UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

MÁSTER UNIVERSITARIO EN INVESTIGACION EN INGENIERIA DE SOFTWARE Y SISTEMAS INFORMATICOS

ITINERARIO DE INGENIERIA DE SOFTWARE

CÓDIGO

TRABAJO DE FIN DE MÁSTER

Mejora de un Sistema de Auditoría Interna para Procesos y Procedimientos a Estaciones de Servicio (SAIPP-ES) utilizando la herramienta ArchE

Estudiante: Vicente Armando Correa Barrera

Profesor: José Félix Estivariz López

Curso: 2020/2021

Convocatoria: septiembre - 2021

CALIFICACIÓN:

AUTORIZACION:

RESUMEN

En las primeras décadas de este siglo, el mundo ha incrementado significativamente su dependencia a diferentes tipos de software, se utilizan en todo tipo de organizaciones tradicionales y emergentes, en nuestra vida diaria a través de dispositivos móviles, redes sociales y con el advenimiento del IoT cada vez más “cosas” de nuestro ecosistema. La relevancia que adquiere la Arquitectura de Software en los tiempos actuales cada vez es mayor pero también el conjunto de decisiones significativas sobre la organización de un sistema de software, la selección de elementos estructurales y sus interfaces, por las que se compone el sistema.

La investigación realizada para este trabajo tiene como objetivo principal proponer mejoras a un Sistema de Auditoria Interna para Procesos y Procedimientos a Estaciones de Servicio (SAIPP-ES), por lo que se revisan los principios y buenas prácticas de los procedimientos para realizar auditorías internas, así como las herramientas de evaluación que son automatizadas en el software propuesto.

El SAIPP-ES fue diseñado con el patrón arquitectónico MVC (Modelo-Vista-Controlador) por lo que se elabora un modelo de este a partir del cual se propondrán mejoras.

Finalmente, el modelo arquitectónico definido para el sistema propuesto es evaluado mediante la herramienta ArchE a través de su marco de razonamiento. Esta herramienta es también responsable de generar recomendaciones de mejora o tácticas, que modifican parcialmente la arquitectura del sistema a fin de satisfacer un atributo de calidad que para este trabajo es el de modificabilidad.

En las conclusiones se analizan los resultados obtenidos y se proponen trabajos futuros que refuercen o mejoren la investigación realizada.

PALABRAS CLAVES

Arquitectura de Software, Atributos de calidad, Características, Tácticas, Componentes, Modificabilidad, Escenarios, Casos de uso.

ABSTRACT

In the first decades of this century, the world has significantly increased its dependence on different types of software, they are used in all kinds of traditional and emerging organizations, in our daily lives through mobile devices, social networks and with the advent of the IoT more and more "things" of our ecosystem. The relevance that Software Architecture acquires in current times is increasing but also the set of significant decisions about the organization of a software system, the selection of structural elements and their interfaces that make up the system.

The main objective of the research carried out for this work is to propose improvements to an Internal Audit System for Processes and Procedures at Service Stations (SAIPP-ES), for which the principles and good practices of the procedures to carry out internal audits are reviewed, as well as the evaluation tools that are automated in the proposed software.

The SAIPP-ES was designed with the architectural pattern MVC (Model-View-Controller) for which a model of it is elaborated from which improvements will be proposed.

Finally, the architectural model defined for the proposed system is evaluated using the ArchE tool through its reasoning framework. This tool is also responsible for generating recommendations for improvement or tactics, which partially modify the architecture of the system in order to satisfy a quality attribute, which for this work is modifiability.

In the conclusions, the results obtained are analyzed and future works are proposed to reinforce or improve the research carried out.

KEYWORDS

Software Architecture, Quality Attributes, Characteristics, Tactics, Components, Modifiability, Scenarios, Use Cases.

AGRADECIMIENTOS

A mi padre, que ya no está con nuestra familia y que el creador lo guarde en su gloria

A mi madre, por siempre haberse siempre sacrificado por mi bienestar y el de toda la familia

A mis hijos por su comprensión en los momentos que sacrifique de estar con ellos por realizar este trabajo de investigación.

A mi profesor, Jose Félix, con quien tome la materia que incentivo haber elegido este tema para mi trabajo de Fin de Master.

Tabla de contenido

Índice de Figuras

Índice de Tablas

1. CAPITULO 1 - Introducción

En el transcurso de esta maestría tuve la oportunidad y el acierto de tomar la materia Arquitecturas para Sistemas de Software, a través de esta pude conocer desde los principios básicos del diseño arquitectónico, los estilos y directrices arquitectónicas hasta analizar a fondo el enfoque de una Arquitectura dirigida por modelos (MDA) y la utilización de los patrones de diseño en aplicaciones reales, todos estos temas despertaron en mi un gran interés por conocer más acerca de estas áreas y entender que rol desempeña un arquitecto de software en los tiempos actuales, ese así como he podido visualizar que un arquitecto desempeña un rol muy amplio y complejo, pues si bien un software nace de un conjunto de requisitos propios de la organización y en su mayoría funcionales, la visión del arquitecto debe abarcar no solo el cumplir estos requisitos sino también de qué forma cumplirlos procurando diseñar un software con características que le permitan perdurar en el tiempo y adaptarse al crecimiento organizacional y del ecosistema.

En el capítulo 2 definiremos los objetivos principales y específicos de esta investigación, enfocados a la mejora de un Software para Auditoría Interna para Procesos y Procedimientos en Estaciones de Servicios (SIIPP-ES) al que aplicaremos los conocimientos adquiridos producto de esta investigación.

En el capítulo 3 y 4 desarrollaremos la investigación a través del marco conceptual que define las bases de una Auditoria Interna, la aplicación de la norma ISO 19011 y del Checklist como herramienta fundamental, así como también los principios de la Arquitectura de Software, como identificar los módulos y características derivadas de los requisitos funcionales y no funcionales del negocio a fin de obtener un diseño arquitectónico básico del software propuesto para la mejora.

En el capítulo 5 se presenta el software SAIPP-ES y se detallan los requerimientos de la organización que dan origen a las responsabilidades y características del sistema, presentado en el diseño arquitectónico del mismo.

En el capítulo 6 se realizará la mejora del software para el atributo de modificabilidad a través de un marco de razonamiento y sugiriendo tácticas para mejorar la arquitectura, todo esto aplicando la herramienta ArchE.

Finalmente, en el capítulo 7 se elaborarán las conclusiones producto de este trabajo de investigación y las recomendaciones respectivas.

2. CAPITULO 2 - Objetivo de la investigación

2.1 Objetivo General

- Analizar el sistema de software propuesto y definir los módulos y su arquitectura
- Utilizar un asistente para el análisis y mejora de arquitecturas de software basado en reglas que nos permita satisfacer un atributo de calidad

2.2 Objetivo Especifico

- Presentar los módulos y el diseño básico de la arquitectura del Sistema de Auditoría Interna para Procesos y Procedimientos a Estaciones de Servicio (SAIPP-ES)
- Presentar los resultados de las mejoras al Sistema de Auditoría Interna para Procesos y Procedimientos a Estaciones de Servicio (SAIPP-ES) con respecto al atributo de Modificabilidad, utilizando la herramienta ArchE.

3. CAPITULO 3 – Auditorías internas y esquemas de automatización

La norma ISO 9000:2015 en [1] es una norma que establece el glosario de términos para todas las normas de gestión de calidad y de sistemas de gestión de calidad desarrolladas por el Comité Técnico ISO/TC 176. En ella se define que la auditoría es:

Un proceso sistemático, independiente y documentado para obtener evidencias objetivas y evaluarlas de manera objetiva con el fin de determinar el grado en que se cumplen los criterios de auditoría.

Vale decir que, para la auditoría interna es válida la misma definición de proceso que para cualquier otro proceso de la empresa.

Es decir, que la auditoría tiene las siguientes características:

- Es un proceso: es decir, que transforma unas entradas en resultados y debe controlarse mediante seguimiento y/o medición.
- Es sistemática: debe realizarse de forma ordenada y paso a paso.
- Es independiente: porque no deben existir vínculos que supongan una amenaza para el juicio del equipo auditor.
- Es documentada: ya que deben conservarse evidencias que soporten la realización de sus actividades y las conclusiones del equipo auditor.

Los requisitos para las auditorías experimentaron un cambio en el año 2018 debido a la actualización de la norma ISO 19011 [2]. Los principales cambios fueron:

- Definición de la auditoría como un “proceso”.
- Agregado del enfoque basado en riesgos a los principios de la auditoría.
- Ampliación de la orientación sobre cómo gestionar el programa de auditoría.
- Ampliación de la orientación sobre cómo planificar y realizar la auditoría.
- Ampliación de los requisitos de competencia genérica para los auditores.
- Ampliación del Anexo A para dar orientación acerca de nuevos aspectos tales como:
    - contexto de la organización
    - liderazgo y compromiso
    - auditorías virtuales
    - cumplimiento
    - cadena de suministro
- Eliminación del anexo que profundizaba aspectos sobre competencias para auditar disciplinas específicas de sistemas de gestión.

3.1 Principios de la Auditoria

La auditoría se caracteriza por depender de varios principios [3]. Estos principios deberían ayudar a hacer de la auditoria una herramienta eficaz y fiable en apoyo de las políticas y controles de gestión, proporcionando

información sobre la cual la organización puede actuar para mejorar su desempeño.

El auditor y las personas que gestionan un programa de auditorías deben tener:

Integridad: Desempeñar su trabajo con honestidad, diligencia y responsabilidad. Observar y cumplir los requisitos legales aplicables; demostrar su competencia al desempeñar su trabajo; desempeñar su trabajo de manera imparcial.

Presentación ecuánime: Es la obligación de informar con veracidad y exactitud.

Debido cuidado profesional: Los auditores deben proceder de acuerdo con la importancia de la tarea que desempeñan y la confianza depositada en ellos por el cliente de la auditoria. Un factor importante al realizar su trabajo con el debido cuidado profesional es tener la aptitud de hacer juicios razonados en todas las situaciones de la auditoria.

Confidencialidad: Los auditores deberían proceder con discreción en el uso y protección de la información adquirida en el curso de sus tareas.

En el proceso de auditoría es fundamental:

Independencia: Los miembros del equipo auditor deben estar libres de cualquier conflicto de intereses.

Enfoque basado en la evidencia: La evidencia de la auditoria debería ser verificable. En general se basará en muestras de la información disponible, ya que una auditoria se lleva a cabo durante un periodo de tiempo delimitado y con recursos finitos. Debe aplicarse un uso apropiado del muestreo, ya que está estrechamente relacionado con la confianza que puede depositarse en las conclusiones de la auditoria.

3.2 Aplicación de la norma ISO 19011

Esta norma proporciona orientación; es decir, no es de cumplimiento obligatorio. Por ello, las organizaciones pueden decidir adoptarla o no, pero no puede certificarse ni puede haber hallazgos por su incumplimiento.

Sin embargo, su uso es recomendado como una buena práctica ya que posee una alineación al ciclo PDCA (Plan-Do-Check-Act), lo que facilita una mejor gestión del proceso de auditoría.

La norma ISO 19011 se centra en los siguientes aspectos [2]:

- principios de la auditoría
- gestión del programa de auditoría
- realización de auditorías
- definición, evaluación y mantenimiento de la competencia de las personas que participan en el proceso de auditoría, es decir, responsables de la gestión del programa de auditoría, los auditores y los equipos auditores.

3.3 Tipos de Auditoria

En [3] se las clasifica de la siguiente forma:

Auditorias de primera parte: (Auditoría interna). Es la auditoría realizada por la empresa a sus propios sistemas o procedimientos de prerrequisitos. Su objetivo es asegurar el mantenimiento y desarrollo del sistema de prerrequisitos.

Puede ser realizada por personal interno (asegurando su independencia con el área a auditar) o personal externo o subcontratado.

Auditorias de segunda parte. Es la auditoría realizada por la empresa a sus proveedores o subcontratistas. Su objetivo es determinar la adecuación de los sistemas de prerrequisitos o de seguridad alimentaria de sus proveedores y evaluar la inocuidad de los alimentos de proveedores / subcontratistas. Puede ser realizado por personal de la empresa o personal externo o subcontratado.

En algunos casos es un requisito contractual entre proveedor- cliente.

Auditoria de tercera parte (Auditoría externa). Auditoría realizada por un organismo independiente a la empresa, proveedores y clientes

3.4 Auditorias in situ y a distancia

El Anexo A de la nueva versión de la norma ISO 19011 [2], incorpora nuevos conceptos del mundo actual. Es así, que brinda orientación acerca de las formas de realizar auditorías a través de los siguientes métodos:

- In situ con interacción humana: estas auditorías se realizan en las instalaciones del auditado, interactuando con el auditado. Quiere decir, que auditor-auditado interactúan en una entrevista.
- In situ sin interacción humana: estas auditorías se realizan en las instalaciones del auditado. El auditor no interactúa con el auditado, sino con equipos, instalaciones y documentos del auditado.
- A distancia con interacción humana: estas auditorías no se realizan en las instalaciones del auditado (sin importar la distancia), interactuando con el auditado. Quiere decir, que auditor-auditado interactúan en una entrevista.
- A distancia sin interacción humana: estas auditorías no se realizan en las instalaciones del auditado (sin importar la distancia). El auditor no interactúa con el auditado, sino con equipos, instalaciones y documentos del auditado.

3.5 Utilización del Checklist para la preparación de la auditoría interna

El uso del Checklist o Lista de verificación durante la auditoría interna [4], busca recabar las pruebas suficientes para asegurar si el proceso auditado cumple con los requisitos definidos. Sin embargo, cuando las bases del proceso concreto no están disponibles, el auditor podrá centrar la atención en la comprobación de procesos de proveedores, la revisión de las entradas del mismo o incluso los pasos realizados en el propio proceso, así como sus resultados y clientes, como manera de revisar que los empleados realizan de manera óptima el proceso y que se está realizando con efectividad.

Checklist: Identificación y seguimiento de problemas

1. Identificación de la auditoría

Institución auditada:

Proyecto: **Iniciador:** **Tipo de auditoría:** ❒ Interna ❒ Externa	**Fase del ciclo de vida** ❒ Planificación ❒ Integración y pruebas ❒ Esp. de Requerimientos ❒ Aceptación y entrega ❒ Diseño ❒ Mantención ❒ Implementación

2. Auditor

Nombre

e-mail	**Fono**

3. Checklist

	Sí	No
• ¿Existen procedimientos que aseguren la detección y corrección de los problemas y/o discrepancias detectadas?		
• ¿Se examinan los informes de problemas y de discrepancias para determinar las posibles causas?		
• ¿Se analiza la relación entre las diferentes actividades de desarrollo para prevenir disconformidades en los productos?		
• ¿Se definen y planifican acciones correctivas? ¿Se asignan los recursos adecuados?		
• ¿Las acciones correctivas son registradas y documentadas minuciosamente?		
• ¿Se revisan y monitorean las acciones correctivas para determinar su efectividad, completitud y complacencia respecto de los estándares?		
• ¿El nivel de gestión apoya las acciones correctivas?		
• ¿Los desarrolladores están de acuerdo en generar informes de problemas y de discrepancias? ¿Los utilizan?		

Checklist: Estado del proyecto

4. Identificación de la auditoría

Institución auditada:

Proyecto: **Iniciador:** **Tipo de auditoría:** ❒ Interna ❒ Externa	**Fase del ciclo de vida** ❒ Planificación ❒ Integración y pruebas ❒ Esp. de Requerimientos ❒ Aceptación y entrega ❒ Diseño ❒ Mantención ❒ Implementación

5. Auditor

Nombre

e-mail	**Fono**

6. Checklist

	Sí	No
• ¿El estado real del proyecto concuerda con la planificación? ¿Si no es así, que tan grande es la brecha?		
• De acuerdo con el plan de proyecto: ¿cuál es el estado de las actividades, recursos, productos de trabajo, hitos?		
• Determinar: (a) fase de desarrollo actual, (b) estado de avance de las actividades, (c) conformación y organización del equipo desarrollador, (d) productos de trabajo, (e) hitos, y (f) resultados de las revisiones.		

Figura 1. Checklist

En definitiva, al preparar un checklist en una auditoría interna, el auditor busca poder realizar una revisión eficiente de un proceso, permitiéndole poder confirmar que los registros existentes sobre los procesos ofrecen la información adecuada para garantizar el cumplimiento de los requisitos y, en caso de que no fuera así, levantaría la instancia pertinente al propietario del proceso concreto informando de la presencia de ciertas inconformidades y la necesidad de realizar una acción correctiva

El checklist se puede utilizar en cualquier área del sistema de gestión, por ejemplo: para evaluar a los proveedores, para realizar controles del producto,

para verificar los productos comprados, o para evaluar la competencia del personal.

Un checklist se puede utilizar con finalidades de evaluación, de control, de análisis, y cómo no, de verificación. Del resultado de un checklist se puede deducir el valor de un indicador, o lo podemos utilizar para comparar entre varias opciones, o establecer una foto fija de la situación actual, en la Figura 1 se muestra un ejemplo.

Por otra parte, los checklist no aportan solamente ventajas, sino que también conllevan un gran riesgo, debido a que pueden hacer que se caiga en la rutina y se conviertan en meros checklist a contestar por SÍ/NO. Algunas veces estos checklist con respuestas del tipo SÍ/NO pueden resultar de utilidad para algunas actividades de auditoría que presente características de inspección. Un ejemplo de este tipo son las auditorías de buenas prácticas de manufactura (BPM).

La eficacia de los checklists se puede evaluar por el carácter de las preguntas. Vamos a ver un ejemplo a continuación de dos tipos de preguntas que se corresponden a un ineficaz checklist y a un eficaz checklist:

1. Pregunta ineficaz – ¿Tiene un procedimiento de Acciones Correctivas?
2. Pregunta eficaz – En relación con las acciones correctivas y tomando como referencia cinco registros ¿las acciones para eliminar la causa raíz se ejecutan de forma eficaz?

Ambas preguntas son válidas, pero en el primero de los casos es posible contestarla sin revisar ningún registro y sin variar año tras año, por lo que no nos aporta ningún elemento para la mejora. En el caso de la segunda pregunta provoca que en cada auditoría interna se **revisen los registros y se dispongan de evidencias** de su cumplimiento más allá del mero procedimiento en sí, pasando del procedimiento a la actividad diaria de la empresa. Además, permite que cada año estas preguntas se actualicen y se adapte a la realidad de la organización.

De esta forma, la Auditoría Interna no caería en la lectura del auditor de una serie de preguntas y promovería la exploración de las distintas respuestas que

se producen durante esta actividad, convirtiéndose en una herramienta de mejora continua.

Algunos de los elementos necesarios para realizar un eficaz checklist son los siguientes:

- Basar el checklist en los distintos procesos que requieren ser auditados.
- No centrarse sólo en los requisitos mínimos de la norma o el documento involucrado, sino que verificar la adecuación de las entradas de cada proceso y la eficacia de los resultados, para detectar elementos o aspectos de mejora.
- Debe tener en cuenta la interrelación entre los distintos procesos de forma que se pueda realizar una adecuada trazabilidad de las pistas de auditoría.
- Es necesario que los puntos o preguntas de los checklist se centren en hechos y datos que permitan una verificación eficaz del cumplimiento, y no en simples respuestas sí/no.
- Hay que adaptar las preguntas para situaciones específicas de acuerdo con las características de la empresa y de los procesos, incluyendo la terminología utilizada por esta.
- Deben ser claras y no convertirse en un guion a leer, sino en una herramienta a seguir como guía del proceso a auditor.

4. CAPITULO 4 – Arquitectura de software

4.1 Arquitectura de Software: definiciones

A través de diferentes autores podemos encontrar definiciones de Arquitectura de software

Len Bass, Paul Clements, Rick Kazman, en [8], formulan una definición que es de su agrado, esta indica que la arquitectura de software de un sistema es el conjunto de estructuras necesarias para razonar acerca de un sistema, las mismas que comprenden elementos de software, las relaciones entre ellos y las propiedades de ambos. Esta definición contrasta con otras definiciones que hablan sobre las decisiones de diseño “tempranas” o “importantes” del sistema.

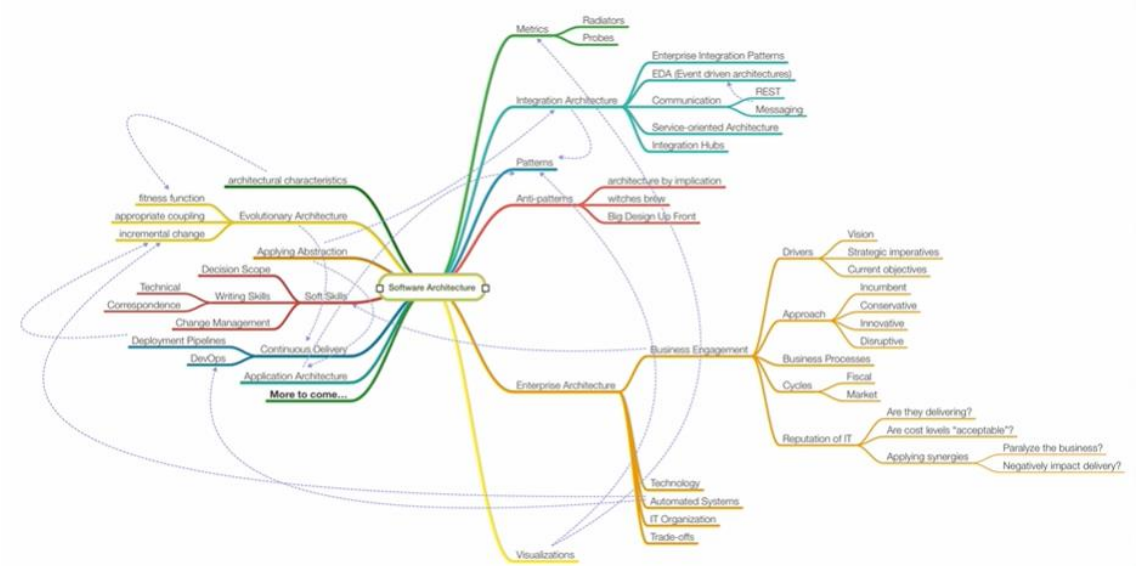


Figura 2. Aspectos de la Arquitectura de Software. [6]

Mark Richards fue consultado para dar una definición agradable, sucinta, concisa, indicando que no era posible y que lo más cercano a lo que había podido llegar era a un mapa mental que comienza a tocar todos los aspectos de la Arquitectura de Software como se muestra en la Figura 2 y que como se puede observar en la parte inferior izquierda de su grafico existe una categoría critica que indica más por venir, debido a que el rol del arquitecto de software se ha venido expandiendo con el transcurso de los años.

