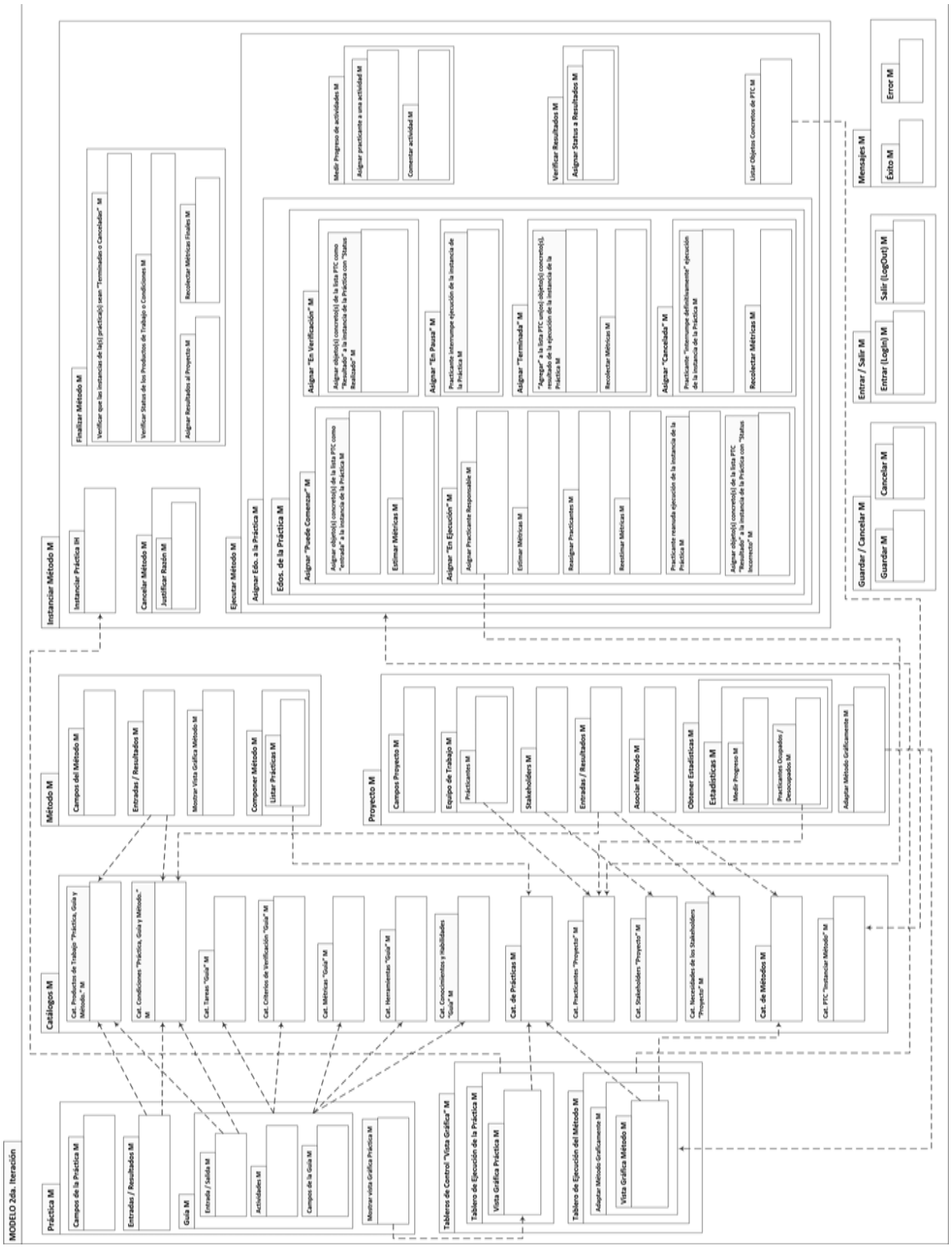


Figura 6.6: Diagrama del Paquete “Modelo” en la segunda iteración.

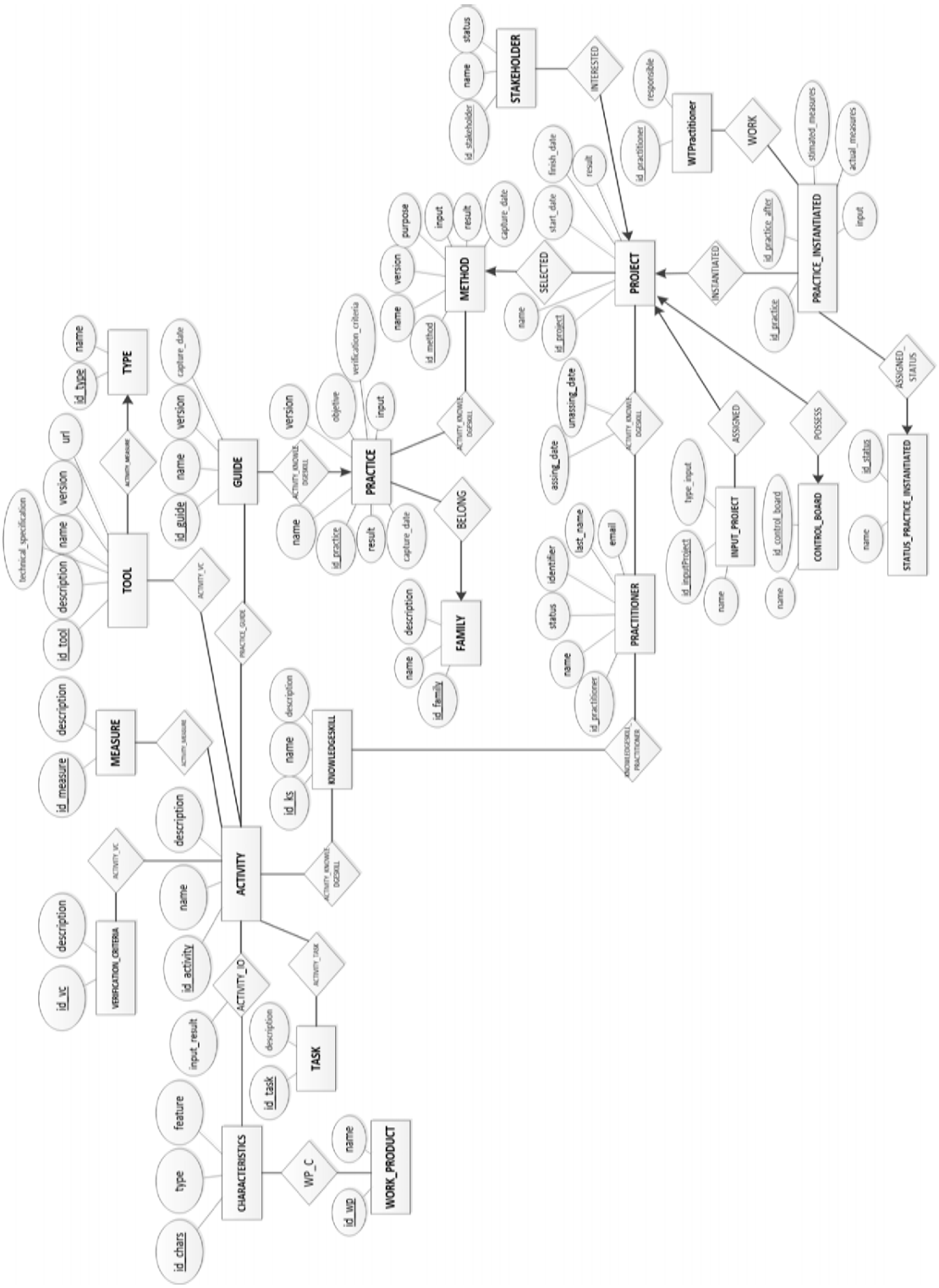


Figura 6.7: Diagrama Entidad/Relación del *Entorno Computacional*

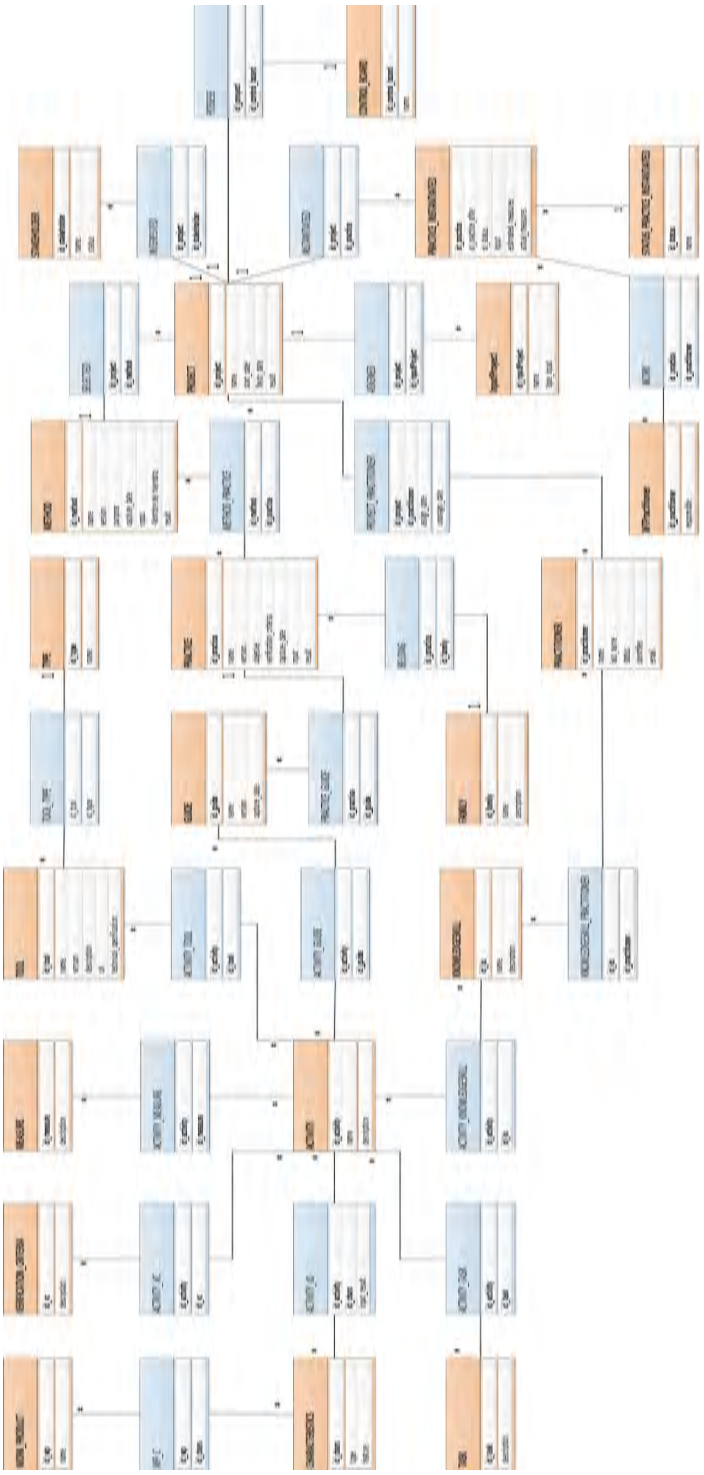


Figura 6.8: Diagrama de Tablas (transformación del diagrama E/R) del *Entorno Computacional*

6.2.3 Ejecución del Método ATAM

La ejecución del método ATAM da como resultado un conjunto de puntos sensibles, de compromiso y de riesgos arquitectónicos. Del método ATAM (sección 4.11), algunos de sus pasos han sido ejecutados en las secciones anteriores, conviene recapitular y comentar brevemente cómo se han ejecutado sus pasos:

1. Presentación del método ATAM.
2. Presentación de los objetivos del negocio.
3. Presentación de la arquitectura.
4. Identificación de los enfoques arquitectónicos.
5. Generación del árbol de utilidad.
6. Análisis de los enfoques arquitectónicos mediante la realización de los escenarios definidos en el árbol de utilidad.
7. Lluvia de ideas y priorización de escenarios.
8. Nuevo Análisis de los enfoques arquitectónicos.
9. Presentación de los resultados.

Aunque los equipos de evaluación y desarrollo pueden ser los mismos, los pasos del método se definen considerando que ambos equipos van a ser diferentes y que todas las partes implicadas (equipo de desarrollo, clientes, usuarios, etcétera), van a estar disponibles para resolver las dudas de los evaluadores. Ni una ni otra circunstancia se dan en este trabajo de tesis, en el que:

- El evaluador ha sido y es parte activa tanto en el proceso de diseño, en el proceso de desarrollo y en el proceso de la evaluación.
- La disponibilidad de los clientes y usuarios finales ha sido muy extensa, y aunque no ha sido posible reunirlos para realizar la evaluación, a lo largo de todo el desarrollo de la arquitectura se ha dispuesto de una gran cantidad de información procedente de los mismos.

Por todo ello, la ejecución del método ATAM se ha centrado en:

- La descripción de los factores del negocio (capítulo 5. KUALI-BEH).

- La descripción de la arquitectura, identificando los estilos arquitectónicos utilizados (secciones 6.1.3 y 6.2.2 descripción de las vistas arquitectónicas haciendo uso del método V&B).
- Realización de los escenarios definidos en el árbol de utilidad.

El paso 1 (presentación del método) se realizó en la sección 4.11 "El Método ATAM". Los pasos 7 y 8 no tienen sentido, puesto que requieren la contribución activa y conjunta de las partes implicadas que no han podido reunirse. Esta ausencia ha sido mitigada aprovechando la información obtenida y plasmándola en el desarrollo de los escenarios de calidad e incorporándolos al árbol de utilidad.

El árbol de utilidad define una taxonomía de los atributos de calidad en la que éstos se van refinando en las sucesivas ramas del árbol hasta llegar a las hojas, las cuales constituyen escenarios de evaluación concretos. La importancia relativa de cada escenario viene dada por su prioridad, la cual es función de dos componentes: la importancia relativa del escenario y el riesgo que implica su relación.

El orden de los escenarios según su prioridad es importante, ya que permite que el proceso de evaluación se centre en los aspectos claves de la arquitectura. Sin embargo, la asignación de las prioridades de todos los escenarios del árbol de utilidad son importantes.

Árbol de Utilidad

El árbol de utilidad definido para la evaluación de la arquitectura se muestra en la figura 6.9, este árbol de utilidad es el mismo que se generó en el paso 2 de la ejecución del método ADD en la sección 6.1.2. En el caso de la modificabilidad, se supone que el escenario se realiza satisfactoriamente si los cambios a realizar no se propagan por la arquitectura y no conllevan una gran complejidad.

