UNIVERSIDAD NACIONAL DE CHIMBORAZO

FACULTAD DE INGENIERÍA

CARRERA DE INGENIERÍA EN SISTEMAS Y COMPUTACIÓN

"Trabajo de grado previo a la obtención del Título de Ingeniero en Sistemas y Computación"

TRABAJO DE GRADUACIÓN

COMPARACIÓN DE LA ARQUITECTURA N - CAPAS VS. ARQUITECTURA N – CAPAS ORIENTADA AL DOMINIO CON .NET, APLICADO EN EL SISTEMA PLAN DE FORTALECIMIENTO Y MEJORAS DE ACREDITACIÓN DE LA UNACH

Autores:

Eddy Navarrete U.

Diego Ortiz R.

Director: Ing. Gonzalo Allauca

Riobamba - Ecuador

2017

Los miembros del Tribunal de Graduación del proyecto de investigación de título: **"Comparación de la Arquitectura N - Capas vs. Arquitectura N – Capas orientada al dominio con .NET, aplicado en el Sistema Plan de Fortalecimiento y Mejoras de Acreditación de la UNACH"**, presentado por los: Sr. Navarrete Uquillas Eddy Mauricio y Sr. Ortiz Ruiz Diego Omar, y dirigida por: Ing. Gonzalo Allauca.

Una vez escuchada la defensa oral y revisado el informe final del proyecto de investigación con fines de graduación escrito en la cual se ha constatado el cumplimiento de las observaciones realizadas, remite la presente para uso y custodia en la biblioteca de la Facultad de Ingeniería de la UNACH.

Para constancia de lo expuesto firman:

Ing. Danny Velasco
Presidente del Tribunal

Ing. Gonzalo Allauca
Miembro del Tribunal

Ing. Fernando Molina
Miembro del Tribunal

AUTORÍA DE LA INVESTIGACIÓN

"La responsabilidad del contenido de este Proyecto de Graduación, corresponde exclusivamente a: Eddy Mauricio Navarrete Uquillas y Diego Omar Ortiz Ruiz, autores del proyecto de investigación, al Ing. Gonzalo Allauca, Director de Tesis; y el patrimonio intelectual de la misma a la Universidad Nacional de Chimborazo.

Eddy M. Navarrete U.

060384169-3

Diego O. Ortiz R.

060442511-6

AGRADECIMIENTO

En el presente trabajo de investigación queremos agradecer a Dios por brindarnos la salud, la vida y por permitirnos hacer realidad nuestro anhelado sueño.

Expresamos nuestro agradecimiento a la **Universidad Nacional de Chimborazo**, la cual nos abrió sus puertas para culminar con éxito una etapa más de nuestras vidas, preparándonos para un futuro competitivo y poder servir a la sociedad con nuestros sólidos conocimientos para el progreso del país.

Un reconocimiento especial a nuestro Director de Proyecto de Investigación, Ing. Gonzalo Allauca, por su calidad humana y todo el apoyo brindado al instruirnos y guiarnos a realizar el presente trabajo investigativo. Al Ing. Fernando Molina por su paciencia y ayuda incondicional y en especial a los Ingenieros Danny Velasco, Henry Paca y Magaly Pérez por la ayuda incondicional para el desarrollo de la misma.

Son muchas las personas que han formado parte de nuestra vida a las que nos encantaría agradecerles su amistad, consejos, apoyo, ánimo y compañía en los momentos más difíciles. Algunas están aquí con nosotros y otras en nuestros recuerdos y en nuestro corazón, sin importar en donde estén queremos darles las gracias por formar parte de nosotros, por todo lo que nos han brindado y por todas sus bendiciones.

Para ellos muchas gracias y que Dios les bendiga

Eddy y Diego

DEDICATORIA

El presente trabajo de investigación va dedicado con mucho cariño y aprecio a mis padres, hermanos y a todos quienes aportaron positivamente a lo largo de mi formación académica, dándome el apoyo e incentivación que necesito para trabajar día a día, ya que son los testigos del trabajo perseverante para lograr un nuevo éxito en mi vida profesional. Por eso y por mucho más le dedico este proceso de formación que constituirá el cimiento fundamental de mi vida profesional y a través de la cual forjare un nuevo presente.

Diego Ortiz

DEDICATORIA

A mi madre Norita.

Por haberme apoyado en todo momento, por sus consejos, sus valores, por la motivación constante que me ha permitido ser una persona de bien, pero más que nada, por su amor.

A mi hermano Ricardo.

Por los ejemplos de perseverancia y constancia que lo caracteriza y que me ha infundado siempre, por el valor mostrado para salir adelante y por su amor.

A mi familia en general, porque me han brindado su apoyo incondicional y por compartir conmigo buenos y malos momentos.

A mis dos amigos.

Que nos apoyamos mutuamente en nuestra formación profesional y que hasta ahora, seguimos siendo amigos: Paúl Reino y a Pedro Larrea por haberme ayudado en el trayecto de mi formación profesional.

ÍNDICE GENERAL

ÍNDICE DE GRÁFICOS

ÍNDICE DE TABLAS

RESUMEN

La presente investigación tiene como objetivo principal la caracterización del estudio comparativo entre las arquitecturas N – Capas y N – capas Orientada al Dominio con .Net, con respecto a la Seguridad y Rendimiento para el desarrollo del Plan de Fortalecimiento y Mejoras de Acreditación de la UNACH.

Esta implementación se evaluó de acuerdo a los Índices de Rendimiento que son: Media de tiempo invertido por una petición, Mediana de tiempo invertido por una petición: significa que el 50% de las muestras tardaron menos del valor reflejado, Tanto por ciento de respuestas con error, El rendimiento en Kb/segundo: igual que la anterior pero con cantidad de datos en lugar de peticiones y al Índice de Seguridad que es Injection SQL, obteniendo como resultado que el Rendimiento en la Arquitectura N – Capas Orientada al Dominio con .Net posee un 49.75% en comparación con la Arquitectura N – Capas que posee un valor de 35.1 %, por lo tanto, el Rendimiento en la Arquitectura N – Capas Orientada al Dominio con .Net es más alto que el de la Arquitectura N – Capas, por lo que el sistema en condiciones particulares de trabajo responde de manera adecuada y eficaz. En cuanto a la seguridad tanto la Arquitecturas N – Capas y la Arquitectura N – Capas Orientada al dominio con .Net poseen un porcentaje del 100% lo que quiere decir que no son vulnerables ante posibles ataques.

Para el desarrollo del aplicativo se utilizó SQL Server 2012 como el gestor de Base de Datos, Visual Studio 2013 como el IDE de desarrollo, Internet Information Services (IIS) como servidor Web, Telerik UI y cristal report para la creación de reportes.

Se concluye que al desarrollar el Sistema de Plan de Mejoras y Acreditación de la UNACH permitirá la automatización de la gestión y contribuirá a mejorar los procesos del Plan de Fortalecimiento y Mejoras para el Departamento de Evaluación y Acreditación de la UNACH.

Abstract

The present research has as main objective the characterization of the comparative study between the N - Layered and N - layered architectures oriented with the domain .Net, regarding to the Security and Performance for the development of the Strengthening Plan and Improvements of Accreditation of UNACH. This implementation was evaluated according to some performance indices such as: average response time per customer request. Average response time per customer request means that 50% of the samples took less than the reflected value, percentage of error, the performance in Kb / second: same as the previous one but with quantity of data instead of requests and to the Security Index that is Injection SQL, obtaining as a result that the performance in the domain - oriented N - layered architecture with .Net showed a 49.75% compared to the N - layered architecture whish showed a value of 35.1%. Therefore, the performance in the domain - oriented N - layered architecture with .Net is higher than the architecture N – layers. Consequently, it is shown the system working in particular conditions responds appropriately and effectively. As for security, both systems showed a percentage of 100% which means they are not vulnerable to possible attacks. During the application SQL Server 2012 was used as the database manager, Visual Studio 2013 as the IDE of development, Internet Information Services (IIS) as Web server and Telerik UI and Crystal Report for report creation. It is concluded that developing the Improvement and Accreditation System Plan of the UNACH will allow the automation of the management and will contribute to improve the processes of the Plan of Strengthening and Improvements for the Department of Evaluation and Accreditation of the UNACH.

Reviewed by: Escudero, Isabel
LANGUAGE CENTER TEACHER

INTRODUCCIÓN

Hoy en día, la Arquitectura de Software permite la comunicación entre todas las partes interesadas en el desarrollo de un sistema basado en un computador. Es así que la Arquitectura constituye un modelo relativamente pequeño y asequible por la vía intelectual sobre cómo está estructurado el sistema y la forma en la que sus componentes interactúan entre sí. (Bass, Clements, & Kazman, 2003)

En la actualidad los sistemas desarrollados en el departamento de Evaluación y Acreditación de la Universidad Nacional de Chimborazo están orientados a soportar los procesos de acreditación institucional y de carreras, basados en los Modelos de Evaluación, emitidos permanentemente por el CEAACES[1], estos plantean el cumplimiento de Criterios, Subcriterios e Indicadores que deben ser gestionados internamente de manera que las evidencias del cumplimiento estén siempre actualizados y se pueda observar la calidad. Como consecuencia de las evaluaciones por parte del CEAACES, específicamente de las carreras, pueden acreditarse o requerir un Plan de Fortalecimiento y Mejoras[2] para lograr la acreditación. Esta situación ha desembocado en la creación de diversos sistemas que permitan dar respuesta a estos requerimientos, pero, es evidente que los tiempos de desarrollo de estos sistemas son demasiado extensos y consuman muchos recursos – o no se aprovechen adecuadamente-, y tengan dificultades al momento de escalar o integrarlos a otros aplicativos. Esta situación motiva la necesidad de implementar un Aplicativo para gestionar el Plan de Fortalecimiento y Mejoras de Acreditación de carreras de la UNACH, aplicando una arquitectura que permita logra una mejor escalabilidad , facilidad de acoplamiento y de fácil mantenimiento, es decir parametrizable, ya que el modelo de evaluación emitido por el CEAACES es modificado continuamente.

[1] Consejo Evaluación Acreditación y Aseguramiento de la Calidad de la Educación Superior

[2] Instructivo para la Elaboración de Planes De Fortalecimiento para las Carreras en Proceso de Acreditación de las Instituciones de Educación Superior.
http://www.ceaaces.gob.ec/sitio/wp-content/uploads/2013/10/INSTRUCTIVO-PARA-LA-ELABORACI%C3%93N-DE-PLANES-DE-FORTALECIMIENTO-PARA-LAS-CARRERAS-EN-PROCESO-DE-ACREDITACI%C3%93N-DE-LAS-INSTITUCIONES-DE-EDUCACI%C3%93N-SUPERIOR.pdf

CAPÍTULO I

1. PROBLEMATIZACIÓN

Hoy en día se han generado múltiples arquitecturas y formas para acoplar la creación de sistemas informáticos, en la medida que los sistemas de software crecen en complejidad, bien sea por número de requerimientos o por el impacto de los mismos, se hace necesario establecer medios para el manejo de esta complejidad. En general, la técnica es descomponer el sistema en piezas que agrupan aspectos específicos del mismo, producto de un proceso de abstracción y que al organizarse de cierta manera constituyen la base de la solución de un problema en particular.

Al igual que en las grandes obras de construcción, para garantizar el éxito en el desarrollo de un aplicativo software se requiere antes que nada de una buena definición de la estructura que se va a seguir, de los distintos elementos o módulos que se van a construir y de cómo interactúan entre ellos de forma segura y eficaz. Un mal trabajo de arquitectura lleva en muchos casos al fracaso del proyecto, y al contrario, un buen trabajo de arquitectura, el producto resultante tenderá a ser robusto, el tiempo y esfuerzo para desarrollarlo más bajo, y algo muy importante, la facilidad para ampliar o extender el desarrollo en un futuro será mucho más alta.

En la actualidad los sistemas desarrollados en el departamento de Evaluación y Acreditación de la Universidad Nacional de Chimborazo están orientados a soportar los procesos de acreditación tanto institucional como de carreras basados en los Modelos emitidos por el CEAACES, estos modelos plantean el cumplimiento de Criterios, Subcriterios e Indicadores que deben ser gestionados internamente de manera que las evidencias del cumplimiento de estos indicadores estén siempre actualizados y sean de calidad. Como consecuencia de las evaluaciones por parte del CEAACES una carrera puede acreditarse o en su defecto requerir de un Plan de Fortalecimiento y Mejoras para lograr la acreditación respectiva. Esta situación ha desembocado en la creación de diversos sistemas que permitan dar respuesta a estos requerimientos, pero al mismo tiempo han conllevado a que los tiempos de desarrollo de estos sistemas sean demasiado extensos consuman mayores recursos, y tengan dificultades al momento de escalar o integrarlos a otros aplicativos.

1.2. IDENTIFICACIÓN Y DESCRIPCIÓN DEL PROBLEMA

Entre un modelo de Arquitectura de Capas y un modelo de Arquitectura N-Capas Orientada al Dominio con .NET, cuál es la mejor para desarrollar una aplicación que permita, automatizar la gestión de los procesos del Plan de Fortalecimiento y Mejoras en el Departamento de Evaluación y Acreditación de la UNACH.

1.3. ANÁLISIS CRÍTICO

Esta investigación caracteriza el estudio comparativo entre dos arquitecturas para el desarrollo del software, analizando sistemáticamente y metódicamente la arquitectura ideal para contribuir a la mejora de procesos institucionales en el Departamento de Evaluación y Acreditación de la UNACH.

1.4. PROGNOSIS

Al realizar el análisis comparativo de la Arquitectura Capas vs Arquitectura N-Capas Orientada al Dominio con .NET, va a dar cabida a desarrollar un sistema informático de manera eficiente y personalizada, especialmente a medida del tipo de aplicación que va a ser diseñada, por lo que se podría proponer una arquitectura para el sistema de Gestión de Recategorización para que se pueda seguir este modelo de manera que la tecnología avanza y pueda adaptarse a ella.

1.5. JUSTIFICACIÓN

En la Universidad Nacional de Chimborazo en el departamento de Evaluación y Acreditación, se viene gestionando diferentes proyectos para la acreditación de la misma, siendo así uno de los pilares fundamentales la recategorización, para lo cual se necesita una aplicación que sirva de apoyo para la automatización en la gestión de sus procesos, incluidos los Planes de Fortalecimiento y Mejoras.

A través de la investigación planteada se desarrollara una aplicación que automatice los procesos de los Planes de Fortalecimiento y Mejoras, que se vienen gestionando en este departamento, el cual ayudará a obtener un mejor desempeño de las diferentes actividades realizadas; contribuyendo como un indicador al momento de la recategorización.

1.6. DELIMITACIÓN

El proyecto de investigación, se limitara al análisis de la Arquitectura N – Capas Orientada al Dominio con .NET, específicamente en la propuesta de una arquitectura personalizada, la cual será robusta, escalable y que posteriormente interactuará con futuras aplicaciones, el cual nos llevará a descubrir parámetros para la evaluación de la arquitectura, enfocados a la optimización de los recursos a consumir en el desarrollo de aplicaciones.

Aplicando esta arquitectura se desarrollara el módulo de Administración, Evaluación y Seguimiento de los Planes de Mejoras Institucionales y Fortalecimiento de carreras y específicamente de la Carrera de Medicina.

1.7. FORMULACIÓN DEL PROBLEMA

¿Cuáles será la mejor arquitectura que permita desarrollar un aplicativo que se acople de manera eficiente a los sistemas del Departamento de evaluación, para automatizar la gestión de los Planes de Fortalecimiento y Mejoras Institucionales?

1.8. OBJETIVOS

1.8.1. Objetivo General

Comparar la Arquitectura Capas vs. Arquitectura N – Capas Orientada al Dominio con .NET, aplicado en el Sistema de Plan de Fortalecimiento y Mejoras de Acreditación de la UNACH.

1.8.2. Objetivos Específicos:

- Analizar las Arquitecturas N - Capas y N – Capas Orientada al Dominio con .NET
- Determinar parámetros de comparación y herramientas que permita seleccionar la mejor arquitectura para la gestión de procesos e interacción con otras aplicaciones.
- Implementar en el departamento de Evaluación y Acreditación de la UNACH, en base a la Arquitectura seleccionada en la investigación.

CAPÍTULO II

2. FUNDAMENTACIÓN TEÓRICO

2.1. ARQUITECTURA DE SOFTWARE

En el enfoque de arquitectura de una construcción, se presume muchos atributos distintos. En el nivel más básico, se considera la forma general de la estructura física. Pero, en realidad, la arquitectura es la manera en la que los distintos componentes del edificio se integran para formar un todo cohesivo[4], es la forma en la que la construcción se adapta a su ambiente y se integra a los demás edificios en la vecindad. La arquitectura no es el software [5]operativo. Es una representación que permite: 1) Analizar la efectividad del diseño para cumplir los requerimientos establecidos, 2) Considerar alternativas arquitectónicas en una etapa en la que hacer cambios al diseño todavía es relativamente fácil y 3) Reducir los riesgos asociados con la construcción del software (Pressman, 2010).

Bass, Clements y Kazman identifican tres razones fundamentales por las que es importante la arquitectura del software:

- Las representaciones de la arquitectura del software permiten la comunicación entre todas las partes (participantes) interesadas en el desarrollo de un sistema basado en computadora.
- La arquitectura resalta las primeras decisiones que tendrán un efecto profundo en todo el trabajo de ingeniería de software siguiente y, también importante, en el éxito último del sistema como entidad operacional.
- La arquitectura "constituye un modelo relativamente pequeño y asequible por la vía intelectual sobre cómo está estructurado el sistema y la forma en la que sus componentes trabajan juntos"
  (Bass, Clements, & Kazman, 2003)

[4] Cohesión hace referencia a la forma en que agrupamos unidades de software (módulos, subrutinas...) en una unidad mayor.
[5] Software es un conjunto de programas, instrucciones y reglas informáticas que permiten ejecutar distintas tareas en una computadora.

Entre las características se mencionan las siguientes, descritas por Shaw y Garlan:

- Composición: permiten la representación del sistema como composición de una serie de partes.
- Configuración: la descripción de la arquitectura es independiente de los componentes que formen parte del sistema.
- Abstracción: describen los roles abstractos que juegan los componentes dentro de la arquitectura.
- Flexibilidad: permiten la definición de nuevas formas de interacción entre componentes.
- Reutilización: permiten reutilizar tanto los componentes como la propia arquitectura.
- Heterogeneidad: permiten combinar descripciones heterogéneas.
- Análisis: permiten diversas formas de análisis de la arquitectura y de los sistemas desarrollados a partir de ella.
  (Shaw & Garlan, 1996)

La descripción de la arquitectura de los sistemas software adquiere gran importancia al aumentar su complejidad. Es por ello que, dado que las expectativas de los usuarios aumentan cada vez más y la gestión de estos sistemas por parte de los diseñadores se hace más difícil, es necesario elegir una arquitectura adecuada para ellos, así como la posterior implementación de cada una de las partes. Esta definición arquitectónica es la que permitirá gestionar los sistemas complejos, centrando su estudio en los componentes que lo forman y en las relaciones que se establecen entre ellos, de modo que, al final, estas partes componentes puedan trabajar conjuntamente para resolver el sistema. (Navasa, 2008)

2.1.1. Definiciones

Para tener una idea más clara, se puede considerar varias definiciones al mismo tiempo. A continuación, se mencionan varias de ellas:

Shaw y Garlan plantean lo siguiente sobre la arquitectura del software: “Desde el primer programa que se dividió en módulos, los sistemas de software han tenido arquitecturas y los programadores han sido los responsables de las interacciones entre los módulos y las propiedades globales del ensamble. Históricamente, las arquitecturas han estado implícitas:

accidentes de implementación o sistemas heredados del pasado. Los desarrolladores de buen software han adoptado con frecuencia uno o varios patrones de arquitectura [6]como estrategias para la organización del sistema, pero los utilizan de manera informal y no tienen manera de hacerlos explícitos en el sistema resultante." (Shaw & Garlan, 1996)

En el número de marzo/abril de 2006 de la IEEE Software, en el artículo de Kruchten, Obbink y Stafford figura la siguiente definición, más elaborada: (Kruchten , Obbink , & Stafford, Marzo-Abril 2006) "La arquitectura del software captura y preserva las intenciones de los diseñadores sobre la estructura y el comportamiento de los sistemas, proporcionando un mecanismo de defensa contra el deterioro de los sistemas antiguos. Esta es la clave para lograr el control intelectual sobre la creciente complejidad de los sistemas." (Kruchten, Obbink, & Stafford, 2006)

De acuerdo a todas estas definiciones que se han generado sobre la Arquitectura del Software, se puede definir como: "La estructura de un sistema, formada por componentes que se relacionan entre ellos." (Bass, Clements, & Kazman, 2012)

Por otra parte, la Arquitectura de Software puede ser considerada como la unión entre los requisitos y el código, la cual desempeña un rol importante entre la especificación de los requisitos de un sistema y la implementación del mismo. Al realizar la representación abstracta del sistema se constituyen una serie de características, mientras otras se ocultan, facilitando así una guía para que los arquitectos o diseñadores puedan apuntar a un modelo de construcción en función de los requisitos. (Pressman, 2010)

2.1.2. Evolución histórica de la Arquitectura del Software

Al momento de estudiar la evolución histórica de la Arquitectura de Software se puede mencionar varios antecedentes. Así se puede empezar citando a Dijkstra cuando, en 1968, propuso que: "Se debería establecer una correcta estructuración de los sistemas software antes de comenzar a programar." (Dijkstra, 1968)

[6] Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones.