La definición de la Rational Unified Process que es adoptada por la IEEE indica que la arquitectura de software es “El concepto de más alto nivel del sistema en su entorno. La arquitectura de un sistema de software en un punto dado en el tiempo es su organización o estructura de componentes significativos que interactúan a través de interfaces, compuestos de componentes e interfaces sucesivamente más pequeños”,

En la definición anterior se pierde un aspecto realmente importante de eso que Ralph Johnson recoge en su refutación de esa definición, que dice que “En la mayoría de los proyectos de software, los desarrolladores expertos que están trabajando en ese proyecto tienen un entendimiento compartido del diseño del sistema. Este entendimiento compartido es llamado *arquitectura*. Este entendimiento incluye como el sistema está dividido en componentes que interactúan a través de interfaces. Estos componentes están compuestos usualmente de componentes más pequeños, pero la arquitectura solo incluye los componentes o interfaces que son entendidos por todos los

desarrolladores". En otras palabras, lo que quiere decir es que no se puede simplemente tomar la estructura de un sistema de arquitectura de software completamente aislada sin comprender las motivaciones de por qué eligieron un enfoque particular, un patrón particular y la forma en que combinan las cosas, porque no entiendes las compensaciones que tuvieron que hacer para llegar a esa decisión arquitectónica final.

Entonces, el punto de esto es que la arquitectura no solo se trata de una comprensión compartida, sino también de transmitir esa comprensión a un grupo de partes interesadas, todas las cuales tienen sus propias prioridades sobre lo que es importante para este software. Y muy a menudo, los arquitectos son los únicos que tienen el alcance suficiente para ver realmente todas estas partes móviles y poder tomar decisiones sobre todas esas partes móviles.

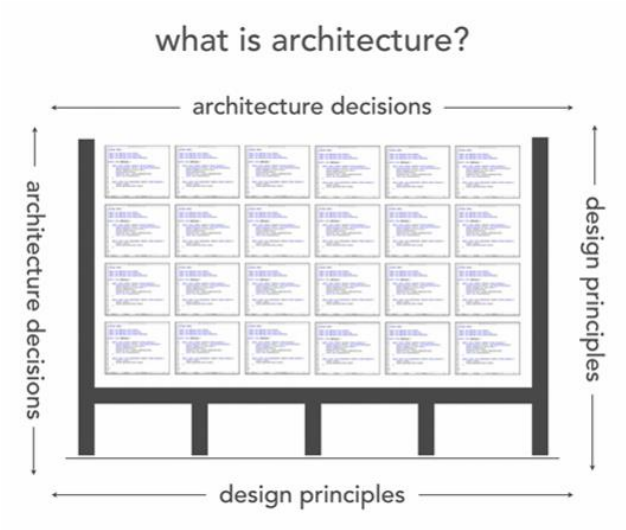


Figura 3. Que es una Arquitectura de Software [6]

En la Figura 3 se muestra una forma de pensar la Arquitectura de software elaborada por Mark Richards y Neal Ford. En esta definición la arquitectura de software consiste en la *estructura* del sistema (denotada por las líneas negras más gruesas soportando la arquitectura), combinada con las características de la arquitectura ("-ilidad") que el sistema debe soportar y finalmente los principios de diseño.

Examinando esta definición la estructura puede definirse, realmente, como dos cosas: la primera es el patrón general de arquitectura que realmente se esté usando, por ejemplo, ¿es micro kernel? ¿Son microservicios?, tal vez es una

arquitectura impulsada por eventos, ver Figura 4 ese es un aspecto de la estructura general, pero la estructura en términos de arquitectura también involucra algunas de las (“-ilidad”) de las que Richards y Ford mencionaba en términos de comprobabilidad, confiabilidad, disponibilidad, este tipo de cosas, ver Figura 5.

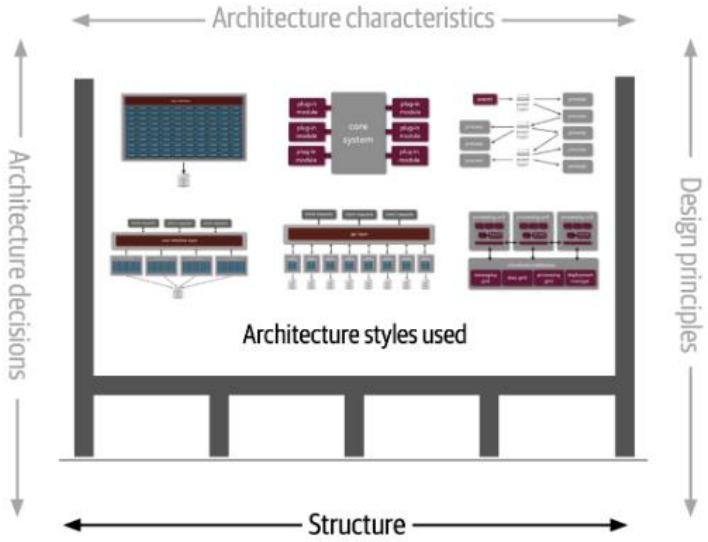


Figura 4. Estilos de Arquitectura [6]

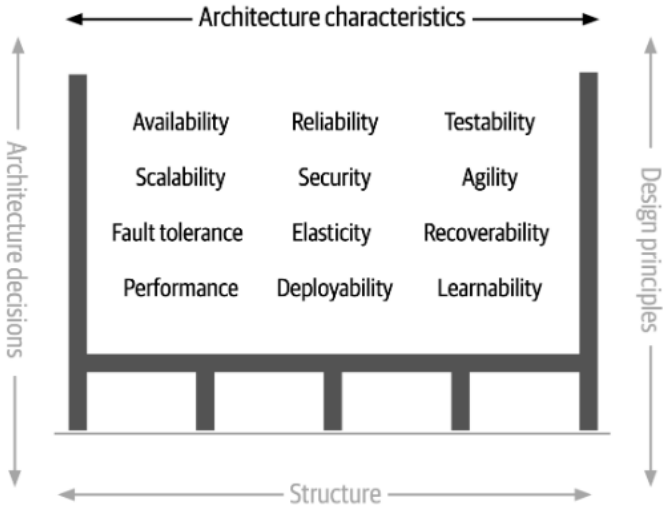


Figura 5. Características de Arquitectura [6]

Pero no basta con decir, por ejemplo, ¿cuál es su arquitectura? Bueno, nuestra arquitectura es microservicios, o nuestra arquitectura está basada en el espacio, esa es la estructura, ese es el patrón de arquitectura general. Pero la arquitectura implica mucho más que solo el patrón general, se trata de esos

principios de diseño y decisiones de arquitectura, esto es lo que compone la arquitectura. Por ejemplo, en una arquitectura en capas, se podría decir que en nuestra estructura tenemos un modelo en capas de n niveles, esa es la estructura general, pero, por ejemplo, los componentes o clases dentro de la capa empresarial o de servicio son los únicos que pueden comunicarse con la capa de persistencia, la de presentación no puede o que todos los objetos compartidos deben residir en un servicio. Esta es una decisión de arquitectura, ver Figura 6.

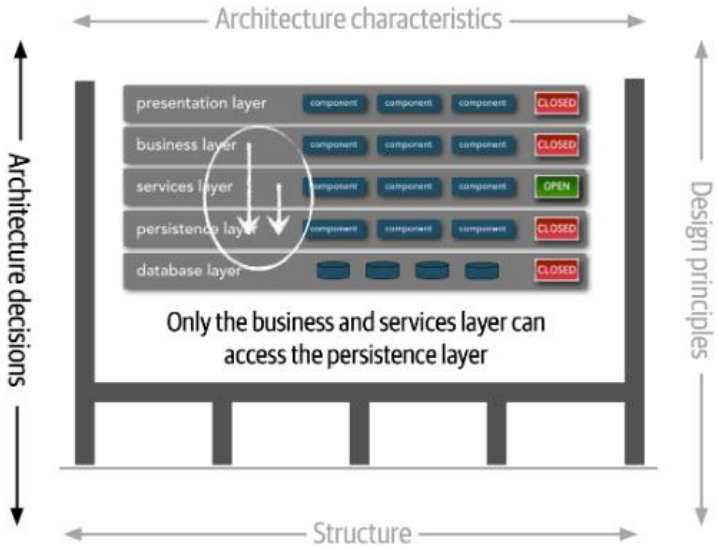


Figura 6. Comunicaciones entre capas de Arquitectura [6]

Con respecto a los principios de diseño, por ejemplo, en una arquitectura de microservicios, uno de esos principios de diseño podría ser que queremos aprovechar la mensajería o la comunicación asincrónica entre servicios siempre que sea posible para aumentar el rendimiento, ver Figura 7. En esencia, estas tres cosas juntas forman lo que realmente es la arquitectura.

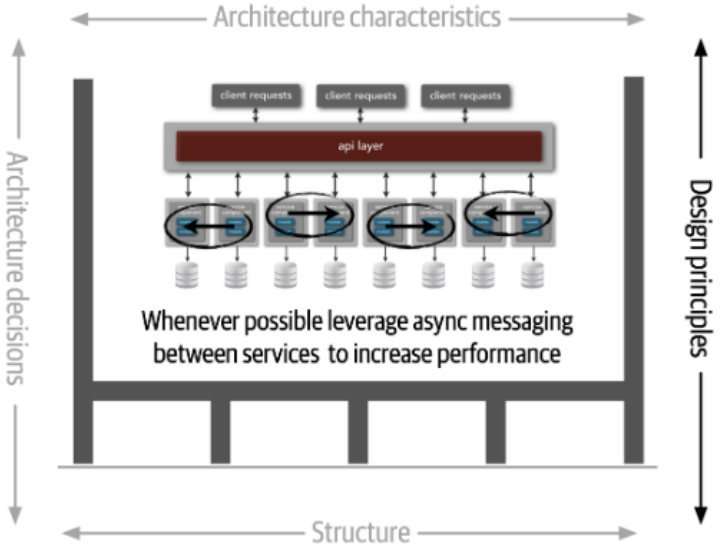


Figura 7. Comunicaciones entre capas de Arquitectura [6]

Al respecto de las decisiones de diseño existe una categorización sistemática parcialmente basada en la definición de Arquitectura de software elaborada por Len Bass, Paul Clements, Rick Kazman en [8], en la cual varias de las categorías se relacionan a la definición de estructuras y la relación entre estas. Las siete categorías de diseño son:

1. Asignación de responsabilidades
2. Modelo de coordinación
3. Modelo de datos
4. Mapeo entre los elementos arquitectónicos
5. Decisiones de tiempo vinculantes
6. Elección de la tecnología

4.2 Funcionalidad

Funcionalidad es la habilidad del sistema para hacer el trabajo por el cual este fue pensado. La funcionalidad no determina la arquitectura, es decir dado un conjunto de funcionalidades requeridas no hay límite para las arquitecturas que se podrían crear para satisfacer esa funcionalidad. Por lo menos, podría dividir la funcionalidad en cualquier número de formas y asignar las subpiezas a diferentes elementos arquitectónicos.

No obstante que la funcionalidad es independiente de cualquier estructura particular, la funcionalidad es lograda asignando responsabilidades a los elementos arquitectónicos. El interés del arquitecto en la funcionalidad está en como esta interactúa con las cualidades y en como restringe a otras.

ISO 25010 define la idoneidad funcional como la capacidad del producto software para proporcionar funciones que satisfagan las necesidades establecidas e implícitas cuando el software se usa en condiciones específicas.

4.3 Caracteristicas de la arquitectura

Cuando se decide resolver un problema o automatizar procesos utilizando un software, se deben levantar una serie de requerimientos o requisitos para aquel sistema. A más de las técnicas para el levantamiento de requisitos generalmente definidas por el proceso de desarrollo del software usado por el equipo, el arquitecto debe considerar muchos otros factores en diseñar una solución de software, tal como se ilustra en la Figura 8.

Los arquitectos deben colaborar en la definición del dominio o requerimientos del negocio, pero una de sus principales responsabilidades o rol implica definir, descubrir y analizar todas las cosas que el software debe hacer que no estén directamente relacionadas al dominio de la funcionalidad, esto se conoce *como características del sistema.*

En varios textos se describen estas características de software con una variedad de términos, uno de ellos es requisitos no funcionales, este término es en sí denigrante, y si bien fue adoptado para diferenciarlo de los requisitos funcionales, un equipo de trabajo podría no poner la atención adecuado a algo no funcional.

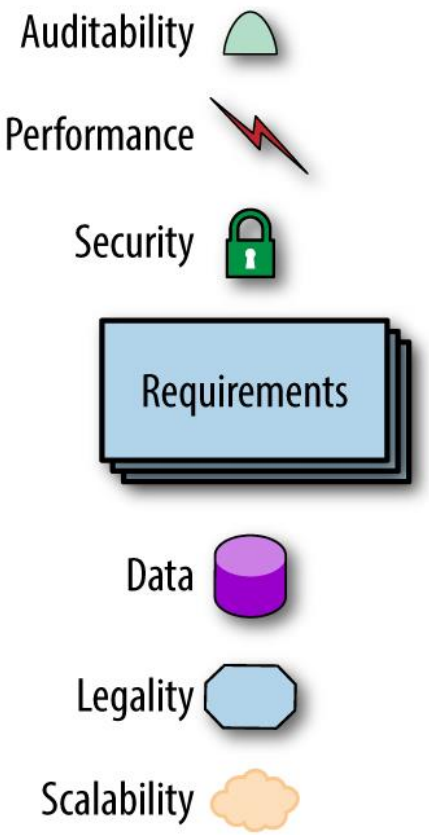


Figura 8. Características de una Arquitectura [6]

También podemos encontrar más comúnmente el termino *atributos de calidad*, definido por Len Bass, Paul Clements, Rick Kazman en [8] como una propiedad del sistema medible o testeable que es usada para indicar que tan bien el sistema satisface las necesidades de sus stakeholders, lo cual de acuerdo a Mark Richards y Neal Ford en [5] implica evaluación de calidad antes que de diseño y prefieren el termino características de arquitectura porque este describe asuntos críticos para el éxito de la arquitectura y consecuentemente el sistema como un todo sin descontar su importancia.

4.4 Atributos de Calidad

Para [8] hay tres clases de atributos de calidad:

- Calidades del sistema
- Calidades del negocio
- Calidades relacionadas con la arquitectura en si misma

Las definiciones proporcionadas para un atributo de calidad no son operacionales y a menudo clasificar estos atributos con relación a los diferentes

aspectos funcionales del sistema es complicado, para resolver esto se recurre a los denominados escenarios de los atributos de calidad.

4.5 Escenarios de Atributos de Calidad

Un escenario de atributo de calidad es un requisito especifico de un atributo de calidad, que consta de seis partes que se detallan en la Tabla 1:

Parte	Descripción
Fuente del estimulo	Representa alguna entidad (un humano, un sistema de cómputo o cualquier otro actuador) que genero el estimulo
Estimulo	El estímulo es una condición que requiere una respuesta cuando esta llega a un sistema
Entorno	El estímulo ocurre bajo ciertas condiciones. El sistema puede estar en una condición de sobrecarga o en operación normal, o en algún otro estado relevante. Para este tipo de sistemas, el entorno debe especificar en qué modo se está ejecutando el sistema.
Artefacto	Se estimula algún artefacto. Esto puede ser una colección de sistemas, todo el sistema, o una parte o partes de él.
Respuesta	La respuesta es la actividad realizada como resultado de la llegada del estímulo.
Medida de la respuesta	Cuando se produce la respuesta, debe ser medible de alguna manera para que se pueda probar el requisito.

Tabla 1.Partes de un escenario de atributo de calidad [8]

En la Figura 9 se muestran las diferentes partes de un escenario de atributo de calidad:

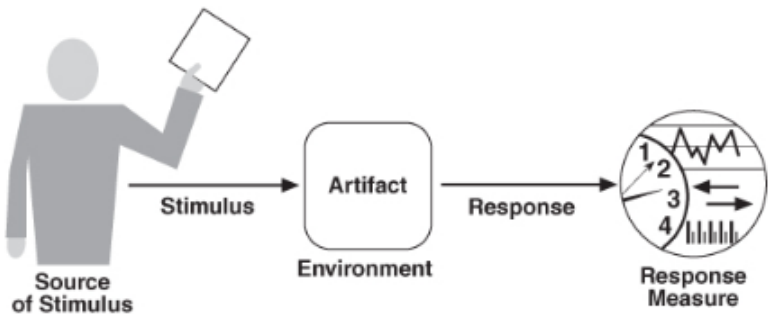


Figura 9. Partes de un escenario de atributos de calidad [8]

Una característica de arquitectura reúne tres criterios:

- Especifica una consideración de diseño que no es de dominio
- Influencia algunos aspectos estructurales del diseño
- Es critica o importante para una aplicación exitosa

Las partes entrelazadas de esta definición se ilustran en la Figura 10, que consisten en tres componentes enumerados, además de algunos modificadores. Las características implícitas rara vez aparecen en los requisitos, pero son necesarios para el éxito del proyecto mientras que las explicitas aparecen en los documentos de requisitos y otras instrucciones específicas.

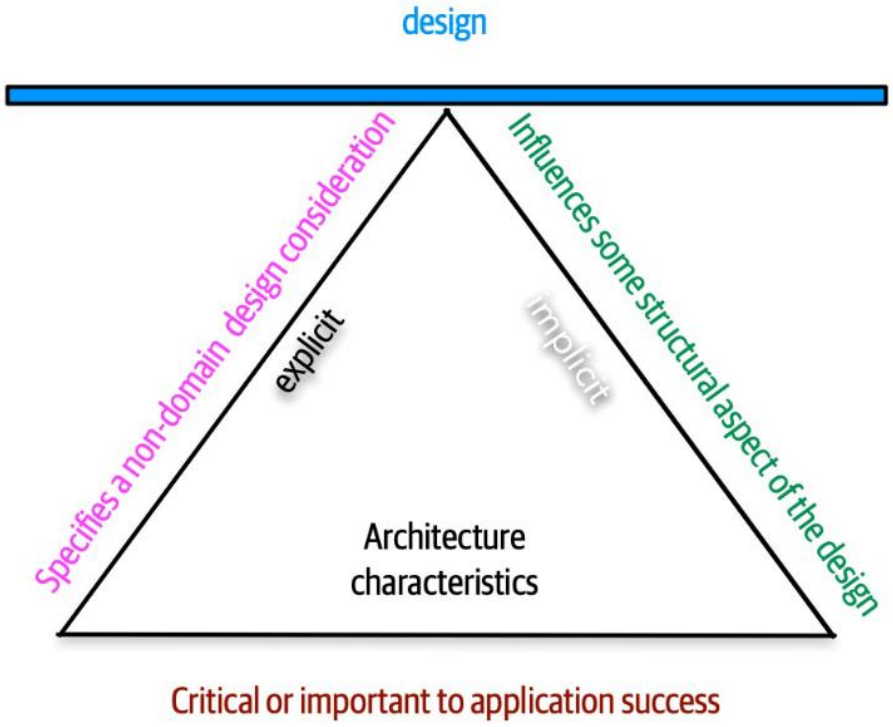


Figura 10. Características implícitas y explicitas de una Arquitectura [6]

La elección de un triángulo es intencional: cada uno de los elementos de definición es compatible con los demás, lo que a su vez es compatible con el diseño general del sistema. El punto de apoyo creado por el triángulo ilustra el hecho de que estas características de la arquitectura a menudo interactúan entre sí, lo que lleva al uso generalizado entre los arquitectos del término compensación.

Podemos identificar dos métodos que se puede utilizar para evaluar y analizar compensaciones:

ATAM-- Architecture Tradeoff Analysis Method (Método de Análisis de Compensación de Arquitectura)
CBAM-- Cost-Benefit Analysis Method (Método de Análisis Costo-Beneficio)

La mayoría de las veces, estas características que necesitamos identificar como arquitectos no provienen de requisitos o historias de usuarios, sino más bien provienen del negocio. Comprender las características generales de la arquitectura antes de formar una solución es la manera correcta de hacer una arquitectura y es el primer paso para elegir qué estilo o patrón de arquitectura se adapta mejor a su problema.

Debido a que el ecosistema de software cambia tan rápido, constantemente aparecen nuevos conceptos, términos, medidas y verificaciones, lo que brinda nuevas oportunidades para las definiciones de las características de la arquitectura.

Los arquitectos suelen separar las características de la arquitectura en amplias categorías. A continuación, se describen varias, junto con algunos ejemplos:

4.6 Características de la arquitectura operacional

Rendimiento: Incluye pruebas de estrés, análisis de picos, análisis de la frecuencia de las funciones utilizadas, la capacidad requerida y los tiempos de respuesta. La aceptación del rendimiento a veces requiere un ejercicio propio, que lleva meses completar.

Un escenario de rendimiento, como se muestra en la Figura 11 comienza con un evento que llega al sistema. Responder correctamente al evento requiere que se consuman recursos (incluido el tiempo). Mientras esto sucede, el sistema puede estar atendiendo simultáneamente otros eventos.

Los eventos pueden llegar en patrones predecibles o distribuciones matemáticas, o ser impredecibles. Un patrón de llegada para eventos se caracteriza por ser periódico, estocástico o esporádico:

- Los eventos periódicos llegan previsiblemente a intervalos de tiempo regulares. Por ejemplo, un evento puede llegar cada 10 milisegundos. La

llegada periódica de eventos se ve con mayor frecuencia en sistemas en tiempo real.

- Llegada estocástica significa que los eventos llegan de acuerdo con alguna distribución probabilística.
- Los eventos esporádicos llegan de acuerdo con un patrón que no es periódico ni estocástico.

La respuesta del sistema a un estímulo se puede medir de la siguiente manera:

Latencia. El tiempo entre la llegada del estímulo y la respuesta del sistema al mismo.

Deadlines en procesamiento. En el controlador del motor, por ejemplo, el combustible debe encenderse cuando el cilindro está en una posición particular, introduciendo así una fecha límite de procesamiento.

El rendimiento del sistema, generalmente dado como el número de transacciones que el sistema puede procesar en una unidad de tiempo.