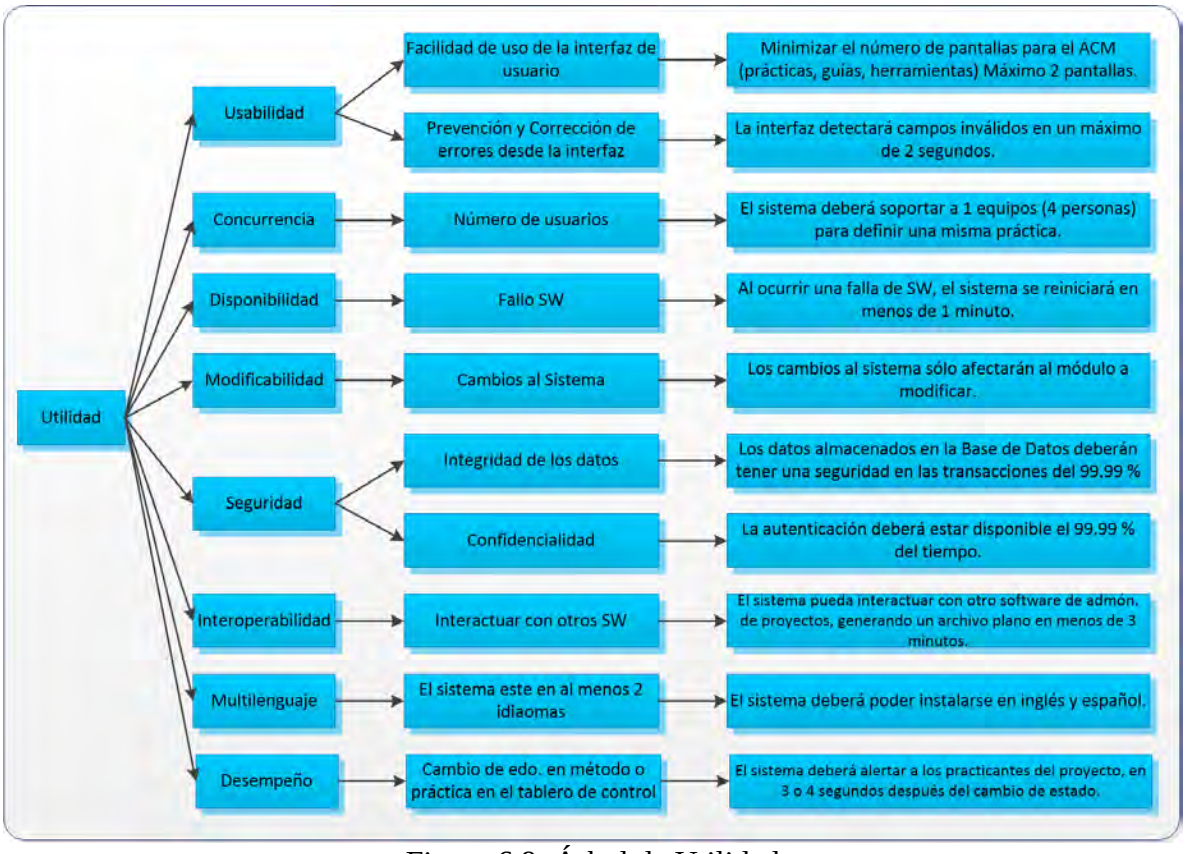


Figura 6.9: Árbol de Utilidad

6.3 Evaluación de la arquitectura vs Drivers Arquitectónicos

Se realizó una segunda tabla comparativa (tabla 6.6) en la cual se listan los paquetes asociados a cada uno de los *Drivers*. Como se pueden observar en las figuras 6.4 a 6.8, se muestra a detalle cada uno de los paquetes que cumplen con los Drivers Arquitectónicos y que estos paquetes se guiaron por el tipo de arquitectura MVC. Las ventajas que obtuvimos al usar este tipo de arquitectura fueron:

- La *escalabilidad*, esta es su principal ventaja, ya que se separa la aplicación en tres capas.
- Otra ventaja es la simplicidad con la que se puede gestionar y mantener al sistema así como también la posibilidad de trabajar en paralelo, es decir, se puede tener a diseñadores trabajando en las vistas (interfaz) mientras que los desarrolladores se pueden centrar en el Controlador y el Modelo. Esto deja claro que si en el futuro se

quieren realizar cambios a la interfaz, tan solo basta con realizar las modificaciones a la "Vista" y la aplicación seguirá funcionando.

- Sencillez para crear distintas representaciones de los mismos datos.
- Facilidad para realizar pruebas unitarias de los componentes.
- Reutilización de los componentes.
- Simplicidad en el mantenimiento del sistema.
- Facilidad para desarrollar prototipos más rápidamente.

Dentro de sus desventajas podemos mencionar las siguientes:

- Tener que apegarse a una estructura predefinida, lo que a veces puede incrementar la complejidad del sistema.
- La distribución de componentes obliga a crear y mantener un mayor número de paquetes.

	Tipo de Driver	Descripción del Driver	Paquetes Asociados
D1	Caso de Uso	Administración de Proyecto, Método, Prácticas, Guías y Herramientas	➢ Admón. Proyecto ➢ Admón. Método ➢ Admón. Prácticas o Admón. Guía ➢ Admón. Catálogos o Cat. Herramientas
D2	Caso de Uso	Mostar vista gráfica de la Práctica	➢ Tableros de Control "Vista Gráfica" o Tablero de Ejecución de la Práctica ▪ Vista Gráfica Práctica
D3	Caso de Uso	Administrar Vista Gráfica del Método	➢ Tableros de Control "Vista Gráfica" o Tablero de Ejecución del Método ▪ Adaptar Método Gráficamente • Vista Gráfica Método
D4	Caso de Uso	Instanciar y Ejecutar Método	➢ Instanciar Método o Ejecutar Método o Instanciar Práctica
D5	Caso de Uso	Cancelar o Finalizar Método	➢ Instanciar Método o Cancelar Método o Finalizar Método
D6	Caso de Uso	Publicación del Tablero de Control de un Proyecto	➢ Tableros de Control "Vista Gráfica" o Tablero de Ejecución del Método o Tablero de Ejecución de la Práctica

D7	Restricción	Todas las herramientas deben ser Software Libre	ALTA
D8	Atributo de Calidad	Concurrente	ALTA
D9	Atributo de Calidad	Facilidad de uso	ALTA
D10	Restricción	Plataforma de desarrollo	ALTA
D11	Restricción	Operación (SMBD, Browser, Web Server)	ALTA
D12	Caso de Uso	Manejo de versiones de Prácticas, Métodos, Proyectos	➢ Admón. Proyecto ➢ Admón. Método ➢ Admón. Prácticas
D13	Caso de Uso	Estadísticas del Proyecto	➢ Admón. Proyecto o Obtener Estadísticas ▪ Estadísticas
D14	Caso de Uso	Adaptar Método Gráficamente	➢ Admón. Proyecto o Adaptar Método Gráficamente ➢ Tableros de Control "Vista Gráfica" o Tablero de Ejecución del Método ▪ Adaptar Método Gráficamente
D15	Restricción	Interfaz de usuario WEB (Mozilla y Chrome)	MEDIA
D16	Atributo de Calidad	Facilidad de crecimiento (Mantenibilidad)	Se modularizó cada caso de uso en distintos paquetes, pensando en que en caso de realizar un cambio se haga sólo a los paquetes afectados por el mismo.
D17	Atributo de Calidad	Disponibilidad	MEDIA
D18	Atributo de Calidad	Seguridad básica de acceso a la herramienta	➢ Entrar / Salir o Entrar (Login) o Salir (LogOut)
D19	Atributo de Calidad	Interoperabilidad con otros sistemas	BAJA
D20	Atributo de Calidad	Facilidad de instalación	BAJA
D21	Atributo de Calidad	Multilenguaje (Español e Inglés)	BAJA

Tabla 6.6: Drivers vs Paquetes de la arquitectura

En la tabla 6.6 se observa que hay campos de los atributos de calidad y restricciones que están con prioridad (Alta, Media y Baja), se optó por dejar esta leyenda en lugar de poner el nombre del paquete o paquetes que satisfacen a estos atributos de calidad y restricciones, ya que no existe paquete alguno que satisfaga propiamente a estos Drivers,

más bien, están en la parte de desarrollo, es decir, se contemplaron para la parte de programación del *Entorno Computacional*, por tal motivo estos Drivers son los puntos sensibles y de riesgo de esta arquitectura, el cumplimiento de estos Drivers Arquitectónicos dependerá de los demás trabajos de tesis al implementar la arquitectura aquí descrita.

Por otra parte, cabe destacar que la arquitectura ya ha sido "mapeada" en el trabajo de tesis "Repositorio de métodos y prácticas de proyectos de software para KUALI-BEH" [Barrera2014], el autor de dicha tesis, dice que la arquitectura definida le fue de utilidad al construir su módulo. Los diagramas proporcionados le fueron de ayuda para organizar sus componentes, identificar sus clases, las que han podido ser incrustadas en los paquetes de la arquitectura y posteriormente en el código final del módulo.

Conclusiones

El objetivo de este trabajo de tesis fue definir la arquitectura de software para el *Entorno Computacional de KUALI-BEH* aplicando los métodos propuestos por el SEI enfocados en la realización del diseño de la arquitectura de software. Dicha arquitectura debería servir de base para la construcción del *Entorno Computacional* llevado a cabo en otros tres proyectos de tesis [Ruíz2014][Barrera2014][Urrutia2014] relacionados con este sistema.

A través de la arquitectura de software se realiza una abstracción de un sistema con el fin de facilitar, entre otras cosas, la comunicación de la organización del sistema entre los involucrados en el proyecto. Dicha abstracción se expresa en términos de componentes, sus propiedades y las relaciones relevantes entre ellos.

La arquitectura de software es el puente entre los objetivos del negocio del sistema y el sistema mismo. No definir el diseño de la arquitectura desde etapas tempranas del desarrollo de software, puede limitar severamente que el producto final satisfaga completamente las necesidades de desempeño, disponibilidad, seguridad del sistema o de los clientes, además el costo de las correcciones relacionadas con problemas en la arquitectura es muy elevado. Es así que establecer la arquitectura de software juega un papel fundamental dentro de todo desarrollo de software.

Al ejecutar los métodos del SEI se pudo observar que son una guía sólida en el diseño, construcción y evaluación de arquitecturas de software, pues a través de ellos se fue viendo cómo iba tomando forma poco a poco la arquitectura final y que realmente estos métodos guían la comprensión de cómo es que esta arquitectura verdaderamente satisface con lo que se pretendía desde un principio que cumpliera el *Entorno Computacional de KUALI-BEH*.

La experiencia de haber ejecutado los métodos del SEI para el *Entorno Computacional de KUALI-BEH*, además de ser un caso de estudio de aplicación real, permitió adquirir experiencia como arquitecto en la definición de la arquitectura.

Como trabajo futuro y temas de investigación futura se propone lo siguiente:

- Acabar de implementar la arquitectura de software del *Entorno Computacional de KUALI-BEH* en dicho sistema de software pues el desarrollo está inconcluso por parte de dos proyectos de tesis.
- Someter los pasos seguidos en esta tesis, pero adecuándolos a un nuevo proyecto, para refinar las actividades que aquí se llevaron a cabo.

Apéndice I Casos de Uso

Casos de Uso del *Entorno Computacional de KUALI-BEH*

Nombre del Caso de Uso: **Administrar Prácticas**

- ACM **Práctica**
  - ACM Nombre, Identificador, Versión, Objetivo
  - ACM Entradas / Resultados (Catálogo de Tipos::Características de Productos de Trabajo y Condiciones)
    - ACM Productos de Trabajo (ACM de tipo y características)
    - ACM Condiciones (ACM de tipo y características)
  - ACM **Guía**
    - ACM Actividades
      - ACM Entrada / Salida (Catálogo de la Práctica)
        - ACM Productos de Trabajo internos
        - ACM Condiciones internas
      - ACM Tareas
      - ACM Criterios de Verificación (Catálogo)
      - ACM Métricas (Catálogo)
      - ACM Herramientas (Catálogo)
      - ACM Conocimientos y Habilidades (Catálogo)
  - Mostrar Vista Textual Práctica
  - Mostrar Vista Gráfica Práctica

Nombre del Caso de Uso: **Administrar Método**

- ACM **Método**
  - ACM Nombre, Identificador, Versión, Propósito
  - ACM Entradas / Resultados (Catálogo)
    - ACM Productos de Trabajo
    - ACM Condiciones
  - Componer Método
    - Listar Prácticas (Catálogo)
  - Mostrar Vista Textual Método
  - Mostrar Vista Gráfica Método
    - Mostrar Prácticas

Nombre del Caso de Uso: **Administrar Proyecto**

- ACM **Proyecto**
  - ACM Nombre, Identificador, Versión, Fechas Inicio / Término
  - ACM Equipo de Trabajo (Catálogo)
    - ACM Practicantes (Catálogo)
  - ACM Stakeholders (Catálogo)
  - ACM Entradas / Resultados (Catálogo de Tipos::Características de Productos de Trabajo y Condiciones)
    - ACM Necesidades de los Stakeholders (Catálogo de Productos de Trabajo y Condiciones concretas asociados con su tipo)
    - ACM Condiciones del Proyecto (Catálogo de Productos de Trabajo y Condiciones concretas asociados con su tipo)
    - ACM Producto de Software (opcional)
  - Asociar Método
  - Obtener estadísticas del proyecto
    - Practicantes ocupados / desocupados
  - Medir progreso

        - Días transcurridos / Días faltantes
    - Adaptar Método Gráficamente
        - Concatenar, Fusionar, Partir, Substituir

Nombre del Caso de Uso: **Instanciar Método**

- **Instanciar práctica**
- **Ejecutar Método**
    - Listar objetos concretos de Productos de Trabajo o Condiciones (Lista PTC) inicializado con Productos de Trabajo y Condiciones de entrada del Proyecto
    - Asignar estado a práctica
        - Asignar *Puede Comenzar* (Depende de la asignación de entradas)
            - Asignar uno o más objetos concretos de la lista PTC como entrada a la instancia de la práctica
        - Asignar *En Ejecución*
            - Asignar practicante responsable a práctica
            - Reasignar practicantes
            - El practicante reanuda la ejecución de la instancia de la práctica
            - Asignar uno o más objetos concretos de la lista PTC como resultado a la instancia de la práctica con status *incorrecto*
        - Asignar *En Verificación*
            - Asignar uno o más objetos concretos de la lista PTC como resultado a la instancia de la práctica con status *realizado*
        - Asignar *En Pausa*
            - El practicante interrumpe la ejecución de la instancia de la práctica
        - Asignar *Terminada* (Depende de cómo se verifique cada uno de sus resultados)
            - Agregar a la lista PTC uno o más objetos concretos resultado de la ejecución de la instancia de la práctica
        - Asignar *Cancelada*
            - El practicante interrumpe definitivamente la ejecución de la instancia de la práctica

- Verificar resultados
- Asignar status a resultados

- Cancelar Método
  - Justificar la razón
- Finalizar Método
  - Instancias de las prácticas *terminadas* o *canceladas*
  - Status de los Productos de Trabajo o Condiciones *verificado*
  - Asignar resultados al Proyecto
  - Recolectar métricas finales

A continuación se muestran los detalles de los casos de uso, del *Entorno Computacional de KUALI-BEH*.

Caso de Uso: Agregar Proyecto

Actor: Practicante Ejecutor
Descripción: En este caso de uso se describe la forma en que el Practicante Ejecutor Agrega un nuevo Proyecto al sistema.
Pre-condiciones: El Practicante Ejecutor, deberá estar registrado en el sistema y haber iniciado sesión. Estar en la pantalla principal del sistema y seleccionar la opción "ADD Project", la cual los enviará a la pantalla principal de ADD Project.
Pos-condiciones: El Proyecto queda guardado en la base de datos del sistema, listo para que se le asocie un Método.

Flujo de Eventos				
Flujo básico:	**Agregar Proyecto**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	Previamente el Practicante Ejecutor debió haber seleccionado la opción "ADD Project".	**2.**	El sistema despliega el formulario con los campos para dar de alta un proyecto.	
3.	El Practicante Ejecutor llena los campos del formulario para agregar un proyecto: • IDENTIFIER • NAME • STAKEHOLDERS • START DATE • FINISH DATE • INPUT ➢ Stakeholders Needs ➢ Project Conditions ➢ Software Product • RESULT • PRACTITIONERS N			
4.	El Practicante Ejecutor verifica la información ingresada y da clic en el botón "Save" para guardar los datos del proyecto.	**5.**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de guardar los datos del formulario.	
6.	El Practicante Ejecutor da clic en el botón "Yes".	**7.**	El sistema verifica los campos del formulario.	R1, R2
		8.	El sistema guarda los datos del formulario en la base de datos y manda mensaje que fueron guardados exitosamente.	
		9.	El sistema regresa a la pantalla principal.	

Flujo Alternativo 1:	**Agregar Proyecto (cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
3.	El Practicante Ejecutor llena los campos del formulario para agregar un proyecto: • IDENTIFIER • NAME • STAKEHOLDERS • START DATE • FINISH DATE • INPUT ➢ Stakeholders Needs ➢ Project Conditions ➢ Software Product • RESULT • PRACTITIONERS N			
4.1	El Practicante Ejecutor presiona el botón “Cancel”.	**5.1**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de cancelar el dar de alta el proyecto.	
6.1	El Practicante Ejecutor presiona el botón “Yes”	**7.1**	El sistema regresa a la pantalla principal.	