En 1969 Brooks e Iverson definieron a la arquitectura como: “la estructura conceptual de un sistema, pero desde la perspectiva de la implementación.” (Brooks & Iverson, 1969)

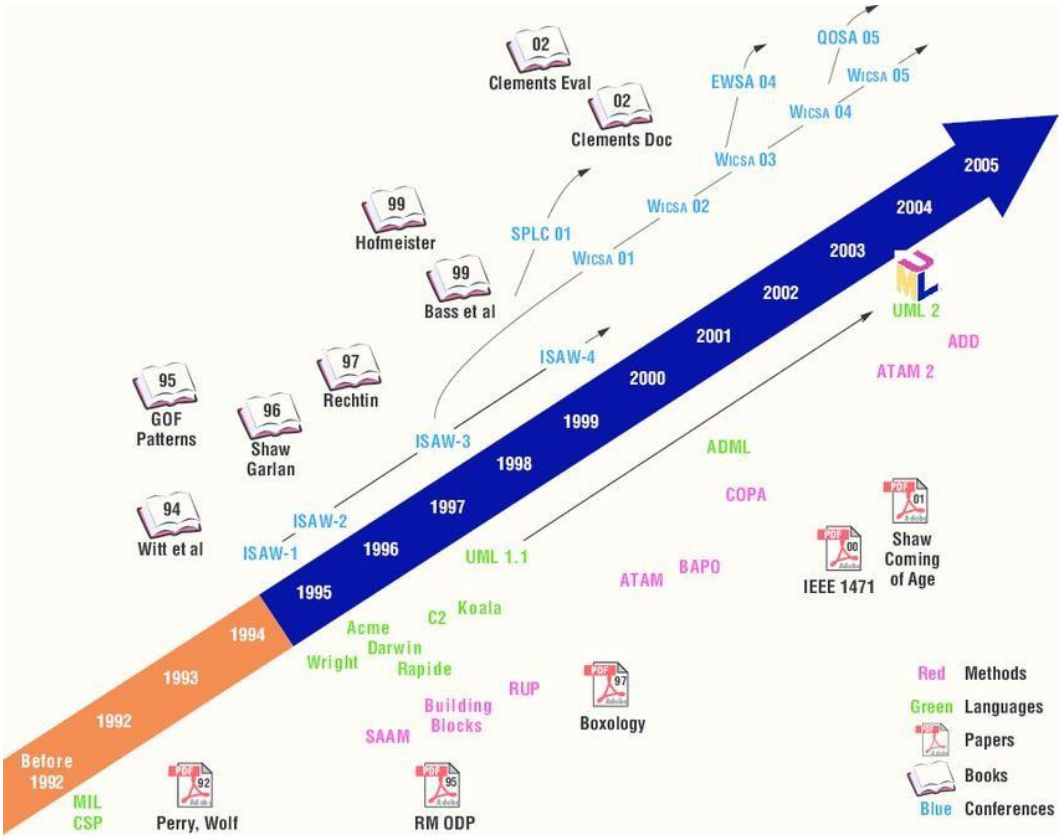
Durante los años 70 surge el diseño estructurado y las primeras investigaciones académicas en materia del diseño de sistemas complejos. Los trabajos que emergen en esta época acerca de la Arquitectura de Software de un sistema se deben a Parnas entre otros. (Parnas, 1972)

Shaw Mary dice que durante los años ochenta, la Arquitectura del Software empieza a tomar fuerza haciendo referencia a la configuración de la estructura de una aplicación. Mary Shaw, en 1984 vuelve a reivindicar las abstracciones[7] de alto nivel, solicitando un espacio para esa reflexión y augurando que el uso de esas abstracciones en el proceso de desarrollo puede resultar en: un nivel de arquitectura del software en el diseño, y por otro lado, que el desarrollo de grandes sistemas requiere abstracciones de alto nivel. (Shaw M. , 1984)

A principio de los años noventa se sugiere por primera vez la Arquitectura del Software en el sentido en el que se conoce actualmente, y surge realmente como disciplina cuando se publican los trabajos de Royce y Royce (Royce & Royce, 1990), Rechtin (Rechtin, 1991), y destacando el artículo de Perry y Wolf. Este trabajo se considera como punto de partida del actual auge de la Arquitectura de Software. Se puede decir que estos autores fundaron la disciplina. Para establecer lo que sucedió en esta década se cita una frase de Perry y Wolf, que ha quedado inscrita en la historia de esta disciplina: “La década de 1990, creemos, será la década de la arquitectura del software.” (Perry & Wolf, 1992)

[7] La abstracción consiste en aislar un elemento de su contexto o del resto de los elementos que lo acompañan.

Gráfico 1: Diez Años de la Arquitectura de Software



Fuente: *(Kruchten, Obbink, & Stafford, 2006)*

2.1.3. Norma IEEE 1471

Esta es la Norma que el IEEE[8] Computer Society propuso en 2000 como estándar para desarrollar sistemas software en términos de descripciones arquitectónicas. En ella se instaura un marco de trabajo o framework[9] conceptual para la descripción arquitectónica. Igualmente se define el contenido y las características de esas descripciones arquitectónicas. Como se sabe, la Arquitectura de Software tiene un gran dominio en el desarrollo de un sistema. Sin embargo, en general, las definiciones de arquitectura no se han aplicado adecuadamente.

[8] IEEE es el Instituto de Ingeniería Eléctrica y Electrónica
[9] Framework es un entorno o ambiente de trabajo para desarrollo; dependiendo del lenguaje normalmente integra componentes que facilitan el desarrollo de aplicaciones

Además, hasta la definición de la Norma no había un marco de trabajo comúnmente aceptado para configurar el pensamiento arquitectónico que facilitara la aplicación y evolución de las prácticas arquitectónicas. En la actualidad se comienza a aplicar conceptos arquitectónicos al desarrollo de los sistemas para lograr beneficios como reducción de costes e incremento de la calidad de los sistemas. Así pues, el objetivo genérico del estándar 1471 es proporcionar la comunicación de las arquitecturas y sentar las bases para desarrollos de calidad y coste reducido, a través de la estandarización de elementos de la descripción arquitectónica. Para ello, la Norma contiene recomendaciones sobre cómo efectuar la representación arquitectónica de un sistema durante su desarrollo. Sin embargo, no se trata de la propuesta de una arquitectura estándar, un proceso arquitectónico o un método, sino de un conjunto de recomendaciones prácticas para la definición de arquitecturas.

La Norma incluye las representaciones arquitectónicas que se usan en: la descripción del sistema y su evolución, la comunicación entre usuarios y desarrolladores, la evaluación y comparación de arquitecturas, las actividades de planificación y gestión del desarrollo, la expresión de características persistentes y guía para el cambio, y la verificación de la implementación relacionada con la descripción arquitectónica. (IEEE, 2000)

2.1.4. Conceptos fundamentales de la arquitectura del software de un sistema

2.1.4.1. Componentes arquitectónicos

El concepto primordial de la arquitectura del software es el de componente: un componente arquitectónico es cada una de las partes o fracciones de composición en las que se divide la funcionalidad de un sistema y cuya unión constituye el sistema completo. Un componente, que se relaciona con el resto del sistema mediante la interfaz que proporciona y que necesita, se puede conocer como una caja negra en la que la aplicación de los principios de encapsulamiento y ocultación de la información resultan fundamentalmente interesantes. Este concepto facilita la evolución de los sistemas al ser fácil cambiar un componente por otro que lo sustituya. Sin embargo, a veces es muy importante considerar ciertos elementos que proporcionen la adaptación de los componentes. Estos elementos adaptadores se suelen constituir a nivel arquitectónico mediante conectores específicos. (Navasa, 2008)

2.1.4.2. Conectores o conexiones arquitectónicas

Se denomina conector a cualquier dispositivo que se capaz de comunicar dos o más elementos y permiten modelar las interacciones entre componentes. La definición de conector arquitectónico fue introducida por Mary Shaw quién planteó separar la funcionalidad de un sistema (componentes) de su interacción (conectores). Los primeros se encargaban de ejecutar su tarea sin preocuparse de cómo se relacionan, mientras que los segundos se encargaban de resolver la comunicación de los primeros. De este modo se diseñó una separación de intereses que eleva el nivel de abstracción al momento de desarrollar un sistema, a la vez que aumenta su modularidad. (Shaw & Garlan, 1994)

2.1.4.3. Puertos

La definición de puerto es similar al de conector, pero no se debe confundir. Se denomina puerto a cada uno de los puntos por los que un componente puede efectuar cualquier tipo de interacción. Se puede decir que es cada uno de los segmentos en los que se fragmenta la interfaz[10] de un componente. Hace referencia a un punto de entrada o de salida de la caja negra que se considera el componente. Según esto, los puertos se representan tanto a los servicios que éste oferta como a los que se precisan. Por tanto, los puertos conforman el entorno externo de los componentes, lo que a su vez condiciona la estructura de la arquitectura. (Navasa, 2008)

2.1.4.4. Vistas

El concepto de vista, que ha sido el último propuesto de manera explícita, permite que una arquitectura de software pueda ser considerado desde diferentes representaciones, de manera que cada una de estas arquitecturas 'parciales' forma una vista; la visión completa del sistema se adquiere por combinación de todas ellas. Históricamente, los primeros enfoques multi-vista fueron Perry y Wolf (Perry & Wolf, 1992) que ya desde sus inicios propusieron que una arquitectura debería tener una vista de flujos, otra de control y otra de recursos. Pero sin duda la propuesta más divulgada es el denominado Modelo 4+1 de Kruchten (Kruchten P. , 1995), que ha sido posteriormente popularizada por su adaptación a UML y su difusión dentro del

[10] Interfaz es la conexión funcional entre dos sistemas o dispositivos de cualquier tipo dando una comunicación entre distintos niveles

campo de la programación orientada a objetos[11]. Este modelo distingue entre cinco vistas que se pueden aplicar al diseño arquitectónico. Estas son: la vista lógica, la de desarrollo, la de proceso, la vista física, y la vista de casos de uso. Cada una describe a un conjunto de características: la vista lógica soporta los requisitos funcionales y describe el modelo de objetos; la vista de desarrollo soporta también la especificación de los requisitos funcionales y la organización estática del sistema; la vista de proceso y la vista física consideran los requisitos no funcionales y se refieren a aspectos como la concurrencia, la distribución o la tolerancia a fallos; la vista de casos de uso incluye los casos de uso y los escenarios para definir y validar la arquitectura. La Norma Recomendada Std-1471 de la IEEE enfatiza la importancia de la definición de vista, adoptando explícitamente una perspectiva multi-vista para la Arquitectura del Software.

2.1.4.5. Abstracción

Es el encapsulamiento[12] propio de la tecnología de objetos, "Una abstracción denota las características esenciales de un objeto que lo distinguen de otras clases de objetos y provee de este modo delimitaciones conceptuales bien definidas, relativas a la perspectiva del observador", la abstracción consiste en extraer las propiedades esenciales o identificar los aspectos de un problema posponiendo o ignorando los detalles menos sustanciales o irrelevantes. (Sarasty España, 2016)

2.1.5. Lenguajes de descripción arquitectónica (LDA)

Una notación que describe y analiza explícitamente las propiedades observables de una arquitectura software, dando soporte a diferentes estilos arquitectónicos a otros niveles de abstracción. (Schneider, 1999)

La gran ventaja de usar un LDA sobre otras notaciones informales es que permiten una mejor comunicación entre el diseñador, implementadores y lectores debido a que la formalidad no deja

[11] Programación Orientada a Objetos es una técnica para desarrollar soluciones computacionales utilizando componentes de software (objetos de software).
[12] Encapsulamiento es el proceso de almacenar en una misma sección los elementos de una abstracción que constituyen su estructura y su comportamiento; sirve para separar el interfaz contractual de una abstracción y su implantación.

lugar a la ambigüedad[13]. Además, admite el análisis formal y temprano de las decisiones de diseño; es decir, una vez que el arquitecto conoce los requisitos de un sistema, decide su estrategia de diseño. A partir de ahí, debe expresar sus características y modelarlo aplicando una convención gráfica o algún lenguaje avanzado de alto nivel de abstracción. Los LDA son lenguajes que admiten modelar la arquitectura del sistema, analizar si es adecuada, determinar sus puntos críticos e, incluso, simular su comportamiento. (Navasa, 2008)

2.1.6. Propiedades

Según Shaw y Garlan los LDA deberían cumplir una serie de propiedades:

- Composición: Un lenguaje debe permitir la representación de un sistema como composición de una serie de partes (componentes) y la comunicación entre ellas (conectores). Igualmente deben admitir la definición de restricciones de diseño en la composición del sistema para dejar claro qué tipo de composiciones se permiten.
- Configuración: La descripción de la arquitectura debe ser independiente de la de los componentes que forman el sistema. Además, los LDA deben admitir realizar descripciones jerárquicas y encapsular subsistemas como componentes en sistemas grandes.
- Abstracción: Por medio de la cual se describen los roles o papeles abstractos que juegan los componentes dentro de la arquitectura.
- Flexibilidad: Permite la definición de nuevas formas de interacción entre componentes.
- Reutilización: Tanto de los componentes como de la propia arquitectura.
- Heterogeneidad: Permite combinar descripciones heterogéneas.
- Análisis de la arquitectura y de los sistemas desarrollados a partir de ella; incluso con la capacidad de formar prototipos del sistema.
  (Shaw & Garlan, 1994)

Según Navasa dice: Las ventajas de un cierto LDA vienen dadas por su poder expresivo para especificar la estructura y el comportamiento, pero también por su forma de uso, la funcionalidad, el rendimiento, la flexibilidad, la capacidad de reutilización, la facilidad de

[13] Ambigüedad es la posibilidad de que algo pueda entenderse de varios modos o de que admita distintas interpretaciones

comprensión y las restricciones tecnológicas. La mayoría de los LDA se complementan mediante herramientas: compiladores, comprobadores de restricciones, simuladores, etc.

Por último, para que un LDA admita la representación de la arquitectura de los sistemas software debería incluir las siguientes características:

- Dinamismo: Su objetivo es hacer explícitos los aspectos arquitectónicos del software para facilitar el desarrollo y evolución de sistemas complejos. Debido al carácter inherentemente dinámico y reconfigurable de muchos de estos sistemas, la capacidad de describir los aspectos que rigen la evolución de una arquitectura es un requisito básico de cualquier LDA. Las arquitecturas dinámicas admiten la replicación, inserción, eliminación y reconexión de sus componentes en tiempo de ejecución.
- Verificación de propiedades: La notación manejada para describir un sistema en un cierto LDA debe basarse en un formalismo que le sirva de soporte para la justificación de las propiedades de dicho sistema en las etapas iniciales del desarrollo, como es el diseño arquitectónico, reduciendo sustancialmente el coste de los errores. Un modelo formal proporciona una definición precisa de una arquitectura de software.
- Desarrollo del sistema y reutilización: El proceso de desarrollo de cualquier sistema de software de cierta complejidad pasa regularmente por una serie de refinamientos sucesivos en los que el sistema se constituye a diferentes niveles de abstracción, que lo van llevando sucesivamente desde la especificación a la implementación. Un buen LDA debe proporcionar mecanismos para el refinamiento de la arquitectura y sus componentes que faciliten este proceso de desarrollo.
  (Navasa, 2008)

Wolf propuso que, como mínimo, un lenguaje debe poder describir: componentes, conectores, configuraciones y restricciones, y además debe cumplir los siguientes requisitos:

- Ayudar en la definición de la arquitectura y en las tareas de refinamiento y validación.
- Definir las reglas sobre las que se construye una arquitectura completa.
- Tener la capacidad de representar la mayoría de los estilos arquitectónicos.

- Tener la capacidad de proporcionar vistas del sistema que expresen la información arquitectónica, a la vez que omita aspectos relacionados con la implementación del sistema. (Wolf, 1997)

2.2. Arquitectura N – Capas

El estilo arquitectural en N - Capas se basa en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas y las responsabilidades la funcionalidad que implementan. (Moquillaza Henriquez, Vega Huerta, & Guerra Grados, 2010)

2.2.1. Capas y Niveles

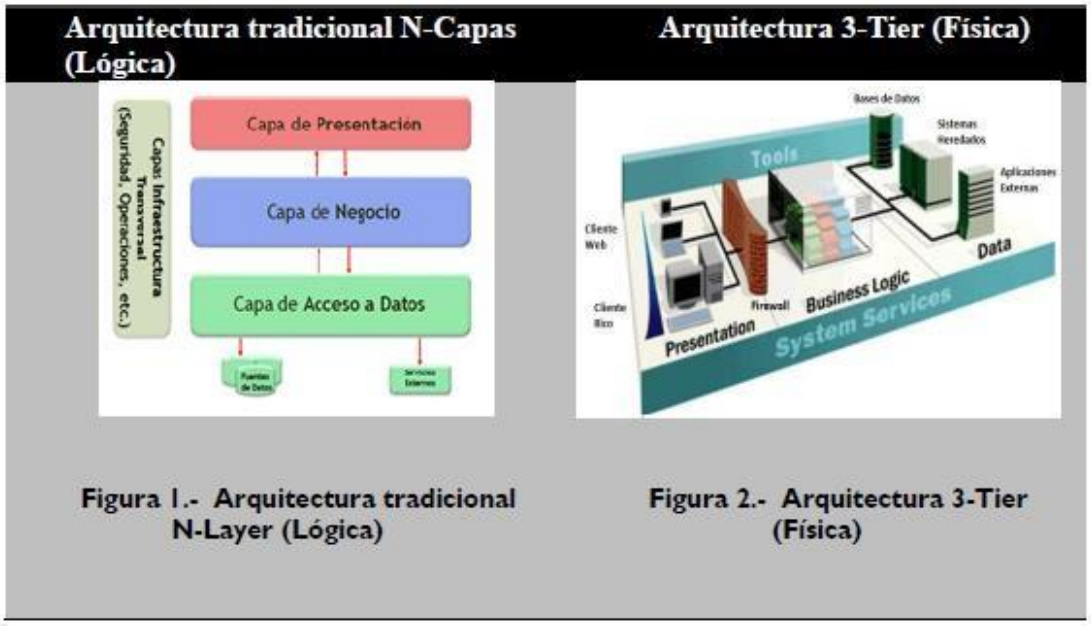
2.2.1.1. Capas

Las Capas (Layers) describen la división lógica de componentes y funcionalidad, y no tienen en cuenta la localización física de componentes en diferentes servidores o en diferentes lugares. (Llorente, Zorrilla Castro, Calvarro Nelson, & Ramos Barroso, 2010)

2.2.1.2. Niveles

Los Niveles (Tiers) se ocupan de la distribución física de componentes y funcionalidad en servidores separados, teniendo en cuenta topología de redes y localizaciones remotas. (Llorente, Zorrilla Castro, Calvarro Nelson, & Ramos Barroso, 2010)

Gráfico 2: N-Tier vs. N-Layer



Fuente: *(Llorente, Zorrilla Castro, Calvarro Nelson, & Ramos Barroso, 2010)*

2.2.2. Tipos de capas

2.2.2.1. Capa De Presentación

Es la responsable de la presentación visual de la aplicación. La capa de presentación enviará mensajes a los objetos de esta capa de negocios o intermedia, la cual o bien responderá entonces directamente o mantendrá un diálogo con la capa de la base de datos, la cual proporcionará los datos que se mandarían como respuesta a la capa de presentación. (Pressman, 2010)

2.2.2.2. Capa De Negocios

Pressman menciona que: Es la responsable del procesamiento que tiene lugar en la aplicación. Por ejemplo, en una aplicación bancaria el código de la capa de presentación se relacionaría simplemente con la monitorización de sucesos y con el envío de datos a la capa de procesamiento. Esta capa intermedia contendría los objetos que se corresponden con las entidades de la aplicación. Esta capa intermedia es la que conlleva la capacidad de mantenimiento y de reutilización.

Contendrá objetos definidos por clases[14] reutilizables que se pueden utilizar una y otra vez en otras aplicaciones. Estos objetos se suelen llamar objetos de negocios y son los que contienen la gama normal de constructores, métodos para establecer y obtener variables, métodos que llevan a cabo cálculos y métodos, normalmente privados, en comunicación con la capa de la base de datos. (Pressman, 2010)

2.2.2.3. Capa De Datos

Esta capa se encarga de acceder a los datos, se debe usar la capa de datos para almacenar y recuperar toda la información de sincronización del Sistema [N – Capas .Net].

Es aquí donde se implementa las conexiones al servidor y la base de datos propiamente dicha, se invoca a los procedimientos almacenados los cuales reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio. (Moquillaza Henriquez, Vega Huerta, & Guerra Grados, 2010)

2.2.3. Características

- Consiste en aislar la lógica de la aplicación y en convertirla en una capa intermedia bien definida y lógica del software.
- Los clientes pidan o envíen información a esta aplicación centralizada, no al gestor de base de datos en el servidor
- Los componentes de la aplicación pueden estar esparcidos en múltiples servidores permitiendo una mayor escalabilidad
- Los problemas de limitación para las conexiones a las bases de datos se minimizan ya que la base de datos solo es vista desde la capa intermedia y no desde todos los clientes. Además de que las conexiones y los drivers de las bases de datos no tienen que estar en los clientes.
- Especialización de la arquitectura cliente-servidor donde la carga se divide en tres partes (o capas) con un reparto claro de funciones: una capa para la presentación (interfaz de

[14] Clase es una plantilla para la creación de objetos de datos según un modelo predefinido. Las clases se utilizan para representar entidades o conceptos

usuario), otra para el cálculo (donde se encuentra modelado el negocio) y otra para el almacenamiento (acceso a datos).

2.2.4. Ventajas

- **Flexibilidad:** Proporciona la posibilidad que los componentes puedan ser modificados, estoy con el fin que puedan realizar sus operaciones sin necesidad de recompilar la aplicación, de esta manera se resguarda el contrato definido para la operación. De igual manera permite la reutilización de los componentes en otros tipos de aplicaciones y no solamente en la aplicación para la que fueron diseñados.
- **Mantenibilidad:** proporciona la tarea de modificar un componente con el fin de corregir errores, optimizar el desempeño, adicionar atributos o adaptarlos a un ambiente variable.
- **Reutilización:** permite que los componentes puedan ser utilizados desde otros componentes o desde otros sistemas. Lo cual quiere decir que inclusive, si los componentes de negocio son consumidos a través de servicios, estos pueden ser reutilizados por otros sistemas internos o externos.
- **Escalabilidad:** es la propiedad que permite que en este caso un componente se pueda adaptar al cambio o puedan crearse nuevos componentes sobre los componentes base para poder especializar más las capacidades de éste, específico para un cliente.
  (Sarasty España, 2016)

2.2.5. Desventajas

- Pueden incrementar el tráfico en la red cuando muchos clientes envían peticiones a un solo servidor. (Salinas, 2011)
- Requiere más balance de carga y tolerancia a las fallas. (Rojas, 2011)
- Complejidad, realización de trabajo innecesario o redundante entre capas.
- Dificultad al corregir la granularidad de las capas.
- Para asegurar estabilidad y calidad, cada capa debe contener sus propias pruebas unitarias.

2.3. Arquitectura N – Capas Orientada al Dominio con .Net

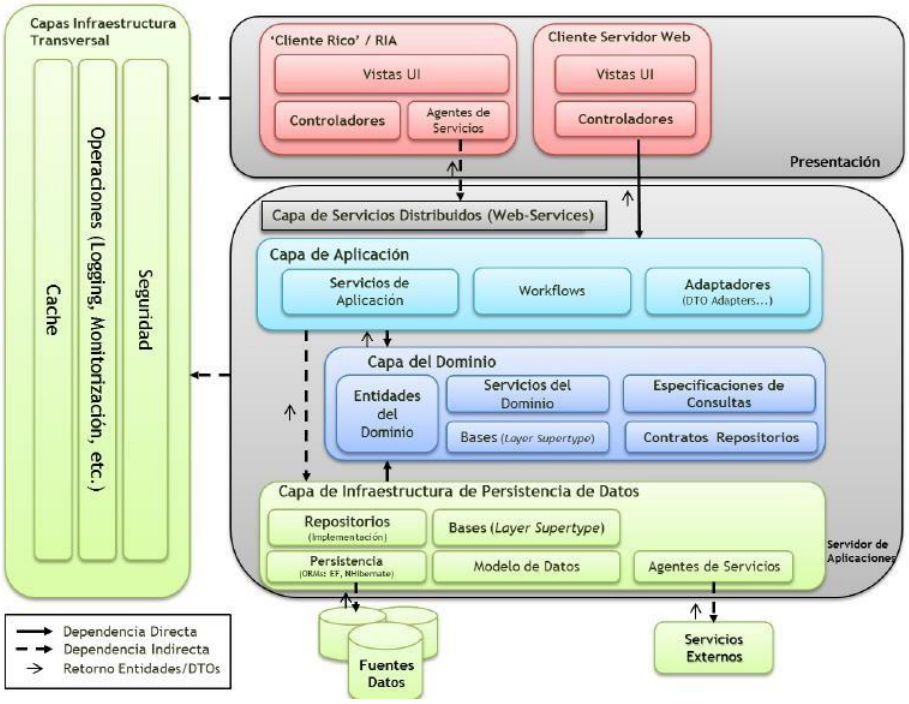
De la Torre, Zorrilla, Calvarro y Ramos dicen: En aplicaciones empresariales complejas que se caracterizan por tener volúmenes de cambios evolutivos considerables y tiempos de vida largos, esta arquitectura proporcionará bases sólidas para su construcción, ya que facilita el mantenimiento, actualización, o en su defecto el traspaso de tecnología y frameworks, por otras versiones más modernas o diferentes, etc. Esta arquitectura hace posible realizar todo esto, con el menor impacto posible en la aplicación, especialmente que no se vea afectada en lo menos posible la capa del Dominio de la aplicación.