La fluctuación de la respuesta: la variación permitida en la latencia.

El número de eventos no procesados porque el sistema estaba demasiado ocupado para responder.

A partir de estas consideraciones, ahora podemos describir las porciones individuales de un escenario general para el rendimiento que se detalla en la Tabla 2.:

Parte	Descripción
Fuente del estimulo	Interno o externo al sistema
Estimulo	Llegada de un evento periódico, esporádico o estocástico
Entorno	Modo operacional: normal, emergente, carga pico o sobrecarga
Artefacto	Sistema o uno o más componentes en el sistema
Respuesta	Eventos de proceso, cambio de nivel de servicio
Medida de la respuesta	Latencia, deadline, rendimiento, fluctuación, tasa de fallas

Tabla 2. Partes para un escenario de rendimiento [8]

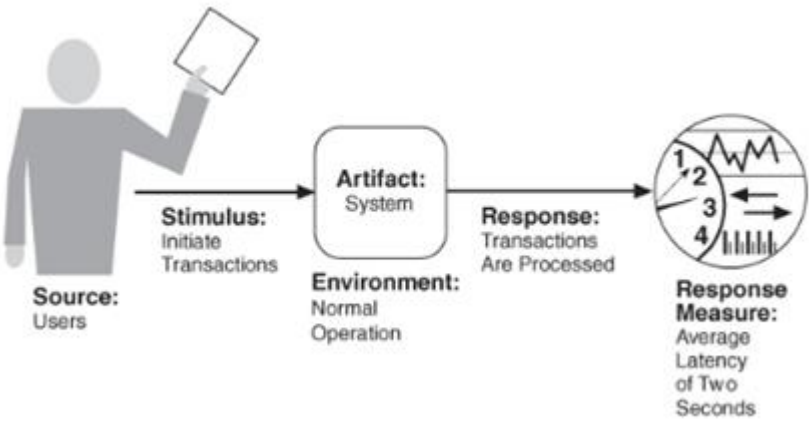


Figura 11. Ejemplo concreto de escenario de rendimiento [8]

Disponibilidad: Cuánto tiempo tendrá que estar disponible el sistema (si está disponible las 24 horas, los 7 días de la semana, se deben implementar pasos para permitir que el sistema esté en funcionamiento rápidamente en caso de falla).

Continuidad: Capacidad de recuperación ante desastres.

Recuperabilidad: Requisitos de continuidad del negocio (por ejemplo, en caso de desastre, ¿con qué rapidez se requiere que el sistema vuelva a estar en línea?). Esto afectará la estrategia de respaldo y los requisitos para hardware duplicado.

Fiabilidad / seguridad: Evalúe si el sistema necesita ser a prueba de fallas o si es de misión crítica de una manera que afecte las vidas. Si falla, ¿le costará a la compañía grandes sumas de dinero?

Robustez: Capacidad para manejar errores y condiciones de límite mientras se ejecuta, si la conexión a Internet se cae o si hay un corte de energía o una falla de hardware.

Escalabilidad: Capacidad para que el sistema funcione y que lo haga a medida que aumenta el número de usuarios o solicitudes.

4.7 Características de la arquitectura estructural

Modificabilidad ¿Qué tan fácil es aplicar cambios y mejorar el sistema?

Para planificar la modificabilidad, un arquitecto debe considerar cuatro preguntas:

¿Qué puede cambiar? Un cambio puede ocurrir a cualquier aspecto de un sistema: las funciones que el sistema computa, la plataforma, el ambiente en el cual el sistema opera, las cualidades que el sistema exhibe, su capacidad, entre otras.

¿Cuál es la probabilidad del cambio? Uno no puede planificar un sistema para todos los cambios potenciales, cualquier cosa puede cambiar, el arquitecto tiene que tomar decisiones difíciles sobre qué cambios son probables y, por lo tanto, qué cambios están soportados y cuáles no.

¿Cuándo se realiza el cambio y quién lo hace? la cuestión de cuándo se realiza un cambio se entrelaza con la cuestión de quién lo hace, se pueden hacer cambios en la implementación, durante la compilación, durante la construcción, durante el seteo de la configuración o durante la ejecución. Un desarrollador, un usuario final o un administrador del sistema también pueden hacer un cambio.

¿Cuál es el costo del cambio? Hacer que un sistema sea más modificable implica dos tipos de costos:

- El costo de introducir los mecanismos para hacer que el sistema sea más modificable.
- El costo de realizar la modificación utilizando los mecanismos.

Un escenario de rendimiento se muestra en la Figura 12. A partir de estas consideraciones, como se muestra en la Tabla 3, podemos ver las porciones del escenario general de modificabilidad:

Parte	Descripción
Fuente del estimulo	Esta parte especifica quién realiza el cambio: el desarrollador, un administrador del sistema o un usuario final.
Estimulo	Esta porción especifica el cambio a realizar. Un cambio puede ser la adición de una función, la modificación de una función existente o la eliminación de una función. También se puede hacer un cambio en las cualidades o la capacidad del sistema, además pueden ocurrir cambios para acomodar nuevas tecnologías de algún tipo
Entorno	Esta parte especifica qué se debe cambiar: componentes o módulos específicos, la plataforma del sistema, su interfaz de usuario, su entorno u otro sistema con el que interopera
Artefacto	Esta parte especifica cuándo se puede realizar el cambio: tiempo de diseño, tiempo de compilación, tiempo de construcción, tiempo de inicio o tiempo de ejecución.
Respuesta	Uno o más de los siguientes: hacer, probar o implementar la modificación.
Medida de la respuesta	Costo en términos de lo siguiente: • Número, tamaño, complejidad de artefactos afectados • Esfuerzo • Tiempo calendario • Dinero (desembolso directo o costo de oportunidad) • Medida en que esta modificación afecta otras funciones o atributos de calidad • Nuevos defectos introducidos

Tabla 3. Partes para un escenario de modificabilidad [8]

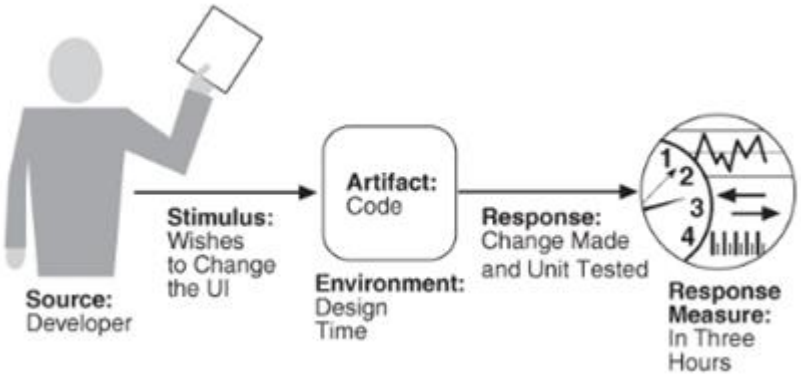


Figura 12. Ejemplo concreto de escenario de modificabilidad [8]

Configurabilidad: Capacidad para que los usuarios finales cambien fácilmente aspectos de la configuración del software (a través de interfaces utilizables).

Extensibilidad: Qué tan importante es conectar nuevas piezas de funcionalidad.

Instalabilidad: Facilidad de instalación del sistema en todas las plataformas necesarias.

Apalancamiento / reutilización: Capacidad para aprovechar componentes comunes en múltiples productos.

Localización: Soporte para múltiples idiomas en pantallas de entrada / consulta en campos de datos; en informes, requisitos de caracteres multibyte y unidades de medida o monedas.

Portabilidad: ¿El sistema necesita ejecutarse en más de una plataforma? (Por ejemplo, ¿la interfaz debe ejecutarse tanto en Oracle como en SAP DB?

Soportabilidad: ¿Qué nivel de soporte técnico necesita la aplicación? ¿Qué nivel de registro y otras instalaciones se requieren para depurar errores en el sistema?

Capacidad de actualización: Posibilidad de actualizar fácil / rápidamente desde una versión anterior de esta aplicación / solución a una versión más reciente en servidores y clientes.

4.8 Características de la arquitectura transversal

Accesibilidad: Acceso a todos sus usuarios, incluidos aquellos con discapacidades como daltonismo o pérdida de audición.

Archivabilidad: ¿Deberán archivarse o eliminarse los datos después de un período de tiempo? (Por ejemplo, las cuentas de los clientes deben eliminarse después de tres meses o marcarse como obsoletas y archivarse en una base de datos secundaria para acceso futuro).

Autenticación: Requisitos de seguridad para garantizar que los usuarios sean quienes dicen ser.

Autorización: Requisitos de seguridad para garantizar que los usuarios solo puedan acceder a ciertas funciones dentro de la aplicación (por caso de uso, subsistema, página web, regla de negocio, nivel de campo, etc.).

Legal: ¿En qué restricciones legislativas funciona el sistema (protección de datos, Sarbanes Oxley, GDPR, etc.)? ¿Qué derechos de reserva requiere la empresa? ¿Alguna normativa sobre la forma en que se debe construir o implementar la aplicación?

Privacidad: Capacidad para ocultar transacciones de los empleados internos de la empresa (transacciones cifradas para que incluso los DBA y los arquitectos de redes no puedan verlas).

Seguridad: ¿Es necesario cifrar los datos en la base de datos? ¿Cifrado para la comunicación de red entre sistemas internos? ¿Qué tipo de autenticación debe existir para el acceso de usuarios remotos?

Soportabilidad: ¿Qué nivel de soporte técnico necesita la aplicación? ¿Qué nivel de registro y otras instalaciones se requieren para depurar errores en el sistema?

Usabilidad / alcanzabilidad: Nivel de capacitación requerido para que los usuarios logren sus objetivos con la aplicación / solución. Los requisitos de usabilidad deben tratarse tan seriamente como cualquier otro problema arquitectónico.

4.9 Arquitectura basada en componentes

Un componente es la manifestación física de un módulo, mientras que un módulo se define como una colección de código relacionado. Los

desarrolladores empaquetan físicamente los módulos de diferentes maneras, a veces dependiendo de su plataforma de desarrollo, así llamamos componentes al empaquetado físico de los módulos.

4.9.1 Alcance de los componentes

Los componentes ofrecen un mecanismo específico del lenguaje para agrupar los artefactos, a menudo anidándolos para crear estratificación. Como se muestra en la Figura 13, los componentes se pueden expresar de las siguientes formas:

- El componente más simple envuelve código a un nivel de modularidad más alto que las clases (o funciones, en lenguajes no orientados a objetos), este a menudo es llamado librería.
- Los componentes también aparecen como subsistemas o capas en la arquitectura.
- Como la unidad de trabajo desplegable para muchos procesadores de eventos.
- Otro tipo de componente, como un servicio que tiende a ejecutarse en su propio espacio de direcciones y se comunica a través de protocolos de red de bajo nivel como TCP / IP o formatos de nivel superior como REST o colas de mensajes, formando unidades independientes desplegables en arquitecturas como microservicios.

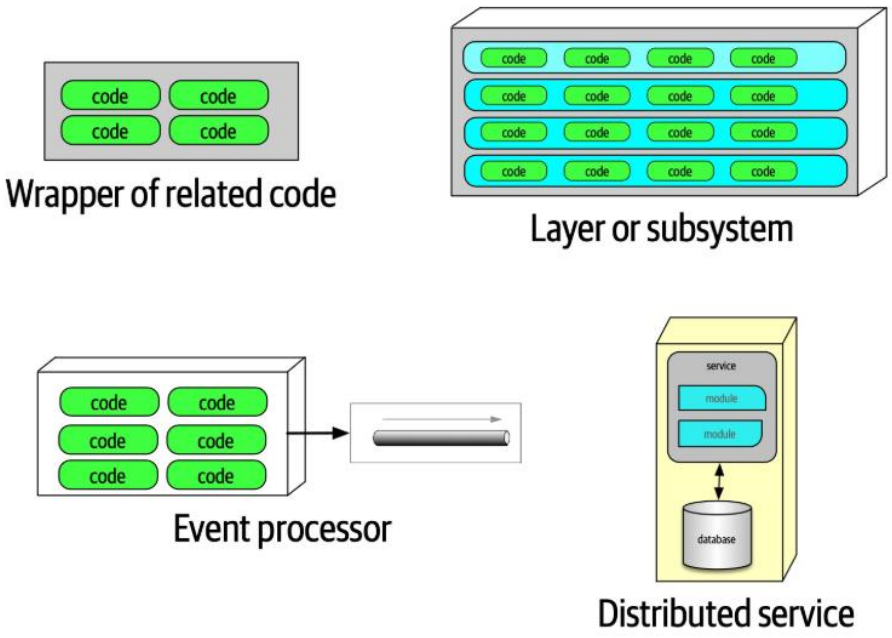


Figura 13. Diferentes variedades de componentes [6]

A menudo es útil tener un mayor nivel de modularidad que el nivel más bajo que ofrece un lenguaje. Por ejemplo, en las arquitecturas de microservicios, la simplicidad es uno de los principios arquitectónicos.

Los componentes forman el bloque de construcción modular fundamental en la arquitectura, lo que los convierte en una consideración crítica para los arquitectos. De hecho, una de las decisiones principales que debe tomar un arquitecto se refiere a la partición de componentes de nivel superior en la arquitectura.

Estos bloques de construcción tienen un conjunto bien definido de operaciones, roles y responsabilidades en la aplicación. La identificación de los componentes define el alcance de estos.

4.9.2 Identificación de componentes

La forma para identificar los componentes pasa por este tipo de proceso, que se describe en la Figura 14:

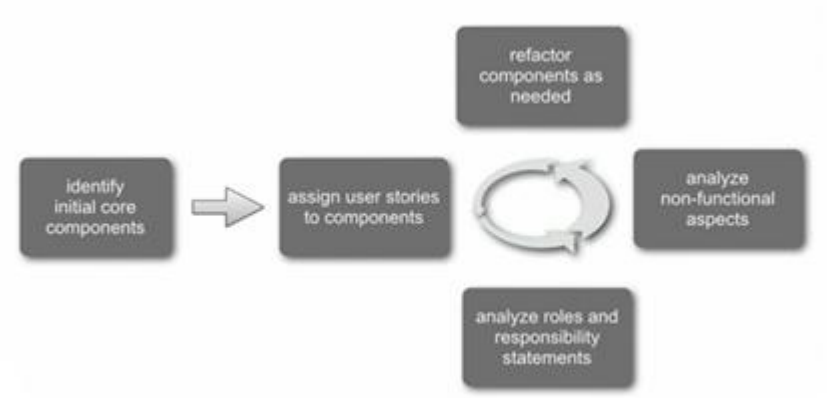


Figura 14. Proceso para identificación de componentes [6]

- Lo primero que se debe hacer es identificar esos componentes centrales iniciales que se convierten en candidatos. Este es el punto de partida, esos componentes generalmente siempre cambiarán. De esta forma inicia el proceso cíclico
- Ahora tomamos esos componentes candidatos y comenzamos a asignar historias de usuarios o requisitos a los componentes. ¿Quién debería hacer cada requisito? Comenzamos a asignar a cada componente lo que debe hacer.
- Luego comenzamos a analizar esos roles y declaraciones de responsabilidad

- A continuación, acoplamos y analizamos aquellos aspectos o requisitos no funcionales, en otras palabras, tal vez disponibilidad o escalabilidad o rendimiento, porque eso puede cambiar lo que hace ese componente o ese nivel de granularidad
- Una vez que hacemos los pasos anteriores, refactorizamos los componentes según sea necesario.

Ya sea demasiado grande o pequeño, está muy acoplado, no hay suficiente cohesión así continuamos este proceso durante todo el ciclo de vida del proyecto y más allá.

Acoplamiento de componentes: está definida como que tanto un componente conoce acerca de otro. Hay tres tipos de acoplamiento:

- Aferente: es el grado en el cual otros componentes son dependientes de uno. Estático
- Eferente: es el grado en el cual un componente objetivo es dependiente de otros componentes. Estático
- Temporal: son componentes que están acoplados debido a dependencias no estáticas o de timing

Además, se tienen cuatro niveles de acoplamiento, son niveles ajustados de acoplamiento que bajan hasta niveles ligeros de acoplamiento. Ahora, por supuesto, lo que queremos lograr en todas nuestras arquitecturas es un acoplamiento ligero y una fuerte cohesión.

- Patológico: es donde un componente se basa en el funcionamiento interno de otro componente. Es el peor nivel
- Externo: es donde múltiples componentes comparten un protocolo impuesto externamente o un determinado formato de datos.
- Control: es donde un componente pasa información a otro componente sobre qué hacer
- Data: es el grado en que los componentes están vinculados a algún contexto de datos compartidos.

Cohesión de componentes: la cohesión es el grado y la manera en que las operaciones de un componente están relacionadas entre sí. Así que ahora no estamos hablando de componentes unidos y dependencias, sino de un componente individual

4.10 Patrones arquitectónicos

En la materia Arquitecturas para sistemas de Software se nos presentó la idea de patrones de diseño, este es un concepto relacionado, pero es un poco diferente en el sentido de que los patrones de diseño generalmente solo hablan de la colaboración de las clases y existe una estructura muy simple. No tiene muchos tipos diferentes de partes móviles.

Los patrones arquitectónicos son diferentes, porque se tienen diferentes tipos de componentes dentro de un patrón arquitectónico. La componibilidad funciona de manera diferente aquí. Y así, aunque la intención es la misma, que es capturar la esencia de la estructura de cómo encajan estas cosas, se usan de manera ligeramente diferente de los patrones de diseño y por supuesto, están en un nivel mucho más alto.

4.11 MVC: Modelo, Vista, Controlador

MVC es un patrón de diseño de software. Describe la separación del software en tres elementos [9]

Modelo: gestiona los datos de una aplicación. Esto debe entenderse en un sentido estricto. Por supuesto, cualquier parte de una aplicación menos que trivial trata con los datos de la aplicación de una forma u otra, pero el modelo de MVC corresponde a elementos de datos visibles para el usuario y posiblemente sujetos a cambios por las interacciones del usuario. El modelo es independiente de la forma en que se representan los datos para el usuario o cualquier flujo de trabajo de la aplicación, por lo que se puede decir que el modelo es la parte central de una aplicación MVC. No es sorprendente que el desarrollo de un modelo sea uno de los primeros pasos de cualquier proyecto de software MVC.

Vista: describe la presentación de los datos y elementos de control (entradas, botones, casillas de verificación, menús, etc.) al usuario. Una vista puede proporcionar diferentes modos, como tablas paginadas o no paginadas, una lista formateada o una lista de enlaces, etc. Una vista también puede usar diferentes tecnologías, como un componente GUI instalado en la PC del usuario, una aplicación en un teléfono móvil o una página web para ser vista en un navegador.

Controlador: maneja la entrada del usuario y prepara el conjunto de datos necesarios para que la parte de la vista haga su trabajo. Mientras que una vista

muestra elementos del modelo, la vista nunca tiene que saber cómo se almacenan y recuperan los datos de algún almacenamiento persistente (base de datos). Esta es la responsabilidad del controlador. Debido a que la entrada del usuario determina qué debe hacer una aplicación a continuación, el controlador también contiene la lógica de la aplicación. Cualquier cálculo y transformación de datos ocurre en la parte de control de MVC.

Existe cierta imprecisión con respecto a los detalles de implementación. Esto proviene de los detalles técnicos del flujo de datos entre los elementos de la vista y los elementos del modelo. MVC no asume cuándo ocurren realmente las actualizaciones para los elementos de vista y los elementos del modelo y qué procedimiento se elige para mantenerlos sincronizados. Por eso, para MVC, encontrará muchos diagramas diferentes en la literatura.

En la Figura 15 se muestra un modelo MVC para Java, que es el lenguaje en que fue desarrollado el SAIPP-ES, en este se aprecia un modelo (almacenado en la memoria) que define el estado de la aplicación; una vista muestra los valores del modelo y envía las interacciones del usuario a un controlador; y el controlador prepara los datos del modelo, maneja la entrada del usuario y, en consecuencia, cambia los valores del modelo, y luego decide qué página de vista mostrar a continuación

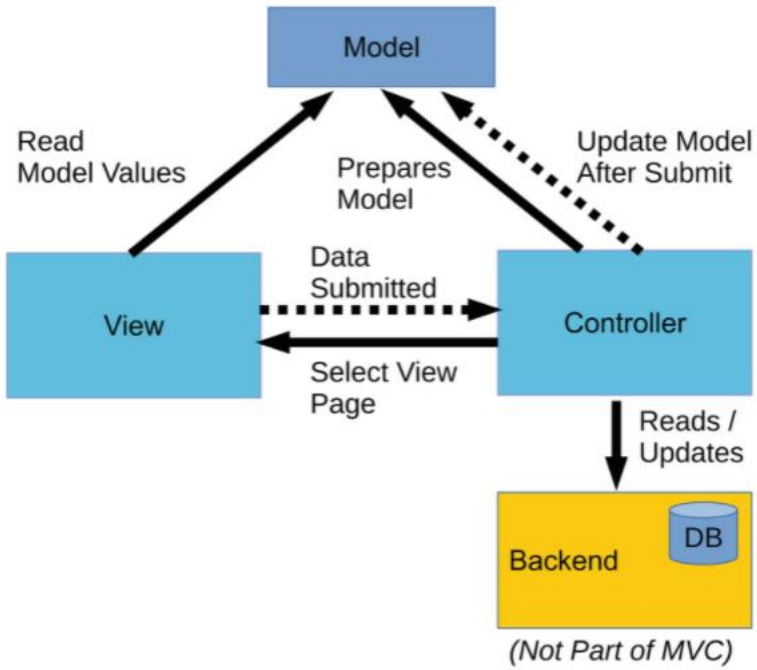


Figura 15. Patrón de diseño MVC para Java [9]

4.12 MVC en aplicaciones web

Las aplicaciones web imponen algunas restricciones si intentamos que funcionen como MVC. La distinción más importante proviene de la naturaleza sin estado del protocolo HTTP, que se utiliza para la comunicación entre la vista (ventana del navegador) y el controlador (servidor HTTP).

Con más detalle, las preguntas importantes sobre MVC para aplicaciones web son las siguientes [9]:

Sesiones: debido a la naturaleza sin estado de HTTP si el navegador envía una solicitud, tal vez porque el usuario ingresó una cadena en un campo de texto y luego presionó el botón Enviar, ¿cómo sabría el servidor qué usuario está realizando la solicitud? Esto generalmente se maneja mediante una sesión, que se identifica mediante un ID de sesión transmitido como una cookie, solicitud o parámetro POST. El framework gestiona las sesiones de forma transparente, por lo que no es necesario crear y mantener sesiones desde el interior del código de la aplicación.