Flujo Alternativo 2:	**Agregar Proyecto (abortar cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
3.	El Practicante Ejecutor llena los campos del formulario para agregar un proyecto: • IDENTIFIER • NAME • STAKEHOLDERS • START DATE • FINISH DATE • INPUT ➢ Stakeholders Needs ➢ Project Conditions ➢ Software Product • RESULT • PRACTITIONERS N			
4.2	El Practicante Ejecutor presiona el botón “Cancel”.	**5.2**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de cancelar el dar de alta el proyecto.	
6.2	El Practicante Ejecutor presiona el botón “No”	**7.2**	El sistema regresa a la pantalla de edición del formulario para dar de alta un proyecto.	

Rebanadas		
Identificador	**Nombre**	**Respuesta del Sistema**
R1	Campos obligatorios vacíos.	El sistema manda un mensaje al usuario indicándole que existen campos obligatorios sin información.
R2	Formato de Fecha inválido.	El sistema manda un mensaje al usuario indicándole que el formato de la fecha no es el correcto.

Caso de Uso: Ver Proyecto

Actor: Practicante Ejecutor
Descripción: En este caso de uso se describe la forma en que el Practicante Ejecutor consulta un Proyecto en el sistema.
Pre-condiciones: El Practicante Ejecutor, deberá estar registrado en el sistema y haber iniciado sesión. Estar en la pantalla principal del sistema y seleccionar la opción “Project”, la cual lo enviará a la pantalla principal de Proyecto.
Pos-condiciones: El Practicante ejecutor puede revisar la información del Proyecto seleccionado.

Flujo de Eventos				
Flujo básico:	**Ver Proyecto**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	El Practicante Ejecutor selecciona un proyecto de la lista mostrada por el sistema.	**2.**	El sistema despliega el formulario (con los campos deshabilitados) con la información del proyecto seleccionado.	
3.	El Practicante Ejecutor da clic en el botón “Cancel”.	**4.**	El sistema regresa a la pantalla principal del sistema.	

Caso de Uso: Agregar Método

Actor: Practicante Documentador (Administrador)
Descripción: En este caso de uso se describe la forma en que el Practicante Documentador, puede "Agregar" un Método.
Pre-condiciones: El Practicante Documentador, deberá estar registrado en el sistema y haber iniciado sesión. Estar en la pantalla principal del sistema y seleccionar la opción "ADD Method".
Pos-condiciones: El sistema tendrá a un nuevo Método, el cual quedará listo para ser compuesto por Prácticas e instanciado a un Proyecto.

Flujo de Eventos				
Flujo básico:	**Agregar Método**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	El Practicante Documentador selecciona la opción "ADD Method".	2.	El sistema despliega el formulario con lo campos para dar de alta un Método.	
3.	El Practicante Documentador llena los campos del formulario para agregar un nuevo Método: • Identifier • Name • Purpose • Input • Result • Practice			
4.	El Practicante Documentador verifica la información ingresada y da clic en el botón "Save" para guardar los datos del Método.	5.	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de guardar los datos del formulario.	
	El Practicante Documentador da clic en el botón "Yes".		El sistema verifica los campos del formulario.	R1
			El sistema guarda el Método (datos del formulario) en la base de datos y manda mensaje que fueron guardados exitosamente.	
			El sistema regresa a la pantalla principal.	

Flujo Alternativo 1:	**Agregar Método (cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
3.	El Practicante Documentador llena los campos del formulario para agregar un nuevo Método: • Identifier • Name • Purpose			

	• Input • Result • Practice			
4.1	El Practicante Documentador presiona el botón “Cancel”.	**5.1**	El sistema pregunta al Practicante Documentador si está seguro de cancelar el dar de alta el Método.	
6.1	El Practicante Documentador presiona el botón “Yes”.	**7.1**	El sistema regresa a la pantalla principal.	

Flujo Alternativo 2:	**Agregar Método (abortar cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
3.	El Practicante Documentador llena los campos del formulario para agregar un nuevo Método: • Identifier • Name • Purpose • Input • Result • Practice			
4.2	El Practicante Documentador presiona el botón “Cancel”.	**5.2**	El sistema pregunta al Practicante Documentador si está seguro de cancelar el dar de alta el Método.	
6.2	El Practicante Documentador presiona el botón “No”.	**7.2**	El sistema regresa a la pantalla de edición del formulario para dar de alta un Método.	

Rebanadas		
Identificador	**Nombre**	**Respuesta del Sistema**
R1	Campos obligatorios vacíos.	El sistema manda un mensaje al usuario indicándole que existen campos obligatorios sin información.

Caso de Uso: Instanciar Método

Actor: Practicante Ejecutor
Descripción: En este caso de uso se describe la forma en que el Practicante Ejecutor, Instancia un Método en un Proyecto.
Pre-condiciones: En el sistema debe de existir al menos un método en la base de datos. Y previamente se debió de haber llenado la plantilla del Proyecto al cual se le desea instanciar un método.
Pos-condiciones: El Proyecto con su Método instanciado queda guardado en la base de datos del sistema.

Flujo de Eventos				
Flujo básico:	**Instanciar Método**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	El Practicante Ejecutor da clic en el botón “Associate Method” de la pantalla principal.	**2.**	El sistema despliega la pantalla para asociar un método a un proyecto. El sistema muestra un listado de los proyectos que se encuentran en la base de datos (Proyectos que NO tienen asociados un método). Botones Habilitados: Botones Deshabilitados: Associate Method, Save, Cancel	
3.	El Practicante Ejecutor selecciona un proyecto de la lista.	**4.**	El sistema despliega la información del proyecto seleccionado en el espacio de visualización del proyecto (con los campos del proyecto deshabilitados). Botones Habilitados: Associate Method Botones Deshabilitados: Save, Cancel	
5.	El Practicante Ejecutor da clic en el botón “Associate Method”.	**6.**	El sistema despliega una lista de los métodos que se encuentran en la base de datos del sistema. Botones Habilitados: Save, Cancel Botones Deshabilitados: Associate Method	
7.	El Practicante Ejecutor selecciona un Método de la lista, acorde al proyecto para asociarlo.	**8.**	El sistema despliega la información del Método seleccionado en el espacio de visualización del Método (con	

			los campos del Método deshabilitados). Botones Habilitados: Save, Cancel Botones Deshabilitados: Associate Method	
9.	El Practicante Ejecutor verifica la información y da clic en el botón “Save” para asociar el Método al Proyecto y guardar.	**10.**	El sistema asocia el Método seleccionado y guarda el Proyecto en la base de datos.	
		11.	El sistema despliega sólo la lista de proyecto en el área de visualización.	

Flujo Alternativo 1:	**Instanciar Método (cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
		8.	El sistema despliega la información del Método seleccionado en el espacio de visualización del Método (con los campos del Método deshabilitados). Botones Habilitados: Save, Cancel Botones Deshabilitados: Associate Method	
9.1	El Practicante Ejecutor presiona el botón “Cancel”.	**10.1**	El sistema despliega sólo la lista de proyecto en el área de visualización.	

Caso de Uso: Componer Método (Instanciar Prácticas)

Actor: Practicante Documentador (Administrador)
Descripción: En este caso de uso se describe la forma en que el Practicante Documentador, puede "componer" un Método a partir de la unión de Prácticas.
Pre-condiciones: El Practicante Documentador, deberá estar registrado en el sistema y haber iniciado sesión. Estar en la pantalla principal del sistema y seleccionar la opción "Compose Method". El sistema deberá contener al menos una práctica.
Pos-condiciones: El sistema tendrá a un nuevo método compuesto, el cual contendrá practicas ya existentes dentro del sistema.

Flujo de Eventos				
Flujo básico:	**Componer Método**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	El Practicante Documentador selecciona la opción "Compose Method".	**2.**	El sistema despliega la vista para la composición de método.	
		3.	El sistema muestra una serie de Prácticas que el Practicante Documentador puede agregar.	R1
4.	El Practicante Documentador selecciona la primer práctica p_1 que cumpla con el input del método.	**5.**	El sistema pregunta al Practicante Documentador si está seguro que el objetivo de la práctica p_1 cumple con el propósito del método.	
6.	El Practicante Documentador da clic en "Yes".	**7.**	El sistema pregunta al Practicante Documentador si está seguro que el input de la práctica p_1 es similar al input del método.	
8.	El Practicante Documentador da clic en "Yes".	**9.**	El sistema agrega la práctica p_1 al método.	
10.	El Practicante Documentador selecciona la práctica p_{i+1} que requiera como input el result de la práctica p_i que escogió anteriormente.	**11.**	El sistema pregunta al Practicante Documentador si está seguro que el objetivo de la práctica p_i cumple con el propósito del método.	
12.	El Practicante Documentador da clic en "Yes".	**13.**	El sistema pregunta al Practicante Documentador si está seguro que el result de la práctica p_i es similar al input de la práctica p_{i+1}.	
14.	El Practicante Documentador da clic en "Yes".	**15.**	El sistema agrega la práctica p_{i+1} al método.	
16.	El Practicante Documentador repite los pasos 10, 12 y 14 las veces que necesite hasta que solo le falte la práctica con la que cumple el result del método.	**17.**	El sistema repite los pasos 11, 13 y 15 mientras el Practicante Documentador siga agregando prácticas.	
18.	El Practicante Documentador selecciona la práctica p_m que	**19.**	El sistema pregunta al Practicante Documentador si está seguro que el	

	como input requiera el result de la última práctica p_i agregada al método y cuyo result cumpla como result del método.		objetivo de la práctica p_m cumple con el propósito del método.	
20.	El Practicante Documentador da clic en "Yes".	**21.**	El sistema pregunta al Practicante Documentador si está seguro que el result de la práctica p_m es similar al result del método.	
22.	El Practicante Documentador da clic en "Yes".	**23.**	El sistema pregunta al Practicante Documentador si está seguro que el conjunto de prácticas asociadas cumplen con el propósito del método.	
24.	El Practicante Documentador da clic en "Yes".	**25.**	El sistema agrega la práctica p_m al método y guarda el nuevo método.	
		26.	El sistema muestra el mensaje "Method Composition Successful".	
		27.	El sistema regresa a la pantalla principal.	

Flujo Alternativo 1:	**Componer Método (cancelar composición)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
		2.	El sistema despliega la vista para la composición de método.	
3.1	El Practicante Documentador presiona el botón "Cancel".	**4.1**	El sistema pregunta al actor si está seguro de cancelar la composición.	
5.1	El Practicante Documentador presiona el botón "Accept".	**6.1**	El sistema regresa a la pantalla principal sin almacenar información.	

Flujo Alternativo 2:	**Componer Método (rechazar cancelación)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
		4.1	El sistema pregunta al actor si está seguro de cancelar la composición.	
5.2	El Practicante Documentador presiona el botón "Cancel".	**6.2**	El sistema cierra la ventana emergente y permite al actor continuar con el flujo principal.	

Flujo Alternativo 3:	**Componer Método (cancelar asociación input)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
		5.	El sistema pregunta al Practicante Documentador si está seguro que el objetivo de la práctica p_1 cumple con el propósito del método.	

6.3	El Practicante Documentador presiona el botón “No”.	7.3	El sistema cierra la ventana emergente y permite al Practicante Documentador volver a seleccionar una práctica.	

Flujo Alternativo 4:	**Componer Método (cancelar asociación por no ser input similar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
		7.	El sistema pregunta al Practicante Documentador si está seguro que el input de la práctica p_1 es similar al input del método.	
8.4	El Practicante Documentador presiona el botón “No”.	9.4	El sistema cierra la ventana emergente y permite al Practicante Documentador volver a seleccionar una práctica.	

Flujo Alternativo 5:	**Componer Método (cancelar asociación práctica)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
		11.	El sistema pregunta al Practicante Documentador si está seguro que el objetivo de la práctica p_i cumple con el propósito del método.	
12.5	El Practicante Documentador presiona el botón “No”.	13.5	El sistema cierra la ventana emergente y permite al Practicante Documentador volver a seleccionar una práctica.	

Flujo Alternativo 6:	**Componer Método (cancelar asociación por no ser input/result similar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
		13.	El sistema pregunta al Practicante Documentador si está seguro que el result de la práctica p_i es similar al input de la práctica p_{i+1}.	
14.6	El Practicante Documentador presiona el botón “No”.	15.6	El sistema cierra la ventana emergente y permite al Practicante Documentador volver a seleccionar una práctica.	

Flujo Alternativo 7:	**Componer Método (cancelar asociación result)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
		19.	El sistema pregunta al Practicante Documentador si está seguro que el	

			objetivo de la práctica p_m cumple con el propósito del método.	
20.7	El Practicante Documentador presiona el botón “No”.	**21.7**	El sistema cierra la ventana emergente y permite al Practicante Documentador volver a seleccionar una práctica.	

Flujo Alternativo 8:	**Componer Método (cancelar asociación por no ser result similar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
		21.	El sistema pregunta al Practicante Documentador si está seguro que el result de la práctica p_m es similar al result del método.	
22.8	El Practicante Documentador presiona el botón “No”.	**23.8**	El sistema cierra la ventana emergente y permite al Practicante Documentador volver a seleccionar una práctica.	

Flujo Alternativo 9:	**Componer Método (cancelar asociación por no cumplir en conjunto)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
		23.	El sistema pregunta al Practicante Documentador si está seguro que el conjunto de prácticas asociadas cumplen con el propósito del método.	
24.9	El Practicante Documentador presiona el botón “No”.	**25.9**	El sistema cierra la ventana emergente y permite al Practicante Documentador volver a seleccionar una práctica.	

Rebanadas		
Identificador	**Nombre**	**Respuesta del Sistema**
R1	No hay prácticas en el sistema.	El sistema manda un mensaje al Practicante Documentador indicándole que no existen prácticas aún en el sistema.

Caso de Uso: Componer Método Gráficamente

Actor: Practicante Documentador (Administrador)
Descripción: En este caso de uso se describe la forma en que el Practicante Documentador, compondrá un Método con Prácticas de forma Gráfica.
Pre-condiciones: El sistema debe tener al menos un Método almacenado.
Pos-condiciones: El sistema tendrá almacenadas las Prácticas correspondientes a un Método.

Flujo de Eventos				
Flujo básico:	**Compose Method Graphic**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	El Practicante Documentador selecciona la opción "Compose Method Graphic".	**2.**	El sistema despliega la vista para la composición de métodos gráficamente.	
		3.	El sistema muestra un cuadro correspondiente al método, dentro de los cuales el Practicante Documentador podrá almacenar las prácticas arrastrándolas.	R1
		4.	El Sistema muestra una serie de prácticas gráficamente, que el Practicante Documentador puede agregar.	
5.	*El* Practicante Documentador arrastra las prácticas que desee agregar al método.			
		6.	El sistema relaciona el método con las prácticas que se vayan arrastrando hacia él.	

Flujo Alternativo 1:	**Compose Method Graphic (Deshacer Composición)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
		6.	El sistema relaciona el método con las prácticas que se vayan arrastrando hacia él.	
7.1	El Practicante Documentador selecciona del cuadro de método, las prácticas que no desee agregar y las arrastra nuevamente hacia fuera del recuadro.			
		8.1	El sistema deshace la relación que existía entre el método y la práctica(as) que se arrastraron hacia fuera del recuerdo.	

Flujo Alternativo 2:	**Compose Method Graphic (Agregar Práctica)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
		4.	El Sistema muestra una serie de prácticas gráficamente, que el Practicante Documentador puede agregar.	R1
5.2	El Practicante Documentador da clic en el botón *Add Practice.*	**6.2**	El Sistema implementa el caso de uso Agregar Práctica.	
		7.2	El Sistema regresa al paso 4 del flujo básico.	