La capa de lógica del Dominio, en aplicaciones muy complejas, están sujetas a sufrir cambios continuos y es importante que estas se puedan modificar, construir y que las capas de la lógica de Dominio se puedan realizar pruebas independientes. Esto conlleva a que el Modelo del Dominio (lógica y reglas de negocio) tenga un mínimo en acoplamiento con el resto de las capas que conforman el sistema (Capas de Presentación, Capas de Infraestructura, Persistencia de Datos, etc.).

La Arquitectura N - Capas Orientada al Dominio está orientada a conseguir un mínimo en acoplamiento entre las diferentes capas, en concreto la autonomía y centro dominante sobre la capa del Modelo de Dominio.

Esta arquitectura tiene como objetivo, estructurar las aplicaciones de una forma limpia y clara la complejidad de estas, utilizando diferentes capas de la arquitectura, implementando el patrón de N – Capas, la cual discierne las diferentes capas y subcapas de la aplicación, y tendencias de las arquitecturas DDD (Domain-Driven Design). (De la Torre Llorente, Zorrilla Castro, Calvarro, & Ramos Barroso, 2010)

Gráfico 3: Arquitectura N – Capas con Orientación al Dominio



Fuente: *(De la Torre Llorente, Zorrilla Castro, Calvarro, & Ramos Barroso, 2010)*

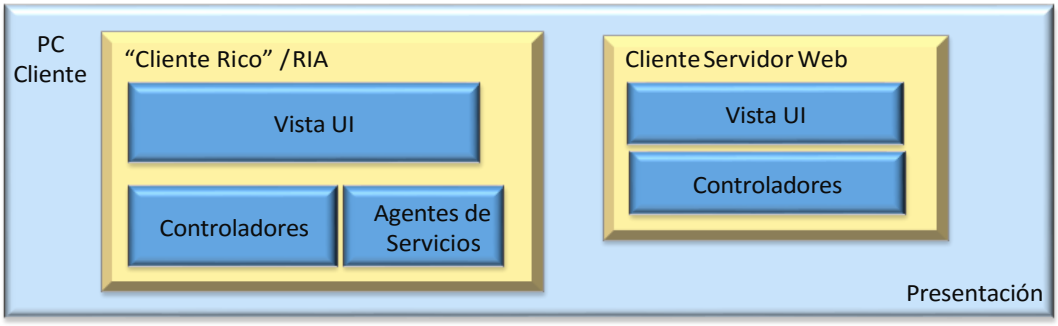
- Capa de Presentación
    - Subcapas de Componentes Visuales (Vistas)
    - Subcapas de Proceso de Interfaz de Usuario (Controladores y similares)
- Capa de Servicios Distribuidos (Servicios-Web[15])
    - Servicios-Web publicando las Capas de Aplicación y Dominio
- Capa de Aplicación
    - Servicios de Aplicación (Tareas y coordinadores de casos de uso)
    - Adaptadores (Conversores de formatos, etc.)
    - Subcapa de Workflows[16] (Opcional)
    - Clases base de Capa Aplicación (Patrón Layer-Supertype)
- Capa del Modelo de Dominio
    - Entidades del Dominio

[15] Servicio Web es una interfaz de software que describe un conjunto de operaciones a las cuales se puede acceder por la red a través de mensajería XML estandarizada.
[16] Workflow (flujo de trabajo) es el estudio de los aspectos operacionales de una actividad de trabajo: cómo se estructuran las tareas, cómo se realizan, cuál es su orden correlativo, cómo se sincronizan

    - Servicios del Dominio
    - Especificaciones de Consultas (Opcional)
    - Contratos/Interfaces de Repositorios
    - Clases base del Dominio (Patrón Layer-Supertype)
- Capa de Infraestructura de Acceso a Datos
    - Implementación de Repositorios‟
    - Modelo lógico de Datos
    - Clases Base (Patrón Layer-Supertype)
    - Infraestructura tecnología ORM
    - Agentes de Servicios externos
- Componentes/Aspectos Horizontales de la Arquitectura
    - Aspectos horizontales de Seguridad, Gestión de operaciones, Monitorización, Correo Electrónico automatizado, etc.

2.3.1. Capa de Presentación

Gráfico 4: Capa de Presentación



Fuente: *Eddy Navarrete/Diego Ortiz*

La capa de presentación se ocupa de mostrar al usuario la información e interpreta las acciones del usuario. (Escalante, 2016)

Según Llorente, Zorrilla y Ramos: En esta capa los componentes, implementan las funcionalidades requeridas, las cuales servirán para que el usuario interactúe con el sistema, en

muchos casos es recomendable que esta capa se subdivida los componentes en diferentes sub-capas, las que se tiene que aplicar patrones como MVC[17], M-V-VM[18], MVP[19].

- **Subcapa de Componentes Visuales (Vistas)**

  Los componentes de esta capa dotan del mecanismo base que el usuario necesita para utilizar el sistema. Estos componentes tienen la funcionalidad de formatear los datos, referente al estilo de letras y controles visuales, y esta capa también recibe la información que el usuario proporciona.

- **Subcapa de control**

  Esta subcapa se encarga de la sincronización de las interacciones del sistema con el usuario, es de mucha utilidad controlar los procesos, utilizando componentes separados con relación a los componentes gráficos. Esto conlleva que la lógica de gestión de estados y los flujos de procesos estén programados en el interior de los controles, que a posterior la lógica sea reutilizable conjuntamente con los patrones de otras interfaces. Esta subcapa nos permitirá realizar diferentes pruebas unitarias de la lógica de presentación. (De la Torre Llorente, Zorrilla Castro, Calvarro, & Ramos Barroso, 2010)

2.3.2. Capa de Servicios Distribuidos (Servicios Web) -Opcional-

Gráfico 5**:** Capa de Servicios Distribuidos (Servicios Web)

Fuente: *Eddy Navarrete/Diego Ortiz*

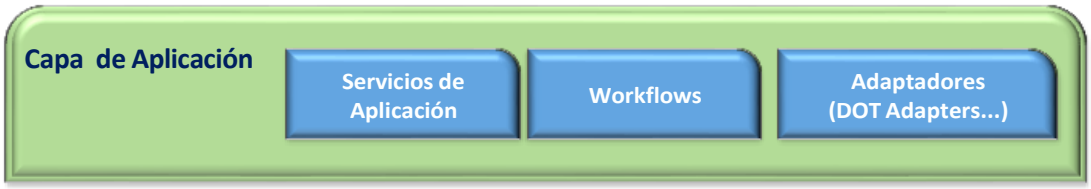
Muchas aplicaciones son proveedoras de servicios para diferentes aplicaciones remotas, habitualmente en la lógica de negocios (capa de negocio interna) estas son publicadas a través de la capa de servicios. La capa de Servicios Web tiene que ser lo más ligera y en esta capa nunca debe contener lógica de negocios, esta capa se comunicará mediante canales de

[17] MVC Modelo Vista Controlador
[18] M-V-VM Modelo Vista Vista Modelo
[19] MVP Modelo Vista Presentador

comunicación y mensajería de datos en forma de acceso remoto. (De la Torre Llorente, Zorrilla Castro, Calvarro, & Ramos Barroso, 2010)

2.3.3. Capa de Aplicación

Gráfico 6**:** Capa de Aplicación



Fuente: *Eddy Navarrete/Diego Ortiz*

En la Arquitectura Orientada al Dominio se propone la implementación de la Capa de Aplicación. En esta capa se define el trabajo que el sistema tiene que hacer, como dirigir a los objetos del dominio y de infraestructura que son los que resuelven los problemas. (Escalante, 2016)

De la Torre, Zorrilla y Ramos dicen: Básicamente esta capa se encarga de la coordinación de la tecnología, como ejecución de unidades de trabajo, transacciones, y en concreto tareas que el sistema requiere.

La Capa de Aplicación cumple el papel de fachada del Dominio, pero no únicamente esta se encarga de simplificar el acceso al Dominio, sino también:

- Organiza en su mayoría la llamada al Repositorio de objetos de la Capa de Persistencia y acceso a datos.
- Se encarga de la agregación y agrupación de las entidades con diferentes datos, para estos puedan ser enviados de una manera eficiente por la Capa de Servicios Web, A este tipo de datos se lo llama DTOs (Data Transfer Object), y el código en la Capa de Aplicación son DTO-Adapters.
- Se encarga de afianzar y asociar las diferentes operaciones del Dominio tomando en cuenta las acciones que el usuario muestra en la interfaz, asociando las operaciones que se llevan a cabo en la Capa de Persistencia y acceso a datos.
- Los estados del sistema que no se encuentran en la Capa del Dominio son mucho más fáciles de darle mantenimiento.

- Organiza entre la Capa del Dominio y las características de infraestructura las diferentes acciones que se pueden ejecutar entre estas, por ejemplo: El momento de ejecutar una transferencia bancario, esta necesita adquirir datos de los recursos de datos haciendo uso de los Repositorios, seguidamente del domino ocupa los objetos con la lógica de negocio de la transferencia y esta puede tener un opción de envió de correo al usuario final, utilizando otro objeto de infraestructura que ejecuta esta operación de envío de correos.

- **Servicios de Aplicación**

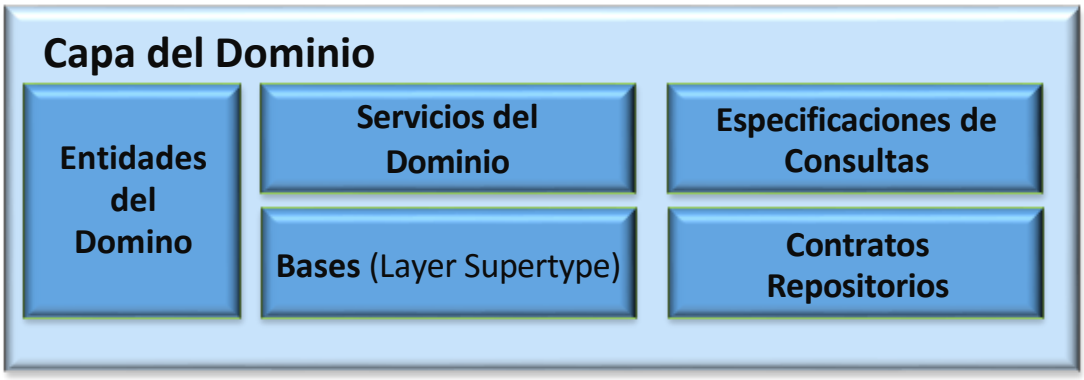
  Existe una gran diferencia entre los Servicios Web y los Servicios de Aplicación en la Arquitectura N - Capas que está Orientada al Domino. En las Capa de Aplicación, Dominio e Infraestructura existe diferentes capas de servicio DDD. Esto quiere decir que las clases agrupan funciones y comportamientos que no son de clases inferiores.
  Los Servicios de Aplicación son los que se encargan de organizar las tareas que deben realizar las capas inferiores (Servicios de Capas del Dominio, incluso los Servicios transversales de la capa de Infraestructura).

- **Workflows de Negocio (Opcional)**

  En diferentes organizaciones las tareas que se deben realizar deben cumplir un determinado proceso, los cuales tienen que cumplir reglas establecidas dependiendo del estado en el que se encuentre la tarea para llegar a su finalización. A este tipo de trabajos y procesos se los denomina Flujos de Trabajo (worflows) que son implementadas con tecnologías específicas y herramientas que gestionan las tareas. (De la Torre Llorente, Zorrila Castro, Calvarro, & Ángel, 2010)

2.3.4. Capa del Dominio

Gráfico 7: Capa del Dominio



Fuente: *Eddy Navarrete/Diego Ortiz*

La Capa del Dominio es la responsable de los conceptos de negocio, dota de resultados del estado en el que se encuentran los diferentes procesos de negocio y codificación de las reglas del dominio. La Capa del Dominio es el kernel[20] del sistema. (Escalante, 2016)

De la Torre, Zorrilla y Ramos dicen: Una vez que se definido los componentes y las características funcionales que estas cumplirán en la aplicación, también se encargarán del encapsulamiento con respecto a la lógica de negocio (lógica el Dominio DDD). Esencialmente son clases que implementan la lógica del dominio dentro de sus funcionalidades, los cuales siguen un patrón de la Arquitectura N – Capas con Orientación al Dominio, la Capa del Dominio no le da importancia a los detalles de la Capa de Persistencia de Datos, ya que este trabajo le corresponde a la Capa de Infraestructura y la Capa de Aplicación es la que se encarga de la coordinación.

A continuación, se especifican los componentes de la Capa de Dominio:

- **Entidades del Dominio**

  Son implementaciones de entidades que cumplen la funcionalidad de guardar y transferir a las diferentes capas datos de las entidades. La principal funcionalidad es que posee la lógica del dominio relativo de cada entidad. Por ejemplo, en campos pre-calculados de la lógica de

[20] El kernel también organiza a la manera en la que se ejecutan los diversos programas que se cargan en memoria.

negocios que tienen validaciones de datos relacionados. Mientras que los objetos en memoria son entidades de datos que el sistema se ejecuta internamente.

En definitiva, las entidades de datos que son utilizadas en el sistema, son objetos de memoria con datos y determinada lógica relacionada, mientras que las clases representan a las entidades que son utilizadas en el sistema. Cuando en las entidades se utilizan solo datos obviando la lógica de la propia entidad dentro de su propia clase, se estará cayendo en el caso del anti-patrón Anemic Domain Model. Es muy importante tomar en cuenta que se puede aplicar a las clases entidades objetos POCO (Plain Old CLR Objects), esto se refiere que las clases son independientes de la tecnología de acceso a los datos.

Lo que se pretende aplicando el diseño Persistence Ignorance, es que las clases del dominio ignoren por completo la estructura de los repositorios y de la tecnología implementada para el acceso a datos.

En el dominio se encuentran las clases entidad, ya que estas son el ente del dominio sin importar la tecnología de la infraestructura, en definitiva, a estas entidades se les tratara como objetos flotantes en la estructura de la arquitectura.

- **Servicios de Dominio**

  Los servicios en las capas del Dominio son clases de compartimiento y también puedes ser funciones con acciones de lógica del dominio. Estas clases tienen que ser clases Stateless ya que no debería tener estados relativos con respecto al dominio, ya que estas serán las que organicen y ejecuten las operaciones contra las entidades del dominio.

  Un ejemplo clásico de un Servicio del Dominio es cuando interactúa con más de una entidades en tiempos iguales. Sin embargo se puede tener un Servicio que ejecute únicamente las operaciones de obtener, actualizar, etc. contra una única entidad raíz.

- **Contratos de Repositorios**

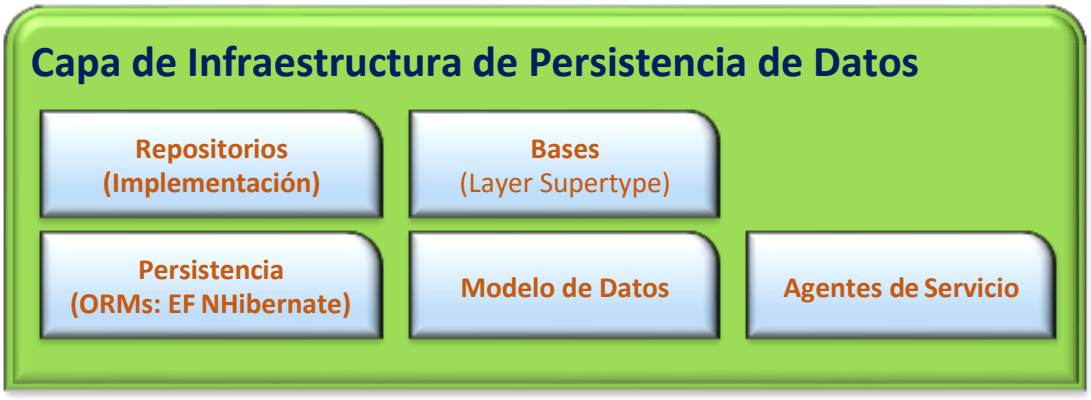
  Se tiene claro que la implementación de estos repositorios se lo tiene que hacer en la capa de Infraestructura, ya que no conforman en la capa del Dominio, en estos Contratos de Repositorio (Interfaces) estará especificado el funcionamiento de cada Repositorio y como

estos se deben integrar con la Capa del Dominio, sin tomar en cuenta como se encuentran implementados internamente.

Estas interfaces/contratos de Repositorios deben estar declarados en el interior de la Capa de Dominio ya que estas interfaces/contratos de Repositorio son escépticos a la tecnología, para que esto sea posible las Entidades del Dominio y los Objetos tienen que ser POCO; es decir, que dichos objetos que almacenan a las entidades y datos tienen que ser también escépticos a las tecnologías de acceso a los datos, esto se logra aplicando el patrón "**Separated Interface Pattern**". Se tiene que saber que estas entidades del Dominio son, los diferentes valores enviados y recibidos por y hacia los Repositorios. (De la Torre Llorente, Zorrilla Castro, Calvarro, & Ramos Barroso, 2010)

2.3.5. Capa de Infraestructura de Acceso a Datos

Gráfico 8: Capa de Infraestructura de Acceso a Datos



Fuente: *Eddy Navarrete/Diego Ortiz*

Esta capa es la que se encargara de la persistencia de datos y la forma lógica para acceder a los datos. Estos datos poder ser de origen interno del sistema o en su defecto, puede acceder a datos externos que son proveídos mediante los Servicios Web. En definitiva esta Capa de Infraestructura de Acceso a Datos provee el ingreso a los datos a las capas superiores, en específico a la Capa de Dominio. Este acceso a los datos se debe ejecutar de una forma desacoplada. (Escalante, 2016)

Según de la Torre, Zorrilla y Ramos describen los componentes de esta capa:

- **Implementación de Repositorios**

  El concepto de Repositorio "Representa todos los objetos de un cierto tipo como un conjunto conceptual" (Evans, 2006). En concreto un repositorio en una tipo clase el cual cumple la función de ejecutar las diferentes operaciones de persistencia y acceso a datos, el que está unido a una determinada tecnología como puede ser: Entity framework[21], NHibernate[22], o también poder ser ADO.NET para la administración de la base de datos. Al implementar la arquitectura de esta forma estamos focalizando la funcionalidad de acceso a los datos, lo que nos permite a posteriori un mantenimiento más eficiente, facilitándonos las configuraciones del sistema.

  Para cada Entidad Raíz del Dominio se deberá implementar un Repositorio. Lo que se pretende decir es que la Entidad Raíz y el Repositorio poseen una relación 1:1. En algunas ocasiones las Entidades Raíz se las puede aislar si se lo requiere, mientras que otras veces la raíz de un "Aggregate", que es un conjunto de entidades "Object Values" más la propia entidad raíz.
  Para ingresar a un determinado Repositorio se lo hará a través du una interfaz ya establecida, una interfaz colocada en el Dominio, de esta manera se lograra eliminar un Repositorio y remplazarlo por otro que puede estar implementado con una tecnología diferente y, esto conlleva a que la Capa de Dominio no se afecte en ninguna de sus instancias.
  La importancia de los Repositorios es que ayudan a la implementación, al estar establecido en la lógica del modelo de Dominio y ocultar la implementación del acceso a datos a través de las interfaces/contratos de Repositorios. A esto se lo llama como "PERSISTENCE IGNORANCE", esto nos quiere decir que el Modelo del Dominio desecha por completo la forma en que persisten o consultan los distintos datos.

[21] Entity Framework es un conjunto de tecnologías de ADO.NET que permiten el desarrollo de aplicaciones de software orientadas a datos.
[22] NHibernate es una biblioteca (marco) basado en .NET para persistir objetos a bases de datos relacionales.

La gran diferencia entre un objeto Data Access de la arquitectura tradicional N – Capas y un Repositorio, es que el objeto Data Access ejecuta directamente las funciones de persistencia y acceso a datos. Mientras que un Repositorio almacena en memoria las funciones que se desean realizar, esto se ejecutara cuando la Capa de Aplicación necesite realizar "n" operaciones de persistencia o acceso a datos en más de una acción al mismo tiempo. Esto se logra implementando el patrón Unidad de Trabajo (Unit of Work), que en determinadas ocasiones los rendimientos de los sistemas aumentan, y en todas las ocasiones, elimina casi en su totalidad las inconsistencias que pudrieran producirse y por último disminuye el tiempo de interrupción en las tablas que se producen al realizar transacciones.

- **Componentes Base (Layer Super Type)**

  En un gran porcentaje de las funciones de acceso a datos, poseen una determinada lógica semejante que suele ser adquirida y construida en componentes distintos y reutilizables. Al realizar este trabajo se está disminuyendo la complejidad de todos los componentes de acceso a datos y lo más importante, es que disminuye la cantidad de código a mantener.

  Esto se puede lograr mediante la implementación del patrón Layered Supertype Pattern, que quiere decir "Si los comportamientos y acciones comunes de un tipo de clases se agrupan en una clase base, esto eliminará muchos duplicados de código y comportamientos" (Martin, 2008).

  En todas las capas de la Arquitectura N – Capas Orientada al Dominio se puede implementar este patrón.

- **Modelo de Datos**

  Habitualmente en aplicaciones como ORM[23] (como Entity Framework) poseen técnicas para definir el modelo de datos a nivel de diagrama (entidad - relación). Esta capa en la que contiene los modelos de Entidad – Relación.

- **Agentes de Servicios remotos/externos**

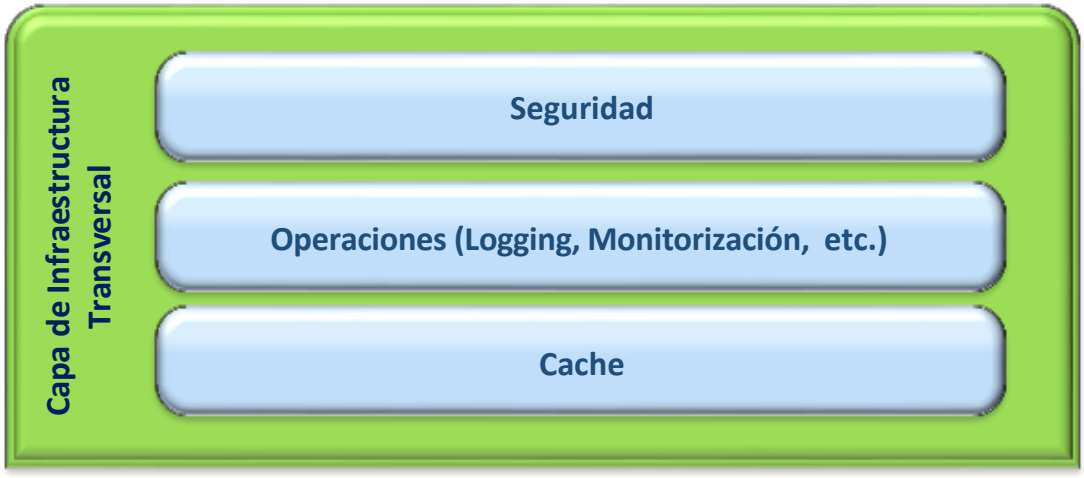
  Cuando los servicios externos como los Servicios Web proporcionan funcionalidades a los componentes de negocio, es importante implementar código que administre la

[23] ORM Object Relational Mapping

interacción entre estos. (Llorente, Zorrilla Castro, Calvarro Nelson, & Ramos Barroso, 2010)

2.3.6. Capas de Infraestructura Transversal/Horizontal

Gráfico 9: Capas de Infraestructura Transversal/Horizontal



Fuente: *Eddy Navarrete/Diego Ortiz*

Según de la Torre, Zorrilla y Ramos describen que esta capa Brindan servicios básicos de soporte a las capas superiores. Estos son componentes unidos a tecnologías específicas para desarrollar sus tareas.