Acceder a los valores del modelo desde la vista: con las aplicaciones web, algún tipo de motor de plantillas generalmente maneja la generación de la vista. Allí, podríamos tener expresiones como $ {user.firstName} para leer el contenido de una entrada de modelo.

Extensión de datos transmitidos: si los datos se envían desde la página web al servidor, básicamente tenemos dos opciones. Primero, se podría transmitir el formulario completo. En segundo lugar, solo los datos que cambiaron podrían enviarse al servidor. Este último reduce el tráfico de la red, pero requiere cierta lógica de secuencia de comandos (JavaScript) para realizar la recopilación de datos en la página web.

Actualización de la vista: con las aplicaciones web, la forma en que se actualiza una vista es crucial. O la página completa se carga después de que el controlador ejecuta una solicitud, o solo aquellas partes de una página web que realmente necesitan una actualización se transmiten desde el servidor al navegador. Nuevamente, el último método reduce el tráfico de la red.

Java EE / Jakarta EE nombra a JSF (Java Server Faces) como el framework web dedicado. JSF, a diferencia de MVC, utiliza un enfoque orientado a componentes para crear aplicaciones web.

5. CAPITULO 5 - Sistema de Auditoria Interna para Procesos y Procedimientos a Estaciones de Servicio (SAIPP-ES)

5.1 Planteamiento del Problema

Primax del Ecuador posee estaciones de servicio en todo el territorio ecuatoriano, en las cuales vende gasolina y diésel en las pistas de abastecimiento, además de productos de consumo y alimentos preparados en las tiendas de conveniencia ubicadas en la misma estación, posee más de 100 estaciones a nivel nacional.

Para el control de sus procesos operativos y de calidad realiza periódicamente auditorías internas a través de visitas de evaluación a las estaciones, en estas utiliza diversos formatos de checklist previamente diseñados en el sistema, los mismos que luego de ser alimentados generan puntajes que son mostrados en matrices de riesgo y reportes consolidados.

Las estaciones están constantemente diversificando su portafolio de servicios tanto de pistas como de tiendas, por lo que, las auditorias deben adaptarse a estos cambios y el sistema debe ser resiliente a los mismos, a través de la creación en el sistema de nuevos tipos de checklist o modificaciones a los ya existentes, lo que debe poder ser realizado apoyándose en la parametrización del sistema, procurando agregar más código al sistema solo para la implementación de nuevas funcionalidades.

5.2 Alcance del Sistema

El Sistema de Auditoria Interna para Procesos y Procedimientos a Estaciones de Servicio (SAIPP-ES) fue desarrollado con el siguiente alcance:

- Sistema Web responsivo con seguridades para el acceso y roles para los diferentes tipos de usuarios
- Parametrización del sistema para crear en tiempo de ejecución estaciones, cargos, tipos de evaluación, tipos de checklist, procesos, KPIs, criterios de evaluación, asignar puntajes a cada criterio, entre otros.
- Crear plantillas para checklist que faciliten la auditoría interna tanto para el control y calidad de los procedimientos además de las actividades que se realizan en una estación de servicio.

- Se podrán crear checklist de dos tipos a partir de una plantilla: de criterio general y de criterio especifico
- Agendamiento de una visita de auditoría interna donde se asigne la estación, el tipo de evaluación y los tipos de checklist con los que realizara su labor cada auditor
- Ejecución de las auditorías a través de la asignación de puntajes a los KPIs contenidos en un checklist, comentarios y evidencias fotográficas.
- Generación automática de matrices de riesgo y reporte de novedades en cada auditoria.
- Ingresar un plan de acción por parte del responsable de la estación luego de cada evaluación.
- Generación de reportes consolidados en formato PDF y Excel

5.3 Requisitos de los interesados (stakeholders) del sistema

El SAIPP-ES está enfocado en el diseño de un conjunto de herramientas (checklists) con las cuales se puedan realizar auditorías que generen resultados medibles a través de reportes, esto será de utilidad a:

- Administrador del sistema
- Auditores
- Responsables de Estación
- Gerente de Auditoria

En la tabla 4 se identifican los requerimientos de uso de cada interesado del sistema atendiendo a los atributos de calidad: funcionalidad, rendimiento, modificabilidad.

CRITERIO	USO/INTERES	USUARIO
FUNCIONALIDAD	Permitir la creación de plantillas de checklists para las evaluaciones de las auditorias	Administrador Gerente de Auditoria
	Permitir el agendamiento de visitas a estaciones	Administrador
	Permitir la ejecución de auditorías utilizando los checklist	Auditores
	Los checklist deben ser fácilmente accesibles	Administrador Auditores

	Permitir la inspección de la información a través de reportes	Administrador Auditores Responsables de Estación Gerente de auditoria
	Ingresar Plan de Acción	Responsables de Estación
USABILIDAD	Función para exportar información de reportes a archivos	Administrador Auditores Responsables de Estación Gerente de auditoria
MANTENIBILIDAD	Se permite crear plantillas de checklist para diversos tipos de evaluación	Administrador
	Dar nuevas funcionalidades al sistema	Administrador

Tabla 4. Requerimientos de Uso de cada involucrado en el sistema

En la tabla 5 tomamos los resultados obtenidos anteriormente y los transformamos en una aproximación a los requisitos del sistema:

#	DESCRIPCION
1	Permitir la creación de plantillas de checklists para las evaluaciones de las auditorias
2	Permitir el agendamiento de visitas a estaciones
3	Permitir la ejecución de auditorías utilizando los checklist
4	Permitir la inspección de la información a través de reportes
5	Ingresar Plan de Acción
6	Los checklist deben ser fácilmente accesibles
7	Se permite crear plantillas de checklist para diversos tipos de evaluación
8	Implementar nuevas funcionalidades al sistema
9	Exportar información a archivos

Tabla 5. Requisitos del sistema

5.4 Casos de uso del sistema

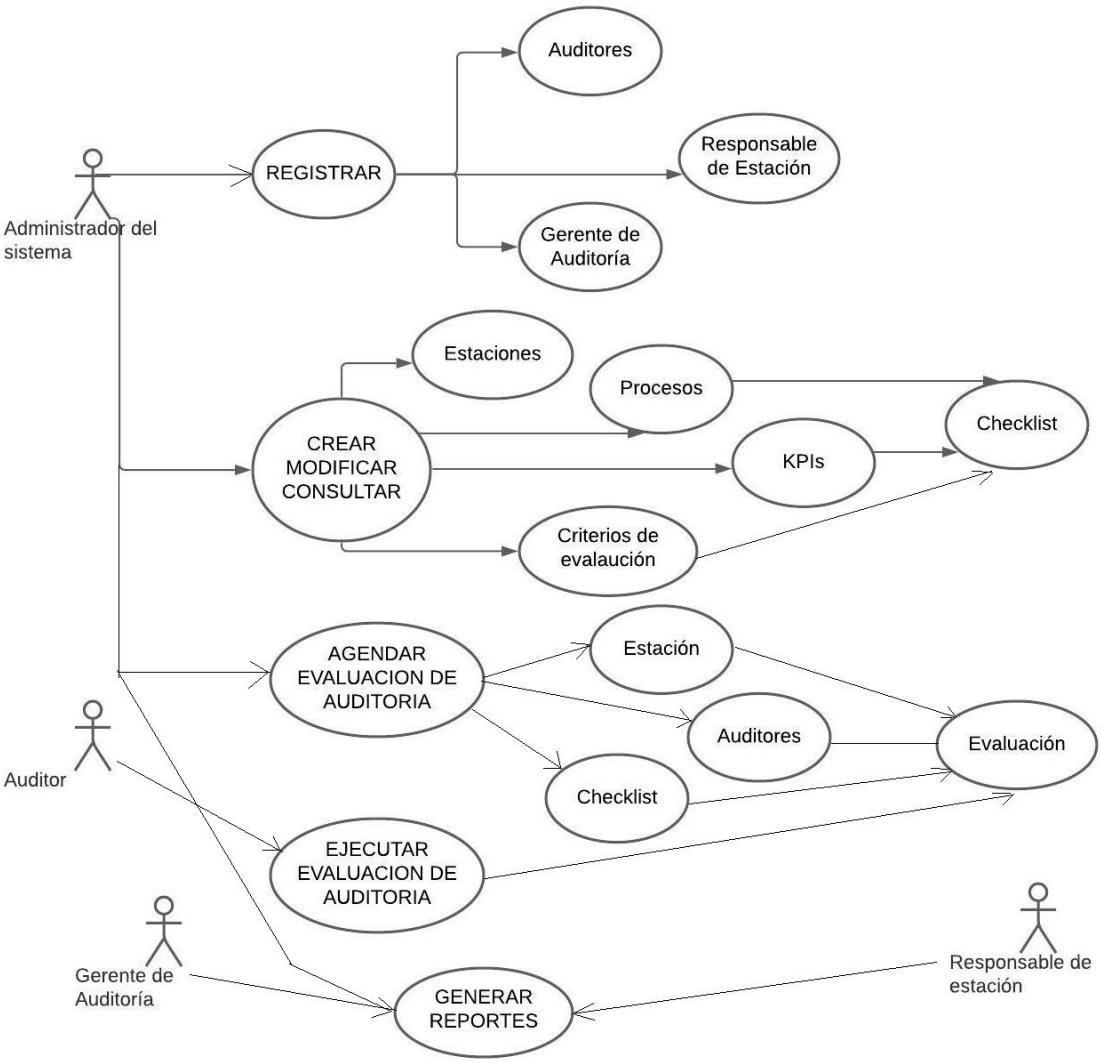


Figura 16. Caso de Uso del Sistema

Del diagrama anterior procedemos a describir los casos de uso que se derivan y en los cuales quedan implícitos los requisitos funcionales.

CASO DE USO - 1	**REGISTRAR ROLES DEL SISTEMA**
Descripción	Los roles que se asignaran a los usuarios del sistema y para que tengan acceso a determinadas opciones
Actores	Administrador de sistema
Precondiciones	Login correcto en el sistema
Postcondiciones	Roles
Flujo normal	

1. Ingreso al sistema 2. Nombre del rol y asignar opciones del sistema 3. Rol creado

Tabla 6. Caso de Uso 1: Registrar roles del sistema

CASO DE USO - 2	**REGISTRAR USUARIOS DEL SISTEMA**
Descripción	Dar de alta nuevos usuarios del sistema
Actores	Administrador de sistema
Precondiciones	Login correcto en el sistema
Postcondiciones	Nuevos Auditores/Responsable de Estación/Gerente de auditoria
Flujo normal	
1. Ingreso al sistema 2. Llenar datos del usuario y asignar un rol 3. Dar de alta al auditor	

Tabla 7. Caso de Uso 2: Registrar usuarios del sistema

CASO DE USO - 3	**CREAR/MODIFICAR/CONSULTAR ESTACION**
Descripción	Se pueden crear nuevas estaciones, consultarlas o modificarlas
Actores	Administrador de sistema
Precondiciones	Login correcto en el sistema
Postcondiciones	Estación creada, modificada o consultada
Flujo normal	
1. Ingreso al sistema 2. Llenar datos de la estación 3. Estación creada o modificada	

Tabla 8. Caso de Uso 3: Crear/Modificar/Consultar Estación

CASO DE USO - 4	**CREAR/MODIFICAR/CONSULTAR PROCESO**
Descripción	Se pueden crear procesos que serán parte de un checklist
Actores	Administrador de sistema
Precondiciones	Login correcto en el sistema
Postcondiciones	Proceso creado
Flujo normal	
1. Ingreso al sistema 2. Ingresar nombre y datos del proceso 3. Proceso creado o modificado	

Tabla 9. Caso de Uso 4: Crear/Modificar/Consultar Proceso

CASO DE USO - 5	**CREAR/MODIFICAR/KPI**
Descripción	Se pueden crear KPIs que serán parte de un proceso
Actores	Administrador de sistema
Precondiciones	Login correcto en el sistema

Postcondiciones	KPI creado
Flujo normal	
1. Ingreso al sistema 2. Seleccionar un proceso y crear un nuevo KPI 3. KPI creado o modificado	

Tabla 10. Caso de Uso 5: Crear/Modificar/KPI

CASO DE USO - 6	**CREAR/MODIFICAR/CRITERIO DE EVALUACION**
Descripción	Se pueden crear criterios de evaluación para un determinado KPI
Actores	Administrador de sistema
Precondiciones	Login correcto en el sistema
Postcondiciones	Criterio de evaluación creado
Flujo normal	
1. Ingreso al sistema 2. Seleccionar un proceso, luego un KPI y crear el nuevo criterio de evaluación para este KPI 3. Criterio de evaluación creado o modificado	

Tabla 11. Caso de Uso 6: Crear/Modificar/Criterio de Evaluación

CASO DE USO - 7	**CREAR/MODIFICAR/ PLANTILLA DE CHECKLIST**
Descripción	Se puede crear una plantilla de checklist para la evaluación en una auditoria
Actores	Administrador de sistema
Precondiciones	Login correcto en el sistema
Postcondiciones	Checklist creado
Flujo normal	
1. Ingreso al sistema 2. Seleccionar un proceso, los KPIs que correspondan a este proceso, repetir este procedimiento para agregar más procesos a la plantilla de checklist 3. Plantilla de Checklist creada o modificada	

Tabla 12. Caso de Uso 7: Crear/Modificar/Checklist

CASO DE USO - 8	**AGENDAR EVALUACION DE AUDITORIA**
Descripción	Se puede agendar una evaluación de Auditoria para una estación
Actores	Administrador de sistema
Precondiciones	Login correcto en el sistema
Postcondiciones	Agenda creada
Flujo normal	
1. Ingreso al sistema 2. Seleccionar una estación, un auditor, una plantilla de checklist, fecha y hora 3. Agenda de auditoria programada	

Tabla 13. Caso de Uso 8: Agendar evaluación de auditoria

CASO DE USO - 9	EJECUTAR EVALUACION DE AUDITORIA
Descripción	Se puede ejecutar una evaluación de auditoria previamente agendada
Actores	Auditor
Precondiciones	Login correcto en el sistema
Postcondiciones	Checklist evaluado
Flujo normal	
1. Ingreso al sistema 2. Seleccionar una agenda en el calendario 3. Llenar los datos en el checklist 4. Evaluación de Auditoria ejecutada	

Tabla 14. Caso de Uso 9: Ejecutar evaluación de auditoria

CASO DE USO - 10	INGRESAR PLAN DE ACCION
Descripción	Se puede ingresar un plan de acción para la evaluación de auditoría realizada a la estación donde es responsable
Actores	Responsable de Estación
Precondiciones	Login correcto en el sistema
Postcondiciones	Plan de acción
Flujo normal	
1. Ingreso al sistema 2. Seleccionar un checklist evaluado 3. Ingresar plan de acción	

Tabla 15. Caso de Uso 10: Ingresar Plan de Acción

CASO DE USO - 11	GENERAR REPORTES
Descripción	Se puede generar reportes de las evaluaciones de auditoria realizadas
Actores	Administrador del sistema/Auditor/Responsable de Estación/Gerente de Auditoria
Precondiciones	Login correcto en el sistema
Postcondiciones	Información en pantalla
Flujo normal	
4. Ingreso al sistema 5. Seleccionar un reporte del menú 6. Seleccionar los criterios en los filtros disponibles 7. Información en pantalla	

Tabla 16. Caso de Uso 11: Generar Reportes

En la tabla 17 se relacionan actores y casos de uso:

ACTOR	FUNCION PRINCIPAL/OBJETIVO	CASOS DE USO
Administrador del Sistema	Acceder a todo el sistema, crear los checklist y agendar las visitas	Registrar Roles de Usuario Registrar usuarios del sistema

		Crear/Modificar/Consultar Estación Crear/Modificar/Consultar Proceso Crear/Modificar/Consultar KPI Crear/Modificar/Consultar Criterio de evaluación Crear/Modificar/Consultar Plantilla de Checklist Agendar Evaluación de Auditoria Generar Reportes
Auditor	Ejecutar las auditorias de evaluación agendadas por el Administrador del Sistema	Ejecutar Evaluación de Auditoria Generar Reportes
Responsable de Estación	Ingresar Plan de Acción y tener acceso a información de los reportes asignados a su rol	Ingresar Plan de Acción Generar Reportes
Gerente de Auditoria	Tener acceso a todos los reportes del sistema	Generar Reportes

Tabla 17. Actores del Sistema

En la tabla 18, se obtienen las funcionalidades del sistema de los casos de uso:

FUNCIONALIDADES	DESCRIPCION
Registrar Roles	Permite añadir un nuevo rol al sistema
Registrar Usuarios	Permite añadir un nuevo usuario al sistema
Crear/Modificar/Consultar Estación	Permite gestionar una nueva estación
Crear/Modificar/Consultar Proceso	Permite gestionar un nuevo proceso
Crear/Modificar/Consultar KPI	Permite gestionar un nuevo KPI
Crear/Modificar/Consultar Criterio de Evaluación	Permite gestionar un nuevo criterio de evaluación
Crear/Modificar/Consultar Plantilla de Checklist	Permite gestionar un nuevo checklist
Agendar Evaluación de auditoria	Permite agendar una evaluación de auditoria
Ejecutar Evaluación de Auditoria	Permite ejecutar una evaluación de auditoria
Ingresar Plan de Acción	Permite ingresar un plan de acción
Generar Reportes	Permite generar reportes de los checklist evaluados

Tabla 18. Funcionalidades del Sistema

5.5 Arquitectura del Sistema SAIPP-ES

El SAIPP-ES es una aplicación web que fue diseñada utilizando el patrón de diseño MVC, ver Figura 17.

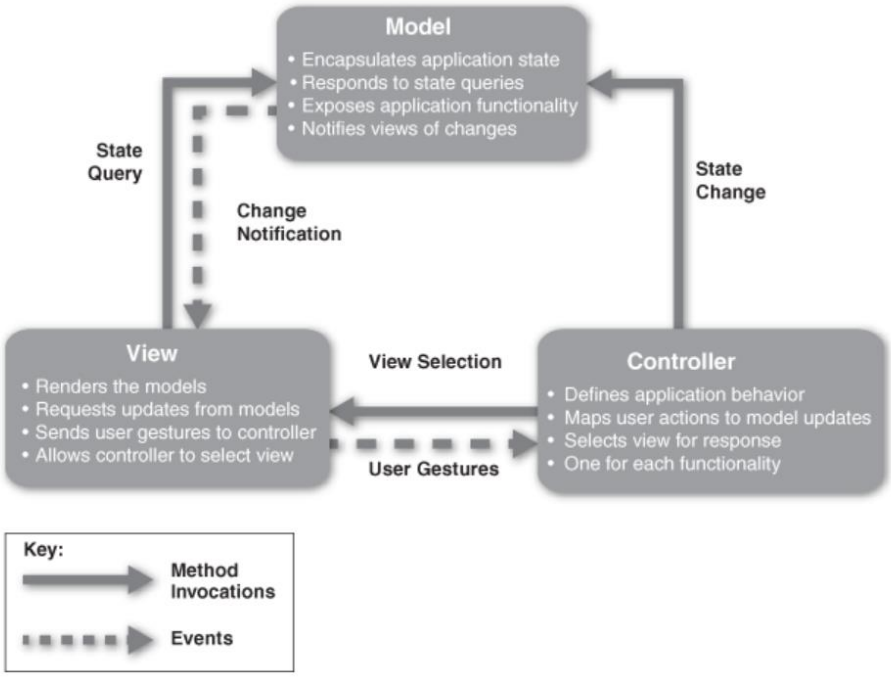


Figura 17. Patrón de Arquitectura MVC[8]

Se procede a agrupar las funcionalidades en paquetes como se muestra en la Figura 18:

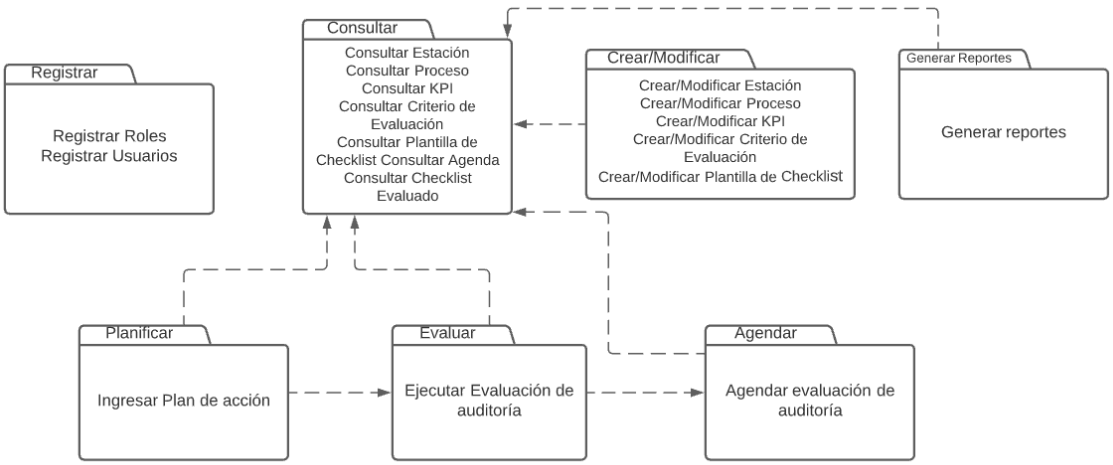


Figura 18. Diagrama de Paquetes del SAIPP-ES

Las dependencias que existen entre paquetes son:

- Consultar depende de que se hayan creado las estaciones, procesos, KPIs, Criterios de Evaluación y Plantilla de Checklist
- Agendar una evaluación de auditoria depende consultar una plantilla de checklist
- Ejecutar depende de consultar y que se haya agendado una evaluación de auditoria
- La planificación depende que se haya ejecutado una evaluación de auditoria
- Generar reportes depende de consultar la información de checklists

Mostramos el diagrama de despliegue en la Figura 19.