Flujo Alternativo 3:	**Compose Method (Ver Práctica)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
		4.	El Sistema muestra una serie de prácticas gráficamente, que el Practicante Documentador puede agregar.	R1
5.3	El Practicante Documentador da clic en una practica para ver sus datos.	**6.3**	El Sistema muestra la información de la práctica implementando el caso de uso Ver Práctica.	
		7.3	El Sistema regresa al paso 4 del flujo básico.	

Rebanadas		
Identificador	**Nombre**	**Respuesta del Sistema**
R1	No hay prácticas en el sistema.	El sistema manda un mensaje a Practicante Documentador indicándole que no existen prácticas aún en el sistema.

Caso de Uso: Modificar Método

Actor: Practicante Documentador (Administrador)
Descripción: En este caso de uso se describe la forma en que el Practicante Documentador, puede "Modificar" un Método.
Pre-condiciones: El Practicante Documentador, deberá estar registrado en el sistema y haber iniciado sesión. Haber seleccionado un Método para realizarle las modificaciones.
Pos-condiciones: El sistema guarda en la base de datos las modificaciones realizadas al Método.

Flujo de Eventos				
Flujo básico:	**Modificar Método**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	Previamente el Practicante Documentador deberá haber seleccionado un Método y dar clic en el botón "Modify".	2.	El sistema habilita los campos del formulario del Método para su modificación. Botones Habilitados: Save, Cancel Botones Deshabilitados: Modify	
3.	El Practicante Documentador modifica los campos deseados en el formulario del Método.			
4.	El Practicante Documentador verifica la información ingresada y da clic en el botón "Save" para guardar los datos del Método.	5.	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de guardar los datos del formulario.	
6.	El Practicante Documentador da clic en el botón "Yes".	7.	El sistema verifica los campos del formulario.	R1
		8.	El sistema despliega un cuadro de diálogo donde pregunta si se desea cambiar la versión del Método.	
9.	El Practicante Documentador da clic en el botón "No".			
		10.	El sistema guarda el Método (datos del formulario) en la base de datos y manda mensaje que fueron guardados exitosamente.	
		11.	El sistema regresa a la pantalla principal.	

Flujo Alternativo 1:	**Modificar Método (cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
3.	El Practicante Documentador modifica los campos deseados en el formulario del Método.			
4.1	El Practicante Documentador presiona el botón "Cancel".	5.1	El sistema pregunta al Practicante Documentador si está seguro de cancelar la modificación del Método.	
6.1	El Practicante Documentador	7.1	El sistema regresa a la pantalla	

	presiona el botón “Yes”.		principal.	

Flujo Alternativo 2:	**Modificar Método (abortar cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
3.	El Practicante Documentador modifica los campos deseados en el formulario del Método.			
4.2	El Practicante Documentador presiona el botón “Cancel”.	**5.2**	El sistema pregunta al Practicante Documentador si está seguro de cancelar la modificación del Método.	
6.2	El Practicante Documentador presiona el botón “No”.	**7.2**	El sistema regresa a la pantalla de edición del formulario para modificar el Método.	

Rebanadas		
Identificador	**Nombre**	**Respuesta del Sistema**
R1	Campos obligatorios vacíos.	El sistema manda un mensaje al usuario indicándole que existen campos obligatorios sin información.

Caso de Uso: Modificar Versión del Método

Actor: Practicante Documentador (Administrador)
Descripción: En este caso de uso se describe la forma en que el Practicante Documentador, puede "Modificar" la Versión de un Método.
Pre-condiciones: El Practicante Documentador, deberá estar registrado en el sistema y haber iniciado sesión. Haber seleccionado un Método para realizarle las modificaciones.
Pos-condiciones: El sistema guarda en la base de datos las modificaciones realizadas al Método.

Flujo de Eventos				
Flujo básico:	**Modificar Método**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	Previamente el Practicante Documentador deberá haber seleccionado un Método y dar clic en el botón "Modify".	2.	El sistema habilita los campos del formulario del Método para su modificación. Botones Habilitados: Save, Cancel Botones Deshabilitados: Modify	
3.	El Practicante Documentador modifica los campos deseados en el formulario del Método.			
4.	El Practicante Documentador verifica la información ingresada y da clic en el botón "Save" para guardar los datos del Método.	5.	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de guardar los datos del formulario.	
6.	El Practicante Documentador da clic en el botón "Yes".	7.	El sistema verifica los campos del formulario.	R1
		8.	El sistema despliega un cuadro de diálogo donde pregunta si se desea cambiar la versión del Método.	
9.	El Practicante Documentador da clic en el botón "No".			
		10.	El sistema guarda el Método (datos del formulario) en la base de datos y manda mensaje que fueron guardados exitosamente.	
		11.	El sistema regresa a la pantalla de visualización del Método con los campos deshabilitados.	

Flujo Alternativo 1:	**Modificar Versión del Método**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
		8.	El sistema despliega un cuadro de diálogo donde pregunta si se desea cambiar la versión del Método.	
9.	El Practicante Documentador da clic en el botón "Yes".	10.1	El sistema despliega un cuadro de diálogo donde pregunta cuál es la nueva versión del Método.	

11.1	El Practicante Documentador ingresa la nueva versión del Método.	**12.1**	El sistema verifica que la nueva versión del Método se válida.	R2
		13.1	El sistema guarda los cambios y manda mensaje que las modificaciones fueron exitosas.	
		14.1	El sistema regresa a la pantalla de visualización del Método con los campos deshabilitados.	

Rebanadas		
Identificador	**Nombre**	**Respuesta del Sistema**
R1	Campos obligatorios vacíos.	El sistema manda un mensaje al usuario indicándole que existen campos obligatorios sin información.
R2	Versión No válida.	El sistema manda un mensaje al usuario indicándole que el número de la versión del Método no es válida (no es válida porque ya existe esta versión).

Caso de Uso: Agregar Práctica

Actor: Practicante Documentador (Administrador).
Descripción: En este caso de uso se describe la forma en que el Practicante Documentador Agrega una nueva Práctica al sistema.
Pre-condiciones: El Practicante Documentador, deberá estar registrado en el sistema y haber iniciado sesión. Estar en la pantalla principal del sistema y seleccionar la opción “ADD Practice”, la cual los enviará a la pantalla principal de Agregar Práctica.
Pos-condiciones: La Práctica queda dada de alta en la base de datos del sistema.

Flujo de Eventos				
Flujo básico:	**Agregar Práctica**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	Previamente el Practicante Documentador debió haber seleccionado la opción “ADD Practice”.	**2.**	El sistema despliega el formulario con los campos para dar de alta una Práctica.	
3.	El Practicante Documentador llena los campos del formulario para agregar una Práctica: • Identifier • Name • Version • Family • Objective • Input • Result • Guide (Manage Guides) • Verification criteria • Measure	**4.**	El sistema despliega la interfaz “Guide Data” con el formulario para llenar los datos de la(s) guía(s) asociada(s) a la Práctica.	
5.	El Practicante Documentador verifica la información ingresada y da clic en el botón “Save” para guardar los datos de la Práctica.	**6.**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de guardar los datos del formulario.	
7.	El Practicante Documentador da clic en el botón “Yes”.	**8.**	El sistema verifica los campos del formulario.	R1
		9.	El sistema guarda la Práctica (datos del formulario) en la base de datos y manda mensaje que fueron guardados exitosamente.	
		10.	El sistema regresa a la pantalla principal.	

Flujo Alternativo 1:	**Agregar Práctica (cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
3.	El Practicante Documentador llena los campos del formulario para agregar una Práctica: • Identifier • Name • Version • Family • Objective • Input • Result • Guide (Manage Guides) • Verification criteria • Measure	**4.**	El sistema despliega la interfaz "Guide Data" con el formulario para llenar los datos de la(s) guía(s) asociada(s) a la Práctica.	
5.1	El Practicante Documentador presiona el botón "Cancel".	**6.1**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de cancelar el dar de alta una Práctica.	
7.1	El Practicante Documentador presiona el botón "Yes"	**8.1**	El sistema regresa a la pantalla principal.	

Flujo Alternativo 2:	**Agregar Práctica (abortar cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
3.	El Practicante Documentador llena los campos del formulario para agregar una Práctica: • Identifier • Name • Version • Family • Objective • Input • Result • Guide (Manage Guides) • Verification criteria • Measure	**4.**	El sistema despliega la interfaz "Guide Data" con el formulario para llenar los datos de la(s) guía(s) asociada(s) a la Práctica.	
5.2	El Practicante Documentador presiona el botón "Cancel".	**6.2**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de cancelar el dar de alta una Práctica.	
7.2	El Practicante Documentador presiona el botón "No"	**8.2**	El sistema regresa a la pantalla de edición del formulario para dar de alta una Práctica.	

Rebanadas		
Identificador	**Nombre**	**Respuesta del Sistema**
R1	Campos obligatorios vacíos.	El sistema manda un mensaje al usuario indicándole que existen campos obligatorios sin información.

Caso de Uso: Modificar Práctica

Actor: Practicante Documentador (Administrador).
Descripción: En este caso de uso se describe la forma en que el Practicante Documentador Modifica una Práctica en el sistema.
Pre-condiciones: El Practicante Documentador, deberá estar registrado en el sistema y haber iniciado sesión. El Practicante Documentador deberá haber seleccionado un Proyecto y un Método asociado al Proyecto, para que el sistema pueda mostrar las Prácticas asociadas al Método. Seleccionar la Práctica a modificar.
Pos-condiciones: La Práctica queda guardada en la base de datos del sistema con los cambios realizados.

Flujo de Eventos				
Flujo básico:	**Modificar Práctica**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	Previamente el Practicante Documentador debió haber seleccionado un Proyecto y un Método asociado al Proyecto, para que el sistema pueda mostrar las Prácticas asociadas al Método.	**2.**	El sistema despliega un listado con las Prácticas asociadas al Método, en el campo de visualización de las Prácticas. Botones Habilitados: Modify Botones Deshabilitados: Save, Cancel	
3.	El Practicante Documentador selecciona una Práctica del listado y da clic en el botón “Modify”.	**4.**	El sistema habilita los campos del formulario de la Práctica para su modificación. Botones Habilitados: Save, Cancel Botones Deshabilitados: Modify	
5.	El Practicante Documentador modifica los campos deseados en el formulario de la Práctica.			
6.	El Practicante Documentador verifica la modificación de la información ingresada en los campos del formulario de la Práctica y da clic en el botón “Save” para guardar los datos de la Práctica.	**7.**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de guardar los datos del formulario.	
8.	El Practicante Documentador da clic en el botón “Yes”.	**9.**	El sistema verifica los campos del formulario.	R1
		10.	El sistema guarda los datos del formulario en la base de datos y manda mensaje que fueron guardados exitosamente.	
		11.	El sistema regresa a la pantalla principal. Botones Habilitados: Modify	

			Botones Deshabilitados: Save, Cancel	

Flujo Alternativo 1:	**Modificar Práctica (cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
5.	El Practicante Documentador modifica los campos deseados en el formulario de la Práctica.			
6.1	El Practicante Documentador presiona el botón “Cancel”.	**7.1**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de cancelar la modificación de la Práctica.	
8.1	El Practicante Documentador presiona el botón “Yes”	**9.1**	El sistema regresa a la pantalla principal.	

Flujo Alternativo 2:	**Modificar Práctica (abortar cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
5.	El Practicante Documentador modifica los campos deseados en el formulario de la Práctica.			
6.2	El Practicante Documentador presiona el botón “Cancel”.	**7.2**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de cancelar la modificación de la Práctica.	
8.2	El Practicante Documentador presiona el botón “No”	**9.2**	El sistema regresa a la pantalla de edición del formulario de la Práctica.	

Rebanadas		
Identificador	**Nombre**	**Respuesta del Sistema**
R1	Campos obligatorios vacíos.	El sistema manda un mensaje al usuario indicándole que existen campos obligatorios sin información.

Caso de Uso: Ver Práctica

Actor: Practicante Documentador (Administrador).
Descripción: En este caso de uso se describe la forma en que el Practicante Documentador Ve la información de una Práctica en el sistema.
Pre-condiciones: El Practicante Documentador, deberá estar registrado en el sistema y haber iniciado sesión. El Practicante Documentador deberá haber seleccionado un Proyecto y un Método asociado al Proyecto, para que el sistema pueda mostrar las Prácticas asociadas al Método. Seleccionar la Práctica que desea revisar**.**
Pos-condiciones: El Practicante Ejecutor ve en la pantalla del sistema toda la información referente a la Práctica seleccionada.

Flujo de Eventos				
Flujo básico:	**Ver Práctica**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	Previamente el Practicante Documentador debió haber seleccionado un Proyecto y un Método asociado al Proyecto, para que el sistema pueda mostrar las Prácticas asociadas al Método.	**2.**	El sistema despliega un listado con las Prácticas asociadas al Método, en el campo de visualización de las Prácticas. Botones Habilitados: Modify Botones Deshabilitados: Save, Cancel	
3.	El Practicante Documentador selecciona una Práctica del listado.	**4.**	El sistema despliega el formulario (con los campos deshabilitados) con la información de la Práctica. Botones Habilitados: Modify Botones Deshabilitados: Save, Cancel	

Caso de Uso: Llenar Formulario de la Guía

Actor: Practicante Documentador (Administrador).
Descripción: En este caso de uso se describe la forma en que el Practicante Documentador llena el formulario de la Guía en el sistema.
Pre-condiciones: El Practicante Documentador, deberá estar registrado en el sistema y haber iniciado sesión. El Practicante Documentador, deberá haber dado clic en el botón Manages Guides de la interfaz Practice Data
Pos-condiciones: La Guía queda guardada con los datos que el Practicante haya escrito.

Flujo de Eventos				
Flujo básico:	**Llenar Formulario de la Guía**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	El Practicante Documentador da clic en el botón "Manages Guides" que se encuentra en la interfaz de Practice Data.	2.	El sistema despliega la interfaz "Guide Data" con el formulario con los campos de la Guía.	
3.	El Practicante Documentador llena los campos del formulario de la Guía: • Activities • Tasks • Tools • Knowledge and Skills			
4.	El Practicante Documentador verifica la información ingresada y da clic en el botón "Save Guide" para guardar los datos de la Guía.	5.	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de guardar los datos del formulario de la Guía.	
6.	El Practicante Documentador da clic en el botón "Yes".	7.	El sistema verifica los campos del formulario.	R1
		8.	El sistema guarda la Guía (datos del formulario) en la base de datos y manda mensaje que fueron guardados exitosamente.	

Flujo Alternativo 1:	**Llenar Formulario de la Guía (2 - N)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	El Practicante Documentador da clic en el botón "Manages Guides" que se encuentra en la interfaz de Practice Data.	2.	El sistema despliega la interfaz "Guide Data" con el formulario con los campos de la Guía.	
3.	El Practicante Documentador llena los campos del formulario de la Guía: • Activities • Tasks			

	• Tools • Knowledge and Skills			
4.1	El Practicante Documentador verifica la información ingresada y da clic en el botón "+" de las pestañas de las guías para agregar una nueva Guía y llenar los campos del formulario mostrados en la interfaz.	**5.1**	El sistema despliega el formulario con los campos de esta nueva Guía. (Los mismos campos para todas las Guías).	
6.1	El Practicante Documentador llena los campos del formulario de la Guía: • Activities • Tasks • Tools • Knowledge and Skills			
7.1	El Practicante Documentador verifica la información ingresada y da clic en el botón "Save Guide" para guardar los datos de la Guía.	**8.1**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de guardar los datos del formulario de la Guía.	
9.1	El Practicante Documentador da clic en el botón "Yes".	**10.1**	El sistema verifica los campos del formulario.	R1
		11.1	El sistema guarda la(s) Guía(s) (datos del formulario) en la base de datos y manda mensaje que fueron guardados exitosamente.	