Hay algunos trabajos implícitos en los componentes de un sistema que veden ser implementadas en diferentes capas, estos trabajos o aspectos transversales codifican diferentes tareas que pueden ser ligadas/usadas por cualquier capa.

Los aspectos transversales más generales son: Seguridad (Validación, Autorización y Autenticación) y funciones que gestionan las operaciones (logging, monitorización, configuración, etc.).

- **Subcapas de Servicios de Infraestructura**

  La idea d Servicio también se aplica en esta capa transversal, esta capa tiene la funcionalidad de asociar las funcionalidades de infraestructura, por ejemplo: envió de correo, supervisar la seguridad, administración de las operaciones, registros, etc. En definitiva los Servicios, asocian todo tipo de tarea de la infraestructura transversal unida a una tecnología concreta.

- **Subcapa de objetos de infraestructura**

  En determinados aspectos de infraestructura transversal, nosotros necesitaremos algunos objetos para implementarlos, ya sea seguridad, monitorización, control, envió de correo, etc. (Llorente, Zorrilla Castro, Calvarro Nelson, & Ramos Barroso, 2010)

2.3.7. Patrón Entities

La característica clave de una entidad es que tiene una identidad. Una entidad es única en el sistema a menos que otra entidad tiene la misma identidad. (C. S. Santos, 2015 12th)

Evans dice: El patrón Entity se fundamenta en que existen objetos que no son esencialmente definidos en base a sus atributos, sino más bien se centra en los hilos de continuidad e identidad los cuales permanecen en el ciclo de vida. Estas Entidades se pueden crear, persistir, recuperar, viajar por medio de la red y volverse a integrarse del otro lado, independientemente que para esta operación se tenga que crear una instancia nueva del objeto, las entidades tendrán que conservarse. El caso de inicio de identidad en un programa de software conlleva a obtener problemas de corrupción de datos y errores en el proceso de ejecución de estos.

Cada Entidad debe poseer un método el cual compare su identidad con otra instancia referente a la misma clase. Muchas veces los objetos tienen la misma identidad que otros, aunque estos no coincidan en los valores de todos sus atributos. Las Entidades tienen que ser distinguibles, aunque existan otros objetos con iguales atributos. Ejemplos obvios de Entities es: Vehículo, Persona, Organización, Factura, País, etc. (Evans, 2006)

2.3.8. Patrón Value Objects

A diferencia de una entidad, un objeto de valor no posee una identidad, siendo sólo un aspecto descriptivo del dominio. Otra característica del valor de los objetos es que sus valores deben ser inmutables, es decir, una vez creados no pueden ser cambiados. (C. S. Santos, 2015 12th)

Según Avream y Marinescu dicen: Ejemplos de Value Objects son: Especie, Correo, Puertas, Ciudades, etc. Al pensar qué objetos son Value Objects y los objetos son Entities depende la como este estructurado el dominio. En un programa los servicios postales, en este caso la Dirección podría ser un Entity y más no un Value Object como en muchos casos.

Value Objects no siempre son simples. En ocasiones un Value Object puede contener muchos atributos y operaciones. En conclusión, son objetos que se utilizan para pasar por parámetro información a otro objeto. Por lo normal estos son transitorios, creados especialmente para realizar la operación y luego desechados. (Avram & Marinescu, 2006)

2.3.9. Patrón Services

Ofrece operaciones específicas que no ahorran estados, es decir, todas las llamadas de servicio, proporcionado por la misma entrada deben tener el mismo resultado. Un servicio está indicado para separar la lógica que trata las diversas entidades u operaciones muy complejas. (C. S. Santos, 2015 12th)

Según Evans dice: Los Servicios son conocidas como actividades, funciones, operaciones y no son cosas. Estos no conservan estado; más bien son stateless. Los Servicios son operaciones ejecutadas como una interfaz suelta en el entorno del dominio, la cual no se encarga de encapsular el estado, como lo hacen las Entities y Value Objects.

Los Servicios tienen una responsabilidad bien definida, y estas responsabilidades y sus interfaces tienen que estar definidas como componente del Modelo de Dominio. Los resultados como los parámetros del Servicio tienen que ser objetos de dominio. (Evans, 2006)

2.3.10. Patrón Modules

Avram y Marinescu dicen: Los módulos son ampliamente utilizados en la mayoría de los proyectos. Es más fácil para obtener la imagen de un modelo grande si nos fijamos en los módulos que contiene, a continuación, en las relaciones entre dichos módulos. Después de la interacción entre los módulos se entiende, se puede empezar a averiguar los detalles en el interior de un módulo. Es una manera sencilla y eficaz para gestionar la complejidad Otra razón para el uso de los módulos está relacionado con la calidad del código.

Es ampliamente aceptado que el código de software debe tener un alto nivel de cohesión y un bajo nivel de acoplamiento. Mientras que la cohesión se inicia en el nivel de clase y el método, se puede aplicar a nivel de módulo. Se recomienda a las clases de grupo muy relacionadas en módulos para proporcionar la máxima cohesión posible. (Avram & Marinescu, 2006)

2.3.11. Patrón Aggregates

Un agregado es un conjunto de objetos asociados que son considerados una unidad. Se previene que se mesclen los objetos, que es esencial para preservar la integridad del modelo de dominio. Un ejemplo de agregada es: Orden, junto con artículos de pedido. (S. A. Soares, 2015)

Según Evans dice: Un ejemplo se Aggregate es una Persona (un Entity) la cual posee una Dirección (un Value Object). Los dos objetos están relacionados una composición. La Dirección es un atributo de la Persona la cual no tienen sentido fuera de la Persona. La Persona es la que se actualiza más no la Dirección, esto quiere decir que se actualiza la dirección que posee una Persona.

Toda Agregación posee una Entidad Raíz (root) y una Entidad Externa (boundary). La entidad Externa determina qué está dentro de la Agregación y qué no. La Entidad Raíz es la Entidad que se encuentra contenida en la Agregación el cual es el punto de ingreso a la Agregación. (Evans, 2006)

2.3.12. Patrón Factories

Los Factories son utilizados para realizar el encapsulamiento de la creación de objetos complejos. En el momento de crear una Entidad, este podría ser muy, ya que puede incluir algunas Entidades y Value Objects. Un claro ejemplo es una Entidad Raíz de una Agregación, la cual implica la creación de algunos objetos con sus respectivas reglas de negocio y consistencia las cuales tienen que ser cumplidas. Para abstraer al cliente y al objeto en sí de este complejo mecanismo es que se provee un Factory que se encargue del trabajo. (Avram & Marinescu, 2006)

2.3.13. Patrón Repositorios

Proporciona mecanismo de inserción, eliminación y los objetos de consulta persistidos de forma transparente para la capa de dominio desde una sola interfaz, abstrayendo la base de datos utilizada. Se debe prever consultas por criterios y evitar consultas directas a la base de datos, que pueden dañar a las reglas de agregaciones y mezclarse con la lógica de negocio de comandos

de la base de datos, por ejemplo. Esto permite que el la capa lógica de negocio sea tratada íntegramente en la capa dominio. (S. A. Soares, 2015)

Según Avram y Marinescu dicen: Los objetos tienen un ciclo de vida a partir de la creación y termina con la eliminación o archivado. En un lenguaje orientado a objetos, uno debe contener una referencia a un objeto con el fin de ser capaz de utilizarlo. Para tener una referencia tal, el cliente debe crear el objeto u obtenerla de otro, por la que atraviesa una asociación existente. Por ejemplo, para obtener un objeto de valor de un agregado, el cliente deberá solicitarla a la raíz del agregado.

El problema ahora es que el cliente debe tener una referencia a la raíz. Para aplicaciones de gran tamaño, esto se convierte en un problema porque hay que asegurarse de que el cliente siempre tiene una referencia al objeto sea necesario, o a otro que tiene una referencia al objeto respectivo. Usando una norma de este tipo en el diseño obligará a los objetos para mantener una serie de referencias que probablemente no mantendrían lo contrario. Esto aumenta el acoplamiento, la creación de una serie de asociaciones que no son realmente necesarios. (Avram & Marinescu, 2006)

2.4. Adaptabilidad y Rendimiento

2.4.1. Adaptabilidad

Es la oportunidad en la cual el software se adapta a un nuevo entorno sin aplicar acciones diferentes o medidas en las cuales sean distintas al medio proporcionado para integrar el software en cuestión.

2.4.2. Rendimiento

Hace referencia al tiempo que tarda en la velocidad de procesamiento, el tiempo de respuesta, consumo de recursos, rendimiento efectivo total y eficacia de una determinada tarea en las condiciones del entorno de trabajo.

Escalabilidad: se refiere al volumen de usuarios simultáneos y/o al volumen de transacciones que pueden realizarse; debe funcionar igualmente con un número pequeño o grande de usuarios. Se traduce en el RNF (Requerimientos no funcionales) Rendimiento (capacidad-escalabilidad). (Losavio & Esteves, 2016)

Sin embargo, durante el desarrollo se ha dificultado la obtención de valores óptimos para estas cualidades, fundamentalmente para la adaptabilidad y otros indicadores asociados a este, mantenibilidad, flexibilidad y escalabilidad. (Arias & Martinez, 2013)

Modularidad

La modularidad es la manifestación más común de la división de problemas. El software se divide en componentes con nombres distintos y abordables por separado, en ocasiones llamados módulos, que se integran para satisfacer los requerimientos del problema.

Productividad

La forma como el software permite a los usuarios emplear cantidades apropiadas de recursos, en relación a la eficacia lograda en un contexto específico de uso.

Para una empresa es muy importante que el software no afecte la productividad del empleado

CAPÍTULO III

3. METODOLOGÍA

3.1. TIPOS DE ESTUDIO

3.1.1. Descriptiva – comparativa

Descriptiva

En base a las variables de estudio expuestas, se detallan las principales características o indicadores de cada una de las arquitecturas propuestas, obteniendo la más adecuada para el desarrollo de aplicaciones de gestión de información como el caso del Aplicativo del Plan de Mejoras y Fortalecimiento de la Carreras de Pregrado de la UNACH.

Para la selección de la arquitectura más adecuada, se toma en consideración los indicadores definidos de seguridad y rendimiento, y si miden aspectos como la Modularidad, Productividad y Escalabilidad junto con sus características más representativas, estableciendo una valoración basada en la bibliografía especializada**.**

Se aplica la siguiente escala de valoración:

Tabla 1 Escala de Valoración

Descripción de la Característica	Valor
No Cumple	1
Cumple Parcialmente	2
Cumple Completamente	3

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 2 Aspecto 1: Modularidad.

Característica	Arquitectura N - Capas	Arquitectura N – Capas Orientada al Dominio con .NET
Reutilización	3	3
Reducción de Costos	2	3
Claridad	2	3
Total	**7**	**9**
Porcentaje	**77,7 %**	**100 %**

Fuente: *Eddy Navarrete/Diego Ortiz*

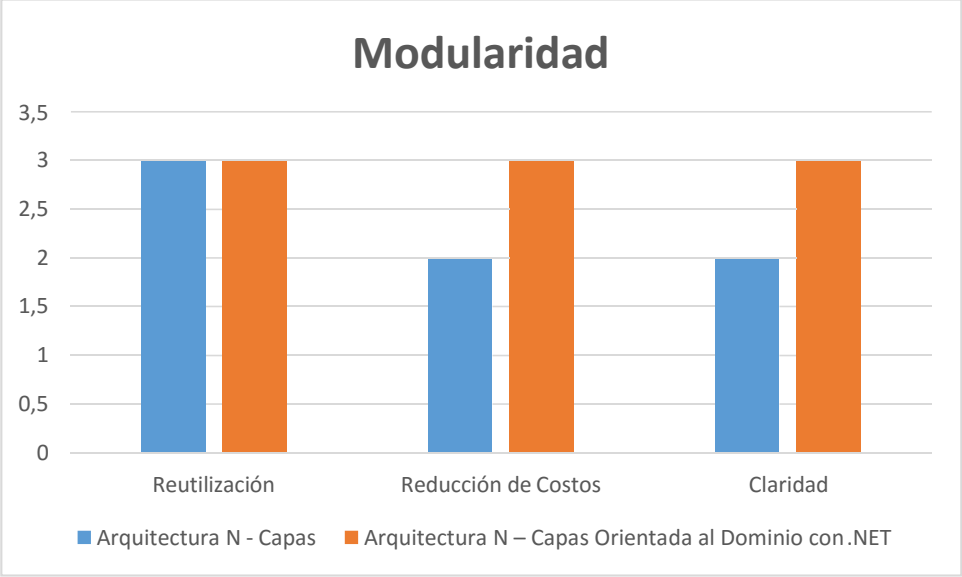
Gómez y Moraleda establecen que en la Reutilización tanto la Arquitectura N – Capas y la Arquitectura N – Capas Orientada al Dominio con .NET los módulos se diseñan teniendo en cuenta otras posibles aplicaciones las cuales resultara inmediata su reutilización.

En reducción de Costos la Arquitectura N – Capas Orientada al Dominio con .NET resulta más económico desarrollar, depurar, documentar, probar y mantener un sistema modular que la Arquitectura N – Capas no lo hace, excepto, si el número de módulos crece innecesariamente.

La Claridad en la Arquitectura N – Capas Orientada al Dominio con .NET es más fácil entender y manejar cada una de las partes del sistema, que tratar de entenderlo como un todo compacto; como lo hace N – Capas. (Gómez & Moraleda, 2014)

Las razones por las que es más adecuado hacer uso de una “Arquitectura N – Capas Orientada al Dominio .NET”, es, especialmente en los casos donde el comportamiento del negocio a automatizar (lógica del dominio) está sujeto a muchos cambios y evoluciones. En este caso específico, disponer de un “Modelo de Dominio” disminuirá el coste total de dichos cambios, y a medio plazo el TCO (Coste Total de la Propiedad) será mucho menor que si la aplicación hubiera sido desarrollada de una forma más acoplada, porque los cambios no tendrán tanto impacto. (Llorente, Zorrilla Castro, Calvarro Nelson, & Ramos Barroso, 2010)

Gráfico 10 - Cuadro estadístico de indicadores de Modularidad.



Fuente: *Eddy Navarrete/Diego Ortiz*

Análisis

Entre estas dos arquitecturas, la Reutilización Cumplen Completamente con esta característica.

En la Arquitectura N – Capas, la Reducción de Costos, Cumple Parcialmente con esta característica.

En la Arquitectura N – Capas Orientada al Dominio con .NET, la Reducción de costos, Cumple Completamente con esta característica.

En la Arquitectura N – Capas, la Claridad Cumple Parcialmente con esta característica.

En la Arquitectura N – Capas Orientada al Dominio con .NET, la Claridad Cumple Completamente con esta característica.

Interpretación

Entre estas dos arquitecturas existe un mismo nivel de reutilización, debido a que dividen a los sistemas en módulos.

La Arquitectura N – Capas tiene un nivel bajo de Reducción de costos, debido a que esta arquitectura desarrolla los sistemas con baja modularidad. (Por: Los autores)

La Arquitectura N – Capas Orientada al Dominio con .NET tiene un nivel alto de Reducción de costos, debido a que desarrolla los sistemas con mayor modularidad.

La Arquitectura N – Capas tiene una la Claridad muy baja, debido a que es más difícil entenderlo como un todo compacto.

La Arquitectura N – Capas Orientada al Dominio con .NET, tiene una la Claridad muy alta, debido a que es más fácil entender y manejar cada una de las partes del sistema.

Después de analizar estas tres características o indicadores de la Modularidad, podemos tener un alto grado de certeza que la Arquitectura N – Capas Orientada al Dominio con .NET, posee un mayor nivel de Modularidad con respecto a la Arquitectura N – Capas. (Gómez & Moraleda, 2014) y (Llorente, Zorrilla Castro, Calvarro Nelson, & Ramos Barroso, 2010)

Tabla 3 Aspecto 2: Productividad.

	Arquitectura N - Capas	Arquitectura N – Capas Orientada al Dominio con .NET
Productividad en Mantenibilidad	2	3
Productividad en Ejecución	3	2
Total	**5**	**5**
Porcentaje	**83,33%**	**83,33 %**

Fuente: *Eddy Navarrete/Diego Ortiz*

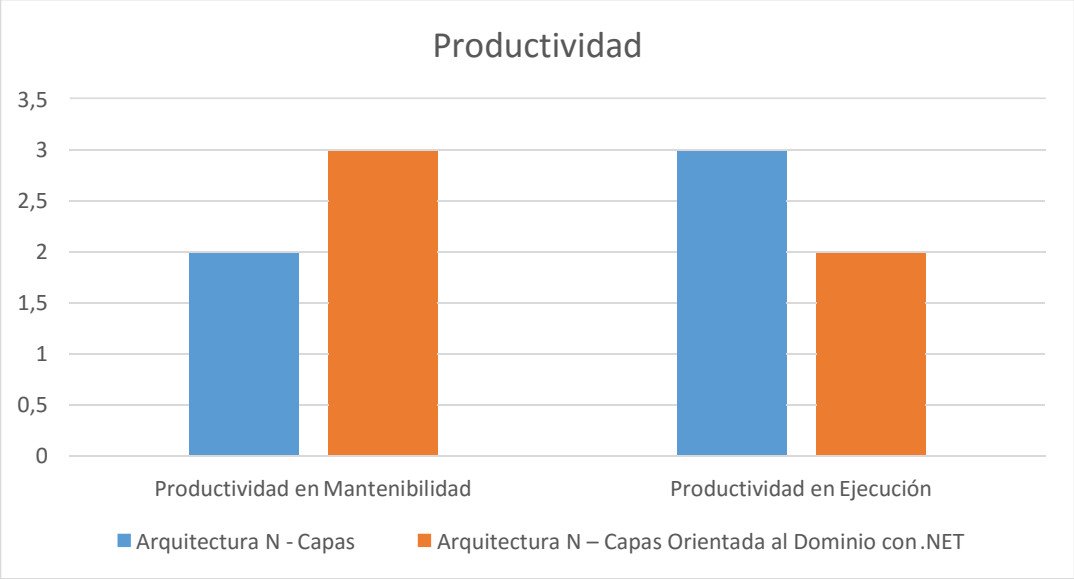
Según Torre Llorente, Zorrilla, Calvarro y Ramos dicen que si la aplicación a realizar es relativamente sencilla y, sobre todo, si las reglas de negocio a automatizar en la aplicación cambiarán muy poco y no se prevén necesidades de cambios de tecnología de infraestructura durante la vida de dicha aplicación, entonces, probablemente la solución no debería seguir el tipo de arquitectura N – Capas Orientada al Dominio con .NET, y más bien se debería seleccionar un tipo de desarrollo/tecnología RAD (Rapid Application Development), como puede ser Arquitectura N - Capas. Es decir, tecnologías de rápida implementación a ser

utilizadas para construir aplicaciones sencillas donde el desacoplamiento entre todos sus componentes y capas no es especialmente relevante.

Las razones por las que es más adecuado hacer uso de una "Arquitectura N - Capas Orientada al Dominio", es especialmente en los casos donde el comportamiento, del negocio a automatizar (lógica del dominio) está sujeto a muchos cambios y evoluciones. En este caso específico, disponer de un "Modelo de Dominio" disminuirá el coste total de dichos cambios. El tener un comportamiento del negocio dinámico encapsulado en una única área del software, disminuye drásticamente la cantidad de tiempo que se necesita para realizar un cambio; este cambio se realizará en un solo sitio y podrá ser convenientemente probado de forma aislada, aunque esto por supuesto dependerá de cómo se haya desarrollado.

El poder aislar tanto como sea posible dicho código del Modelo del Dominio, disminuye las posibilidades de tener que realizar cambios en otras áreas de la aplicación (lo cual siempre puede afectar con nuevos problemas, regresiones, etc.). Esto es de vital importancia si se desea reducir y mejorar los ciclos de estabilización y puesta en producción de las soluciones. (Llorente, Zorrilla Castro, Calvarro Nelson, & Ramos Barroso, 2010)

Gráfico 11 Cuadro estadístico de indicadores de Productividad.



Fuente: *Eddy Navarrete/Diego Ortiz*

Análisis

En la Arquitectura N – Capas la Productividad cumple parcialmente con esta característica.

En la Arquitectura N – Capas Orientada al Dominio con .NET la Productividad en mantenibilidad cumple completamente con esta característica.

En la Arquitectura N – Capas la Productividad cumple completamente con esta característica.

En la Arquitectura N – Capas Orientada al Dominio con .NET la Productividad en ejecución cumple parcialmente con esta característica.

Interpretación

La Arquitectura N – Capas tiene un nivel bajo en Productividad referente a la mantenibilidad, debido a que esta arquitectura básicamente se conforma por tres capas lo que conlleva a que el código no se encuentre lo suficientemente distribuido y localizado en la aplicación. (Llorente, Zorrilla Castro, Calvarro Nelson, & Ramos Barroso, 2010)

La Arquitectura N – Capas Orientada al Dominio con .NET tiene un alto nivel en Productividad en cuanto a la mantenibilidad, debido a que esta arquitectura presenta una subdivisión de cada una de las capas y en especial cuenta con la capa de Dominio la cual concentra las principales funcionalidades del sistema. (Llorente, Zorrilla Castro, Calvarro Nelson, & Ramos Barroso, 2010)

La Arquitectura N – Capas tiene un alto nivel en cuanto a la Productividad en ejecución, debido a que esta arquitectura básica cuenta con tres capas, en el momento de ejecutar los procesos no tiene que recorrer muchas capas de la aplicación produciendo que estos procesos se ejecuten con mayor rapidez. (Llorente, Zorrilla Castro, Calvarro Nelson, & Ramos Barroso, 2010)

La Arquitectura N – Capas Orientada al Dominio con .NET tiene un bajo nivel en cuanto a la Productividad en ejecución, debido a que esta arquitectura cuenta con un mayor número de capas, en el momento de ejecutar los procesos estos tendrán un mayor tiempo de ejecución debido a que tienen que recorrer por un mayor número de capas. (Llorente, Zorrilla Castro, Calvarro Nelson, & Ramos Barroso, 2010)

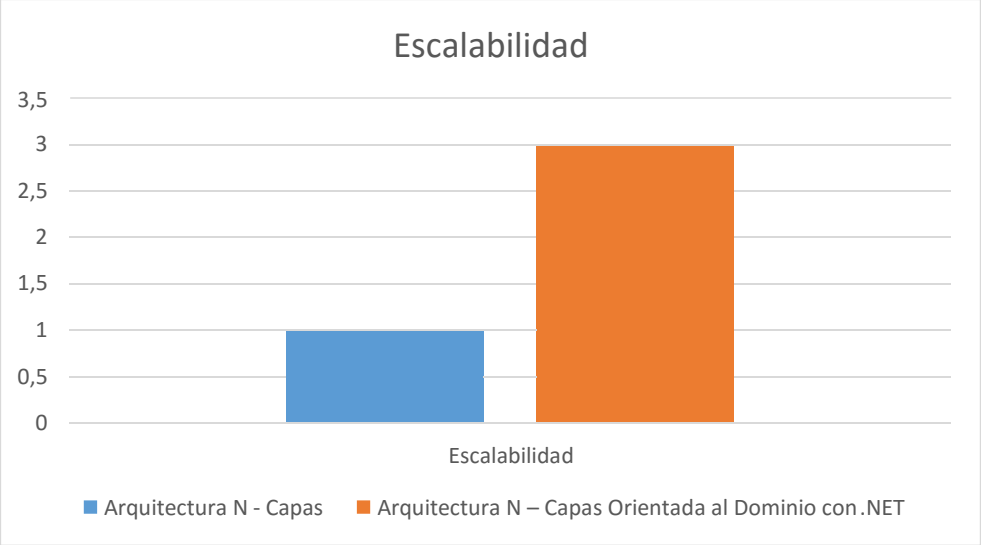
Tabla 4 Aspecto 3: Escalabilidad.