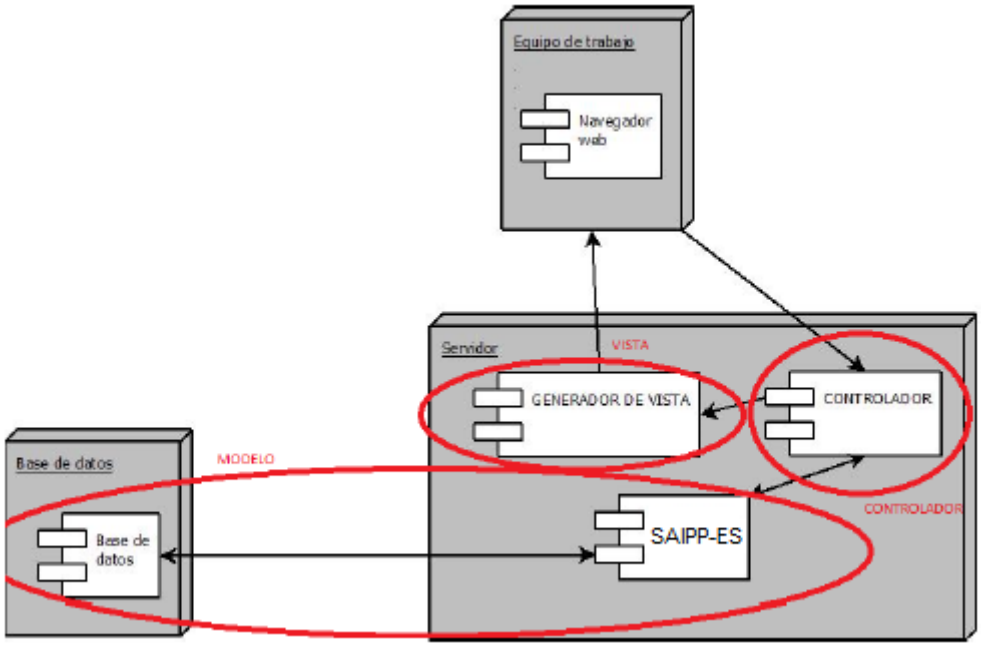


Figura 19. Diagrama de Despliegue del SAIPP-ES

5.6 Requisitos de Calidad

El sistema deberá ser diseñado con una alta parametrización de forma tal que, si se requiere realizar evaluaciones de auditorías internas a nuevos procesos o actividades, estás puedan ser ejecutadas utilizando nuevas plantillas de checklist, creadas previamente por el Administrador del sistema.

Un checklist es considerado la principal herramienta utilizada durante la evaluación de una auditoría interna, este se debe poder crear en tiempo de ejecución, en base a una plantilla previamente diseñada a la medida de los requerimientos de evaluación del auditor.

Se han definido dos tipos de checklist: de criterio general y de criterio especifico.

Un checklist de criterio general, permite escoger un tipo de criterio con el cual serán evaluados todos los KPIs que conforman los procesos a auditar, ver ejemplo en tabla 19

KPI	Criterio de evaluación
Limpieza de mesas	Cumple/No cumple
Limpieza del piso	Cumple/No cumple
Uso de desinfectantes	Cumple/No cumple

Tabla 19. Proceso: buenas prácticas de limpieza

Un checklist de criterio especifico es similar al de criterio general con la diferencia que los criterios de evaluación pueden ser customizados para cada KPI de un proceso, ver ejemplo en tabla 20.

KPI	Criterios de evaluación
Arqueo de caja	En cero/Menor a $1/Mayor a $1
Comprobantes de venta	Completos/Incompletos
Cumplimiento de Horario	Puntual/Atraso/Falta

Tabla 20. Proceso: cierre de caja

Por ende, el principal atributo de calidad que debe contemplar la arquitectura de este sistema es que los cambios requeridos por el Auditor ante un nuevo tipo de evaluación de auditoría interna puedan ser realizados en tiempo de ejecución apoyado en una amplia parametrización del sistema o en su defecto que el tiempo de desarrollo para realizar dichos cambios por parte del ingeniero de software sea el menor posible y sin que la arquitectura del sistema se vea mayormente comprometida.

6. CAPITULO 6 - Mejora del SAIPP-ES utilizando la herramienta ArchE

6.1 Herramienta ArchE

ArchE es una herramienta experimental desarrollada por el SEI (Software Engineering Institute) y cuya última versión es la 3.0 desarrollada en 2008. Su carácter es experimental y su finalidad es fundamentalmente académica. Para este trabajo de fin de maestría se utilizará para evaluar la arquitectura del sistema en base al atributo de calidad de modificabilidad realizando propuestas para mejorar la arquitectura bajo evaluación, de manera que el arquitecto o diseñador pueda elegir la mejor solución.

El objeto de esta herramienta se centra en que podamos transformar de una forma sistemática un conjunto de requerimientos tanto funcionales como en forma de atributos de calidad y determinados por lo general por el arquitecto de software en forma de escenarios y llevarlos a un diseño arquitectónico que cumpla con estos escenarios

Según [10], ArchE contiene conocimientos de atributos de calidad, pero no conoce ningún dominio del problema. Por tal motivo, ArchE puede ofrecer recomendaciones sobre cómo satisfacer requisitos de atributos de calidad, pero no conoce que significan estas recomendaciones para el arquitecto con respecto al particular dominio del sistema.

ArchE es un sistema basado en reglas, en el cual los modelos de atributos de calidad son vistos como marcos. ArchE utiliza Jess (intérprete basado en reglas de Java). Además, ArchE está implementado como una aplicación Eclipse, la cual proporciona una familiarización inmediata con la interfaz de usuario y un concepto de operación para cualquiera que haya usado Eclipse.

En la Figura 26 se muestra el flujo general de ArchE:

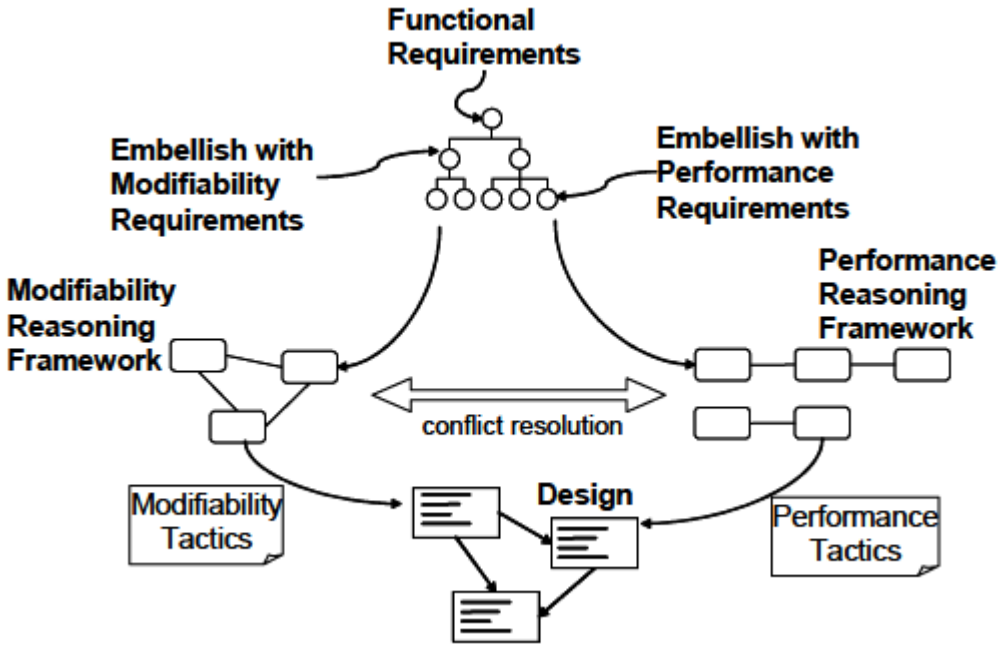


Figura 20. Flujo general de ArchE [10]

Según [10], los conceptos más importantes que se pueden observar en este grafico se detallan a continuación:

Escenarios de atributos de calidad: de conformidad con lo indicado en el capítulo 3 se tiene una estructura formal para estos escenarios que incluye seis partes: estímulo, fuente del estímulo, entorno, artefacto que es estimulado, respuesta y medida de respuesta.

Marcos de razonamiento: es un conjunto de conocimientos sobre un atributo de calidad en particular, este incluye métodos para calcular una respuesta medida (el parámetro dependiente), dada una colección de parámetros independientes. También incluye tácticas arquitectónicas que permiten ajustar parámetros independientes para afectar (y controlar) el valor del parámetro dependiente.

Responsabilidades: es una actividad realizada por el software que está siendo diseñado. ArchE utiliza las responsabilidades como un medio para expresar los requisitos funcionales, como una parte integral de los escenarios de atributos de calidad y como un medio para integrar los modelos producidos por varios marcos de razonamiento de atributos de calidad.

Una vez definidos estos conceptos, ArchE funciona de la siguiente manera: [10]

Dentro de ArchE los requisitos funcionales se presentan como grafos de responsabilidades y son complementados con requisitos de atributos de calidad, en la forma de escenarios de atributos de calidad.

Para cada atributo de calidad, existen marcos de razonamiento que convierten dichos escenarios de calidad en modelos específicos de atributos de calidad. Cada modelo representa un diseño que satisface los requisitos especificados para dicho atributo de calidad.

Los marcos de razonamiento resuelven conflictos entre diferentes modelos de atributos de calidad para crear un modelo global que satisfaga todos los requisitos de atributos de calidad. Este modelo global se convierte entonces en una representación arquitectónica del diseño.

Este principio de diseño se basa en los siguientes puntos:

- Establecer requisitos funcionales
- Dependencias entre requisitos
- Requisitos de atributos de calidad
- Diseño inicial conteniendo sólo el sistema
- Comprobar que el diseño es bueno, produciendo modelos de atributos de calidad que proporcionen información acerca de los atributos de calidad
- Extraer información requerida del modelo desde el diseño (interpretación)
- Correr el modelo para calcular los valores para los requisitos de atributos de calidad (evaluación)

- Intentar una mejora, mediante un conjunto de tácticas que mejoren la arquitectura

- Interpretar el modelo para determinar posibles tácticas

- Aplicar las tácticas al diseño mediante el cambio de elementos, relaciones y sus propiedades.

6.2 Definición de escenarios

En este se analizan los costos de realizar modificaciones en el sistema, como afectan al sistema y a otros módulos. Aquí se valora la habilidad del sistema para que sea flexible frente a los cambios.

Las mediciones se realizan realizando cambios en los diferentes módulos y observando cuanto costo tienen (entendiendo como costo el tiempo que conlleva realizar dichos cambios).

En la arquitectura que planteamos para medirlo estableceremos unos valores de modificabilidad asociados a cada módulo (en días de trabajo). Se utilizará ArchE para verificar si se cumplen los escenarios propuestos y de no ser así se utilizarán las tácticas que esta herramienta nos proponga para conseguir el escenario marcado en los escenarios de modificabilidad.

Escenario 1:

Elemento	Descripción
Fuente	Ingeniero de software
Estimulo	Modificación del Módulo de Agendar
Artefacto	Sistema
Entorno	En tiempo de mantenimiento
Respuesta	Modificación de parámetros del Módulo de agendar
Medida	En 27 días

Escenario 2:

Elemento	Descripción
Fuente	Ingeniero de software
Estimulo	Modificación de Modulo de Evaluar
Artefacto	Sistema
Entorno	En tiempo de mantenimiento
Respuesta	Modificación de criterios de evaluación
Medida	En 25 días

Tabla 21. Escenarios de modificabilidad

6.3 Evaluación de la Arquitectura en ArchE

6.3.1 Definición de funciones y responsabilidades

De acuerdo con el diagrama de casos de uso del sistema que se muestra en la figura 16, podemos agrupar todos los casos de uso en un conjunto de funcionalidades básicas del sistema, como se muestra en la tabla 22 y que relacionamos anteriormente en la tabla 5.

ID	Descripción
01	Vista
02	Controlador
03	Generar de Reportes
04	Consultar
05	Crear-Modificar
06	Agendar
07	Evaluar
08	Planificar

Tabla 22. Relación de funciones del sistema SAIPP-ES

Una vez definidas las funciones estas son ingresadas en ArchE, ver Figura 21:

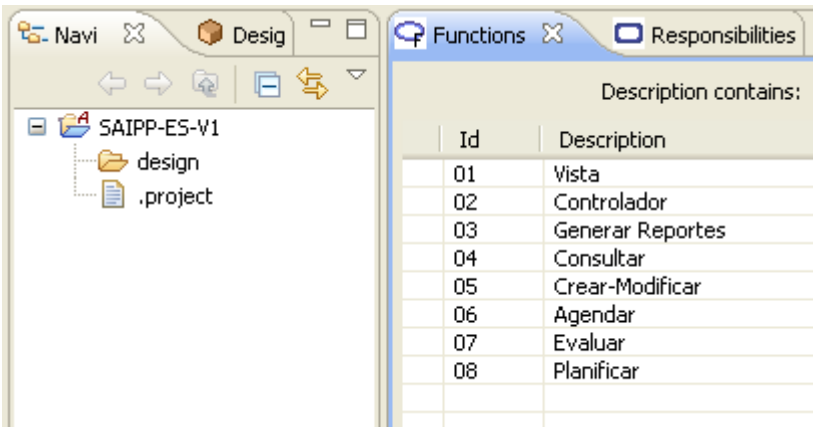


Figura 21. Definición de funciones

A continuación, ArchE genera automáticamente las responsabilidades y el mapeo de las funciones con las responsabilidades, tal como se muestra en la Figura 22 y 23. El arquitecto puede posteriormente modificar o asignar nuevas responsabilidades

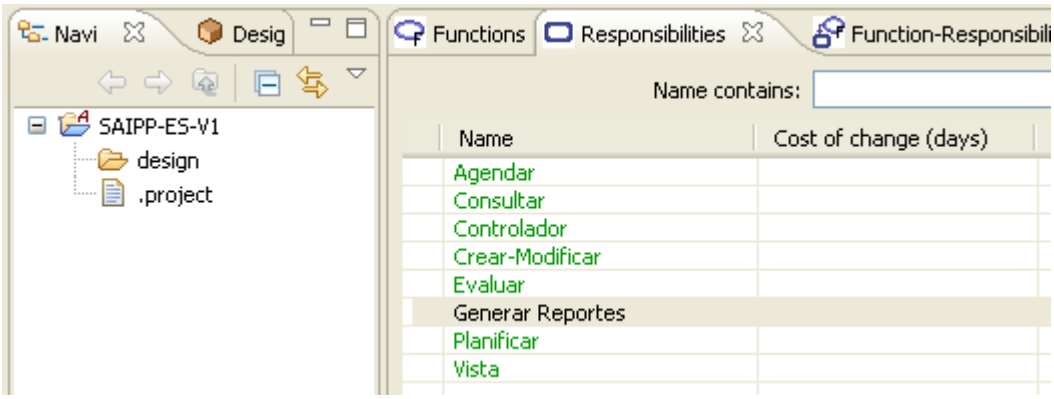


Figura 22. Definición de responsabilidades

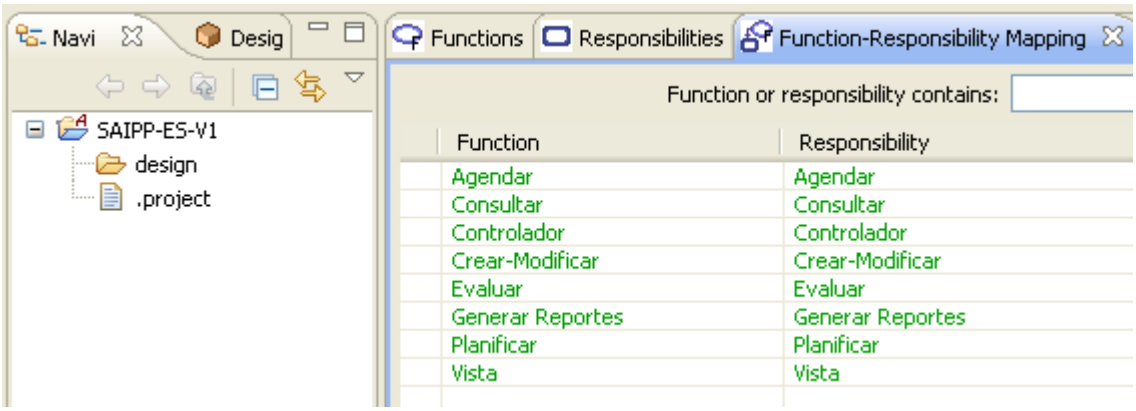


Figura 23. Mapeo de funciones y responsabilidades

6.3.2 Definición de atributos de las Responsabilidades

Este es un paso muy importante puesto que ArchE luego de definir los escenarios realizara un cálculo automático de los días requeridos para realizar las modificaciones planteadas. Inicialmente ArchE asigna un valor promedio de costo en días para cada una de las responsabilidades, para este caso asignó un costo de 7.5 a cada una.

Procedemos a definir los atributos de cada responsabilidad con valores aproximados a la realidad, con esto indicamos a la herramienta ArchE el tiempo de costo en días, ver Figura 24.

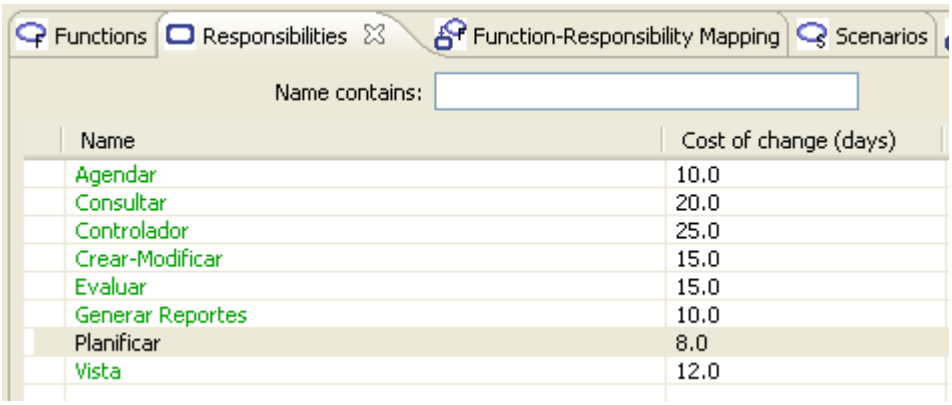


Figura 24. Atributos de responsabilidades

6.3.3 Definición de relaciones

A continuación, se muestran en la Tabla 23 las diferentes relaciones de dependencia que representan el impacto de la modificabilidad de una responsabilidad sobre otra y en la Figura 25 como quedan ingresadas en la herramienta. Es muy importante arrancar ahora los marcos de razonamiento en este caso el de modificabilidad para que se puedan establecer dichas relaciones.

Responsabilidad padre	Vista	Controlador	Generar Reportes	Consultar	Crear-Modificar	Agendar	Evaluar	Planificar
Vista		D				D	D	D
Controlador			D			D	D	D
Generar Reportes				D				
Consultar								
Crear-Modificar		D						
Agendar				D				
Evaluar				D		D		
Planificar				D			D	

Tabla 23. Mapeo de relaciones

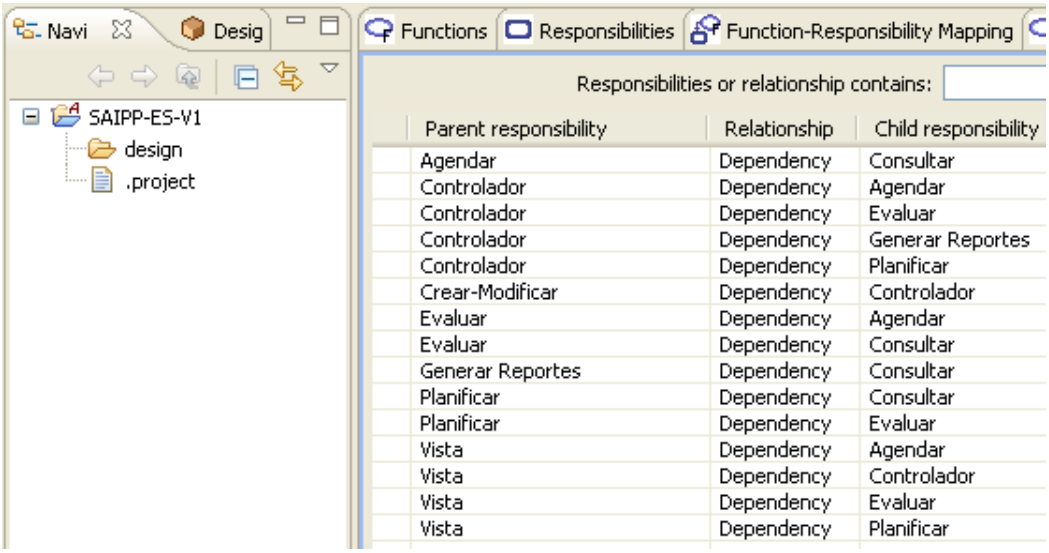


Figura 25. Relaciones de dependencia

6.3.4 Definición de Escenarios

A continuacion se procede a ingresar los escenarios en la herramienta ArchE.

El escenario M1 se muestra ingresado en la herramienta en la Figura 26.