Rebanadas		
Identificador	**Nombre**	**Respuesta del Sistema**
R1	Campos obligatorios vacíos.	El sistema manda un mensaje al usuario indicándole que existen campos obligatorios sin información.

Caso de Uso: Llenar Formulario de las Herramientas

Actor: Practicante Documentador (Administrador).
Descripción: En este caso de uso se describe la forma en que el Practicante Documentador llena el formulario de las Herramientas en el sistema.
Pre-condiciones: El Practicante Documentador, deberá haber dado clic en el botón Manages Tools de la interfaz Guide Data
Pos-condiciones: La Herramienta queda guardada con los datos que el Practicante haya escrito.

Flujo de Eventos				
Flujo básico:	**Llenar Formulario de las Herramientas**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	El Practicante Documentador da clic en el botón “Manages Tools” que se encuentra en la interfaz de Guide Data.	2.	El sistema despliega la interfaz “Tool Data” con el formulario con los campos de la Herramienta.	
3.	El Practicante Documentador llena los campos del formulario de la Herramienta: • Name • Version • Type • Description • Technical Specifications • Download URL			
4.	El Practicante Documentador verifica la información ingresada y da clic en el botón “Save Tool” para guardar los datos de la Herramienta.	5.	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de guardar los datos del formulario de la Herramienta.	
6.	El Practicante Documentador da clic en el botón “Yes”.	7.	El sistema verifica los campos del formulario.	R1
		8.	El sistema guarda la Herramienta (datos del formulario) en la base de datos y manda mensaje que fueron guardados exitosamente.	

Flujo Alternativo 1:	**Llenar Formulario de la Herramienta (2 - N)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	El Practicante Documentador da clic en el botón “Manages Tools” que se encuentra en la interfaz de Guide Data.	2.	El sistema despliega la interfaz “Tool Data” con el formulario con los campos de la Herramienta.	
3.	El Practicante Documentador llena los campos del formulario de la			

	Herramienta: • Name • Version • Type • Description • Technical Specifications • Download URL			
4.1	El Practicante Documentador verifica la información ingresada y da clic en el botón "+" de las pestañas de las Herramientas para agregar una nueva Herramienta y llenar los campos del formulario mostrados en la interfaz.	**5.1**	El sistema despliega el formulario con los campos de esta nueva Herramienta. (Los mismos campos para todas las Herramientas).	
6.1	El Practicante Documentador llena los campos del formulario de la Herramienta: • Name • Version • Type • Description • Technical Specifications • Download URL			
7.1	El Practicante Documentador verifica la información ingresada y da clic en el botón "Save Tool" para guardar los datos de la(s) Herramienta(s).	**8.1**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de guardar los datos del formulario de la Herramienta.	
9.1	El Practicante Documentador da clic en el botón "Yes".	**10.1**	El sistema verifica los campos del formulario.	R1
		11.1	El sistema guarda la(s) Herramienta(s) (datos del formulario) en la base de datos y manda mensaje que fueron guardados exitosamente.	

Rebanadas		
Identificador	**Nombre**	**Respuesta del Sistema**
R1	Campos obligatorios vacíos.	El sistema manda un mensaje al usuario indicándole que existen campos obligatorios sin información.

Caso de Uso: Ciclo de vida de las Prácticas

Actor: Practicante Documentador

Descripción: En estos casos de uso se describe la forma en que se le asignan diferentes estados a las Prácticas en un Proyecto.

DEFINICIÓN	
Caso de uso:	**"Pasar al estado" In-Execution (Practice in state Can-Start)**
Actor Principal:	Practicante Documentador *("ET" Equipo de Trabajo)*
Descripción:	La práctica en estado Can-Start "CS" cambia al estado In-Execution "IE".
Pre-condiciones:	Que una instancia de práctica esté en estado Can-Start "CS".
Pos-condiciones:	La instancia de práctica se encuentra en estado In-Execution "IE".

Flujo de Eventos				
Flujo básico:	**"Pasar al estado" In-Execution (Practice in state Can-Start)**			
Actor		**Sistema**		
Paso	**Acción**	**Paso**	**Acción**	**Rebanada**
1.	El *ET* elige una instancia de práctica en el menú "Practice List" que esté en estado Can-Start "CS".	**2.**	El Sistema marca seleccionada dicha práctica, "Practice CS".	
		3.	El Sistema despliega la información de la práctica instanciada que está en estado Can-Start "CS" que fue seleccionada.	
		4.	En el Sistema se visualiza que las entradas a la instancia de práctica ya están asignas, mostrando una "Palomita Verde" y un "Tache" en "Result".	
		5.	El Sistema habilita los botones "Modify", "Save" y "Cancel".	
		6.	El Sistema habilita el botón "+" para agregar al o los responsables.	
	El *ET* da clic en el botón "+" para agregar responsables.	**7.**	El Sistema despliega un cuadro emergente en donde se lista con check list los practitioners que participan en ese proyecto.	
8.	El *ET* selecciona al o los responsables de la actividad y da clic en el botón "Accept".	9.	El sistema muestra los nombres que se seleccionaron en el campo Responsible.	
10.	El *ET* da clic en el botón "Save".	**11.**	El sistema guarda al o los responsables "Responsible".	
		12.	El Sistema habilita el botón "In-	

			Execution", y el botón "Modify".	
13.	El *ET* da clic en el botón "In-Execution".	**14.**	El Sistema muestra en la parte de "Practice List" que la instancia de práctica cambia de estado a In-Execution "IE".	
		15.	El Sistema deshabilita los botones de la parte superior y en la parte de información de práctica no muestra nada.	

DEFINICIÓN	
Caso de uso:	**Cancelled**
Actor Principal:	Practicante Documentador *("ET" Equipo de Trabajo)*
Descripción:	La instancia de práctica en estado In-Execution "IE" cambia al estado Cancelled "C".
Pre-condiciones:	Que una instancia de práctica esté en estado In-Execution "IE".
Pos-condiciones:	La instancia de práctica se encuentra en estado Cancelled "C".

Flujo de Eventos				
Flujo básico:		**Cancelled**		
Actor		**Sistema**		
Paso	**Acción**	**Paso**	**Acción**	**Rebanada**
1.	El *ET* elige una instancia de práctica en el menú "Practice List" que esté en estado In-Execution "IE".	**2.**	El Sistema marca seleccionada dicha práctica, In-Execution "IE".	
		3.	El Sistema despliega la información de la práctica instanciada que está en estado In-Execution "IE" que fue seleccionada.	
		4.	En el Sistema se visualiza que las entradas a la instancia de práctica ya están asignas, mostrando una "Palomita Verde" y un "Tache" en "Result".	
		5.	El Sistema habilita los botones "In-Verification", "Stand-By", "Cancelled" y "Modify".	
6.	El *ET* da clic en el Botón "Cancelled".	**7.**	El Sistema muestra en la parte de "Practice List" que la instancia de práctica cambia de	

			estado a Cancelled "C".	
		8.	El Sistema deshabilita los botones de la parte superior y en la parte de información de práctica no muestra nada.	

DEFINICIÓN	
Caso de uso:	**Stand-By**
Actor Principal:	Practicante Documentador *("ET" Equipo de Trabajo)*
Descripción:	La instancia de práctica en estado In-Execution "IE" cambia al estado Stand-By "SB".
Pre-condiciones:	Que una instancia de práctica esté en estado In-Execution "IE".
Pos-condiciones:	La instancia de práctica se encuentra en estado Stand-By "SB".

Flujo de Eventos				
Flujo básico:	**Stand-By**			
Actor		**Sistema**		
Paso	**Acción**	**Paso**	**Acción**	**Rebanada**
1.	El *ET* elige una instancia de práctica en el menú "Practice List" que esté en estado In-Execution "IE".	2.	El Sistema marca seleccionada dicha práctica, In-Execution "IE".	
		3.	El Sistema despliega la información de la práctica instanciada que está en estado In-Execution "IE" que fue seleccionada.	
		4.	En el Sistema se visualiza que las entradas a la instancia de práctica ya están asignas, mostrando una "Palomita Verde" y un "Tache" en "Result".	
		5.	El Sistema habilita los botones "In-Verification", "Stand-By", "Cancelled" y "Modify".	
6.	El *ET* da clic en el botón "Stand By".	7.	El Sistema muestra en la parte de "Practice List" que la instancia de práctica cambia de estado a Stand By "SB".	

		8.	El Sistema deshabilita los botones de la parte superior y en la parte de información de práctica no muestra nada.	

DEFINICIÓN	
Caso de uso:	**In-Execution (Estando en Stand-By)**
Actor Principal:	Practicante Documentador *("ET" Equipo de Trabajo)*
Descripción:	La instancia de práctica en estado Stand-By "SB" cambia al estado In-Execution "IE".
Pre-condiciones:	Que una instancia de práctica esté en estado Stand-By "SB".
Pos-condiciones:	La instancia de práctica se encuentra en estado In-Execution "IE".

Flujo de Eventos				
Flujo básico:		**In-Execution (Estando en Stand-By)**		
Actor		**Sistema**		
Paso	**Acción**	**Paso**	**Acción**	**Rebanada**
1.	El *ET* elige una instancia de práctica en el menú "Practice List" que esté en estado Stand-By "SB".	2.	El Sistema marca seleccionada dicha práctica, Stand-By "SB".	
		3.	El Sistema despliega la información de la práctica instanciada que está en estado Stand-By "SB" que fue seleccionada.	
		4.	En el Sistema se visualiza que las entradas a la instancia de práctica ya están asignas, mostrando una "Palomita Verde" y un "Tache" en "Result".	
		5.	El Sistema habilita el "In-Execution".	
6.	El *ET* da clic en el botón "IE" para reiniciar la instancia de práctica a In-Excution "IE".	7.	El Sistema muestra en la parte de "Practice List" que la instancia de práctica cambia de estado a In-Execution "IE".	
		8.	El Sistema deshabilita los botones de la parte superior y en la parte de información de práctica no muestra nada.	

DEFINICIÓN	
Caso de uso:	**In-Verification**
Actor Principal:	Practicante Documentador *(“ET” Equipo de Trabajo)*
Descripción:	La instancia de práctica en estado In-Execution “IE” cambia al estado In-Verification “IV”.
Pre-condiciones:	Que una instancia de práctica esté en estado In-Execution “IE”.
Pos-condiciones:	La instancia de práctica se encuentra en estado In-Verification “IV”.

Flujo de Eventos				
Flujo básico:	**In-Verification**			
Actor		**Sistema**		
Paso	**Acción**	**Paso**	**Acción**	**Rebanada**
1.	El *ET* elige una instancia de práctica en el menú “Practice List” que esté en estado In-Execution “IE”.	**2.**	El Sistema marca seleccionada dicha práctica, In-Execution “IE”.	
		3.	El Sistema despliega la información de la práctica instanciada que está en estado In-Execution “IE” que fue seleccionada.	
		4.	En el Sistema se visualiza que las entradas a la instancia de práctica ya están asignas, mostrando una “Palomita Verde” y un “Tache” en “Result”.	
		5.	El Sistema habilita los botones “In-Verification”, “Stand-By”, “Cancelled” y “Modify”.	
6.	El *ET* da clic en el botón “In-Verification”.	**7.**	El Sistema muestra en la parte de “Practice List” que la instancia de práctica cambia de estado a In-Verification “IV”.	
		8.	El Sistema deshabilita los botones de la parte superior y en la parte de información de práctica no muestra nada.	

DEFINICIÓN	
Caso de uso:	**In-Execution (Partiendo en verificación, WT verifies result as incorrect)**
Actor Principal:	Practicante Documentador *("ET" Equipo de Trabajo)*
Descripción:	La instancia de práctica en estado In-Verification "IV" pero el resultado no es correcto entonces cambia al estado In-Execution "IE".
Pre-condiciones:	Que una instancia de práctica esté en estado In-Verification "IV".
Pos-condiciones:	La instancia de práctica se encuentra en estado In-Execution "IE".

Flujo de Eventos				
Flujo básico:	**In-Execution (Partiendo en verificación, WT verifies result as incorrect)**			
Actor		**Sistema**		
Paso	**Acción**	**Paso**	**Acción**	**Rebanada**
1.	El *ET* elige una instancia de práctica en el menú "Practice List" que esté en estado In-Verification "IV".	**2.**	El Sistema marca seleccionada dicha práctica, In-Verification "IV".	
		3.	El Sistema despliega la información de la práctica instanciada que está en estado In-Verification "IV".	
		4.	En el Sistema se visualiza que las entradas a la instancia de práctica ya están asignas, mostrando una "Palomita Verde" y un "Palomita Amarilla" en "Result" que los resultados se están verificando.	
5.	El *ET* verifica que los resultados son incorrectos y da clic en el botón "In-Execution".	**6.**	El Sistema muestra en la parte de "Practice List" que la instancia de práctica cambia de estado a In-Execution "IE".	
		7.	El Sistema deshabilita los botones de la parte superior y no muestra ninguna información en la página principal.	

DEFINICIÓN	
Caso de uso:	**Finished**
Actor Principal:	Practicante Documentador *(“ET” Equipo de Trabajo)*
Descripción:	La Instanciada de práctica In-Verification “IV”, el resultado de la práctica es correcto entonces se cambia al estado Finished “F”.
Precondiciones:	Que una instancia de práctica esté en estado “In-Verification”.
Poscondiciones:	La instancia de práctica se encuentra en estado Finished “F”.

Flujo de Eventos				
Flujo básico:	**Finished**			
Actor		**Sistema**		
Paso	**Acción**	**Paso**	**Acción**	**Rebanada**
1.	El *ET* elige una instancia de práctica en el menú “Practice List” que esté en estado In-Verification “IV”.	**2.**	El Sistema marca seleccionada dicha práctica, In-Verification “IV”.	
		3.	El Sistema despliega la información de la práctica instanciada que está en estado In-Verification “IV”.	
		4.	En el Sistema se visualiza que las entradas a la instancia de práctica ya están asignas, mostrando una “Palomita Verde” y un “Palomita Amarilla” en “Result” que los resultados se están verificando.	
5.	El *ET* verifica que los resultados son correctos y da clic en el botón “Finished”.	**6**	Si la salida de esta practica corresponde a las entradas de otra practica N y con esto todas las entradas de la practica N están listas entonces cambia de estado a Can-Start “CS” la practica N.	
		7.	El Sistema muestra en la parte de “Practice List” que la instancia de práctica cambia de estado Finished “F” y la practica N cambia a estado “CS”.	
		8.	El Sistema deshabilita los botones de la parte superior y no muestra ninguna información en la página principal.	

Caso de Uso: Agregar Practicante

Actor: Practicante Ejecutor
Descripción: En este caso de uso se describe la forma en que el Practicante Ejecutor Agrega un nuevo Practicante al sistema.
Pre-condiciones: El Practicante Ejecutor, deberá estar en la pantalla principal del sistema y seleccionar la opción "Practitioner", la cual lo enviará a la pantalla del Practicante.
Pos-condiciones: El Practicante queda guardado en la base de datos del sistema, listo para que sea elegido a formar parte de un equipo de trabajo.

Flujo de Eventos				
Flujo básico:	**ADD Practicante**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	El Practicante Ejecutor da clic en el botón "Practitioner" de la pantalla principal.	**2.**	El sistema despliega la pantalla principal del "Practicante"; con el listado de los Practicantes que se encuentran en la base de datos del sistema. Botones Habilitados: Save, Cancel Botones Deshabilitados: ADD New Practitioner, Modify, Remove	
3.	El Practicante Ejecutor da clic en el botón "ADD New Practitioner".	**4.**	El sistema despliega el formulario con los campos para dar de alta a un Practicante. Botones Habilitados: Save, Cancel Botones Deshabilitados: ADD New Practitioner, Modify, Remove	
5.	El Practicante Ejecutor llena los campos del formulario para agregar un Practicante: • First Name • Last Name • Identifier • E-Mail • Knowledge & Skills			
6.	El Practicante Ejecutor verifica la información ingresada y da clic en el botón "Save" para guardar los datos del Practicante.	**7.**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de guardar los datos del formulario.	
8.	El Practicante ejecutor da clic en el botón "Yes".	**9.**	El sistema verifica los campos del formulario.	R1

		10.	El sistema guarda los datos del formulario en la base de datos y manda mensaje que fueron guardados exitosamente.	
		11.	El sistema regresa a la pantalla principal del "Practicante". Botones Habilitados: ADD New Practitioner, Cancel Botones Deshabilitados: Modify, Remove, Save	

Flujo Alternativo 1:	**ADD Practicante (cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
5.	El Practicante Ejecutor llena los campos del formulario para agregar un Practicante: • First Name • Last Name • Identifier • E-Mail • Knowledge & Skills			
6.1	El Practicante Ejecutor presiona el botón "Cancel".	**7.1**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de cancelar el dar de alta al Practicante.	
8.1	El Practicante Ejecutor presiona el botón "Yes".	**9.1**	El sistema regresa a la pantalla principal del "Practicante".	