	Arquitectura N - Capas	Arquitectura N – Capas Orientada al Dominio con .NET
Escalabilidad	1	3
Total	**1**	**3**
Porcentaje	**33,3 %**	**100 %**

Fuente: *Eddy Navarrete/Diego Ortiz*

La Arquitectura N – Capas Orientada al Dominio con .NET se caracterizan por tener una vida relativamente larga y un volumen de cambios evolutivos considerable. Por lo tanto, en estas aplicaciones es muy importante todo lo relativo al mantenimiento de la aplicación, la facilidad de actualización, o la sustitución de tecnologías y frameworks/ORMs (Object relational mapping) por otras versiones más modernas o incluso por otros diferentes, etc. El objetivo es que todo esto se pueda realizar con el menor impacto posible sobre el resto de la aplicación. En definitiva, que los cambios de tecnologías de infraestructura de una aplicación no afecten a capas de alto nivel de la aplicación, especialmente, que afecten lo mínimo posible a la capa del "Dominio de la aplicación". Mientras que la Arquitectura N – Capas no cuenta con estas características anteriormente descritas. (Llorente, Zorrilla Castro, Calvarro Nelson, & Ramos Barroso, 2010)

Gráfico 12 Cuadro estadístico de indicadores de Escalabilidad.



Fuente: *Eddy Navarrete/Diego Ortiz*

Análisis

En la Arquitectura N – Capas la Escalabilidad no cumple con esta característica.

En la Arquitectura N – Capas Orientada al Dominio con .NET la Escalabilidad cumple completamente con esta característica.

Interpretación

La Arquitectura N – Capas tiene un nivel bajo en cuanto a la Escalabilidad, debido a que esta arquitectura tiende a tener un alto nivel de acoplamiento y un bajo nivel de cohesión debido a su estructura como tal. (Llorente, Zorrilla Castro, Calvarro Nelson, & Ramos Barroso, 2010)

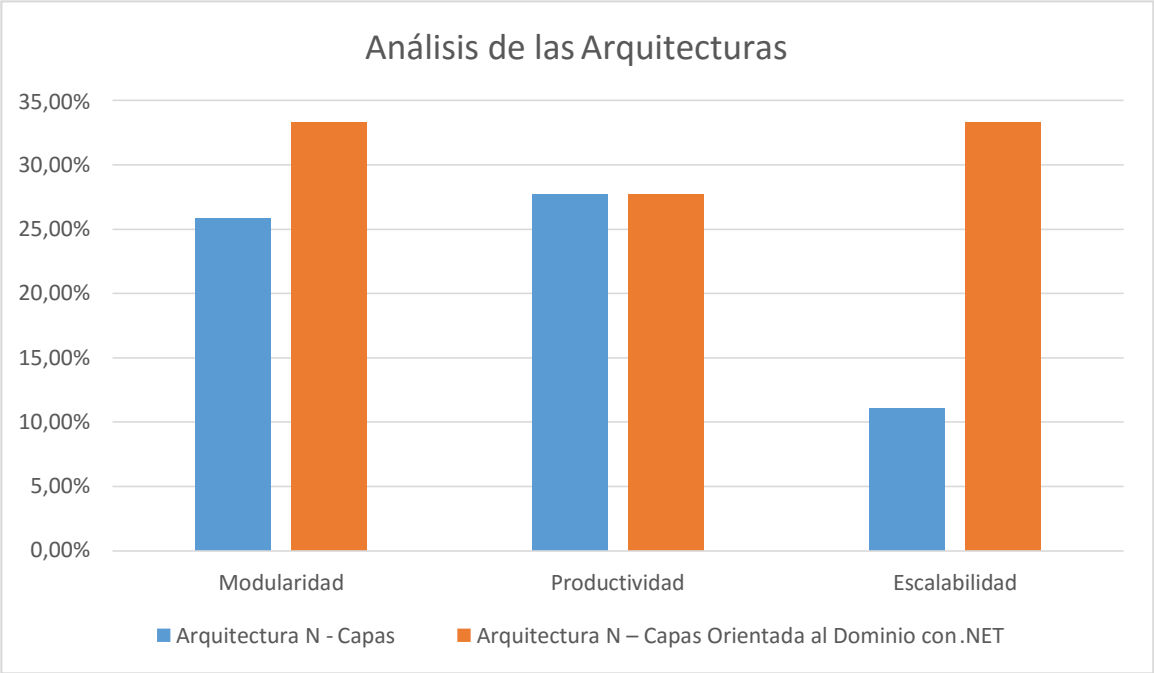
La Arquitectura N – Capas Orientada al Dominio con .NET tiene un alto nivel en cuanto a la Escalabilidad, debido a que las características de esta arquitectura presentan un nivel bajo de acoplamiento y un nivel alto de cohesión, debido a que el código se encuentra mejor distribuido en las capas y centraliza las funciones principales en la capa de Dominio. (Llorente, Zorrilla Castro, Calvarro Nelson, & Ramos Barroso, 2010)

Tabla 5 Análisis de las Arquitecturas N – Capas y N – Capas Orientada al Dominio con .NET.

Arquitectura	Modularidad	Productividad	Escalabilidad	Total
Arquitectura N - Capas	25,87 %	27,74 %	11,08 %	**64,69 %**
Arquitectura N – Capas Orientada al Dominio con .NET	33,33 %	27,74 %	33,33 %	**94,40 %**

Fuente: *Eddy Navarrete/Diego Ortiz*

Gráfico 13 Cuadro estadístico de los aspectos entre las Arquitecturas N – Capas y N – Capas orientada al Dominio con .NET.



Fuente: *Eddy Navarrete/Diego Ortiz*

Gracias al estudio comparativo entre estas dos arquitecturas se ha logrado determinar que la Arquitectura N – Capas Orientada al Dominio con .NET es un 94,40% más adaptable en relación con la Arquitectura N – Capas, partiendo de los parámetros de Modularidad y Escalabilidad se pudo evidenciar las fortalezas de esta arquitectura.

Se concluye que la Arquitectura N – Capas Orientada al Dominio con .NET se adapta mucho mejor a otros sistemas informáticos. (Por: Los autores)

3.1.2. Metodología de Investigación

Tabla 6. Metodología Research.

CRITERIO	DETALLE
FOCALIZACIÓN	• Analizar las Arquitecturas N - Capas y N – Capas Orientada al Dominio con .NET • Determinar parámetros de comparación y herramientas que permita seleccionar la mejor

	arquitectura para la gestión de procesos e interacción con otras aplicaciones. • Implementar en el departamento de Evaluación y Acreditación de la UNACH, en base a la Arquitectura seleccionada en la investigación.
ESTRATEGIA DE BÚSQUEDA:	Área: Software, Sub área: Programación **Propósito de la búsqueda**: Caracterizar los aspectos de la Arquitectura N - Capas y de la Arquitectura N – Capas Orientada al Dominio con .NET
FUENTES DE INFORMACIÓN	Enciclopedias, revistas científicas, actas de conferencias, libros, repositorios de Universidades y sociedades académicas, Sitios web de empresas de tecnología.
MOTORES DE BÚSQUEDA	Google scholar, IEEE
CRITERIOS DE BÚSQUEDA	• allintitle: "Arquitectura de Software" • definitions of "Software Architecture" • allintitle: "Arquitectura N Capas" • Arquitectura N Capas • Patrón de arquitectura n-capas con orientación al dominio del año 2016 • allintitle: "n Capas" • Patrón de arquitectura n-capas con orientación al dominio • Arquitectura n-capas con orientación al dominio
CRITERIOS DE SELECCIÓN	• Arquitectura de Software • Arquitectura de Software N – Capas • Arquitectura de Software N – Capas Orientado al dominio con .NET
CRITERIOS DE EXCLUSIÓN	• No Incluir patentes • No Incluir citas • Journals & Magazines

	• Intervalo de años 2010 - 2012 • Resultados mayores a cinco años.
CRITERIOS DE EVALUACIÓN DE CONTENIDO	Exactitud, objetividad, cobertura, vigencia, relevancia en función de los objetivos de la investigación.
ANÁLISIS DE INFORMACIÓN	Realiza un análisis comparativo de las arquitecturas en función de los objetivos, variables e indicadores, seleccionamos la más adecuada de acuerdo a los requerimientos y se aplica en un caso de estudio.

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 7 Resultado de la Metodología Research.

Metodología Research								
Tema	**Motor de Búsqueda**	**Query**	**Resultados**	**Repetidos**	**Validos**	**No Validos**	**Intervalo de tiempo**	**Restricciones**
Arquitectura de Software	https://scholar.google.com	allintitle: "Arquitectura de Software"	42	3	11	28	2011 -2016	No Incluir patentes No Incluir citas
Arquitectura de Software	http://ieeexplore.ieee.org/	definitions of "Software Architecture"	13	0	1	12	2010 - 2015	Journals & Magazines
Arquitectura N - Capas	https://scholar.google.com	allintitle: "Arquitectura N Capas"	1	0	1	0	2011 - 2016	No Incluir patentes No Incluir citas
Arquitectura N - Capas	http://ieeexplore.ieee.org/	Arquitectura N Capas	2	0	0	2	2010 - 2012	Journals & Magazines

Arquitectura de Software y N - Capas	https://scholar.google.com	patrón de arquitectura n-capas con orientación al dominio del año 2016	9	0	1	8	2011 - 2016	No Incluir patentes No Incluir citas
N - Capas	https://scholar.google.com	allintitle: "n Capas"	4	0	1	3	2010 - 2016	No Incluir patentes No Incluir citas
Patrón de arquitectura n-capas con orientación al dominio	https://scholar.google.com	Patrón de arquitectura n-capas con orientación al dominio	7	0	1	6	2016	No Aplica
Patrón de arquitectura n-capas con orientación al dominio	http://ieeexplore.ieee.org/	Patrón de arquitectura n-capas con orientación al dominio	14	0	3	11	2007 - 2016	No Aplica
Arquitectura n-capas con orientación al dominio	https://scholar.google.com	Arquitectura n-capas con orientación al dominio	63	0	1	62	2010 -2012	No Incluir patentes No Incluir citas
		TOTAL	**155**	**3**	**20**	**132**		

Fuente: *Eddy Navarrete/Diego Ortiz*

3.2. OPERACIONALIZACIÓN DE VARIABLES

3.2.1. HIPÓTESIS

El análisis comparativo de modelos permitirá demostrar que el modelo de Arquitectura N – Capas orientada al dominio con .NET es más adaptable que el modelo de Arquitectura de N – Capas, en la implementación del Sistema de Plan de Fortalecimiento y Mejoras de Acreditación de la UNACH.

3.2.2. IDENTIFICACIÓN DE VARIABLES

Variable Independiente.

- Los modelos de Arquitectura de Capas vs la Arquitectura N – Capas Orientada al Dominio con .NET

Variable Dependiente.

- Adaptabilidad del sistema de gestión de información

Tabla 8. Operacionalización de Variables.

VARIABLE	TIPO	DEFINICIÓN CONCEPTUAL	DIMENSIÓN	INDICADORES
Los modelos de Arquitectura de Capas vs la Arquitectura N – Capas Orientada al Dominio con .NET	Independiente	Definir un análisis comparativo general entre las dos arquitecturas, nos permitirá la elección de la arquitectura que se adecue a los requerimientos de la aplicación. Definiremos un análisis específico de acuerdo a la infraestructura de los sistemas que actualmente está en operación en el departamento de Evaluación y Acreditación de la UNACH, que nos permitirá implementar la arquitectura adecuada para que esta sea escalable, modulable, cohesivo y acoplable.	• Evaluación • Comparación	• Escalabilidad • Modularidad • Productividad

Adaptabilidad del sistema de gestión de información	Dependiente	Aplicando la arquitectura adecuada para desarrollar se obtendrá una aplicación que interactuara y se adaptara con otros sistemas, con el mínimo de recursos disponibles y tiempo	• Seguridad. • Rendimiento.	• Numero de Vulnerabilidades • La media de tiempo invertido por una petición. • La mediana de tiempo invertido por una petición: significa que el 50% de las muestras tardaron menos del valor reflejado. • El tanto por ciento de respuestas con error. • El rendimiento en Kb/segundo: igual que la anterior pero con cantidad de datos en lugar de peticiones

Fuente: *Eddy Navarrete/Diego Ortiz*

CAPÍTULO IV

4. RESULTADOS

Para el análisis estadístico de los indicadores de la variable dependiente se van utilizar las herramientas SQL Injection y Apache JMeter, para lo cual se realizan tres mediciones, un antes sin implementar el módulo de Plan de Mejoras y dos después de haber implementado el módulo de Plan de mejoras. El antes se mide con una aplicación desarrollada en N – Capas que se encuentre en producción y el después se mide en el mismo aplicativo que se encuentra desarrollado en N – Capas y también en el módulo de Plan de Mejoras y Fortalecimiento de Carreras de la UNACH.

Esta investigación se aplica en el mes de Febrero del 2017 en el Departamento de Evaluación y Acreditación de la UNACH.

Al momento de ejecutar las pruebas con la herramienta Apache JMeter (Anexo 2), se identifica los siguientes parámetros en:

Gráfico de Resultados

- **Datos:** muestra los valores actuales de los datos.
- **Media:** representa la Media.
- **Mediana:** dibuja la Mediana.
- **Desviación:** muestra la Desviación Estándar (una medida de la Variación).
- **Rendimiento:** representa el número de muestras por unidad de tiempo.

Reporte Resumen

- **Label:** El nombre de la muestra (conjunto de muestras).
- **# Muestras:** El número de muestras para cada URL.
- **Media:** El tiempo medio transcurrido para un conjunto de resultados. (Para nuestro estudio la Media del tiempo invertido por una petición ideal será de 200ms)
- **Mín:** El mínimo tiempo transcurrido para las muestras de la URL dada.
- **Máx:** El máximo tiempo transcurrido para las muestras de la URL dada.
- **Error %:** Porcentaje de las peticiones con errores.

- **Rendimiento:** Rendimiento medido en base a peticiones por segundo/minuto/hora.
- **Kb/sec:** Rendimiento medido en Kilobytes por segundo.
- **Avg. Bytes:** Tamaño medio de la respuesta de la muestra medido en bytes (erróneamente, en JMeter 2.2 muestra el valor en kB).

Con esta herramienta se pudo evidenciar los diferentes resultados que tanto para el Sistema DEA sin el módulo de Plan de Mejoras como también para el Sistema DEA con el módulo de Plan de Mejoras arrojo diferentes valores, tomando en cuenta los diferentes casos en los cuales se realizaron las pruebas, es así que se tomó las pruebas con lo siguiente:

- 10 usuarios en 30 segundos
- 60 usuarios en 180 segundos
- 120 usuarios en 360 segundos

4.1. Indicador 1: Rendimiento

Tabla 9. Índices de Rendimiento

Índice	Porcentaje
Media de tiempo invertido por una petición.	20 %
Mediana de tiempo invertido por una petición: significa que el 50% de las muestras tardaron menos del valor reflejado.	20 %
Tanto por ciento de respuestas con error.	20 %
El rendimiento en Kb/segundo: igual que la anterior pero con cantidad de datos en lugar de peticiones	20 %

Fuente: *Eddy Navarrete/Diego Ortiz*

Para evaluar el rendimiento entre la Arquitectura N – Capas y la Arquitectura N – Capas Orientada al dominio con .Net, se ha tomado en cuenta cinco índices mostrados en la Tabla 9.

A estos índices se le asignado un porcentaje del 20 %, los cuales demuestran las peticiones realizadas al servidor y como este responden en un tiempo determinado.

Se ha realizado las mediciones con 120 usuarios en un máximo de tiempo de 360 segundos ya que la aplicación realizada esta diseñada para una alta concurrencia de usuarios cuyo rendimiento no se vea afectado.

4.2. Indicador 2: Seguridad

Tabla 10. Índices de Seguridad

Índice		Porcentaje
Injection SQL	Vulnerables (0 – 99%)	100 %
	No Vulnerables (100%)	

Fuente: *Eddy Navarrete/Diego Ortiz*

Para evaluar la seguridad entre las Arquitecturas N – Capas y la Arquitectura N – Capas Orientada al dominio con .Net, se ha visto necesario ejecutar sentencias de Inyección SQL para medir las diferentes vulnerabilidades que poseen tanto la Arquitectura N – capas en el Sistema DEA y la Arquitectura N – Capas Orientada al dominio con .Net acoplada al Sistema DEA.

4.3. Comparación y valoración entre la Arquitectura N – Capas y la Arquitectura N – capas Orientada al Dominio con .NET, en base al Rendimiento.

Tabla 11. Medición de los Índices de Rendimiento

Nª	Usuarios	Segundos	Indicador	Sistema DEA (sin el modulo Plan de Mejoras)			Sistema DEA (con el modulo Plan de Mejoras) Arquitectura N – Capas Orientada al Dominio con .Net		
1	120	360s	Media de tiempo invertido por una petición.	451 ms	44.35 %	8.87 %	301 ms	66.45 %	13.29 %

			Mediana de tiempo invertido por una petición. (El tiempo ideal de referencia es de 150 ms.)	539 ms	27.93	5.59%	191 ms	78.53 %	15.71 %
			Tanto por ciento de respuestas con error.	0%	100%	20 %	0 %	100%	20%
			El rendimiento en Kb/segundo: igual que la anterior pero con cantidad de datos en lugar de peticiones (el rendimiento ideal de referencia es de 100kb/s)	3.21 kb/s	3.21 %	0.64 %	3.82 kb/s	3.82 %	0.75 %
TOTAL (100%)				**35.1%**			**49.75%**		

Fuente: *Eddy Navarrete/Diego Ortiz*

4.4. Comparación y valoración entre la Arquitectura N – Capas y la Arquitectura N – capas Orientada al Dominio con .NET, en base a la Seguridad

Tabla 12. Medición de los Índices de Seguridad

Parámetro	Sistema DEA (sin el modulo Plan de Mejoras)	Sistema DEA (con el modulo Plan de Mejoras) Arquitectura N – Capas Orientada al Dominio con .Net
Injection SQL	100 %	100%
TOTAL (100%)	**100%**	**100%**

Fuente: *Eddy Navarrete/Diego Ortiz*

4.5. Cuadro Comparativo entre la Arquitectura N – Capas y la Arquitectura N – Capas Orientada al Dominio con .NET

Tabla 13. Resultados de Rendimiento y Seguridad

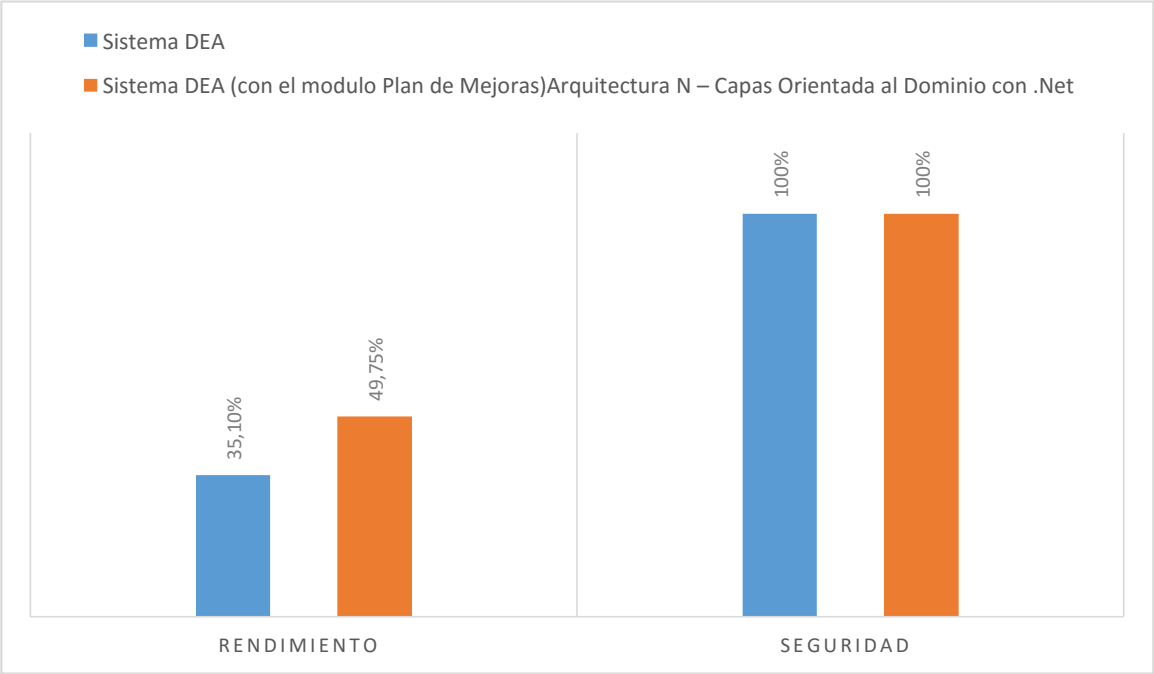
Experimentos		Arquitectura N - Capas	Arquitectura N – Capas Orientada al dominio con .Net
Indicadores	**Rendimiento**	35.1%	49.75%
	Seguridad	100%	100%

Fuente: *Eddy Navarrete/Diego Ortiz*

4.6. Análisis de los resultados

Los datos obtenidos mediante la experimentación, indican en cuento al Rendimiento la Arquitectura N – Capas Orientada al Dominio con .Net posee un 49.75% en la Arquitectura N – Capas que posee un valor de 35.1 %, por lo tanto, el Rendimiento en la Arquitectura N – Capas Orientada al Dominio con .Net es más alto que el de la Arquitectura N – Capas, por lo que el sistema en condiciones particulares de trabajo responde de manera adecuada y eficaz. En cuanto a la seguridad tanto la Arquitecturas N – Capas y la Arquitectura N – Capas Orientada al dominio con .Net poseen un porcentaje del 100% lo que quiere decir se no son vulnerables ante posibles ataques.