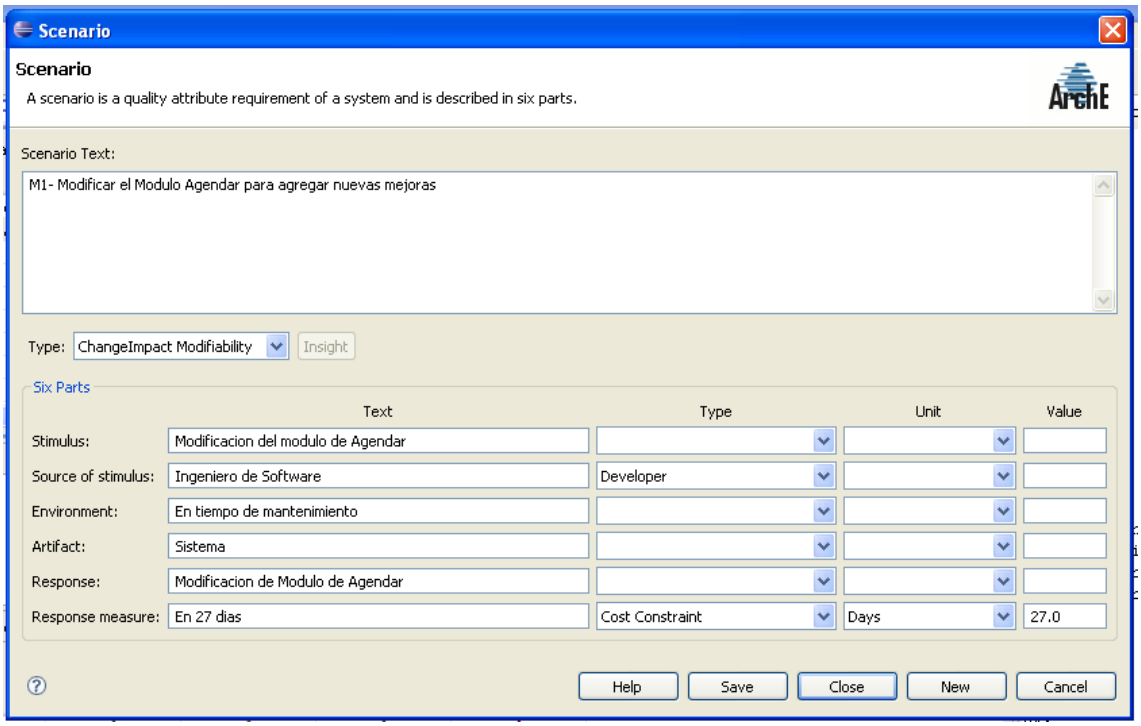


Figura 26. Definición de escenario M1

El escenario M2 se muestra ingresado en la herramienta en la Figura 27.

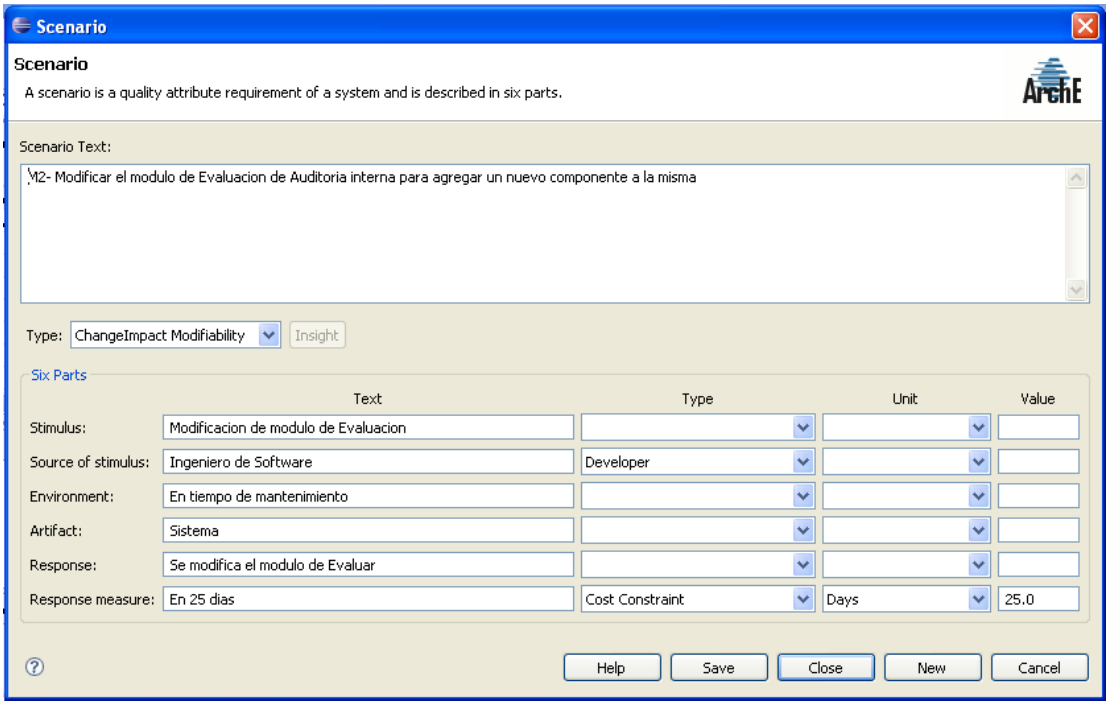


Figura 27. Definición de escenario M2

En el parametro Response measure se tiene que ingresar una cantidad de dias que represente con holgura las modificaciones que se deben realizar para cumplir con este escenario puesto que ArchE en el mapeo de Escenarios a Responsabilides realiza un calculo automatico tomando como base los valores definidos en dias para el costo de cambios de cada responsabilidad.

Hasta este paso si consultamos los escenarios ingresados se puede observar que estos se cumplen, ver figura 28.

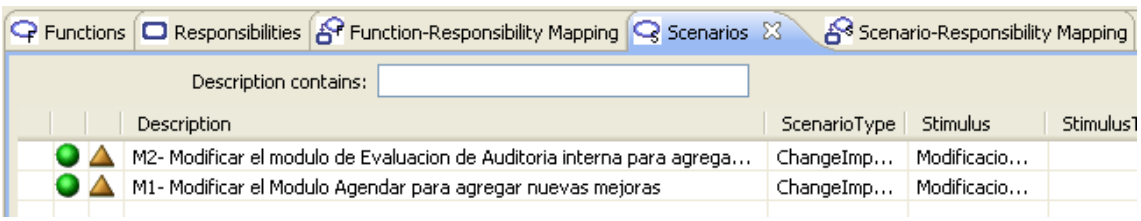


Figura 28. Cumplimiento de escenarios

6.3.5 Mapeo de Escenarios a Responsabilidades

Se procede a mapear los escenarios a las responsabilidades, tal como se muestra en la tabla 24.

	ESCENARIOS	
Responsabilidades	**M1**	**M2**
Vista		
Controlador		
Generar Reportes		
Consultar	X	
Crear-Modificar		
Agendar	X	X
Evaluar		X
Planificar		

Tabla 24. Mapeo de escenarios y responsabilidades

Se procede a ingresar el mapeado en la herramienta ArchE, ver la Figura 29.

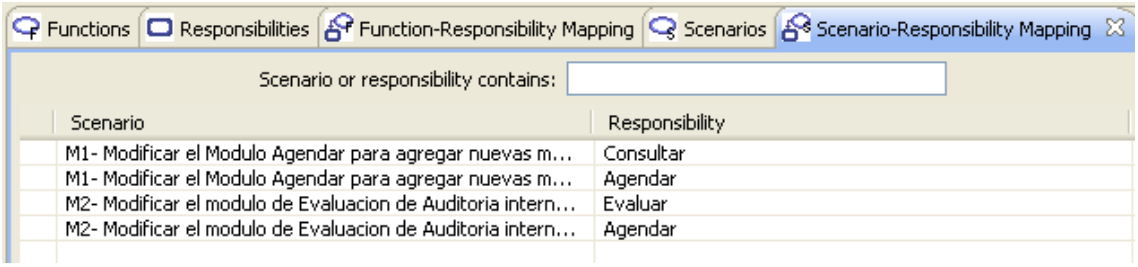
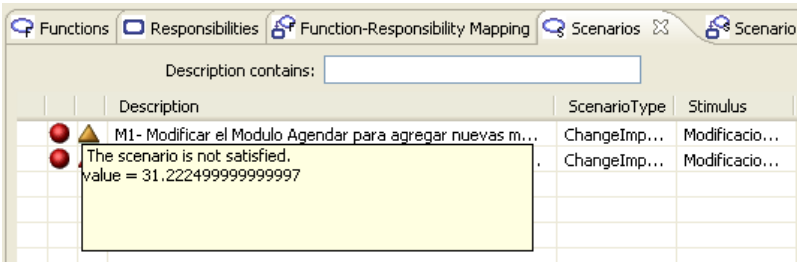


Figura 29. Mapeo de escenario/responsabilidad

6.3.6 Evaluación mediante la herramienta ArchE

Luego de haber introducido las responsabilidades, los escenarios y habiendo realizado el mapeo entre ambos, se ejecutan automáticamente los marcos de razonamiento y se obtienen los siguientes resultados iniciales en la Figura 30, ninguno de los escenarios se cumple.



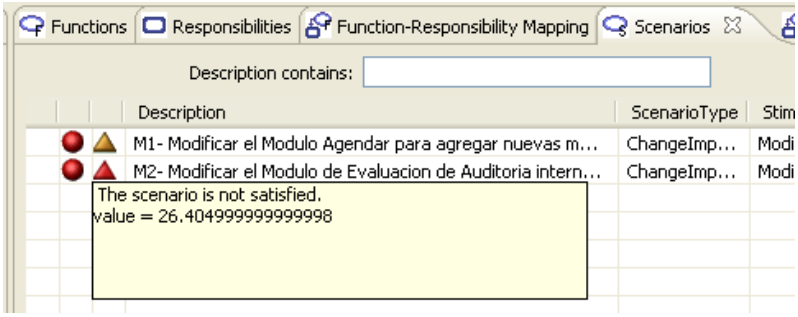


Figura 30. Resultados de los escenarios de modificabilidad

Podemos observar en la Figura 31 una vista de las relaciones de dependencias que se han establecido.

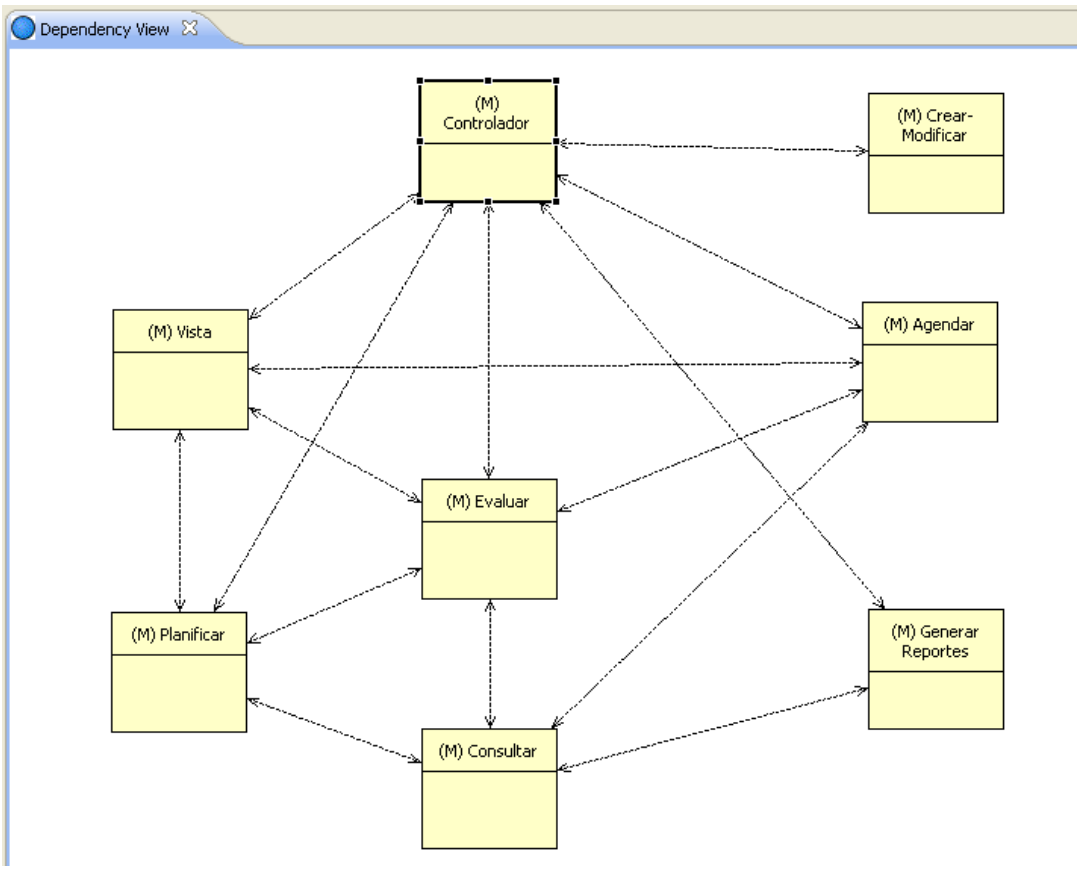


Figura 31. Resultados de los escenarios de modificabilidad

La herramienta ArchE nos ofrece de forma visual su valoración utilizando un sistema de colores que se interpretan de la siguiente manera:

Para indicar si el escenario cumple o no un círculo puede ser

- Rojo: no se cumple
- Verde: si se cumple

Para indicar el estado del escenario luego de la última modificación un triángulo con un color que indica:

- Verde: si se han mejorado las condiciones
- Amarillo: si no han incidido
- Rojo: si las condiciones han empeorado

Se puede observar que ninguno de los escenarios se cumple puesto que están marcados con un círculo rojo, para el escenario M1 estimamos 27 días para el cambio, pero la suma de los cambios de los módulos que afecta ArchE nos indica que es 31.22, mientras que para el escenario M2 estimamos 25 días para el cambio, pero la suma de los cambios de los módulos que afecta ArchE nos indica que es 26.4

A fin de cumplir el escenario revisemos las tácticas que nos propone ArchE, ver Figura 32

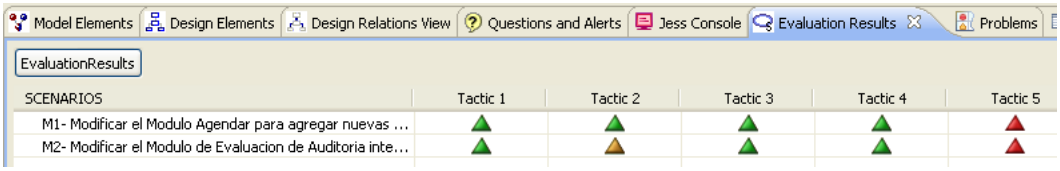


Figura 32. Tácticas propuestas por ArchE

Observamos que las tacticas 1, 3 y 4 aportan a la mejora de los escenarios M1 y M2, la tactica 2 solo mejora el escenario M1, mientra que la tactica 5 si se aplica aumentaria los costos por lo que se descarta.

En la sección de notificaciones en la viñeta “Questions and Alerts”, ver Figura 33, se indica lo siguiente:

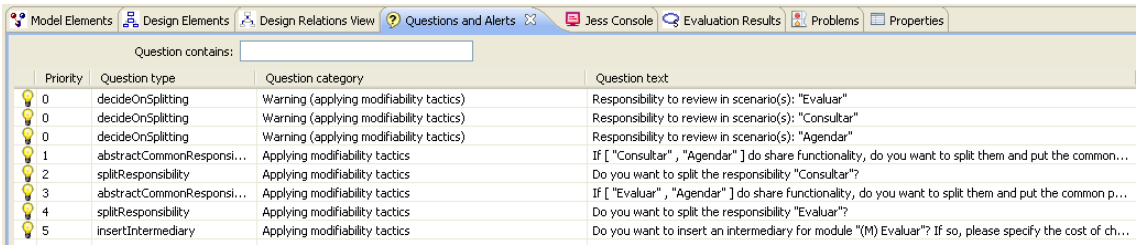


Figura 33. Tácticas propuestas por ArchE

- Alertas: nos indican sugerencias generales para el sistema, tienen prioridad 0
- Tácticas: nos indican propuestas de mejora con respecto a los módulos y su funcionalidad con el objetivo de cumplir los escenarios

La herramienta ArchE nos presenta 3 warnings y 5 tácticas:

Warnings:

- "decideOnSplitting": Responsability to review in "Evaluar"
  Nos sugiere tomar la decisión de dividir la responsabilidad de la función "Evaluar"
- "decideOnSplitting": Responsability to review in "Consultar"
  Nos sugiere tomar la decisión de dividir la responsabilidad de la función "Consultar"
- "decideOnSplitting": Responsability to review in "Agendar"
  Nos sugiere tomar la decisión de dividir la responsabilidad de la función "Agendar"

Tácticas:

- Táctica 1 (abstractCommonResponsibilities): nos indica que si "Consultar" y "Agendar" comparten funcionalidad se las puede dividir y poner sus partes comunes en un módulo separado
- Táctica 2 (splitResponsibility): nos recomienda dividir la responsabilidad "Consultar"
- Táctica 3 (abstractCommonResponsibilities): nos indica que si "Evaluar" y "Agendar" comparten funcionalidad se las puede dividir y poner sus partes comunes en un módulo separado
- Táctica 4 (splitResponsibility): nos recomienda dividir la responsabilidad "Evaluar"
- Táctica 5 (insertIntermediary): nos recomienda insertar un intermediario para el módulo "Evaluar" y que de aplicarlo se deberá especificar el costo de este cambio

Disponemos de cinco tácticas que permitirán mejorar nuestros escenarios, por lo que luego de analizar la más conveniente aplicamos la táctica 4 que permitirá cumplir con el escenario M2, ver figura 34

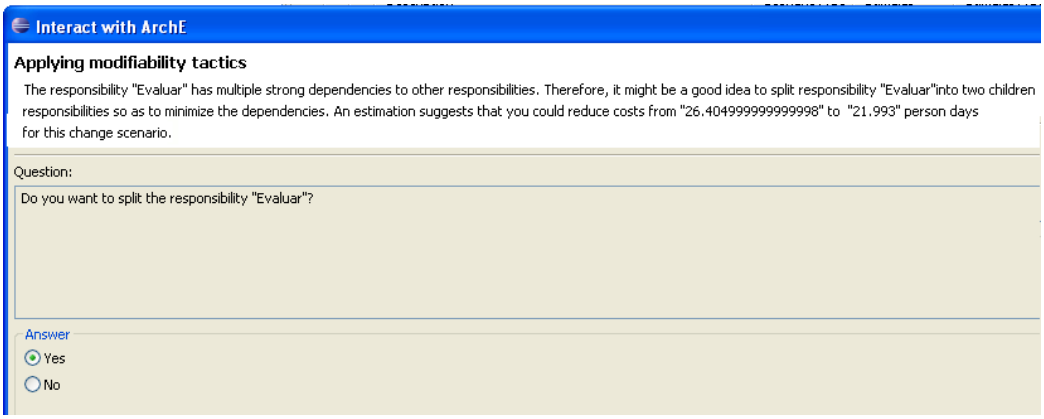


Figura 34. Aplicación de táctica 2

Adicionalmente la aplicación de la táctica 2 sugiere que podría reducir el costo actual de “26.404” a “21.993” días-persona para el cambio de este escenario.

En la figura 35 podemos observar en la Vista de dependencias que el módulo “Evaluar” ya no se encuentra y más bien se ha dividido su funcionalidad en dos nuevos módulos “Evaluar_child_21” y “Evaluar_child_22” lo que permite cumplir con el costo en días del escenario M2.

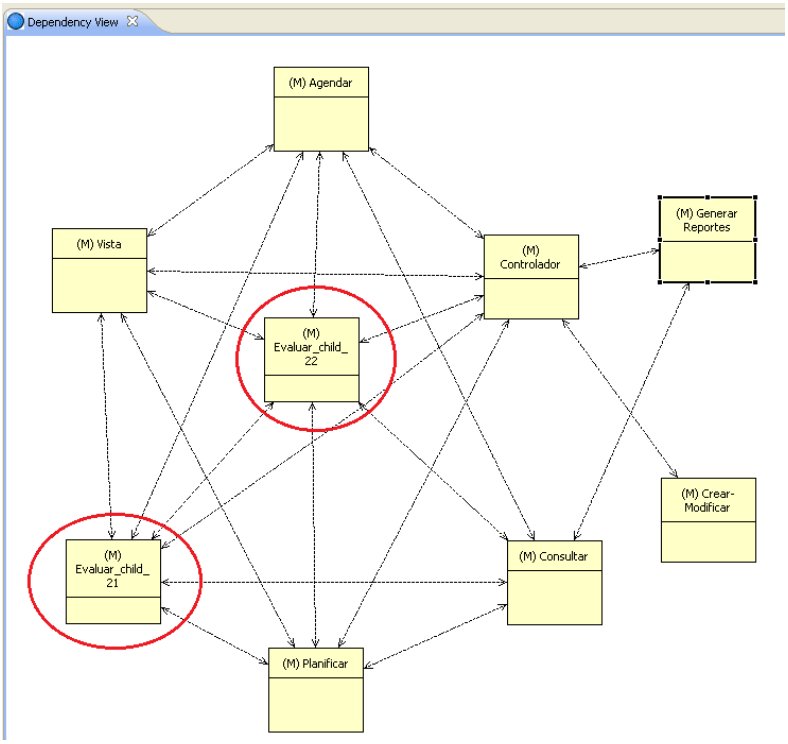


Figura 35. Vista de dependencias posterior a la aplicación de táctica 2

La aplicación de la táctica 3, ver Figura 36, también permite reducir el costo de días y por ende cumplir con el escenario M2, por lo que en un nuevo proyecto de AchE se procedió a ingresar la misma configuración y aplicamos esta táctica para observar su resultado.

.

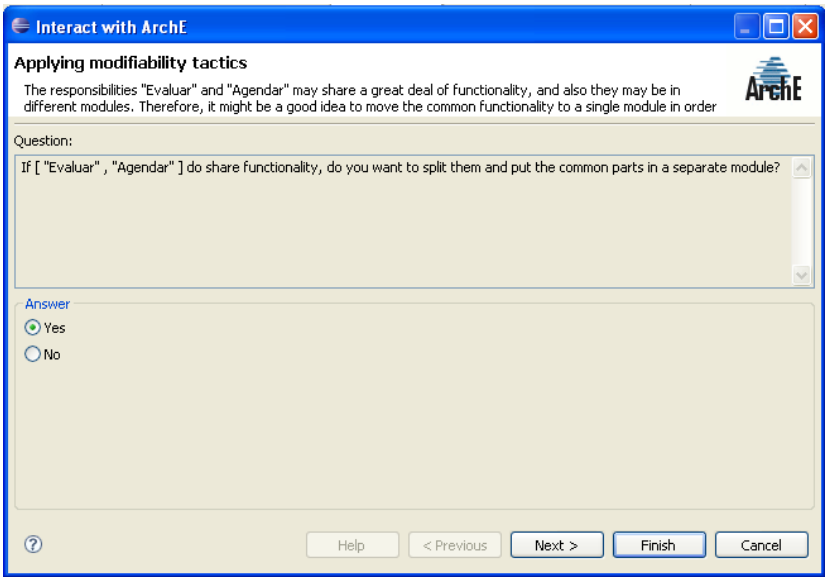


Figura 36. Vista de dependencias posterior a la aplicación de táctica 3

En la figura 37 podemos observar que se ha introducido un nuevo módulo "Evaluar/Agendar_Shared17" el cual comparte funcionalidades de los módulos "Evaluar" y "Agendar", los cuales se mantienen en la Vista de dependencia.