Flujo Alternativo 2:	**ADD Practicante (abortar cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
5.	El Practicante Ejecutor llena los campos del formulario para agregar un Practicante: • First Name • Last Name • Identifier • E-Mail • Knowledge & Skills			
6.2	El Practicante Ejecutor presiona el botón "Cancel".	**7.2**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de cancelar el dar de alta al Practicante.	

8.2	El Practicante Ejecutor presiona el botón “No”	**9.2**	El sistema regresa a la pantalla de edición del formulario para dar de alta al Practicante. Botones Habilitados: Save, Cancel Botones Deshabilitados: ADD New Partitioner, Modify, Remove	

Rebanadas		
Identificador	**Nombre**	**Respuesta del Sistema**
R1	Campos obligatorios vacíos.	El sistema manda un mensaje al usuario indicándole que existen campos obligatorios sin información.

Caso de Uso: Modificar Practicante

Actor: Practicante Ejecutor
Descripción: En este caso de uso se describe la forma en que el Practicante Ejecutor Modifica la información de un Practicante en el sistema.
Pre-condiciones: El Practicante Ejecutor, deberá estar en la pantalla principal del sistema y seleccionar la opción "Practitioner", la cual lo enviará a la pantalla del Practitioner.
Pos-condiciones: La modificación de los datos del Practicante queda guardada en la base de datos del sistema.

Flujo de Eventos				
Flujo básico:	**Modificar Practicante**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	El Practicante Ejecutor da clic en el botón "Practitioner" de la pantalla principal.	**2.**	El sistema despliega la pantalla principal del "Practicante"; con el listado de los Practicantes que se encuentran en la base de datos del sistema. Botones Habilitados: ADD New Practitioner, Cancel Botones Deshabilitados: Modify, Remove, Save	
3.	El Practicante Ejecutor selecciona un Practicante de la lista mostrada por el sistema.	**4.**	El sistema despliega el formulario (con los campos deshabilitados) con la información del Practicante seleccionado. Botones Habilitados: Modify, Remove , Cancel Botones Deshabilitados: ADD New Practitioner, Save	
5.	El Practicante Ejecutor da clic en el botón "Modify".	**6.**	El sistema habilita los campos del formulario del Practicante para su modificación. Botones Habilitados: Save, Cancel Botones Deshabilitados: ADD New Practitioner, Modify, Remove	
7.	El Practicante Ejecutor modifica los campos deseados en el formulario del Practicante.			
8.	El Practicante Ejecutor verifica la modificación de la información ingresada en los campos del formulario del Practicante y da clic en el botón "Save" para guardar los datos del Practicante.	**9.**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de guardar los datos del formulario.	

10.	El Practicante ejecutor da clic en el botón "Yes".	**11.**	El sistema verifica los campos del formulario.	R1
		12.	El sistema guarda los datos del formulario en la base de datos y manda mensaje que fueron guardados exitosamente.	
		13.	El sistema regresa a la pantalla principal del "Practicante". Botones Habilitados: ADD New Practitioner, Cancel Botones Deshabilitados: Modify, Remove, Save	

Flujo Alternativo 1:	**Modificar Practicante (cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
7.	El Practicante Ejecutor modifica los campos deseados en el formulario del Practicante.			
8.1	El Practicante Ejecutor presiona el botón "Cancel".	**9.1**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de cancelar la modificación de los datos del Practicante.	
10.1	El Practicante Ejecutor presiona el botón "Yes".	**11.1**	El sistema regresa a la pantalla principal del "Practicante". Botones Habilitados: ADD New Practitioner, Cancel Botones Deshabilitados: Modify, Remove, Save	

Flujo Alternativo 2:	**Modificar Practicante (abortar cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
7.	El Practicante Ejecutor modifica los campos deseados en el formulario del Practicante.			
8.2	El Practicante Ejecutor presiona el botón "Cancel".	**9.2**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de cancelar la modificación de los datos del Practicante.	
10.2	El Practicante Ejecutor presiona el botón "No"	**11.2**	El sistema regresa a la pantalla de modificación del formulario	

			del “Practicante”. Botones Habilitados: Save, Cancel Botones Deshabilitados: ADD New Practitioner, Modify, Remove	

Rebanadas		
Identificador	**Nombre**	**Respuesta del Sistema**
R1	Campos obligatorios vacíos.	El sistema manda un mensaje al usuario indicándole que existen campos obligatorios sin información.

Caso de Uso: Ver Practicante

Actor: Practicante Ejecutor
Descripción: En este caso de uso se describe la forma en que el Practicante Ejecutor ve la información de un Practicante en el sistema.
Pre-condiciones: El Practicante Ejecutor, deberá estar en la pantalla principal del sistema y seleccionar la opción "Practitioner", la cual lo enviará a la pantalla principal del Practitioner.
Pos-condiciones: El Practicante Ejecutor ve en la pantalla del sistema toda la información referente al Practicante seleccionado.

Flujo de Eventos				
Flujo básico:	**Ver Practicante**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	El Practicante Ejecutor da clic en el botón "Practitioner" de la pantalla principal.	**2.**	El sistema despliega la pantalla principal del "Practicante"; con el listado de los Practicantes que se encuentran en la base de datos del sistema. Botones Habilitados: ADD New Practitioner, Cancel Botones Deshabilitados: Modify, Remove, Save	
3.	El Practicante Ejecutor selecciona un Practicante de la lista mostrada por el sistema.	**4.**	El sistema despliega el formulario (con los campos deshabilitados) con la información del Practicante seleccionado. Botones Habilitados: Modify, Remove, Cancel Botones Deshabilitados: ADD New Practitioner, Save	
5.	El Practicante Ejecutor da clic en el botón "Cancel".	**6.**	El sistema regresa a la pantalla principal del "Practicante".	

Caso de Uso: Remover (Eliminar) Practicante

Actor: Practicante Ejecutor
Descripción: En este caso de uso se describe la forma en que el Practicante Ejecutor Elimina de la lista de Practicantes activos de la organización, a un Practicante de la lista.
Pre-condiciones: El Practicante Ejecutor, deberá estar en la pantalla principal del sistema y seleccionar la opción "Practitioner", la cual lo enviará a la pantalla principal del Practitioner.
Pos-condiciones: Se elimina al Practicante de la lista de Practicantes activos del sistema.

Flujo de Eventos				
Flujo básico:	**Remove Practicante**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	El Practicante Ejecutor da clic en el botón "Practitioner" de la pantalla principal.	**2.**	El sistema despliega la pantalla principal del "Practicante"; con el listado de los Practicantes que se encuentran en la base de datos del sistema. Botones Habilitados: ADD New Practitioner, Cancel Botones Deshabilitados: Modify, Remove, Save	
3.	El Practicante Ejecutor selecciona un a Practicante de la lista mostrada por el sistema.	**4.**	El sistema despliega el formulario (con los campos deshabilitados) con la información del Practicante seleccionado. Botones Habilitados: Modify, Remove, Cancel Botones Deshabilitados: ADD New Practitioner, Save	
5.	El Practicante Ejecutor da clic en el botón "Remove".	**6.**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de eliminar de la lista de Practicantes activos al Practicante seleccionado.	
7.	El Practicante ejecutor da clic en el botón "Yes".	**8.**	El sistema elimina/da de baja al Practicante.	
		9.	El sistema regresa a la pantalla principal del "Practicante". Botones Habilitados: ADD New Practitioner, Cancel Botones Deshabilitados: Modify, Remove, Save	

Flujo Alternativo 1:		**Remove Practicante (cancelar)**		
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
5.	El Practicante Ejecutor da clic en el botón “Remove”.	**6.**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de eliminar de la lista de Practicantes activos al Practicante seleccionado.	
7.1	El Practicante Ejecutor presiona el botón “Cancel”.	**8.1**	El sistema regresa a la pantalla principal del “Practicante”. Botones Habilitados: ADD New Practitioner, Cancel Botones Deshabilitados: Modify, Remove, Save	

Caso de Uso: Agregar Stakeholder

Actor: Practicante Ejecutor
Descripción: En este caso de uso se describe la forma en que el Practicante Ejecutor Agrega un nuevo Stakeholder al sistema.
Pre-condiciones: El Practicante Ejecutor, deberá estar en la pantalla principal del sistema y seleccionar la opción "Stakeholder", la cual lo enviará a la pantalla del Stakeholder.
Pos-condiciones: El Stakeholder queda guardado en la base de datos del sistema, listo para que sea elegido para formar parte de un proyecto.

Flujo de Eventos				
Flujo básico:	**Agregar Stakeholder**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	El Practicante Ejecutor da clic en el botón "Stakeholder" de la pantalla principal.	**2.**	El sistema despliega la pantalla principal del "Stakeholder"; con el listado de los Stakeholders que se encuentran en la base de datos del sistema. Botones Habilitados: Save, Cancel Botones Deshabilitados: ADD New Stakeholder, Modify, Remove	
3.	El Practicante Ejecutor da clic en el botón "ADD New Stakeholder".	**4.**	El sistema despliega el formulario con los campos para dar de alta a un Stakeholder. Botones Habilitados: Save, Cancel Botones Deshabilitados: ADD New Stakeholder, Modify, Remove	
5.	El Practicante Ejecutor llena los campos del formulario para agregar un Stakeholder: • First Name • Last Name • Identifier • E-Mail • Rol			
6.	El Practicante Ejecutor verifica la información ingresada y da clic en el botón "Save" para guardar los datos del Stakeholder.	**7.**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de guardar los datos del formulario.	
8.	El Practicante Ejecutor da clic en el botón "Yes".	**9.**	El sistema verifica los campos del formulario.	R1

		10.	El sistema guarda los datos del formulario en la base de datos y manda mensaje que fueron guardados exitosamente.	
		11.	El sistema regresa a la pantalla principal del "Stakeholder". Botones Habilitados: ADD New Stakeholder, Cancel Botones Deshabilitados: Modify, Remove, Save	

Flujo Alternativo 1:	**Agregar Stakeholder (cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
5.	El Practicante Ejecutor llena los campos del formulario para agregar un Stakeholder: • First Name • Last Name • Identifier • E-Mail • Rol			
6.1	El Practicante Ejecutor presiona el botón "Cancel".	**7.1**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de cancelar el dar de alta al Stakeholder.	
8.1	El Practicante Ejecutor presiona el botón "Yes".	**9.1**	El sistema regresa a la pantalla principal del "Stakeholder".	

Flujo Alternativo 2:	**Agregar Stakeholder (abortar cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
5.	El Practicante Ejecutor llena los campos del formulario para agregar un Practicante: • First Name • Last Name • Identifier • E-Mail • Rol			
6.2	El Practicante Ejecutor presiona el botón "Cancel".	**7.2**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de cancelar el dar de alta al Stakeholder.	

8.2	El Practicante Ejecutor presiona el botón “No”	**9.2**	El sistema regresa a la pantalla de edición del formulario para dar de alta Stakeholder. Botones Habilitados: Save, Cancel Botones Deshabilitados: ADD New Stakeholder, Modify, Remove	

Rebanadas		
Identificador	**Nombre**	**Respuesta del Sistema**
R1	Campos obligatorios vacíos.	El sistema manda un mensaje al usuario indicándole que existen campos obligatorios sin información.

Caso de Uso: Modificar Stakeholder

Actor: Practicante Ejecutor
Descripción: En este caso de uso se describe la forma en que el Practicante Ejecutor Modifica la información de un Stakeholder en el sistema.
Pre-condiciones: El Practicante Ejecutor, deberá estar en la pantalla principal del sistema y seleccionar la opción "Stakeholder", la cual lo enviará a la pantalla del Stakeholder.
Pos-condiciones: La modificación de los datos del Stakeholder queda guardada en la base de datos del sistema

Flujo de Eventos				
Flujo básico:	**Modificar Stakeholder**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	El Practicante Ejecutor da clic en el botón "Stakeholder" de la pantalla principal.	**2.**	El sistema despliega la pantalla principal del "Stakeholder"; con el listado de los Stakeholders que se encuentran en la base de datos del sistema. Botones Habilitados: ADD New Stakeholder, Cancel Botones Deshabilitados: Modify, Remove, Save	
3.	El Practicante Ejecutor selecciona un Stakeholder de la lista mostrada por el sistema.	**4.**	El sistema despliega el formulario (con los campos deshabilitados) con la información del Stakeholder seleccionado. Botones Habilitados: Modify, Remove , Cancel Botones Deshabilitados: ADD New Stakeholder, Save	
5.	El Practicante Ejecutor da clic en el botón "Modify".	**6.**	El sistema habilita los campos del formulario del Stakeholder para su modificación. Botones Habilitados: Save, Cancel Botones Deshabilitados: ADD New Stakeholder, Modify, Remove	
7.	El Practicante Ejecutor modifica los campos deseados en el formulario del Stakeholder.			
8.	El Practicante Ejecutor verifica la modificación de la información ingresada en los campos del formulario del Stakeholder y da clic en el botón "Save" para guardar los datos del Stakeholder.	**9.**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de guardar los datos del formulario.	

10.	El Practicante ejecutor da clic en el botón “Yes”.	**11.**	El sistema verifica los campos del formulario.	R1
		12.	El sistema guarda los datos del formulario en la base de datos y manda mensaje que fueron guardados exitosamente.	
		13.	El sistema regresa a la pantalla principal del “Stakeholder”. Botones Habilitados: ADD New Stakeholder, Cancel Botones Deshabilitados: Modify, Remove, Save	

Flujo Alternativo 1:	**Modificar Stakeholder (cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
7.	El Practicante Ejecutor modifica los campos deseados en el formulario del Stakeholder.			
8.1	El Practicante Ejecutor presiona el botón “Cancel”.	**9.1**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de cancelar la modificación de los datos del Stakeholder.	
10.1	El Practicante Ejecutor presiona el botón “Yes”.	**11.1**	El sistema regresa a la pantalla principal del “Stakeholder”. Botones Habilitados: ADD New Stakeholder, Cancel Botones Deshabilitados: Modify, Remove, Save	

Flujo Alternativo 2:	**Modificar Stakeholder (abortar cancelar)**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
7.	El Practicante Ejecutor modifica los campos deseados en el formulario del Stakeholder.			
8.2	El Practicante Ejecutor presiona el botón “Cancel”.	**9.2**	El sistema despliega un cuadro de diálogo donde pregunta si está seguro de cancelar la modificación de los datos del Stakeholder.	
10.2	El Practicante Ejecutor presiona el botón “No”	**11.2**	El sistema regresa a la pantalla de modificación del formulario	

			del “Stakeholder”. Botones Habilitados: Save, Cancel Botones Deshabilitados: ADD New Stakeholder, Modify, Remove	

Rebanadas		
Identificador	**Nombre**	**Respuesta del Sistema**
R1	Campos obligatorios vacíos.	El sistema manda un mensaje al usuario indicándole que existen campos obligatorios sin información.