Gráfico 14. Cuadro estadístico de Rendimiento y Seguridad entre las Arquitecturas N – Capas y Arquitectura N – Capas orientada al Dominio con .NET



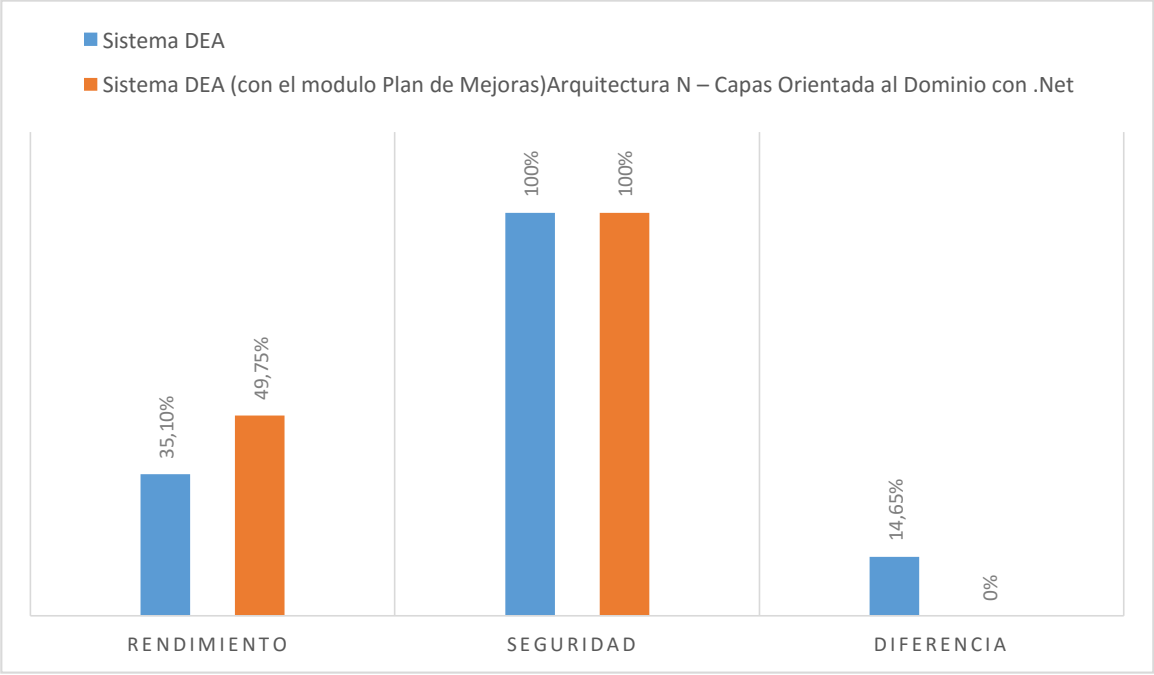
Fuente: *Eddy Navarrete/Diego Ortiz*

Así después de las comparaciones valorizadas se tiene como resultado que en Rendimiento la Arquitectura N – Capas (Sistema DEA) tiene un porcentaje de 35.10% mientras él la Arquitectura N – capas orientada al Dominio con .Net (Sistema DEA con el módulo Plan de Mejoras) posee un porcentaje de 49.75%, de la misma manera en cuanto a la Seguridad tanto la

Arquitectura N – Capas (Sistema DEA) y la Arquitectura N – capas orientada al Dominio con .Net (Sistema DEA con el módulo Plan de Mejoras) tiene un porcentaje del 100%.

Se procede a realizar el cálculo matemático para identificar cual es la diferencia de valores entre Rendimiento y Seguridad.

Gráfico 15. Cuadro estadístico de resultados diferenciales entre las Arquitecturas N – Capas y N – Capas orientada al Dominio con .NET.



Fuente: *Eddy Navarrete/Diego Ortiz*

4.7. Comprobación de la Hipótesis

A través de los resultados obtenidos y con el manejo de la herramienta Apache JMeter y SQL Injection, se puede precisar que la Arquitectura N – Capas Orientada al Dominio con .Net incide en un 14.65% más con respecto al Rendimiento que la Arquitectura N – Capas. Por lo tanto, se puede deliberar que la Arquitectura N – Capas Orientada al Dominio con .Net es la que mejor al momento de desarrollar la aplicación **Planes de Mejoras** ya que se adapta a los diferentes Sistemas y con la que se va a tener un mayor Rendimiento y de igual forma en cuanto a la Seguridad tanto la Arquitectura N – Capas como la Arquitectura N – Capas Orientada al Dominio con .Net no poseen un número mayor de vulnerabilidades.

CAPÍTULO V

5. PROPUESTA

5.1 Titulo de la propuesta

Desarrollo del Sistema de Plan de Fortalecimiento y Mejoras de Acreditación de la UNACH aplicando la Arquitectura N – Capas Orientada al Dominio con .NET

5.2 Introducción

Determinado el análisis entre la Arquitectura Capas vs. Arquitectura N – Capas Orientada al Dominio con .NET, se desarrolla el módulo de Administración, Evaluación y Seguimiento de los Planes de Mejoras Institucionales y Fortalecimiento de carreras de la UNACH, específicamente de la Carrera de Medicina, la cual tendrá una estructura basada en la Arquitectura N – Capas Orientada al Dominio con .NET,

5.2.1. Elaboración de los Planes de Fortalecimiento para Carreras

El Plan de Fortalecimiento para carreras debe ser elaborado con el propósito fundamental de elevar los niveles de calidad de las carreras de la IES (Instituto de Educación Superior); éste debe contemplar un plazo de ejecución en el periodo de un año, con la posibilidad de que al finalizar el mismo, la IES pueda presentar un nuevo plan para un segundo año; además, es importante recalcar que la IES debe contemplar acciones de mejora, que no correspondan a actividades cotidianas de su quehacer universitario.

Adicionalmente se deben considerar otros insumos importantes como el Informe de Autoevaluación presentado, el Plan de Mejoras/Plan de Fortalecimiento Institucional, documentos de la carrera y otros documentos y directrices institucionales que la IES considere importantes para mejorar la calidad de sus carreras.

5.2.1.1 Estructura del Plan de Fortalecimiento para carreras

El Plan de Fortalecimiento para carreras tiene la siguiente estructura:

Documento del Plan. - En el cual se exponen todos los elementos del plan como: antecedentes, filosofía institucional de la IES, información general y específica de la carrera, análisis del informe de evaluación de la carrera emitido por el CEAACES, definición de estrategias y la relación de las mismas con los objetivos institucionales.

La IES podrá establecer objetivos propios que no necesariamente estén ligados a los indicadores del modelo, pero que ayuden a la mejora continua de la carrera y de la institución.

Plan de acción. - Corresponde al detalle de las acciones que implementará la IES en la carrera para mejorar la calidad de manera progresiva e integral, el que se traduce en el cronograma de trabajo. A partir de las estrategias propuestas en el apartado anterior (documento del plan), se deberá establecer actividades, períodos de ejecución, responsable, presupuesto y los medios de verificación.

Estructura de Seguimiento Interno.- Corresponde al diseño de los mecanismos para llevar a cabo un riguroso proceso de análisis autocrítico y un diálogo reflexivo sobre la totalidad de las actividades de una carrera, con amplia participación de sus integrantes, a fin de superar los obstáculos existentes y considerar los logros alcanzados, para mejorar la eficiencia y calidad académica.

5.2.2 Elementos del Documento del Plan

El plan de fortalecimiento para carreras debe contemplar los siguientes elementos:

5.2.2.1 Antecedentes

En esta sección se realiza una breve presentación de la institución y de la carrera para la cual se está planteando el plan de fortalecimiento. Se deberá hacer referencia al proceso de evaluación de carreras y la normativa que soporta la elaboración y presentación del plan en mención.

5.2.2.2 Filosofía institucional de la IES

En esta sección se debe presentar la filosofía institucional (misión, visión, valores y/o principios institucionales), que corresponde a los propósitos declarados por la IES, así también deberá incluir los propósitos de la carrera.

5.2.2.3 Información de la Carrera

En esta sección la IES debe presentar información general y específica de la carrera: nombre completo de la carrera, título que otorga, duración, tipo de sede donde se imparte la carrera, si es el caso el nombre de la extensión, objetivo general y específicos de la carrera, perfil de ingreso y egreso de la misma, información del equipo coordinador de la

carrera, así como de la persona responsable del Plan que presentará la IES y que será el nexo con el CEAACES.

Gráfico 16. Información de Carrera

Datos Generales de la Carrera		
Nombre completo de la carrerra:		
Título que otorga la carrera:		
Duración de la Carrera (incluido proceso de titulación)	Número de Créditos:	
	Número de Semestres:	
Tipo de sede en que se imparte la carrera	Matriz: ☐	
	Extensión: ☐	
Nombre de la extensión donde se imparte la carrerra (si aplica)		

Datos Específicos de la Carrera
Objetivo General: -
Objetivos Específicos: - -
Perfil de ingreso de la carrera:
Perfil de egreso de la carrera:

Datos Administrativos de la Carrera (equipo coordinador de la carrera o su equivalente)			
Nombres y apellidos:	Correo electrónico:	Teléfono convencional de contacto (ext)	Celular

Datos Responsable del Plan de Fortalecimiento para la Carrera			
Nombres y apellidos:	Correo electrónico:	Teléfono convencional de contacto(ext)	Celular

Fuente: *CEAACES*

5.2.2.4 Análisis de Resultados de la Evaluación de la Carrera

En este apartado, la IES debe realizar un análisis profundo de los resultados obtenidos por cada indicador en el proceso de evaluación de la carrera identificando los factores que influyeron en los resultados obtenidos, sean estos internos (fortalezas, debilidades) o externos (oportunidades, amenazas). Es importante que el análisis que desarrolla la IES responda a condiciones reales de la carrera, puesto que este se constituye en la base para la definición de las estrategias; así también debe incluir las metas que la IES pretende alcanzar durante el período de ejecución del Plan de Fortalecimiento.

La IES, dentro de este apartado, podrá establecer objetivos que no necesariamente estén relacionados con los indicadores del modelo, pero que le permitan mejorar continuamente sus actividades académicas e institucionales, y por cada objetivo propuesto deberá establecer su meta correspondiente.

5.2.2.5 Definición de Estrategias

En esta sección se deben definir la(s) estrategia(s) que impulsarán y permitirán alcanzar las metas propuestas que han sido definidas en el punto anterior. La información obtenida del análisis de los resultados de evaluación es fundamental en esta fase, puesto que las estrategias podrán apoyarse en los factores anteriormente identificados (utilización de las fortalezas, aprovechar las oportunidades, eliminar o disminuir debilidades y amenazas que estén afectando al bajo desempeño en los indicadores del modelo de evaluación del CEAACES).

Las mismas deben ser concebidas para ejecutarse durante los 12 meses de duración del Plan de Fortalecimiento de la Carrera (que podría extenderse a 24 meses).

De las estrategias definidas se desprenderá el plan de acción; por lo tanto, su enfoque debe permitir alcanzar mejoras progresivas e integrales en los indicadores evaluados por el CEAACES.

Es fundamental que la IES además de trabajar en los indicadores del modelo de evaluación del CEAACES en los cuales tiene un bajo desempeño, no descuide aquellos que alcanzaron resultados aceptables u óptimos, diseñando estrategias que garanticen su mejora continua.

Las estrategias que se definan deben apoyar la consistencia interna y externa; deben ser actividades que impacten considerablemente a la calidad de la educación y de los profesionales que está formando y deben proyectar a la carrera mucho más allá del cumplimiento base (modelo CEAACES).

Gráfico 17. Matriz de Análisis Situacional

Indicador del modelo de evaluación	Valor óptimo del indicador	Calificación obtenida en la evaluación del CEAACES	Factores que influyeron en los resultados de evaluación de la Carrera	Meta propuesta	Estrategias

Fuente: *CEAACES*

Adicionalmente, la IES puede proponer estrategias que pese a no estar directamente relacionadas con la mejora en los indicadores del modelo de evaluación de la carrera definido por el CEAACES, contribuyen en el mejoramiento de la calidad de la educación de la carrera. Esta información puede ser presentada en el siguiente formato.

Gráfico 18. Estrategias Adicionales

Objetivo	Meta propuesta	Estrategias

Fuente: *CEAACES*

5.2.2.6 Relación Estrategias - Objetivos Institucionales

La IES debe analizar y realizar un alineamiento de las estrategias planteadas en el punto anterior con los Objetivos Institucionales de la IES, a los cuales contribuye con la ejecución de la estrategia; esto con la finalidad de identificar de qué forma el mejoramiento de las carreras, contribuye al mejoramiento institucional.

Con esta herramienta se pretende que la IES ejecute acciones que vayan más allá de los criterios que analiza el CEAACES y que las acciones a implementar se alineen con el enfoque estratégico de la IES.

Dicha relación se debe establecer en el formato que se presenta a continuación:

Gráfico 19. Relación Estrategias – Objetivos Institucionales

Posibles casos	Estrategias	Objetivos Institucionales
Caso 1: Relación una estrategia con un solo objetivo	1.	1.
Caso 2: Relación una estrategia con dos o más objetivos	1.	1. 2.
Caso 3: Relación dos o más estrategias con uno o más objetivos	1. 2.	1. 2.

Fuente: *CEAACES*

Es importante que la IES considere los posibles casos expuestos en la matriz anterior al momento de alinear las estrategias con los indicadores del modelo de evaluación del CEAACES y los objetivos institucionales.

La IES debe considerar que los objetivos institucionales contemplan a su matriz, extensiones y sedes.

5.2.3 Elementos del Plan de Acción

El Plan de Fortalecimiento para las Carreras contemplará un plan de acción que deberá contener OBLIGATORIAMENTE al menos los siguientes elementos:

5.2.3.1 Indicadores del modelo de evaluación de la Carrera (CEAACES)

Se deben colocar al menos los indicadores del modelo de evaluación de carreras utilizado por el CEAACES.

5.2.3.2 Desempeño obtenido en la evaluación CEAACES

Por indicador del modelo de evaluación del entorno de aprendizaje del CEAACES, se debe incluir el resultado obtenido en el informe de evaluación del entorno de aprendizaje de la carrera.

5.2.3.3 Meta Propuesta

En función del análisis desarrollado por la IES, se debe incluir el resultado por indicador que se compromete alcanzar la carrera dentro del periodo máximo de ejecución del plan.

5.2.3.4 Estrategias

Para este punto, se debe trasladar al plan de acción, las estrategias definidas mediante el análisis de resultados de la evaluación de la carrera, en el documento del plan. Vale señalar que en lo concerniente al modelo de evaluación del CEAACES su enfoque debe basarse en cerrar las brechas identificadas en el informe de evaluación de la carrera; sin embargo, es recomendable que la IES maneje un planteamiento más ambicioso con miras hacia el mejoramiento continuo de la carrera.

5.2.3.5 Actividades

Corresponden al conjunto de acciones o tareas a implementar y que se desprenden de las estrategias definidas en el documento del plan. Las actividades deben estar contempladas para un período de un año. Las actividades propuestas deben ser entendibles y verificables.

5.2.3.6 Cronograma

Fechas de ejecución: Cada actividad debe tener una fecha de inicio y una fecha de finalización. Los plazos establecidos deben corresponder a la duración racional de la actividad, evitando contemplar plazos exagerados para actividades simples o plazos muy cortos para actividades complejas. También debe considerarse un plazo adecuado debido a imprevistos que pudieran surgir. El formato de fecha señalará únicamente mes y año (mm/año).

5.2.3.7 Responsable

Cada actividad o tarea debe tener un responsable para su ejecución. Las personas responsables no necesariamente son quienes ejecutan las actividades, pero si quienes garantizan su cumplimiento dentro de los plazos establecidos, así como la eficacia y eficiencia de las acciones implementadas. Los responsables deben proveer las evidencias de realización (medios de verificación).

Se debe señalar el nombre del cargo, más no el nombre de la persona.

5.2.3.8 Presupuesto

Es el valor monetario destinado para la ejecución de las tareas o actividades que serán implementadas. El presupuesto establecido debe ser realista, obedecer a un proceso de

análisis responsable. La IES deberá verificar que el presupuesto esté aprobado por la autoridad competente de la institución.

5.2.3.9 Medios de Verificación

Corresponde a un detalle de los instrumentos o medios a través de los cuales se acreditará el cumplimiento de las actividades y objetivos establecidos. Los medios de verificación proporcionan al evaluador evidencia de cumplimiento. Se debe cuidar que los medios de verificación sean adecuados y suficientes, y que permitan verificar el cumplimiento de las actividades.

5.3. Objetivos

5.3.1. Objetivo General

Desarrollar un Aplicativo para Administración, Evaluación y Seguimiento del Plan de Fortalecimiento y Mejoras de Acreditación de la UNACH aplicando la Arquitectura N – Capas Orientada al Dominio con .NET

5.3.2. Objetivos Específicos:

- Implementar en el Departamento de Evaluación y Acreditación de la UNACH, en base a la Arquitectura seleccionada en la investigación.
- Medir los indicadores de seguridad y rendimiento en el aplicativo desarrollado

5.4. Fundamentación Científico -Técnica

En la actualidad, la automatización de los sistemas en los diferentes departamentos constituye un desafío fundamental que busca optimizar aún más el proceso productivo. Este trabajo se enmarca dentro de un proyecto de implantación e implementación cuyo principal objetivo es la automatización de los procesos del Plan de Fortalecimiento y Mejoras en la Acreditación de la Carrera de Medicina de la Facultad de Ciencias de la Salud de la UNACH.

Es por esto que el desarrollo de Software informático está teniendo gran acogida en la actualidad, dentro de este proceso de desarrollo se tiene las metodologías agiles que son muy flexibles para afrontar cambios en la planificación del desarrollo de software y promover el trabajo en equipo, por estas razones se propone desarrollar el software con la metodología SCRUM.

5.5 Descripción de la propuesta

5.5.1. Ámbito

El proyecto de software se centra en el desarrollo de un sistema que permita la automatización de los procesos en el Plan de Fortalecimiento y Mejoras en la Acreditación de la Carrera de Medicina de la Facultad de Ciencias de la Salud de la UNACH.

Sin olvidar que se trata de un desarrollo específicamente informático pero lleva implícito la creación de ciertos procedimientos administrativos.

5.5.2. Visión General

5.5.2.1. Datos Generales

Nombre de la Institución: Universidad Nacional de Chimborazo

Beneficiario: Carrera de Medicina

Técnico o Encargado: (Director de carrera)

Ubicación: UNACH Campus Norte "Ms. Edison Riera R." Avda. Antonio José de Sucre, Km. 1 1/2 Vía a Guano, Facultad de Ciencias de la Salud.

5.5.2.2 Ingeniería de Requerimientos

En este punto se detallarán los requisitos funcionales y no funcionales establecidos en coordinación con el usuario y el equipo de desarrollo.

Tabla 14. Listado de Requerimientos

ID	DESCRIPCIÓN	TIPO
1	El sistema permitirá ingresar Planes de Mejoras	ENTRADA
2	El sistema permitirá modificar los Planes de Mejoras	ENTRADA
3	El sistema permitirá eliminar los Planes de Mejoras	ENTRADA
4	El sistema permitirá consultar los Planes de Mejoras	CONSULTA
5	El sistema permitirá ingresar Estrategias	ENTRADA

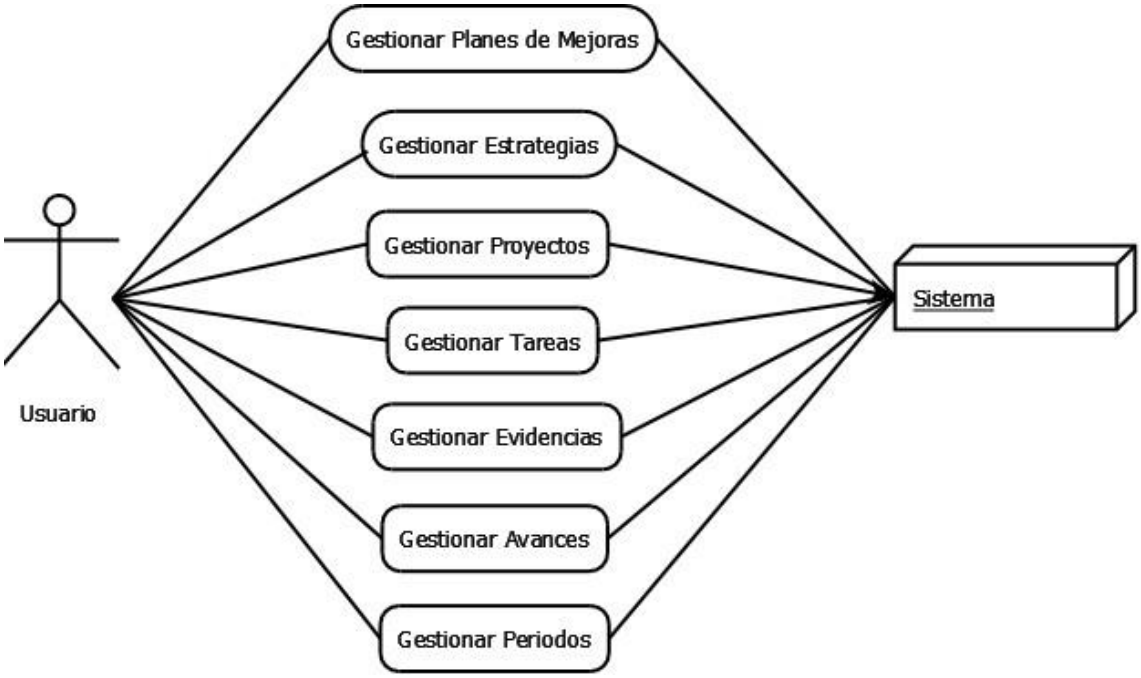
6	El sistema permitirá modificar las Estrategias	ENTRADA
7	El sistema permitirá eliminar las Estrategias	ENTRADA
8	El sistema podrá consultar las diferentes Estrategias que existen.	CONSULTA
9	El sistema permitirá ingresar Proyectos	ENTRADA
10	El sistema permitirá modificar los Proyectos	ENTRADA
11	El sistema permitirá eliminar los Proyectos	ENTRADA
12	El sistema podrá consultar los diferentes Proyectos que existen.	CONSULTA
13	El sistema permitirá ingresar Tareas	ENTRADA
14	El sistema permitirá modificar las Tareas	ENTRADA
15	El sistema permitirá eliminar las Tareas	ENTRADA
16	El sistema podrá consultar las diferentes Tareas que existen y su capacidad en cada una de ellas.	CONSULTA
17	El sistema permitirá ingresar Evidencias a las Tareas.	ENTRADA
18	El sistema permitirá modificar las Evidencias.	ENTRADA
19	El sistema permitirá eliminar las Evidencias.	ENTRADA
20	El sistema podrá consultar las Evidencias.	CONSULTA
21	El sistema podrá ingresar Archivos en las Evidencias	ENTRADA
22	El sistema permitirá ingresar datos de los Avances de las Tareas.	ENTRADA
23	El sistema permitirá modificar los Avances de las Tareas.	ENTRADA
24	El sistema permitirá eliminar Avances a las Tareas.	ENTRADA
25	El sistema podrá consultar los diferentes Avances que existen a cada una de las Tareas.	CONSULTA
26	El sistema podrá consultar las diferentes Planes, Proyectos por periodos académicos.	CONSULTA
27	El sistema podrá consultar a las personas responsables de cada uno de los proyectos.	CONSULTA