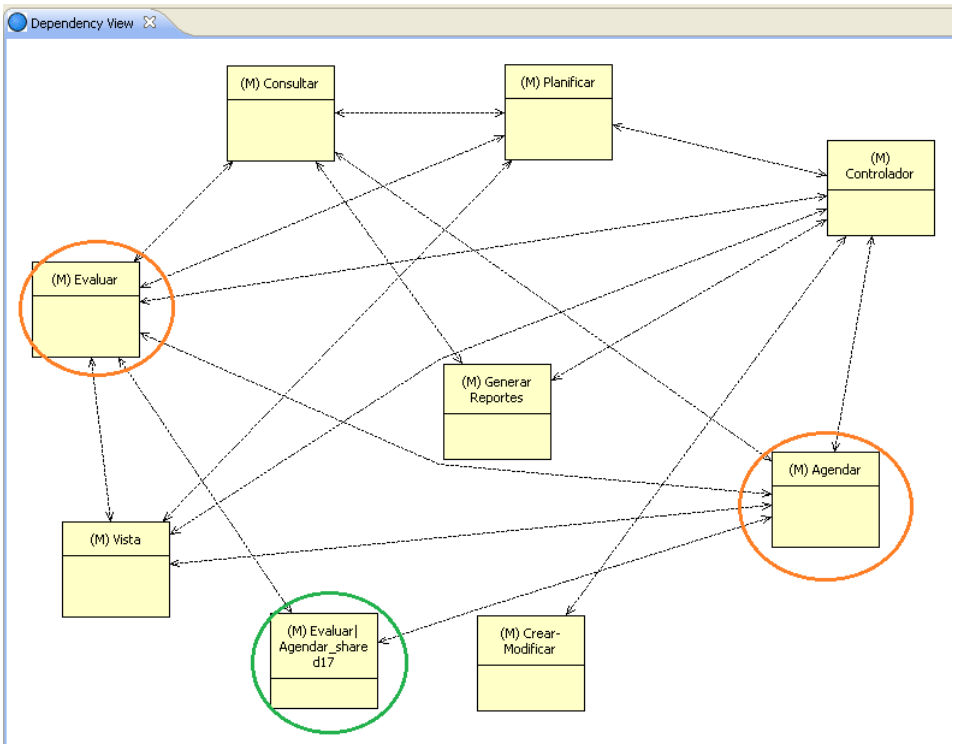


Figura 37. Vista de dependencias posterior a la aplicación de táctica 3

Regresando al proyecto de ArchE inicial, luego de aplicar la táctica 2, se vuelven a evaluar los escenarios automáticamente y procedemos a revisar como quedaron nuestros escenarios.

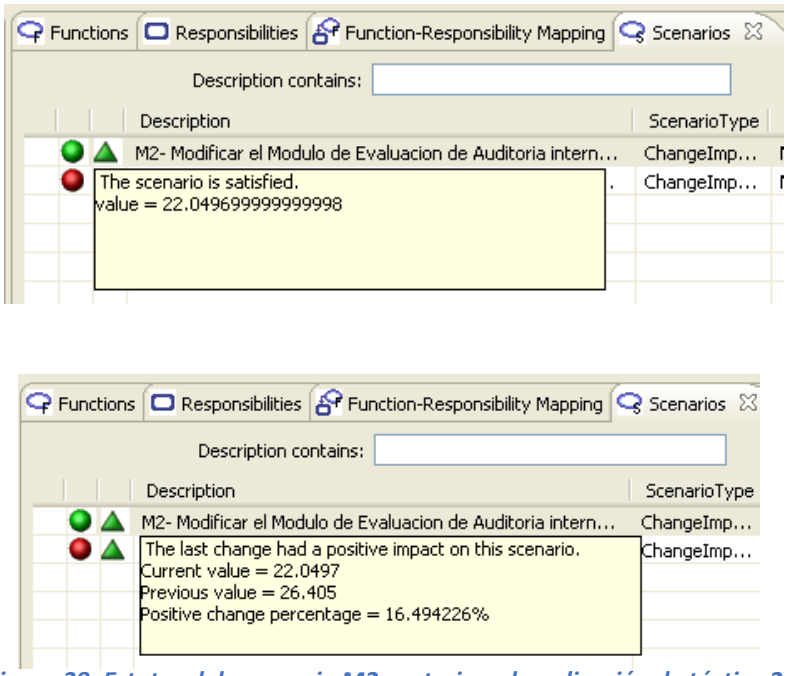


Figura 38. Estatus del escenario M2 posterior a la aplicación de táctica 2

Podemos observar en la figura 38 que el escenario M2 se cumple con un valor de 22.049 que es el esperado, sin embargo, el escenario M1 aún no se cumple como se muestra en la Figura 39.

.

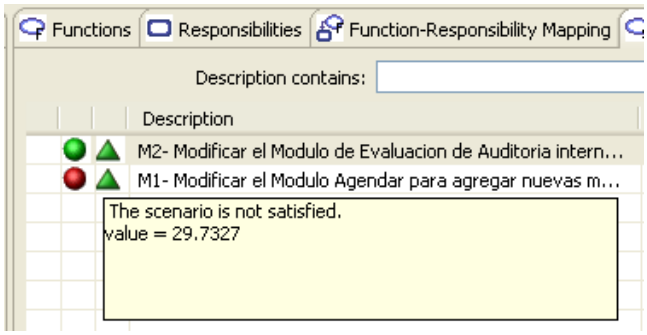


Figura 39. Estatus del escenario M1 posterior a la aplicación de táctica 2

Nuevamente observamos las tácticas que nos recomienda ArchE, ver Figura 40.

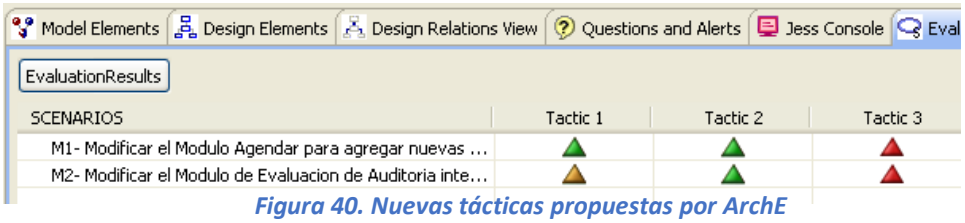


Figura 40. Nuevas tácticas propuestas por ArchE

Observamos que solo la táctica 2 es completamente favorable para ambos escenarios, más aún considerando que ya tenemos satisfecho el escenario M2. Esta táctica favorece al escenario M1 por lo que podría ser una buena opción, mientras que la táctica 3 no favorece a ninguno de los escenarios.

En este punto es necesario realizar una valoración exhaustiva de cada una de las tácticas propuestas a fin de determinar cuál es más recomendable aplicar, para esto revisamos las indicaciones de la viñeta “Questions and Alerts”, ver Figura 38.

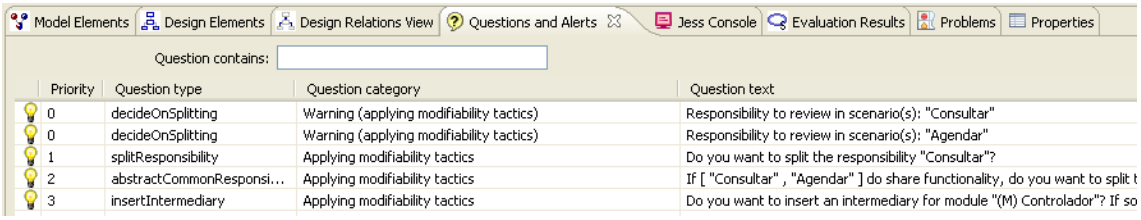


Figura 41. Alertas y Preguntas recomendadas por ArchE

Analisis de Tactica 1

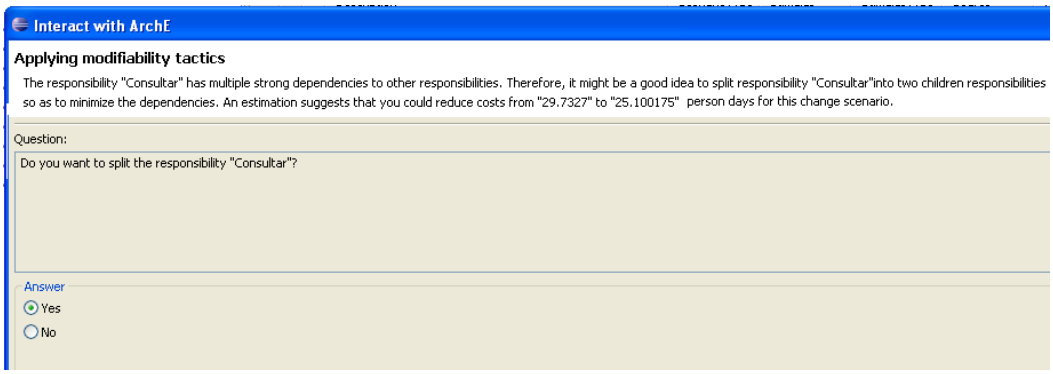


Figura 42. Recomendación de Táctica 1

En la táctica 1, ver la Figura 42, se nos plantea que la responsabilidad “Consultar” tiene múltiples y fuertes dependencias con otras responsabilidades

y que podría ser muy conveniente dividir la responsabilidad “Consultar” en dos responsabilidades hijas para de esta forma minimizar las dependencias.

La táctica 1 nos pregunta ¿Queremos dividir la responsabilidad “Consultar”?, para determinar su conveniencia debemos revisar como se afecta el costo del cambio al aplicarla, para esto veamos en la viñeta “Evaluation Results” la predicción del costo del cambio.

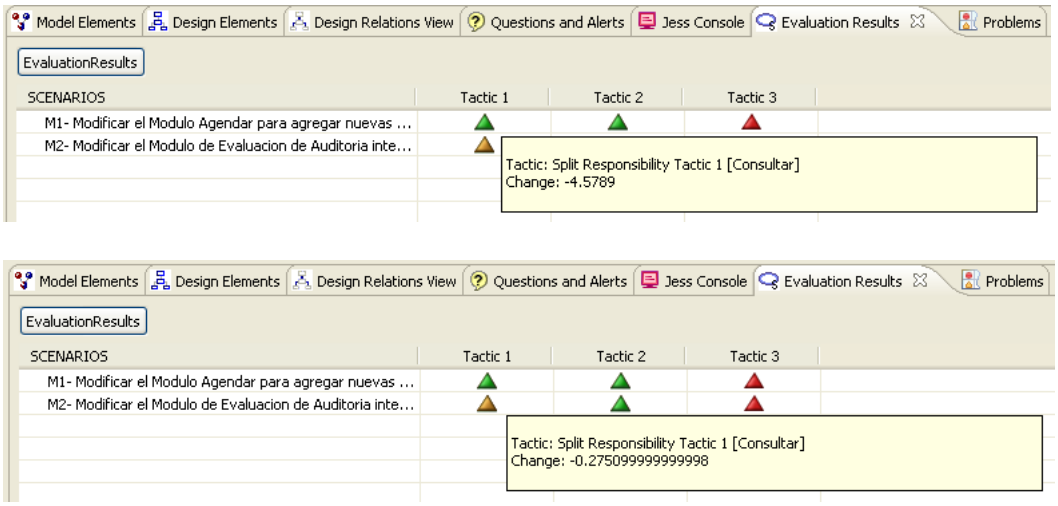


Figura 43. Costo del cambio con táctica 1

Como podemos observar en la Figura 43 la aplicación de la táctica 1 reduce en 4.5789 el costo en el escenario M1, mientras que en el escenario M2 el costo se reduce en 0.275.

Analisis de Tactica 2

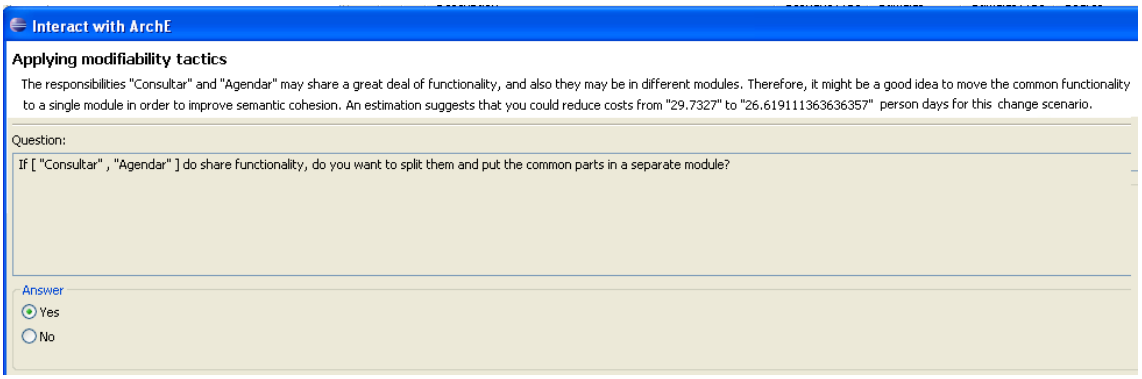


Figura 44. Recomendación de Táctica 2

En la táctica 2, ver la Figura 44, se nos plantea que las responsabilidades “Consultar” y “Agendar” pueden compartir varias funcionalidades y que también

podrían estar en diferentes módulos, por lo que podría ser muy conveniente mover esta funcionalidad que tienen en común a un nuevo módulo para de esta forma mejorar la cohesión semántica.

La táctica 2 nos pregunta que, ¿Si "Consultar" y "Agendar" comparten funcionalidad, quisiéramos dividirlas y poner sus partes comunes en un módulo separado? Para determinar su conveniencia debemos revisar como se afecta el costo del cambio al aplicarla, para esto veamos en la viñeta "Evaluation Results" la predicción del costo del cambio.

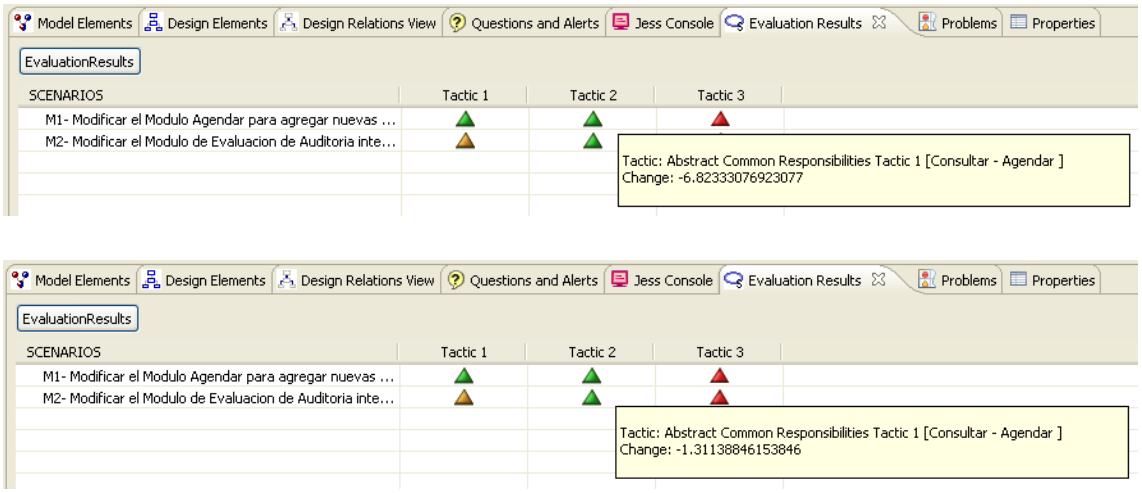


Figura 45. Costo del cambio con táctica 2

Como podemos observar en la Figura 45 la aplicación de la táctica 2 reduce en 6.823 el costo en el escenario M1, mientras que en el escenario M2 el costo se reduce en 1.311.

Analisis de Tactica 3

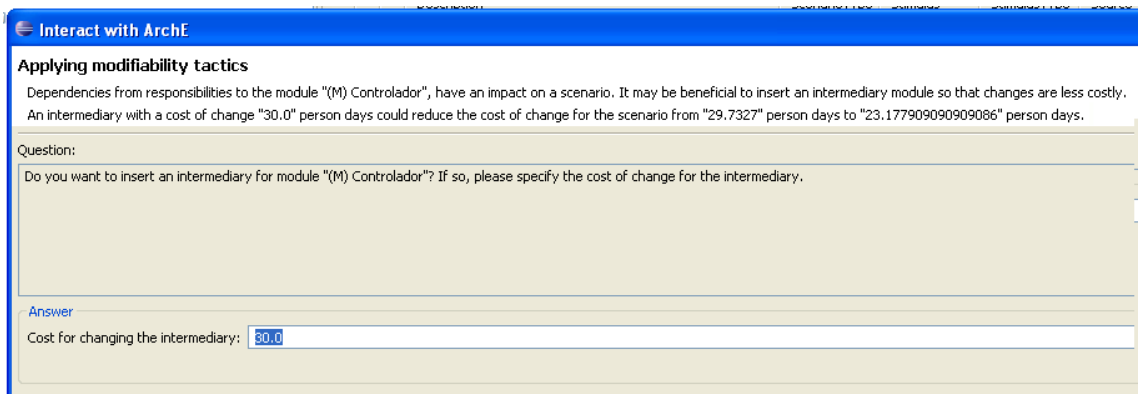


Figura 46. Recomendación de Táctica 3

En la táctica 3, ver la Figura 46, se nos plantea que las dependencias de responsabilidades hacia el módulo “Controlador” tienen un impacto sobre un escenario (en este caso a ambos) y que podría ser beneficioso insertar un módulo intermediario de tal forma que los cambios sean menos costosos.

La táctica nos pregunta ¿Queremos insertar un intermediario para el módulo “Controlador”? de ser afirmativo debemos especificar el costo del cambio para este módulo intermediario, el costo sugerido es de 30.0, para determinar su conveniencia debemos revisar como se afecta el costo del cambio al aplicarla, para esto veamos en la viñeta “Evaluation Results” la predicción del costo del cambio.

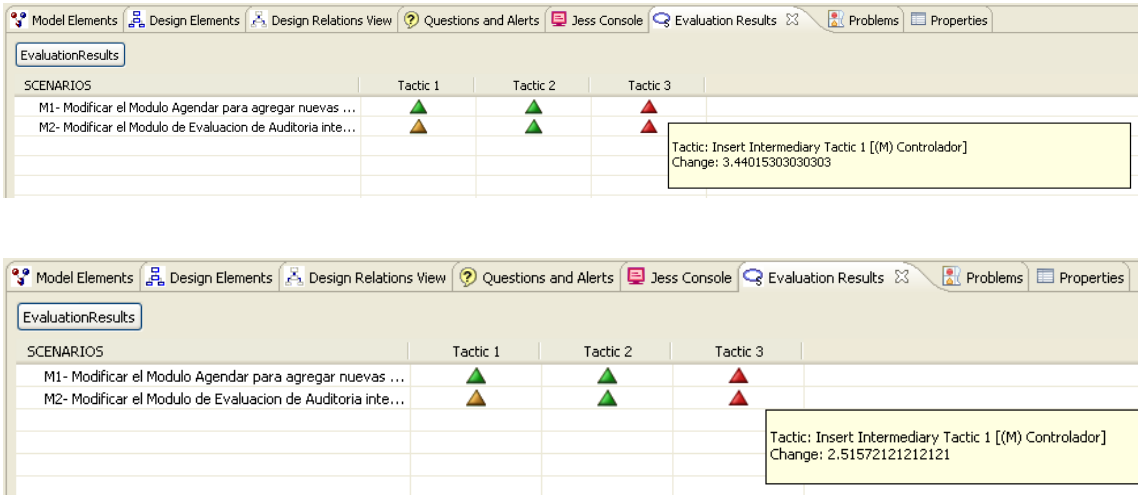


Figura 47. Costo del cambio con táctica 4

Como podemos observar en la Figura 47 la aplicación de la táctica 3 incrementa en 3.4401 el costo en el escenario M1, mientras que en el escenario M2 el costo se incrementa en 2.515.

En base al análisis realizado con las 3 tácticas propuestas por ArchE seleccionamos la táctica 1 puesto que tiene un impacto más favorable al escenario M1, este se reduce en 4.5789 mientras que el costo del cambio sobre el escenario M2 es mínimo, se reduce en 0.275, además de ser la única táctica que nos permite dividir el módulo “Consultar” que es un cambio en la misma línea que se tomó para satisfacer el escenario “M2”.

Procedemos a aplicar la táctica 2, se ejecutan de forma automática los marcos de razonamiento y se obtienen los siguientes resultados:

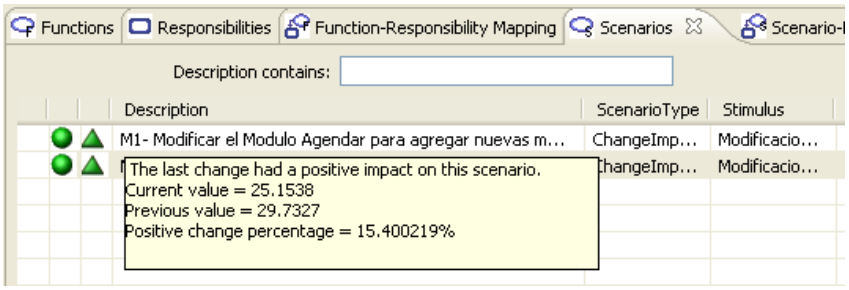


Figura 48. Escenario M1 cumplido

En la Figura 48, vemos que el escenario M1 se ha cumplido y el costo del cambio se estableció en 25.1538

Como se esperaba el costo del cambio en el escenario M2 se mejoró en un mínimo y se estableció en 21.774, ver Figura 49.