Caso de Uso: Ver Stakeholder

Actor: Practicante Ejecutor
Descripción: En este caso de uso se describe la forma en que el Practicante Ejecutor ve la información de un Stakeholder en el sistema.
Pre-condiciones: El Practicante Ejecutor, deberá estar en la pantalla principal del sistema y seleccionar la opción "Stakeholder", la cual lo enviará a la pantalla principal del Stakeholder.
Pos-condiciones: El Practicante Ejecutor ve en la pantalla del sistema toda la información referente al Stakeholder seleccionado.

Flujo de Eventos				
Flujo básico:	**Ver Stakeholder**			
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Rebanada**
1.	El Practicante Ejecutor da clic en el botón "Stakeholder" de la pantalla principal.	**2.**	El sistema despliega la pantalla principal del "Stakeholder"; con el listado de los Stakeholders que se encuentran en la base de datos del sistema. Botones Habilitados: ADD New Stakeholder, Cancel Botones Deshabilitados: Modify, Remove, Save	
3.	El Practicante Ejecutor selecciona un Stakeholder de la lista mostrada por el sistema.	**4.**	El sistema despliega el formulario (con los campos deshabilitados) con la información del Stakeholder seleccionado. Botones Habilitados: Modify, Remove, Cancel Botones Deshabilitados: ADD New Stakeholder, Save	
5.	El Practicante Ejecutor da clic en el botón "Cancel".	**6.**	El sistema regresa a la pantalla principal del "Stakeholder".	

Apéndice II Escenarios de atributos de calidad

Escenarios de atributos de calidad

Escenario crudo:	*Alertas sobre los estados de (proyecto, método, práctica)*
Objetivos de negocio correspondientes:	*El sistema soporta KUALI-BEH*
Atributos de calidad relevantes:	*Interoperabilidad, Usabilidad, Desempeño, Funcionalidad*
Estímulo:	*Cambio de estado en método o práctica en el tablero de control*
Fuente de estímulo:	*Practicantes*
Entorno:	*Operación normal durante la realización de un proyecto*
Artefacto:	*Entorno Computacional de KUALI-BEH*
Respuesta:	*Alerta a los practicantes del proyecto*
Medida de la respuesta:	*3 o 4 segundo después del cambio de estado.*

Escenario crudo:	*Asociar una práctica a un método cuando se desea definir un método.*
Objetivos de negocio correspondientes:	*El sistema soporta KUALI-BEH*
Atributos de calidad relevantes:	*Usabilidad, Funcionalidad*
Estímulo:	*Componer Método.* *Establecer un método mediante la selección de prácticas disponibles en el sistema.* *Tener una colección de prácticas a seleccionar.* *Tener seleccionado un método.*
Fuente de estímulo:	*Practicantes*
Entorno:	*Operación normal.*
Artefacto:	*Entorno Computacional de KUALI-BEH*
Respuesta:	*Las prácticas quedan asociadas a un método.*
Medida de la respuesta:	*2 a 3 segundos después de asignar una práctica al método seleccionado.*

Escenario crudo:	*Que el sistema se pueda visualizar en varios lenguajes*
Objetivos de negocio correspondientes:	*El sistema soporta KUALI-BEH*
Atributos de calidad relevantes:	*Usabilidad, Funcionalidad*
Estímulo:	*Selección de opción de idioma.*
Fuente de estímulo:	*Usuario que Instala el Sistema.*
Entorno:	*Durante la Instalación.*
Artefacto:	*Programa de Instalación del Sistema KUALI-BEH.*
Respuesta:	*El Entorno Computacional de KUALI-BEH queda instalado con el idioma seleccionado.*
Medida de la respuesta:	*Tiempo de Instalación será el mismo independientemente del idioma.*

Escenario crudo:	*Que asociar un método a un proyecto no sean más de dos pantallas.*
Objetivos de negocio correspondientes:	*El Entorno Computacional soporta KUALI-BEH*
Atributos de calidad relevantes:	*Usabilidad, Desempeño, Funcionalidad*
Estímulo:	*Asociar Método a Proyecto.* *Composición de proyecto por uno o varios métodos.* *Haber seleccionado un proyecto previamente.* *Decidir de una colección de métodos aquellos que vayan acorde al proyecto creado y asociarlos.*
Fuente de estímulo:	*Practicantes*
Entorno:	*Operación normal al momento de definir los métodos que conformarán un proyecto.*
Artefacto:	*Entorno Computacional de KUALI-BEH*
Respuesta:	*El proyecto tendrá asociado uno o varios métodos correspondientes con su objetivo.*
Medida de la respuesta:	*En a lo más dos pantallas y el tiempo de respuesta debe ser a lo más 3 segundos por cada asociación.*

Escenario crudo:	*Varios Usuarios definan una misma práctica*
Objetivos de negocio correspondientes:	*El Entorno Computacional soporta KUALI-BEH*
Atributos de calidad relevantes:	*Concurrencia, Usabilidad, Desempeño, Funcionalidad*
Estímulo:	*Caso de Uso Colaboración para definir una practica. Documentación de la definición de una práctica de la forma de trabajo utilizando la tecnología de la "mesa colaborativa". Fase 1. Inserción individual de los formularios (práctica, guías, herramientas) Fase 2. Discusión de la información de cada integrante del equipo Fase 3. Unión en el documento final (colaborativa) y salvar*
Fuente de estímulo:	*Usuarios*
Entorno:	*Operación normal con un equipo de 4 personas*
Artefacto:	*Entorno Colaborativo*
Respuesta:	*Practica definida y guardada.*
Medida de la respuesta:	*En cuatro pantallas y el tiempo de respuesta es menor a 5 segundos por intervención.*

Escenario crudo:	*El sistema pueda interactuar con otro software de administración de proyectos*
Objetivos de negocio correspondientes:	*El Entorno Computacional soporta KUALI-BEH*
Atributos de calidad relevantes:	*Usabilidad, Interoperabilidad, Funcionalidad*
Estímulo:	*Generación de un nuevo plan de proyecto en otro Sistema a partir de la información ya almacenada en el tablero de proyecto de KUALI-BEH.*
Fuente de estímulo:	*Usuarios*
Entorno:	*Operación normal, cuando se desee exportar la información almacenada en el tablero de control de proyectos.*
Artefacto:	*Entorno Computacional de KUALI-BEH*
Respuesta:	*Generación de un nuevo archivo con formato compatible para otro sistema.*
Medida de la respuesta:	*En solo una pantalla y tiempo de respuesta menor a 3 segundos*

Escenario crudo:	*Manipulación de versiones (se actualiza la versión de una práctica y su método asociado)*
Objetivos de negocio correspondientes:	*El Entorno Computacional soporta KUALI-BEH*
Atributos de calidad relevantes:	*Usabilidad, Desempeño, Modificabilidad, Funcionalidad*
Estímulo:	*Modificación de una práctica o un método*
Fuente de estímulo:	*Practicantes*
Entorno:	*Operación normal durante la realización de un proyecto*
Artefacto:	*Entorno Computacional de KUALI-BEH*
Respuesta:	*Se crea una nueva práctica con las modificaciones hechas y se guarda con la nueva versión y automáticamente cambia la versión del método al cual esta asociada la práctica.*
Medida de la respuesta:	*Tiempo menor a 3 segundos.*

Apéndice III Diagramas de Comunicación

Diagrama de Comunicación “Administrar Catálogos”

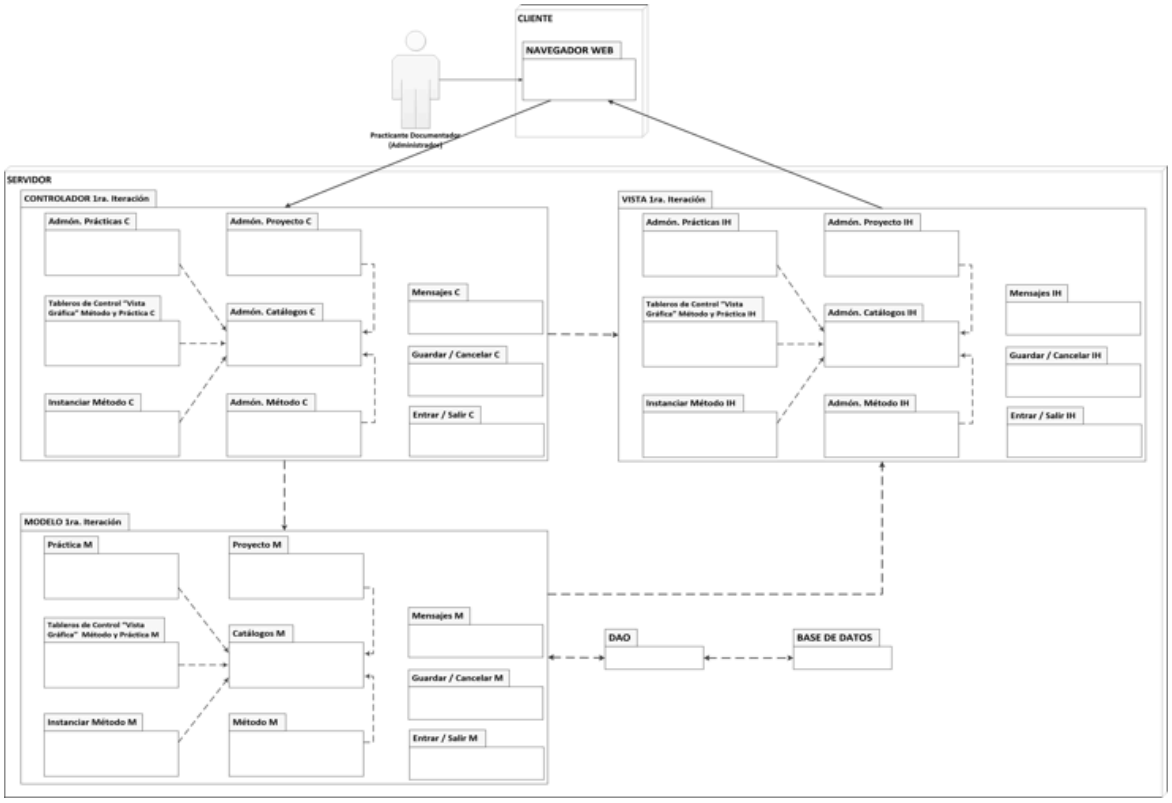


Diagrama de Comunicación “Administrar Método”

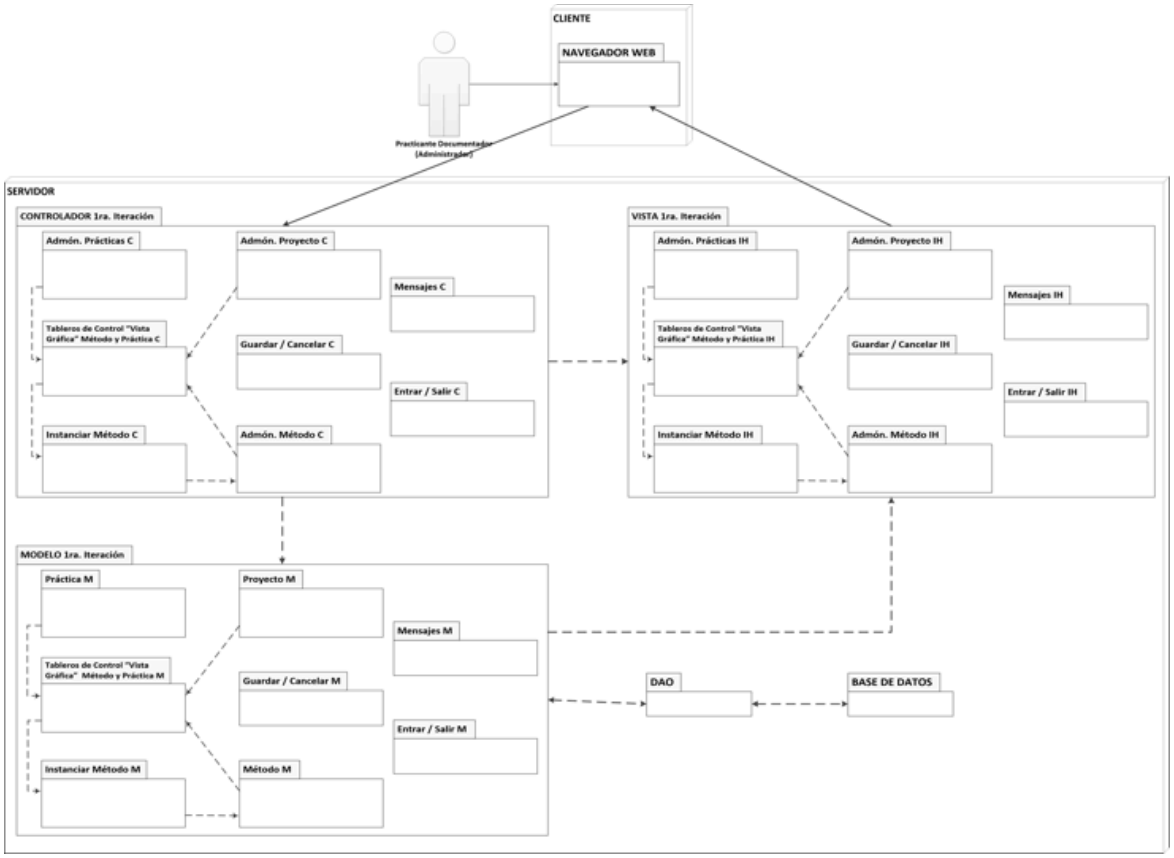


Diagrama de Comunicación “Instanciar Método”

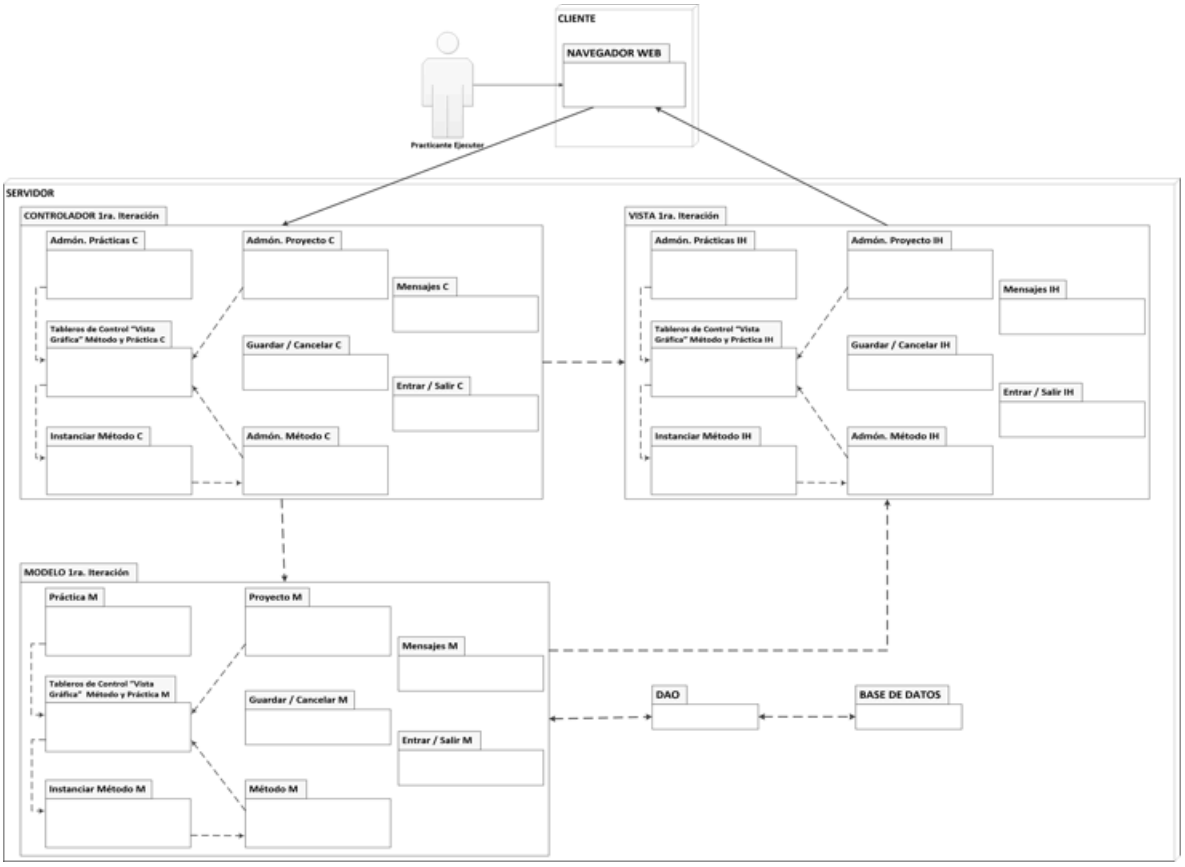


Diagrama de Comunicación “Administrar Proyecto”