28	El sistema tendrá la capacidad de consultar las Tareas que se están ejecutándose y sus Avances	CONSULTA

Fuente: *Eddy Navarrete/Diego Ortiz*

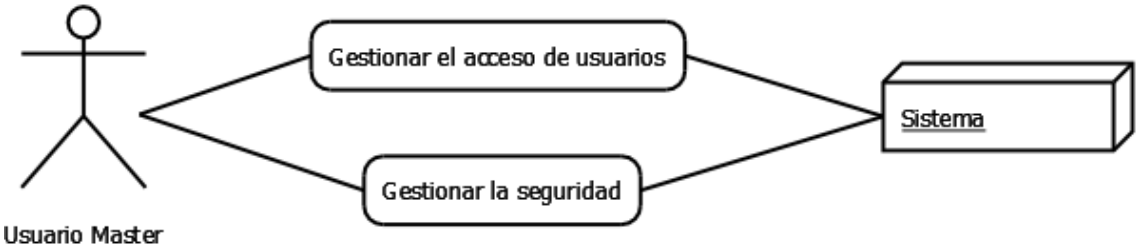
5.5.2.3. Diagramas de Caso de Uso

Gráfico 20. Funciones del Usuario en el Sistema



Fuente: *Eddy Navarrete/Diego Ortiz*

Gráfico 21. Funciones del Usuario Master en el Sistema

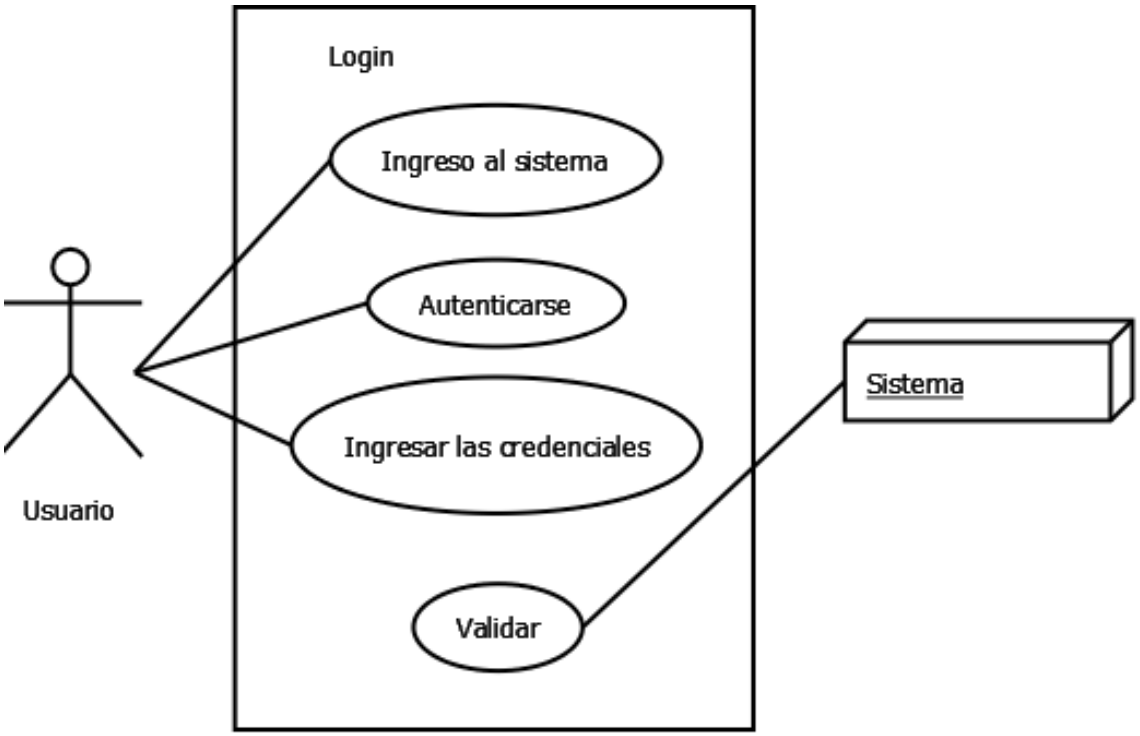


Fuente: *Eddy Navarrete/Diego Ortiz*

Gráfico 22. Funciones del Usuario Master en el Sistema

Fuente: *Eddy Navarrete/Diego Ortiz*

Gráfico 23. Funciones del Usuario para acceder al Sistema



Fuente: *Eddy Navarrete/Diego Ortiz*

5.5.3. Roles

Tabla 15: Roles

Miembro	Grupo	Roles XP	Metodología
Diego Ortiz	Tesista	Programador	XP
Eddy Navarrete	Tesista	Programador	
Ing. Gonzalo Allauca	Tutor	Director	

Fuente: *Eddy Navarrete/Diego Ortiz*

5.5.4. Plan General del Proyecto

La planificación de este proyecto se basará en la metodología ágil XP, detallaremos las iteraciones asignándoles un tiempo de duración y una fecha de inicio que es el 11 de abril de 2016.

Se va a desarrollar los módulos de:

- Plan de Fortalecimiento y Mejoras de la Acreditación de la Carrera de medicina de la UNACH.

Para su desarrollo se utilizará el programa Microsoft Project el cual ayudará a llevar un orden cronológico sobre el desarrollo del sistema de acuerdo a la metodología ágil XP.

5.5.5. Historias de Usuarios

Una historia de usuario es una representación de un requisito de software escrito en una o dos frases utilizando el lenguaje común del usuario. Son requeridas en las metodologías de desarrollo ágiles para la especificación de requisitos.

Se utiliza para estimar tiempos de desarrollo de la parte de la aplicación que describen, así como el verificar si el programa cumple con los requisitos del usuario.

Tabla 16: Historias de Usuario

N ª	Nombre	Prioridad	Riesgo	Esfuerzo	Iteración
1	Análisis y Diseño de la Base de Datos	Alta	Alto	Alto	1
2	Gestión de Acceso de Usuarios	Alta	Alto	Medio	2
3	Gestión de la Seguridad	Alta	Medio	Medio	2

4	Gestión de Planes de Mejoras	Media	Medio	Medio	2
5	Gestión de Estrategias	Media	Medio	Medio	2
6	Gestión de Proyectos	Alta	Alto	Alto	2
7	Gestión de Tareas	Media	Medio	Medio	2
8	Gestión de Evidencias	Media	Medio	Medio	2
9	Gestión de Avances	Media	Medio	Medio	2
10	Emisión de Reportes y Consultas	Alta	Alto	Medio	3

Fuente: *Eddy Navarrete/Diego Ortiz*

5.5.6. Desarrollo de las Iteraciones

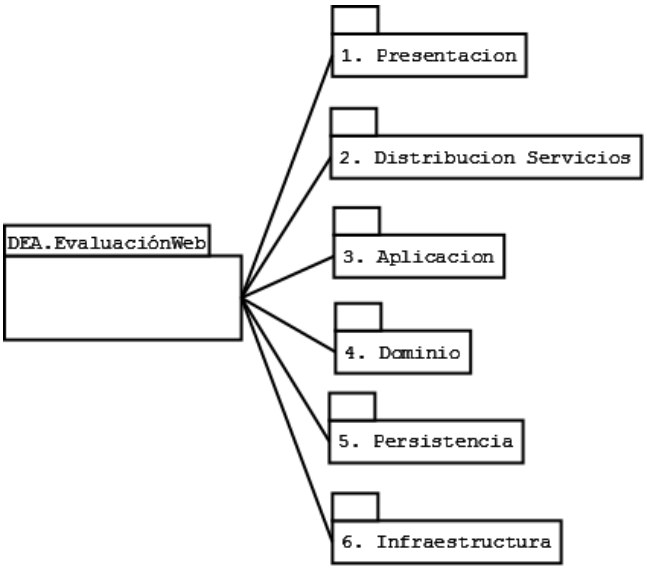
5.5.6.1. Iteración I

En esta fase se desarrollaron las diferentes metáforas del sistema que son:

Diseño de la Arquitectura del Sistema.

Para definir la arquitectura del sistema a desarrollar, se lo representa mediante el Diagrama Arquitectura N – Capas con Orientación al Dominio de .NET.

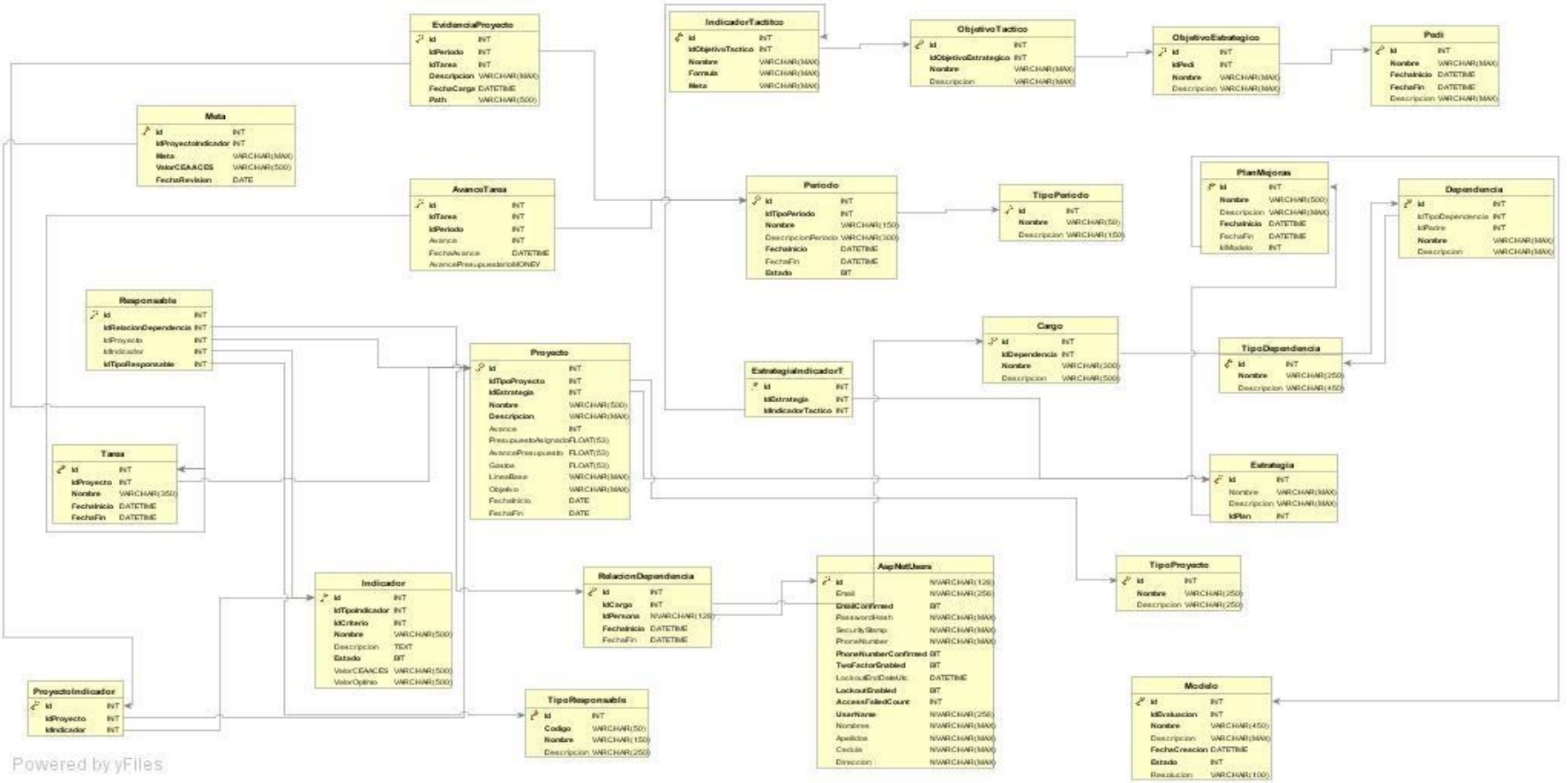
Gráfico 24. Arquitectura N – Capas con Orientación al Dominio de .NET



Fuente: *Eddy Navarrete/Diego Ortiz*

Diseño de la Base de Datos.

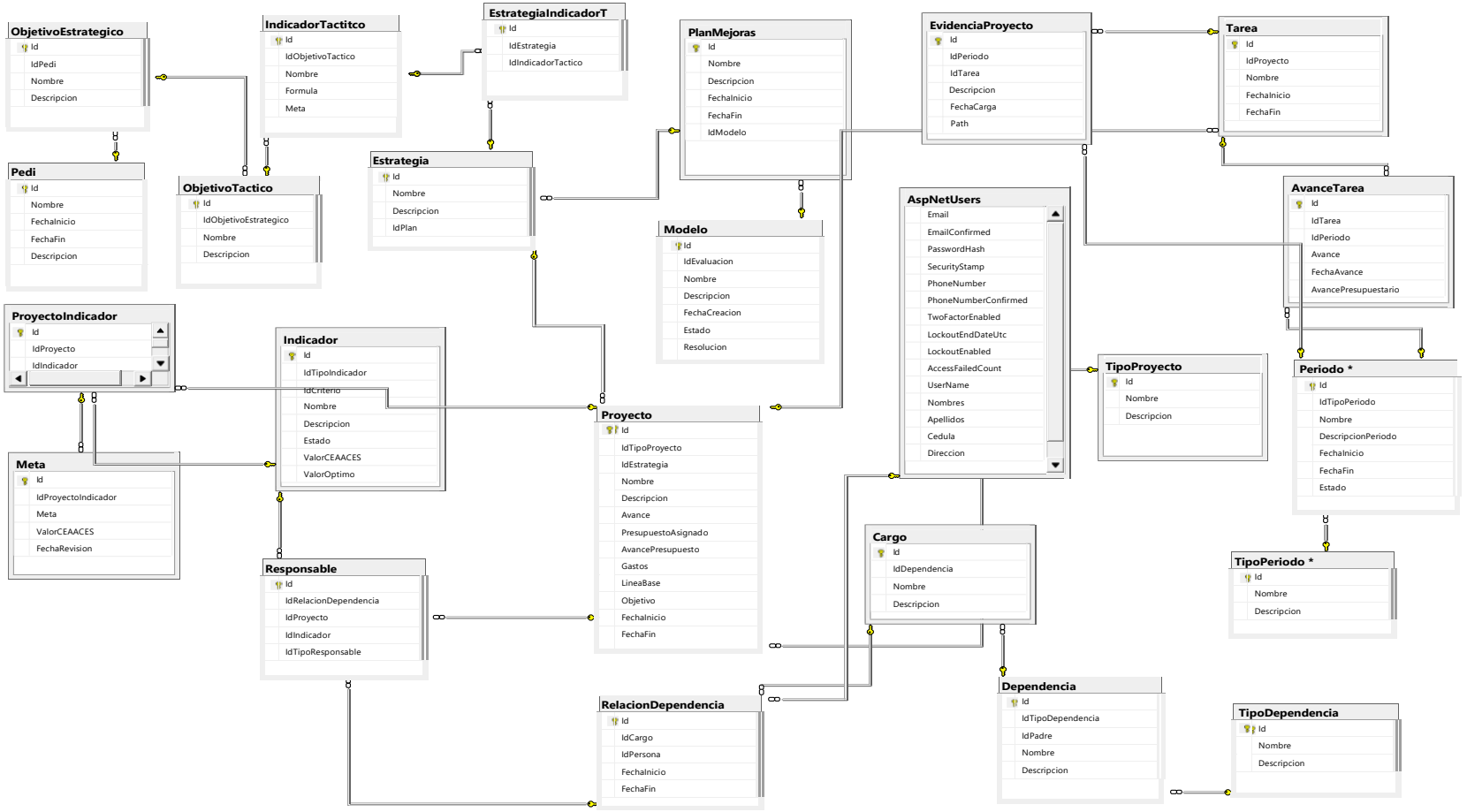
Gráfico 25. Diseño Lógico Diagrama Entidad – Relación



Fuente: *Eddy Navarrete/Diego Ortiz*

Diseño Físico: Se implementó la base de datos en el DBMS SQL Server

Gráfico 26. Diseño Físico de la Base de Datos



Fuente: *Eddy Navarrete/Diego Ortiz*

Diccionario de Datos: A continuación se enlistará el diccionario de datos.

Tabla 17. Tabla Proyecto

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	int	No	Código principal del proyecto, autonumérico
2	IdTipoProyecto	int	No	Código del tipo de Proyecto que puede ser: Gasto Corriente y de Inversión. FK: Tabla Tipo de Proyecto
3	IdEstrategia	int	No	Código de la Estrategia asignada al proyecto. FK: Tabla Estrategia
4	Nombre	varchar(500)	No	Nombre del Proyecto
5	Descripción	varchar(MAX)	No	Descripción del Proyecto
6	AvanceFisico	int	No	Avance del proyecto
7	PresupuestoAsignado	float	No	Presupuesto asignado al proyecto
8	AvancePresupuestario	float	No	Avance del presupuesto asignado al proyecto
9	LineaBase	varchar(MAX)	No	Línea Base del Proyecto.
10	Objetivo	varchar(MAX)	No	Objetivo principal del proyecto
11	FechaInicio	datetime	No	Fecha de inicio del Proyecto
12	FechaFin	datetime	No	Fecha de culminación del proyecto

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 18. TipoProyecto

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	Int	No	Código principal del Tipo de Proyecto, autonumérico
2	Nombre	varchar(500)	No	Nombre del Tipo de Proyecto
3	Descripción	varchar(MAX)	No	Descripción del Tipo de Proyecto

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 19. PlanMejora

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	Int	No	Código principal del Plan de Mejoras, autonumérico
2	Nombre	varchar(500)	No	Nombre del plan de Mejora
3	Descripción	varchar(MAX)	No	Descripción del Plan de Mejora
4	FechaInicio	datetime	No	Fecha de inicio del plan de mejora
5	FechaFin	datetime	No	Fecha fin del plan de mejora
6	IdModelo	Int	No	Código del Modelo

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 20. Tarea

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	Int	No	Código principal de la Tarea, autonumérico
2	IdProyecto	Int	No	Código del Proyecto. FK: Tabla Proyecto
3	Nombre	varchar(500)	No	Nombre de la Tarea
4	FechaInicio	datetime	No	Fecha de inicio de la Tarea
5	FechaFin	datetime	No	Fecha fin de la Tarea

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 21. Periodo

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	int	No	Código principal del Periodo, autonumérico
2	IdTipoPeriodo	int	No	Código del Tipo de Periodo, este puede ser: Anual, Semestral y Plan de Mejoras FK: Tabla TipoPeriodo
3	Nombre	varchar(500)	No	Nombre del periodo
4	DescripcionPeriodo	varchar(MAX)	No	Descripción del Periodo
5	FechaInicio	datetime	No	Fecha de Inicio del Periodo
6	FechaFin	datetime	No	Fecha Fin del Periodo
7	Estado	bit	No	El estado muestra si está vigente o no el periodo con 0 y 1.

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 22. EvidenciaPM

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	int	No	Código principal de EvidenciaPM, autonumérico
2	IdPeriodo	int	No	Código del Periodo. FK: Periodo
3	IdTarea	int	No	Código de la Tarea. FK: Tarea
4	Descripcion	varchar(MAX)	No	Descripción de la EvidenciaPM
5	FechaCarga	datetime	No	Fecha de la carga de las evidencias
6	Path	varchar(500)	No	Es la ruta de subida de la evidencia

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 23. AvanceTarea

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	int	No	Código principal del Avance de la Tarea, autonumérico
2	IdTarea	int	No	Código de la Tarea. FK: Tabla Tarea
3	IdPeriodo	int	No	Código del Periodo. FK: Tabla Periodo
4	Avance	int	No	Avance de la Tarea dentro de un Periodo
5	AvancePresupuestario	float	No	Avance presupuestario de la Tarea.

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 24. TipoPeriodo

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	int	No	Código principal del Tipo de Periodo, autonumérico
2	Nombre	varchar(500)	No	Nombre del Tipo de Periodo
3	Descripción	varchar(MAX)	No	Descripción del Tipo de Periodo

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 25. TipoDependencia

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	int	No	Código principal del Tipo de Dependencia, autonumérico
2	Nombre	varchar(500)	No	Nombre del Tipo de Dependencia
3	Descripción	varchar(MAX)	No	Descripción del Tipo de Dependencia

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 26. Indicador

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	int	No	Código principal del Indicador, autonumérico
2	IdTipoIndicador	int	No	Código del Tipo de Indicador. FK: Tabla TipoIndicador
3	IdCriterio	int	No	Código del Criterio. FK: Tabla Criterio
4	Nombre	varchar(500)	No	Nombre del Indicador
5	Descripcion	varchar(MAX)	No	Descripción del Indicador.
6	Estado	Int	No	Estado del Indicador

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 27. Dependencia

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	Int	No	Código principal de Dependencia, autonumérico
2	IdTipoDependencia	Int	No	Código del Tipo de dependencia. FK: Tabla TipoDependencia
3	IdPadre	Int	No	Código del Padre, para tener un orden jerárquico de datos.
4	Nombre	varchar(500)	No	Nombre de la dependencia.
5	Descripcion	varchar(MAX)	No	Descripción de la Dependencia.

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 28. Cargo

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	Int	No	Código principal del Cargo, autonumérico
2	IdDependencia	Int	No	Código de dependencia. FK: Tabla Dependencia.
3	Nombre	varchar(500)	No	Nombre del Cargo.
4	Descripcion	varchar(MAX)	No	Descripción del Cargo.

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 29. ProyectoIndicador

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	int	No	Código principal del Proyecto Indicador, autonumérico
2	IdProyecto	int	No	Código del Proyecto. FK: Tabla Proyecto
3	IdIndicador	int	No	Código del Indicador.

				FK: Tabla Indicador.
4	ValorCeaaces	int	No	Valor numérico que el Ceaaces establece
5	Meta	int	No	Valor numérico al que se desea llegar.

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 30. Responsable

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	int	No	Código principal del Responsable, autonumérico
2	IdRelacionDependencia	int	No	Código de Relación Dependencia. FK: Tabla RelacionDependencia
3	IdProyecto	int	No	Código del Proyecto. FK: Tabla Proyecto.
4	IdIndicador	int	No	Código del Indicador. FK: Tabla Indicador.
5	IdTipoResponsable	int	No	Código del Tipo Responsable. FK: Tabla TipoResponsable.