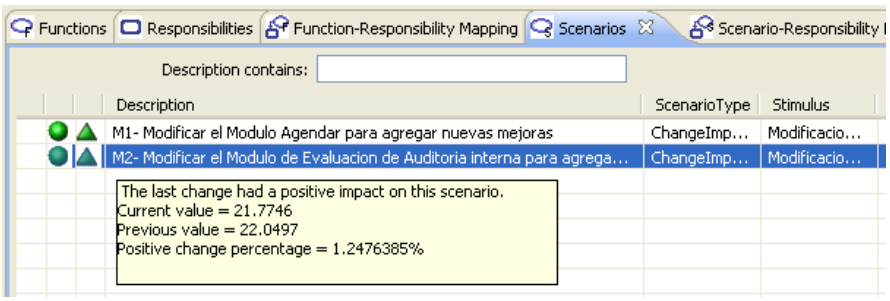


Figura 49. Escenario M2 con mínima mejora de costo de cambio

En la Figura 50 observamos que en la Vista de dependencia se ha aplicado lo sugerido en la táctica, se dividió en dos el módulo “Consultar” compartiendo sus responsabilidades.

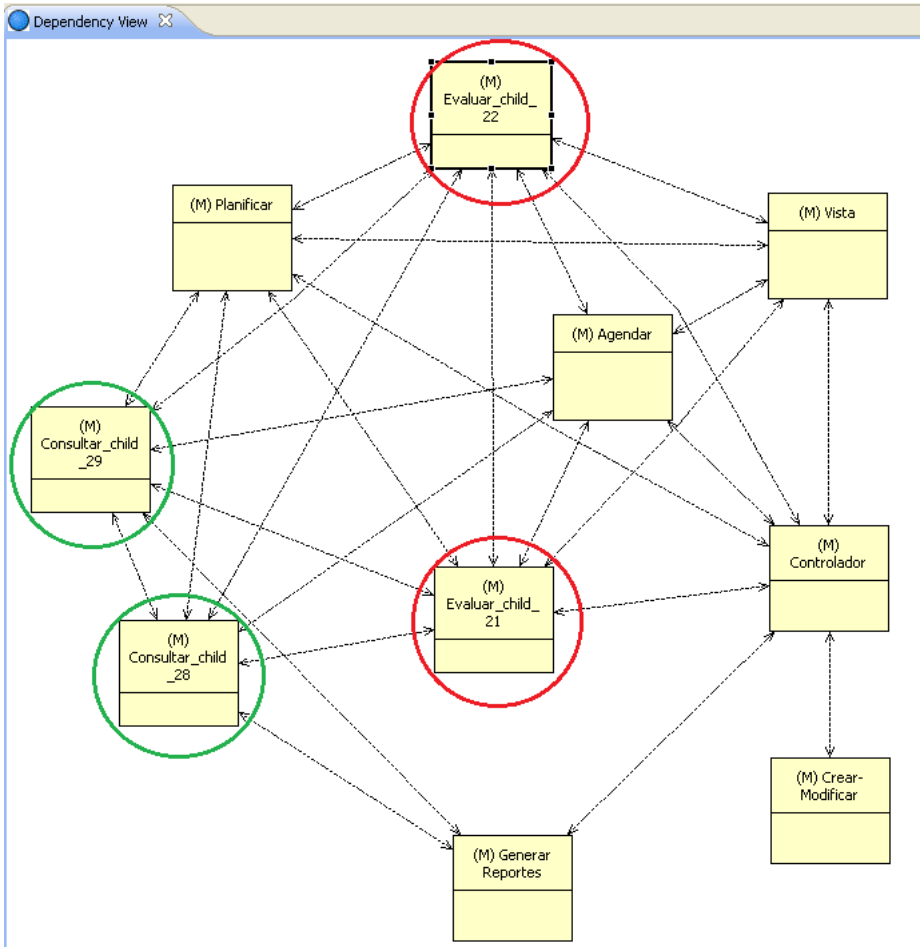


Figura 50. Vista de dependencias con escenarios M1 y M2 satisfechos

En la Figura 51 podemos observar el mapeo final de las responsabilidades con respecto a cada uno de los escenarios.

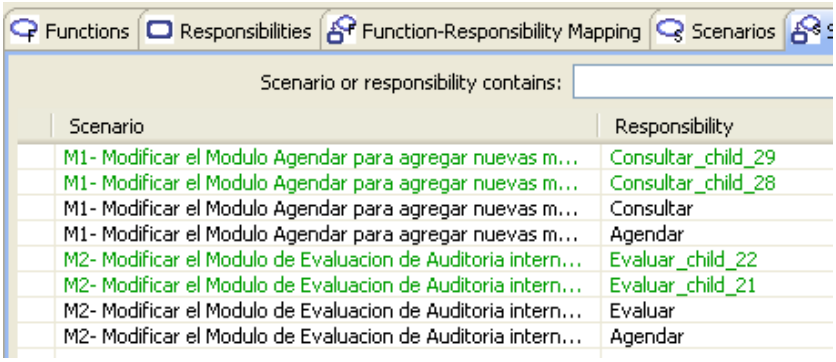


Figura 51. Mapeo de Responsabilidades para los dos escenarios

6.4 Valoración de los resultados

Con los resultados obtenidos procederemos a realizar una comparativa entre los costos iniciales de los escenarios antes de aplicar las tácticas propuestas por ArchE para cumplir con el requisito de calidad de Modificabilidad y los que se obtuvieron luego de aplicar dichas tácticas.

Los atributos de costo antes de aplicar las tácticas eran, ver Figura 52:

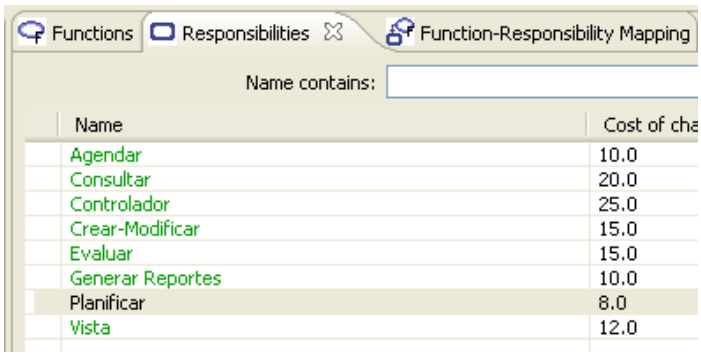


Figura 52. Costo del cambio inicial para cada responsabilidad

Luego de aplicar la táctica propuesta por ArchE esta dividió el módulo “Evaluar” en dos responsabilidades diferentes con las características compartidas del módulo “Evaluar” que permitió satisfacer el escenario M2. Se obtuvieron los siguientes costos, ver Figura 53.

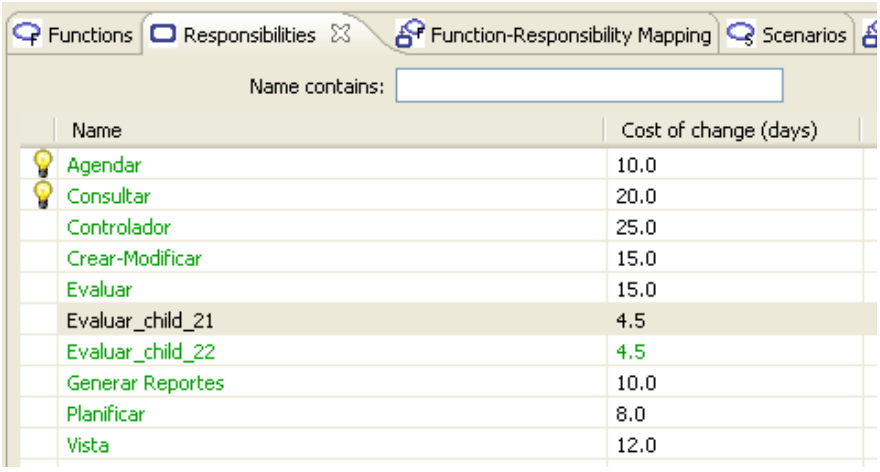


Figura 53. Costo del cambio de cada responsabilidad con el escenario M2 satisfecho

Observamos que la responsabilidad “Evaluar” que inicialmente tenía un costo de 15.0 ahora se comparte, pero también se optimiza a 4.5 para

“Evaluar_child_21” y 4.5 para “Evaluar_child_22”, la reducción del costo es de 6.0

Para satisfacer el escenario M2 teníamos la alternativa de aplicar otra táctica que tomaba parte de las responsabilidades del módulo “Evaluar” y “Agendar” y las asignaba a un nuevo módulo “Evaluar/Agendar_Shared17”. Se obtuvieron los siguientes costos, ver Figura 54.

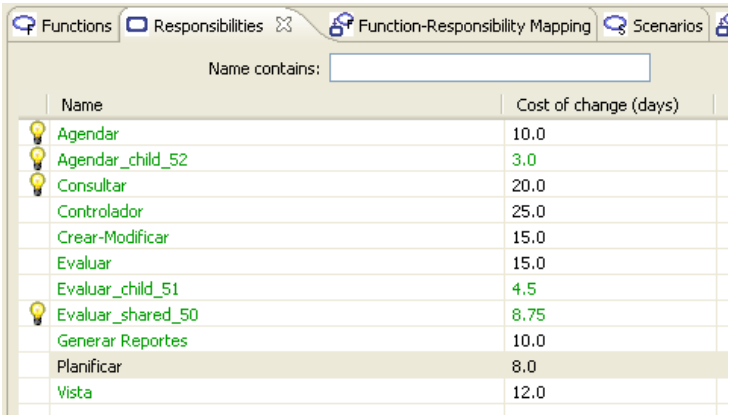
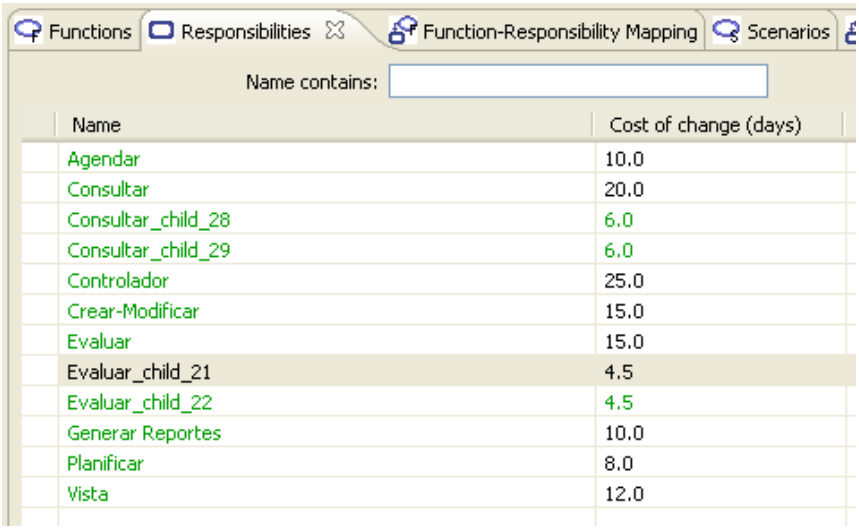


Figura 54. Costo del cambio de cada responsabilidad con el escenario M1 satisfecho

Observamos que de la responsabilidad “Evaluar” se generan dos nuevas responsabilidades “Evaluar_child_51” con un costo de 4.5 y “Evaluar_shared_50” con un costo de 8.75 y de la responsabilidad “Agendar” se generó una nueva responsabilidad “Agendar_child_52” con un costo de 3.0, estos se cohesionan en un nuevo módulo “Evaluar|Agendar_shared17, con un costo de cambio total de 16.25. Esta táctica no se aplicó.

Finalmente, luego de que el marco de razonamiento reformula nuevas tácticas, aplicamos una que divide el módulo “Consultar”, en dos responsabilidades diferentes que permiten satisfacer el escenario M1. Se obtuvieron los siguientes costos, ver Figura 55.



Name	Cost of change (days)
Agendar	10.0
Consultar	20.0
Consultar_child_28	6.0
Consultar_child_29	6.0
Controlador	25.0
Crear-Modificar	15.0
Evaluar	15.0
Evaluar_child_21	4.5
Evaluar_child_22	4.5
Generar Reportes	10.0
Planificar	8.0
Vista	12.0

Figura 55. Costo del cambio de cada responsabilidad con escenarios M1 y M2 satisfechos

Observamos que la responsabilidad “Consultar” que inicialmente tenía un costo de 20.0 ahora se comparte, pero también se optimiza a 6.0 para “Consultar_child_28” y 6.0 para “Consultar_child_29”, la reducción del costo es de 8.0

Con base en la Figura 19, los nuevos módulos tendrán las siguientes funciones repartidas:
Evaluar_child_21 y Evaluar_child_22:

- Agendar
- Consultar

Consultar_child_28:

- Consultar Estación
- Consultar Proceso
- Consultar KPI
- Consultar Criterio de Evaluación

Consultar_child_29:

- Consultar Plantilla de Checklist
- Consultar Agenda
- Consultar Checklist Evaluado

Con estas nuevas responsabilidades y módulos derivados de las tácticas aplicadas obtendremos las mejoras en el diseño para que se cumplan los escenarios de calidad propuestos.

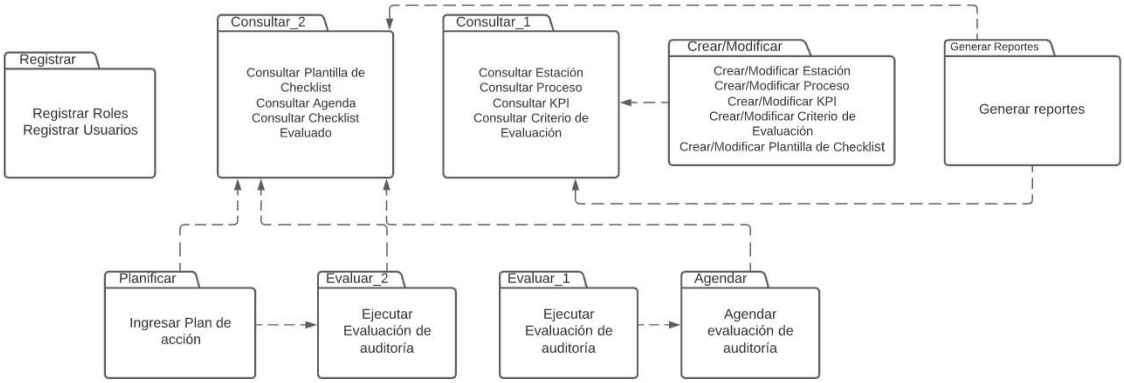


Figura 56. Diagrama de Paquetes del SAIPP-ES luego de aplicados los cambios

En la Figura 56 se muestra cómo se reorganizan los paquetes que dan soporte a la Arquitectura propuesta, luego de aplicar las tácticas y satisfacer los escenarios propuestos.

7. CAPITULO 7 - Conclusiones y Trabajos futuros

7.1 Conclusiones

En el presente trabajo de investigación se ha cumplido con los objetivos planteados, esto es, presentar los módulos y el diseño básico de la arquitectura del Sistema de Auditoria Interna para Procesos y Procedimientos a Estaciones de Servicio (SAIPP-ES) y los resultados de las mejoras al mismo con respecto al atributo de Modificabilidad, utilizando la herramienta ArchE.

Con relación a los diversos estilos de arquitectura se pudo observar el alto grado de relevancia que tiene el arquitecto de software para determinar el estilo que más se adapte no solo a los requerimientos funcionales sino a las características o atributos de calidad que son necesarios que se cumplan para que un sistema pueda tener un largo ciclo de vida, cumpliendo con las características inicialmente planteadas y adaptándose con ciertos cambios en el diseño a los nuevos escenarios que el ámbito tecnológico impone, todo esto de una forma medible y fácilmente adaptable.

La arquitectura definida para el sistema evaluado está basada en el Modelo-Vista-Controlador, la misma que se diseñó utilizando un proceso de identificación de requisitos funcionales y no funcionales, dando como resultado una arquitectura versátil, altamente modificable y escalable.

El tipo de arquitectura definida facilitó el uso de una herramienta como ArchE, la misma que permitió realizar un análisis adecuado del cual se obtuvieron tácticas que fueron aplicadas para mejorar los escenarios inicialmente no cumplidos, más aún que ArchE desconoce completamente la funcionalidad y semántica de la arquitectura en cuestión y más bien el rol del arquitecto toma relevancia al interpretar dichas tácticas y su aplicabilidad.

Se evaluaron los resultados obtenidos siendo favorables en términos de una reducción en el costo de los cambios al repartir las funcionalidades de los módulos “Consultar” y “Evaluar”, mejorando de forma sustancial el atributo de modificabilidad del diseño arquitectónico del sistema SAIPP-ES, hemos pasado de un costo total de modificación para el módulo “Evaluar” de 15 días a subdividirlo en dos cuyos costos son 4.5 respectivamente. Dependiendo de que módulos necesitemos modificar se obtuvo una mejora del 70% menos del

tiempo de modificación, mientras que para el módulo “Evaluar” de 20 días se pasó a subdividirlo en dos cuyos costos son 6.0 respectivamente. Dependiendo de que módulos necesitemos modificar se obtuvo también una mejora del 70% menos del tiempo de modificación.

Los escenarios propuestos en la herramienta ArchE deben parametrizarse con restricciones no tan exigentes y el parámetro Response Measure debe tener la suficiente holgura en días a fin de que puedan ser satisfechos los escenarios planteados al aplicar las tácticas propuestas puesto que, el marco de razonamiento hace los cálculos automáticos en base a los costos de cada responsabilidad.

Tomando la recomendación de [20] debemos tener muy presente que los cambios que realiza el usuario en las dependencias y parámetros de ArchE, para que estos permanezcan en otras sesiones, es necesario grabarlos en la base de datos FactBase, tal y como ya se ha explicado anteriormente, por lo que, si no se hace a través del menú correspondiente, se puede perder bastante trabajo hecho si se cierra la herramienta antes de hacerlos persistentes.

Así también, cuando se ejecuta la opción de hacer persistentes los cambios con Persist FactBase, si se hubiera ejecutado un marco y hubiera tácticas propuestas, todos estos análisis se anulan, y vuelve a ejecutarse cada marco para proponer nuevas tácticas con los cambios realizados.

7.2 Trabajos futuros

La investigación en el ámbito de la arquitectura de software está en constante evolución, por lo que el presente trabajo de investigación puede ser utilizado como un punto de apoyo para proponer el análisis de diferentes tipos de sistemas que permitan aplicar los diversos estilos arquitectónicos vigentes y de esta forma proponer diseños novedosos y que pueden ser utilizados de modelos para su aplicación práctica y ágil a diferentes dominios.

Con respecto a la herramienta ArchE no es mucho lo que puede realizarse a futuro puesto que existen apenas dos marcos de razonamiento disponibles y

las actualizaciones a esta herramienta ya han dejado de realizarse desde hace más de 10 años.

8. CAPITULO 9 – Bibliografía

[1] ISO 900:2015, Sistemas de Gestión de Calidad – Fundamentos y Vocabulario, 2015, https://www.iso.org/obp/ui/es/#iso:std:iso:9000:ed-4:v1:es:sec:A

[2] ISO 19011:2018, Directrices para la auditoria de los Sistemas de Gestión, 2018, https://www.iso.org/obp/ui#iso:std:iso:19011:ed-3:v1:es

[3] Maria Isabel Isaza Posada, 2012, Guía Metodológica en la realización de auditorías a proveedores de insumos críticos para el servicio de alimentación, Trabajo de Grado, Universidad de Medellín, Colombia.

[4] Preparación de un Checklist de Auditoría Interna, 2016, https://www.escuelaeuropeaexcelencia.com/2016/07/preparar-una-check-list-auditoria-interna/

[5] Mark Richards; Neal Ford. Fundamentals of Software Architecture. O´Reily Media, Inc. 2020

[6] Mark Richards; Neal Ford. Software Architecture Fundamentals – Architectural Thinking. O´Reily Media, Inc. January 2020

[7] Mark Richards; Neal Ford. Software Architecture Fundamentals – Architectural Styles. O´Reily Media, Inc. January 2020

[8] Paul Clements; Rick Hazman; Len Bass. Software Architecture in Practice. Addison Wesley Professional, First Edition. 2021

[9] Peter Spath; Beginning Java MVC 1.0: Model View Controller Development to Build Web, Cloud, and Microservices Applications; Apress, Third Edition. 2012

[10] Bachmann, Felix; Bass, Len; Klein, Mark; Preliminary Design of ArchE: A Software Architecture Design Assistant. September 2003. Technical Report. CMU/SEI-2003-TR.

[11] Bachmann, Felix; Bass, Len; Bianco, Philip; Klein, Mark. Using ArchE in the Classroom: One Experience. September 2007. Technical Note. CMU/SEI-2007-TN-001.

[12] Bachmann, Felix; Bass, Len; Klein, Mark. Illuminating the Fundamental Contributors to Software Architecture Quality. August 2002. Technical Report. CMU/SEI-2002-TR-025. ESC-TR- 2002-025.

[13] Bachmann, Felix; Klein, Mark. Methodical Design of Software Architecture Using an Architecture Design Assistant (ArchE). 2005 by Software Engineering Institute - Carnegie Mellon University. Pittsburgh.

[14] Bachmann, Felix; Bass, Len; Bianco, Phil. Software Architecture Design with ArchE. Marzo de 2007. Software Engineering Institute. Carnegie Mellon University. Pittsburgh.

[15] Bass, Len. ArchE – An Architecture Design Assistant. August 2, 2007. Software Engineering Institute. Carnegie Mellon University. Pittsburgh.

[16] SEI, Software Engineering Insitute. Architecture Expert (ArchE) Tool., 2007 http://www.sei.cmu.edu/architecture/arche.html

[17] Shaw, M., Garlan, D. Software Architecture: Perspectives on an Emerging Discipline, 1996, Prentice Hall.

[18] Juan Antonio Perez-Campanero Atanasio, septiembre 2010, Desarrollo de Software y de las Arquitecturas de Software, Maestría, UNED, España.

[19] Jesús Martin Fernández, septiembre 2010, Arquitecturas Software: Gestión de los atributos de calidad de un sistema y su incorporación en ArchE para la mejora del análisis, evaluación y soporte en la toma de decisiones durante el desarrollo arquitectónico, Maestría, UNED, España.

[20] Jose Antonio Miranda Rodera, junio 2015, ArchE: Evaluación de Arquitecturas y Toma de decisiones. Aplicación al análisis de arquitecturas en el sector Aeronáutico, Maestría, UNED, España.

[21] Rafael Elías Cárdenas Rodriguez, septiembre 2014, ARCHE, Arquitecturas y toma de decisiones, Maestría, UNED, España.

[22] Pablo Garcia Diaz, septiembre 2013, ArchE: Evaluación de Arquitecturas y Toma de decisiones, Maestría, UNED, España.