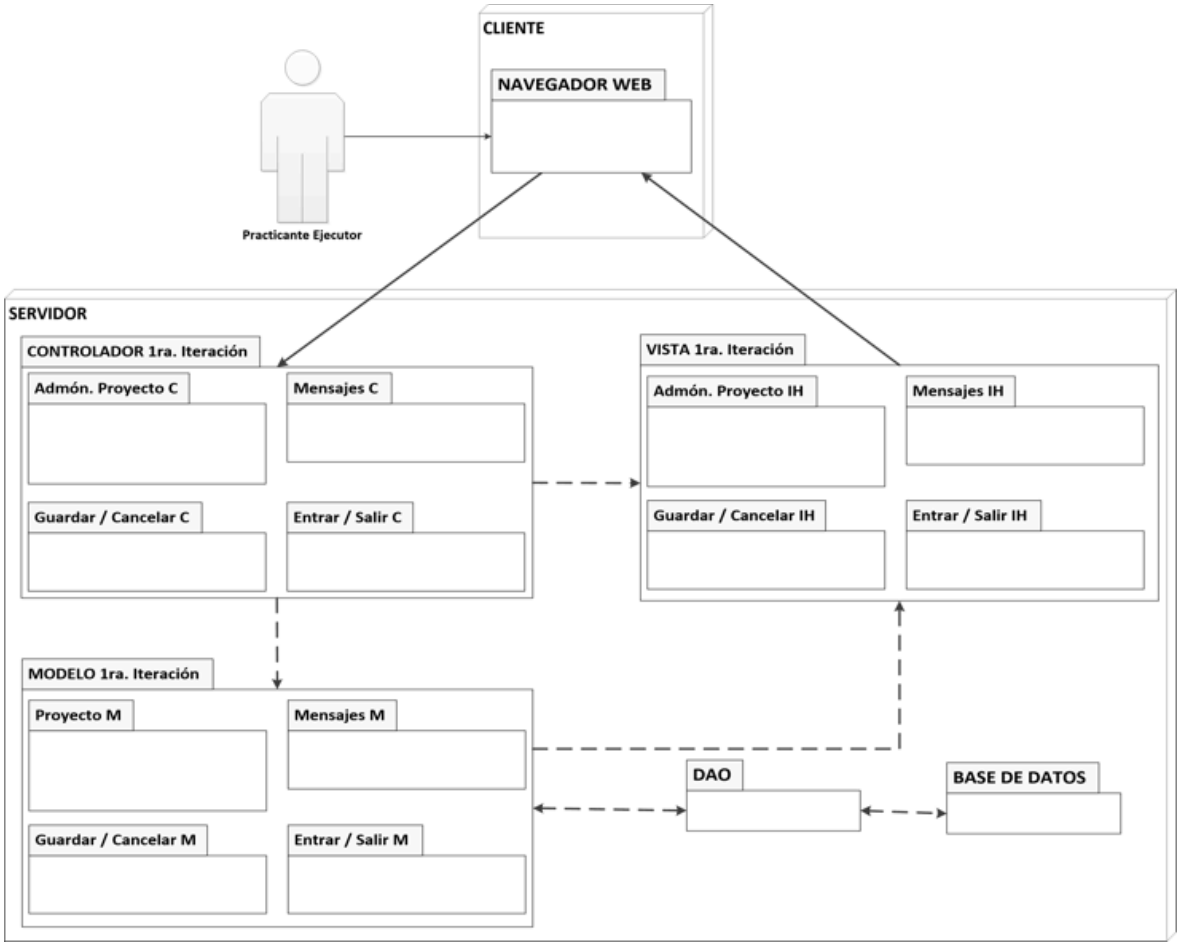
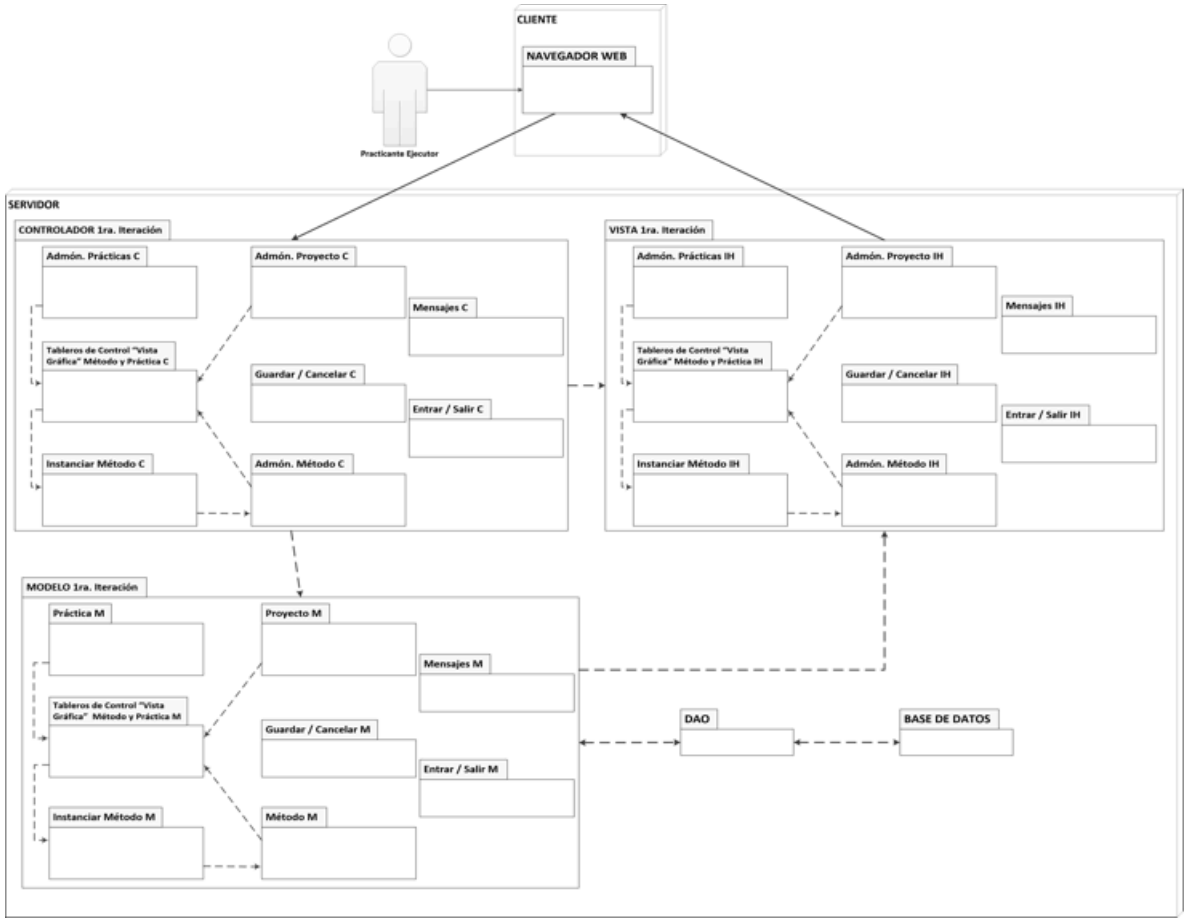


Diagrama de Comunicación “Tableros de Control”



Referencias Bibliográficas

[Barbacci2003] Mario R. Barbacci, Robert J. Ellison, Anthony J. Lattanze, Judith A. Stafford, Charles B. Weinstock, William G. Wood. "Quality Attribute Workshops (QAWs) 3º Edición". Technical Report. Software Engineering Institute. CMU/SEI-2003-TR-016. Octubre 2003

[Barrera2014] Rodrigo Alberto Barrera Hernandez. "Repositorio de Métodos y Prácticas de proyectos de software para KUALI-BEH". Tesis de maestría en ingeniería, Posgrado en Ciencia e Ingeniería de la Computación, UNAM.

[Bass2012] Lean Bass, Paul Clements y Rick Kazman. "Software Architecture in Practice 3ra Edición". SEI Series in Software Engineering. Addison-Wesley. Septiembre 2012

[Buschmann1996] Frank Buschmann. "A System of Patterns. Patterns-Oriented Software Architecture". Chichester, UK: John Wiley & Sons, 1996

[Casales2011] María Evelia Casales Cabrera. "Análisis de Arquitecturas de Software en ambientes de desarrollo ágil". Tesis de maestría, Universidad Nacional Autónoma de México. Distrito Federal, México. 2011

[Clements2002] Paul Clements, Paul Kazman y Mark Klein. "Evaluating Software Architectures: Methods and Case Studies". Boston: SEI Series in Software Engineering. Addison Wesley. 2002

[ClementsNorthrop1996] Paul C. Clements y Linda M. Northrop. "Software Architecture: An Executive Overview. Technical Report". CMU/SEI-96-TR-003, ESC-TR-96-003. Febrero de 1996

[CSE2013] Página web CSE "Object-Oriented Application Frameworks" [citado por última vez diciembre 2013]
URL: http://www.cse.wustl.edu/~schmidt/CACM-frameworks.html

[Dijkstra1968] Edsger W. Dijkstra. "Go-To statement considered harmful". ACM Communications of the ACM, Volumen 11, Número 3, Marzo de 1968

[ET2013] Página web etutorials.org. [citado por última vez diciembre 2013].
URL:
http://etutorials.org/Programming/Software+architecture+in+practice,+second+edition/Part+One+Envisioning+Architecture/Chapter+1.+The+Architecture+Business+Cycle/1.1+Where+Do+Architectures+Come+From/

[Fielding2000] Roy Thomas Fielding. "Architectural styles and the design of network-based software architectures". Tesis doctoral, University of California, Irvine, 2000.

[Galeon2014] Página web Galeon "Requerimientos de Software". [citado por última vez enero 2014]
URL: http://requerimientos.galeon.com

[Hofmeister2000] Christine Hofmeister, Robert Nord y Dilip Soni. “Applied Software Architecture”. MA: Addison-Wesley. 2000

[IEEE1471] Software Engineering Standards Committee of the IEEE Computer Society, IEEE Recommended Practice for Architecture Description of Software-Intensive System, IEEE Std 1471-2000
URL: http://standards.ieee.org/

[ISO/IEC 25060] ISO/IEC TR 25060:2010
URL:
http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=35786

[JavaM2013] Página web JavaMéxico. “Que es Data Access Object” [citado por última vez diciembre 2013].
URL:http://www.javamexico.org/blogs/richardmx/que_es_data_access_object

[Johnson1990] T.L. Johnson, A.D. Barker. “Trends in shop floor control: Modularity hierarchy and decentralization”. 1990

[Kruchten1995] Philippe Kruchten. “Architectural Blueprints - The 4+1 View Model of Software Architecture”. IEEE Software. Noviembre de 1995

[Kruchten2004] Philippe Kruchten. “The Rational Unified Process An Introduction”. Boston, MA: Addison-Wesley. 2004

[KruchtenJarr2013] Página web Jarroba.com. [citado por última vez diciembre 2013].
URL: http://www.jarroba.com/modelo-“41”-vistas-de-kruchten-para-dummies/

[Lane1990] Thomas G. Lane. “Studying Software Architecture Through Design Spaces and Rules. Technical Report”. CMU/SEI-90-TR-18 ESD-90-TR-219, noviembre de 1990.

[ObjSoft2013] Página web Object-Oriented Software Engineering. Diagrama Patrón MVC [citad por última vez diciembre 2013].
URL: http://pl.cs.jhu.edu/oose/lectures/swing.shtml

[Oktaba2012] Hanna Oktaba, Miguel E. Morales Trujillo. “KUALI-BEH: Software Project Common Concepts v1.1”. Agosto 2012
https://docs.google.com/a/kualikaans.mx/viewer?a=v&pid=sites&srcid=a3VhbGkta2FhbnMubXh8a3VhbGkta2FhbnN8Z3g6MTYxMmQ2OWUxODJkZDI1YQ

[OMG2012] Página web OMG [citado por última vez agosto de 2012]
URL: http://www.omg.org/

[Oracle2013]	Página web Oracle. “Core J2EE Patterns- Data Access Object” [citado por última vez diciembre 2013]. URL: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html
[Pastor2002]	Juan Ángel Pastor Franco. “Evaluación y desarrollo incremental de una arquitectura de software de referencia para sistemas de teleoperación utilizando métodos formales”. Tesis doctoral, Universidad Politectnica de Cartagena. Depto de Tecnologías de la Información y de la Comunicación. 2002
[Pressman2009]	Roger S. Pressman. “Software Engineering: A Practitioner´s Approach”. Septima Edición. McGraw Hill, Enero 2009.
[RAE2013]	Página oficial de la Real Academia de la Lengua Española [citado por última vez diciembre 2013]. http://rae.es
[Rational2013]	Página oficial de IBM Rational Software Corporation. [citado por última vez diciembre 2013]. URL: http://www-01.ibm.com/software/rational
[Reynoso2004]	Carlos Reynoso, Nicolás Kicillof. Estilos Arquitectónicos en la Estrategía de Arquitectura de Microsoft. Versión 1.0, Marzo de 2004. Universidad de Buenos Aires URL: www.carlosreynoso.com.ar/archivos/arquitectura/Estilos.PDF
[Ruíz2014]	C. Eraím Ruíz Sánchez. “Selección y Adaptación de Métodos en proyectos de software aplicando KUALI-BEH”. Tesis de maestría en ingeniería, Posgrado en Ciencia e Ingeniería de la Computación, UNAM.
[SEI2013]	Página oficial del Software Engineering Institute [citado por última vez diciembre 2013]. http://www.sei.cmu.edu/
[SelicRumbaugh1998]	Bran Selic & Jim Rumbaugh. “Using UML for Modeling Complex Real-Time Systems”. Rational Whitepaper. Marzo de 1998. [citado por última vez noviembre 2013] URL:http://www.ibm.com/developerworks/rational/library/content/03July/1000/1155/1155_umlmodeling.pdf
[Shaw1995]	Mary Shaw. “Comparing architectural design styles”. IEEE Software 0740-7459, 1995.
[ShawGarlan1995]	Mary Shaw y David Garlan. “An introduction to Software Architecture”. CMU Software Engineering Institute Technical Report, CMU/SEI-94-TR-21, ESC-TR-94-21, 1995.
[Shaw1989]	Mary Shaw. “Larger Scale Systems Require Higher-Level Abstractions”. ACM SIGSOFT Software Engineering Notes, Páginas 143-146,Volumen 14, Número 3, Mayo de 1989

[Taylor2010] Richard N. Taylor, Nenad Medvidovic y Eric M. Dashofy: Software Architecture Foundations, Theory and Practice. Wiley 2010

[Testing2012] Página web testing-ing2012 [citad por última vez diciembre 2013]. URL: http://testing-ing2012.wikispaces.com/Testing+No+Funcional

[Urrutia2014] José Luis Urrutia Velázquez. “Tableros de Control para el seguimiento de proyectos de software aplicando KUALI-BEH”. Tesis de maestría en ingeniería, Posgrado en Ciencia e Ingeniería de la Computación, UNAM.

[WSEI2013] Wikipedia. “Def. Software Engineering Institute” [citado por última vez en noviembre de 2013]. http://es.wikipedia.org/wiki/Software_Engineering_Institute