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 31. Estrategia

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	int	No	Código principal de Estrategia, autonumérico
2	IdPlan	int	No	Código del Plan. FK: Tabla PlanMejora
3	Nombre	varchar(MAX)	No	Nombre de la Estrategia
4	Descripcion	varchar(MAX)	Si	Descripción de la Estrategia

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 32. Pedi

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	int	No	Código principal del Pedi, autonumérico
2	Nombre	varchar(MAX)	No	Nombre del Pedi
3	FechaInicio	datetime	No	Fecha de Inicio del Pedi
4	FechaFin	datetime	No	Fecha Fin del Pedi
5	Descripcion	varchar(MAX)	Si	Descripción del Pedi

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 33. ObjetivoEstrategico

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	int	No	Código principal del ObjetivoEstrategico, autonumérico
2	IdPedi	int	No	Código del Pedi. FK: Tabla Pedi
3	Nombre	varchar(MAX)	No	Nombre del objetivo Estratégico
4	Descripcion	varchar(MAX)	Si	Descripción del objetivo Estratégico

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 34. ObjetivoTactico

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	int	No	Código principal del ObjetivoTactico, autonumérico
2	IdObjetivoEstrategico	int	No	Código del ObjetivoEstrategico. FK: Tabla ObjetivoEstrategico

3	Nombre	varchar(MAX)	No	Nombre del Objetivo Táctico
4	Descripcion	varchar(MAX)	Si	Descripción del Objetivo Táctico

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 35. IndicadorTactico

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	int	No	Código principal del IndicadorTactico, autonumérico
2	IdObjetivoTactico	int	No	Código del ObjetivoTactico. FK: Tabla ObjetivoTactico
3	Nombre	varchar(MAX)	No	Nombre del IndicadorTactico
4	Formula	varchar(MAX)	No	Formula del IndicadorTactico
5	Meta	varchar(MAX)	No	Meta del IndicadorTactico

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 36. EstrategiaIndicadorT

ítem	Nombre	Tipo	Nulos	Descripción
1	Id	int	No	Código principal de EstrategiaIndicadorT, autonumérico
2	IdEstrategia	int	No	Código de la Estrategia. FK: Tabla Estrategia
3	IdIndicadorTactico	int	No	Código del IndicadorTactico. FK: Tabla IndicadorTactico

Fuente: *Eddy Navarrete/Diego Ortiz*

Estándar de Codificación.

Se implementará el estándar de codificación CamelCase porque es el más sencillo y conocido ya que utiliza reglas muy simples y entendibles.

Este tipo de escritura, el CamelCase, se caracteriza porque las palabras van unidas entre sí sin espacios; con la peculiaridad de que las primeras letras de cada término se encuentra en mayúscula para hacer más legible el conjunto.

Admite dos posibles combinaciones entre mayúsculas y minúsculas:

- UpperCamelCase, cuando la primera letra de cada una de las palabras es mayúscula. Ejemplo: EjemploDeUpperCamelCase.
- lowerCamelCase, igual que la anterior con la excepción de que la primera letra es minúscula. Ejemplo: ejemploDeLowerCamelCase.

- **Diseño de las Interfaces de Usuario.**

5.5.6.2. Iteración II

Tabla 37. Historia de Usuario Gestión de Acceso de Usuario

HISTORIA DE USUARIO	
Número: 1	**Usuario:** Administrador Master, Usuario DEA
Nombre Historia: Gestión de Acceso de Usuarios	
Prioridad: Alta	**Riesgo en desarrollo:** Alto
Esfuerzo: Medio	**Interacción asignada:** 2
Programador responsable: Diego Ortiz / Eddy Navarrete	
Descripción: Antes de poder interactuar con el sistema se requiere las credenciales (usuario y contraseña) para poder acceder al sistema.	
Observaciones: Usuario DEA	

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 38. Historia de Usuario Gestión de la Seguridad

HISTORIA DE USUARIO	
Número: 2	**Usuario:** Administrador Master
Nombre Historia: Gestión de la Seguridad	
Prioridad: Alta	**Riesgo en desarrollo:** Alto
Esfuerzo: Medio	**Interacción asignada:** 2
Programador responsable: Diego Ortiz / Eddy Navarrete	
Descripción: El Administrador Master una vez que ingrese al sistema podrá crear un nuevo usuario (Usuario DEA) y asignar privilegios.	
Observaciones: El Administrador Master tendrá usuarios registrados en la Base de Datos y podrá Gestionar los mismos.	

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 39. Historia de Usuario Gestión de Mejoras

HISTORIA DE USUARIO	
Número: 3	**Usuario:** Usuario DEA
Nombre Historia: Gestión de Planes de Mejoras	
Prioridad: Alta	**Riesgo en desarrollo:** Medio
Esfuerzo: Medio	**Interacción asignada:** 2
Programador responsable: Diego Ortiz / Eddy Navarrete	
Descripción: El Usuario DEA una vez que haya ingresado al sistema, podrá añadir un nuevo Plan de Mejora y almacenarlo en la Base de Datos.	
Observaciones: El Usuario DEA podrá gestionar los Planes de Mejoras de la Base de Datos.	

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 40. Historia de Usuario Gestión de Estrategia

HISTORIA DE USUARIO	
Número: 4	**Usuario:** Usuario DEA
Nombre Historia: Gestión de Estrategias	
Prioridad: Medio	**Riesgo en desarrollo:** Medio
Esfuerzo: Medio	**Interacción asignada:** 2
Programador responsable: Diego Ortiz / Eddy Navarrete	
Descripción: El Usuario DEA una vez que haya ingresado al sistema, podrá añadir una nueva Estrategia y almacenarla en la Base de Datos.	
Observaciones: El Usuario DEA podrá gestionar las Estrategias de la Base de Datos.	

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 41. Historia de Usuario Gestión de Proyectos

HISTORIA DE USUARIO	
Número: 5	**Usuario:** Usuario DEA
Nombre Historia: Gestión de Proyectos	
Prioridad: Alta	**Riesgo en desarrollo:** Alto
Esfuerzo: Alta	**Interacción asignada:** 2
Programador responsable: Diego Ortiz / Eddy Navarrete	
Descripción: El Usuario DEA una vez que haya ingresado al sistema, podrá añadir un nuevo Proyecto y almacenarlo en la Base de Datos.	
Observaciones: El Usuario DEA podrá gestionar los Proyectos de la Base de Datos.	

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 42. Historia de Usuario Gestión de Tareas

HISTORIA DE USUARIO	
Número: 6	**Usuario:** Usuario DEA
Nombre Historia: Gestión de Tareas	
Prioridad: Media	**Riesgo en desarrollo:** Medio
Esfuerzo: Medio	**Interacción asignada:** 2

Programador responsable: Diego Ortiz / Eddy Navarrete
Descripción: El Usuario DEA una vez que haya ingresado al sistema, podrá añadir una nueva Tarea y almacenarla en la Base de Datos.
Observaciones: El Usuario DEA podrá gestionar las Tareas de la Base de Datos.

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 43. Historia de Usuario Gestión de Evidencias

HISTORIA DE USUARIO	
Número: 7	**Usuario:** Usuario DEA
Nombre Historia: Gestión de Evidencias	
Prioridad: Media	**Riesgo en desarrollo:** Medio
Esfuerzo: Medio	**Interacción asignada:** 2
Programador responsable: Diego Ortiz / Eddy Navarrete	
Descripción: El Usuario DEA una vez que haya ingresado al sistema, podrá añadir una nueva Evidencia y almacenarla en la Base de Datos.	
Observaciones: El Usuario DEA podrá gestionar las Evidencias de la Base de Datos.	

Fuente: *Eddy Navarrete/Diego Ortiz*

Tabla 44. Historia de Usuario Gestión de Avances

HISTORIA DE USUARIO	
Número: 8	**Usuario:** Usuario DEA
Nombre Historia: Gestión de Avances	
Prioridad: Media	**Riesgo en desarrollo:** Medio
Esfuerzo: Medio	**Interacción asignada:** 2
Programador responsable: Diego Ortiz / Eddy Navarrete	
Descripción: El Usuario DEA una vez que haya ingresado al sistema, podrá añadir un nuevo Avance y almacenarlo en la Base de Datos.	
Observaciones: El Usuario DEA podrá gestionar los Avances de la Base de Datos.	

Fuente: *Eddy Navarrete/Diego Ortiz*

5.5.6.3. Iteración III

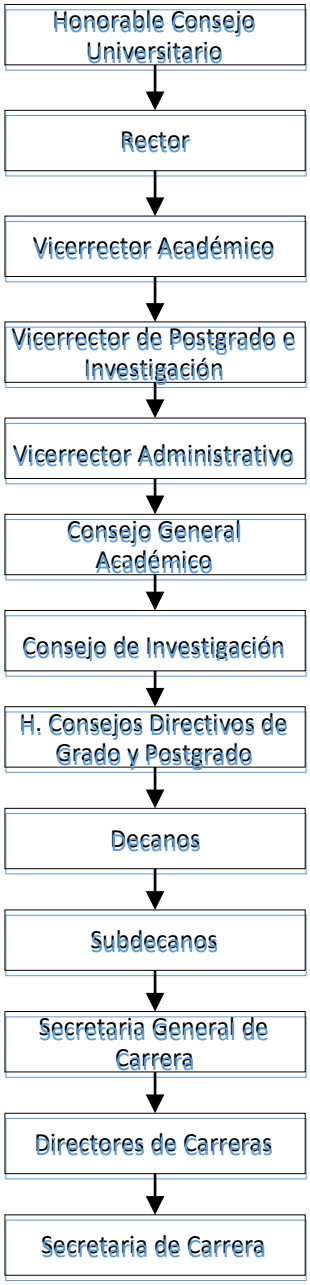
Tabla 45. Historia de usuario Emisión de reportes y Consultas

HISTORIA DE USUARIO	
Número: 9	**Usuario:** Usuario DEA
Nombre Historia: Emisión de Reportes y Consultas	
Prioridad: Alta	**Riesgo en desarrollo:** Medio
Esfuerzo: Alto	**Interacción asignada:** 3
Programador responsable: Diego Ortiz / Eddy Navarrete	
Descripción: El Usuario DEA podrá imprimir un reporte y su historial de acuerdo a la actividad que seleccione.	
Observaciones: El Usuario DEA podrá gestionar los Avances de la Base de Datos.	

Fuente: *Eddy Navarrete/Diego Ortiz*

5.6 Diseño Organizacional.

Gráfico 27. Diseño Organizacional



Fuente: *Eddy Navarrete/Diego Ortiz*

5.7 Monitoreo y Evaluación de la propuesta

El monitoreo y la evaluación es por excelencia el instrumento que proporciona la información básica para facilitar la toma de decisiones, permite hacer un análisis de:

¿Cuáles son los problemas?, ¿Cómo se pueden enfrentar?. ¿Cuáles son los logros?, ¿Cómo se pueden consolidar?, ¿Cuál es el impacto de las actividades desarrolladas en la propuesta?

CONCLUSIONES

- La Arquitectura N – Capas incrementa el tráfico en la red cuando muchos clientes envían peticiones a un solo servidor, por lo contrario en la Arquitectura N – Capas Orientada al Dominio con .Net el tráfico en la red se mantuvo estable con la misma cantidad de clientes al hacer las peticiones a un mismo servidor.
- Gracias al desarrollo de la aplicación de Plan de Fortalecimiento y Mejoras de Acreditación de la UNACH, se ha logrado automatizar los procesos de Acreditación tanto Institucional como de Carreras, además permite la gestión de las evidencias y así alcanzar los índices de Acreditación impuestos por el CEAACES.
- Mediante la experimentación e investigación se observó la diferencia que existe en cuanto al rendimiento de la aplicación, en la Arquitectura N – Capas Orientada al Dominio con .Net el rendimiento es del 14.65% mayor que en la Arquitectura N – Capas, ya que en esta arquitectura concentra la lógica de negocio en el dominio.
- A través de la separación de la capa de dominio del resto de capas el diseño es mucho más limpio en cada capa, ayudando en el despliegue del sistema distribuido, permitiendo que diferentes capas sean situadas de forma flexible en diferentes servidores o clientes, de manera que se minimizó el exceso de comunicación y se mejoró el rendimiento.

RECOMENDACIONES

- Después de haber desarrollado el Sistema Plan de Fortalecimiento y Mejoras de Acreditación de la UNACH, sería interesante implementar y expandir sus funcionalidades, ya que la Arquitectura N – Capas Orientada al Dominio con .Net permite que las aplicaciones sean fáciles de escalar y sobre todo integrar nuevos módulos al sistema.
- En el desarrollo de aplicaciones empresariales complejas y de gran envergadura, la lógica de negocio es de vital importancia. La Arquitectura N – Capas Orientada al Dominio con .Net es una alternativa a ser implementada ya que ofrece importantes beneficios en cuanto a calidad, estabilidad y, sobre todo, facilita el mantenimiento futuro de las aplicaciones, debido al desacoplamiento entre sus componentes.
- En aplicaciones que estén sujetas a cambios continuos y evoluciones, se sugiere utilizar la Arquitectura N – Capas Orientada al Dominio con .Net, situando toda la lógica de negocio en una sola parte del software, disminuye drásticamente la cantidad de tiempo que se necesita para realizar dichos cambios, de estar forma disminuye las posibilidades de tener que realizar cambios en otras áreas de la aplicación.

BIBLIOGRAFÍA

Arias, Y. M., & Martinez, N. S. (2013). Estilo Arquitectonico para el sistema integrado de gestión Cedrux. *GECONTEC: revista Internacional de Gestión del Conocimiento y la Tecnología*, 3 - 13.

Arizabaleta Rodríguez, A. &. (2012). Especificación de una arquitectura empresarial de software utilizando el framework TOGAF.

Avalos Pérez, M. A. (2015). Implementación de un modelo alternativo de arquitectura de software para proyectar y construir sistemas lógicos de computadoras.

Avram, A., & Marinescu, F. (2006). *Domain Driven Design Quickly.* InfoQ.com Enterprise Software Development.

Bass, L., Clements, P., & Kazman, R. (2003). Software Architecture in Practice. *Addison-Wesley 2ª edicion.*

Bass, L., Clements, P., & Kazman, R. (2012). Software Architecture in Practice. *Addison Wesley, 3ª edicion.*

Brooks Jr, F. P. (1995). *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley .

Brooks, F. P., & Iverson, K. E. (1969). Automatic Data Processing. *New York. Wiley. ISBN:0471106054.*

C. S. Santos, D. M. (2015 12th). A Study of Test Techniques for Integration with Domain Driven Design. *Information Technology - New Generations (ITNG),* (págs. 373-378). Las Vegas.

Cardacci, D. G. (2015). Arquitectura de software académica para la comprensión del desarrollo de software en capas (No. 574). . *Universidad del CEMA.*

Contreras, M. R. (2013). ARQUITECTURA DE SOFTWARE PARA EL SERVICIO DE SOPORTE DE TECNOLOGÍA DE INFORMACIÓN BASADA EN SERVICIOS WEB. . *REVISTA COLOMBIANA DE TECNOLOGIAS DE AVANZADA (RCTA)*, 2(20).

De la Torre Llorente, C., Zorrila Castro, U., Calvarro, J., & Ángel, R. B. (2010). *Guía de Arquitectura N - Capas orientada al Dominio con .Net.* España: Krasis Consulting S. A.

De la Torre Llorente, C., Zorrilla Castro, U., Calvarro, J., & Ramos Barroso, Á. M. (2010). *Guía de Arquitectura N - Capas orientada al Dominio con .Net.* España: Krasis Consulting S. A.

Dijkstra, E. (1968). The Structure of the Multiprogramming System. *Communications of the ACM*.

Dustdar, S. &. (2014). *Service Engineering.* . Springer.

Escalante, L. C. (2016). *El patrón de arquitectura n-capas con orientación al dominio como solución en el diseño de aplicaciones empresariales.* Trujillo: Tecnología & Desarrollo.

Evans, E. (2006). *Domain Driver Design.* Pearson Education.

Galicia, C. (2013). Desarrollo de sistemas web con n-capas y metodologías ágiles en la universidad tecnológica de Tehuacán. . *de Cuerpos Académicos*, 166.

Gómez, S., & Moraleda, E. (2014). *Aproximación a la Ingeniría de Software.* Editorial Universitaria Ramon Areces.

IEEE. (2000). Recommended Practice for Architectural Description of Software-Intensive Systems. *IEEE Presss*.

Jiménez, L. B. (2012). Arquitectura de software para el producto “Mis Mejores Cuentos”. *Serie Científica*, 5(5).

Kruchten, P. (1995). The 4+1 View Model of Architecture. *IEEE Software, vol 12, Nº 6*.

Kruchten, P., Obbink, H., & Stafford, J. (Abril de 2006). The Past, Present, and Future for Software Architecture. *IEEE Software*.

Llorente, C. d., Zorrilla Castro, U., Calvarro Nelson, J., & Ramos Barroso, M. Á. (2010). *GUÍA DE ARQUITECTURA N-CAPAS ORIENTADA AL DOMINIO CON .NET 4.0.* Krasis Consulting, S. L.

Losavio, F., & Esteves, Y. (2016). Modelado del Negocio como Técnica Centrada en la Calidad del Software para el Análisis del Dominio del Aprendizaje Electrónico. *IV Simposio Científico y Tecnológico en Computación / SCTC 2016 / ISBN: 978-980-12-8407-9. Universidad Central de Venezuela, Caracas, Venezuela*.

Malavolta, P. L. (2013). What Industry Needs from Architectural Languages: A Survey. *IEEE Transactions on Software Engineering vol. 39, no. 6*, 869-891.

Martin, F. (2008). *Refactoring:Improving the Design of Existing Code (Object Technology Series).* ADDISON WESLEY LONGMAN INC DIV PEARSON SUITE 300.

Martínez, A. M. (s.f.). ARQUITECTURA DE SOFTWARE PARA EL DESARROLLO DE LABORATORIOS VIRTUALES.

Moquillaza Henriquez, S., Vega Huerta, H., & Guerra Grados, L. (2010). Programación en N capas. *Revista de investigación de Sistemas e Informática*, 7(2), 57-67.

Navasa, A. (2008). *Marco de trabajo para el desarrollo de arquitecturas software orientado a aspectos.* Cáceres.

Parnas, D. (1972). On the Criteria for Decomposing Systems into Modules. *Communications of the ACM*.

Peña, A. D. (2016). Arquitectura de Software para el sistema de visualización médica Vismedic. *Revista Cubana de Informática Médica*, 16(1).

Perry, D. E., & Wolf, A. L. (1992). Foundations for the Study of Software Architecture. *SOFTWARE ENGINEERING NOTES vol 17 no 4*.

Pressman, R. S. (2010). *Ingenieria de Software. Un enfoque práctico* . McGRAW-HILL INTERAMERICANA.

Rechtin, E. (1991). Systems Architecting: Creating and Building Complex Systems. *Prentice Hall*.

Rodriguez, H. (2014). Arquitectura de software OpenCNC.

Rojas, M. &. (2011). Una arquitectura de software para la integración de objetos de aprendizaje basada en servicios web.

Royce, W. E., & Royce, W. W. (1990). Software Architecture. Integrating Process and Technology. *TRW Queso, vol 14, nº 1*.

S. A. Soares, M. B. (2015). Dribbling complexity in model driven development using Naked Objects. *Computing Conference (CLEI)*, (págs. 1-11). Arequipa.

Salinas, E. C. (2011). Arquitectura orientada a servicios para software de apoyo para el proceso personal de software. *Ingeniare. Revista chilena de ingeniería*, 19(1), 40-52.

Sarasty España, H. F. (2016). Documentación y análisis de los principales frameworks de arquitectura de software en aplicaciones empresariales. *Doctoral dissertation, Facultad de Informática*.

Schneider, J.-G. (1999). Components, Scripts, and Glue: A conceptual framework for software composition. *Tesis Doctoral*.

Shaw, M. (1984). Abstraction Techniques in Modern Programming Languages. *IEEE Software, pp. 10-26.*

Shaw, M., & Garlan, D. (1994). Characteristics of Higher-Level Languages for Software Architecture. *Technical Report CMU-CS-94-210, Carnegie Mellon University.*

Shaw, M., & Garlan, D. (1996). *Software Architecture: perspectives on an emerging discipline.* Prentice Hall.

Vernon. (2013). *Implementing domain-driven design. .* Addison - Wesley .

Vernon, V. (2013). *Implementing domain-driven design.* Addison - Wesley.

Wirfs-brock, R. J. (2015). Driven to Discovering Your Design Values. *IEEE Software*, 9-11.

Wolf, A. (1997). Succeedings of the Second International Software Architecture Workshop (ISAW-2). *ACM SIGSOFT Software Engineering Notes*, 45-56.

Y. Karam, T. B. (2013). Support for self-optimized organizational patterns. *Systems Conference (SysCon)*, (págs. 235-243). Orlando.

ANEXOS

ANEXO 1

MARCO ADMINISTRATIVO

1.1. RECURSOS

1.1.1. Recursos Hardware

Nombre	Descripción	Disponibilidad
Portatil Dell Inspiron Core I7 Portatil Hp I7 Pavilion Dv7-6163us	Herramientas de desarrollo	100%
Impresora HP Laserjet Pro P1102w	Herramientas de utilidad	100%

1.1.2. Recursos Software

Nombre	Descripción
C# y HTML	Lenguajes de programación a utilizarse en la codificación del sistema
Visual Studio (Versión 2015)	Entorno de desarrollo integrado en el que se desarrollará y depurará las aplicaciones
SQL-Server	Es un sistema de gestión de base de datos objeto-relacional, que almacenará los datos.

1.2. PRESUPUESTO

Actividad	Cantidad	P/Unitario	TOTAL
Útiles de oficina	20	$ 0.50	$ 10
Hojas de papel bond	2000	$ 0.02	$ 40
Tinta de impresora	4	$ 30	$ 120

Copias	600	$ 0.03	$ 18
Anillados del proyecto	3	$ 5	$ 15
Empastados del Proyecto	10	$ 4	$ 40
Internet (horas)	300	$ 0.60	$ 180
Imprevistos (10 %)	-	-	$ 40.80
TOTAL			**$ 465.30**

1.3. CRONOGRAMA

Cronograma de Actividades

MES	1				2				3			
ACTIVIDADES	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
Análisis de requerimientos												
Entrevista con el cliente y el equipo de trabajo para la recolección de requerimientos.												
Entrevista con el cliente y el equipo de trabajo para la recolección de requerimientos												
Análisis de los requerimientos.												
Presentación del informe final de los requerimientos.												
Socialización de los requerimientos al equipo de desarrollo.												
Diseño												
Diseño de la interfaz												
Aprobación de la Interfaz												

Codificación												
Análisis e instalación del software a utilizar												
Desarrollo de la aplicación												
Pruebas y Despliegue												
Verificación y puesta en marcha de la aplicación												
Mantenimiento y Funcionamiento												
Confirmación del funcionamiento de la aplicación												
Cierre												
Ejecución de la aplicación												

ANEXO 2

MEDICIONES

Herramienta para la medición de indicadores

- Apache JMeter

El Apache JMeter es un software de código abierto, una aplicación diseñada totalmente en JAVA para medir el rendimiento y comportamiento de servidores mediante pruebas. Originalmente se diseñó para probar aplicaciones Web, pero se ha ampliado desde entonces a otras funciones.

Se utilizar para probar el rendimiento tanto de los recursos estáticos y dinámicos (archivos, Servlets, scripts de Perl, objetos Java, bases de datos - consultas, servidores FTP y mucho más). Se puede utilizar para simular una carga pesada en un servidor, la red o un objeto para poner a prueba su resistencia o para analizar el rendimiento global en diferentes tipos de carga. Puede usarlo para hacer un análisis gráfico de rendimiento o para probar el comportamiento de diferentes elementos con un gran volumen de carga y concurrencia.

Algunos de los tipos de Servidor que se pueden probar son:

- Web HTTP y HTTPS.
- SOAP.
- Base de datos a través de JDBC.
- **LDAP.**

Plan de Medición del Sistema

Para realizar las mediciones ocuparemos las siguientes herramientas que contiene JMeter

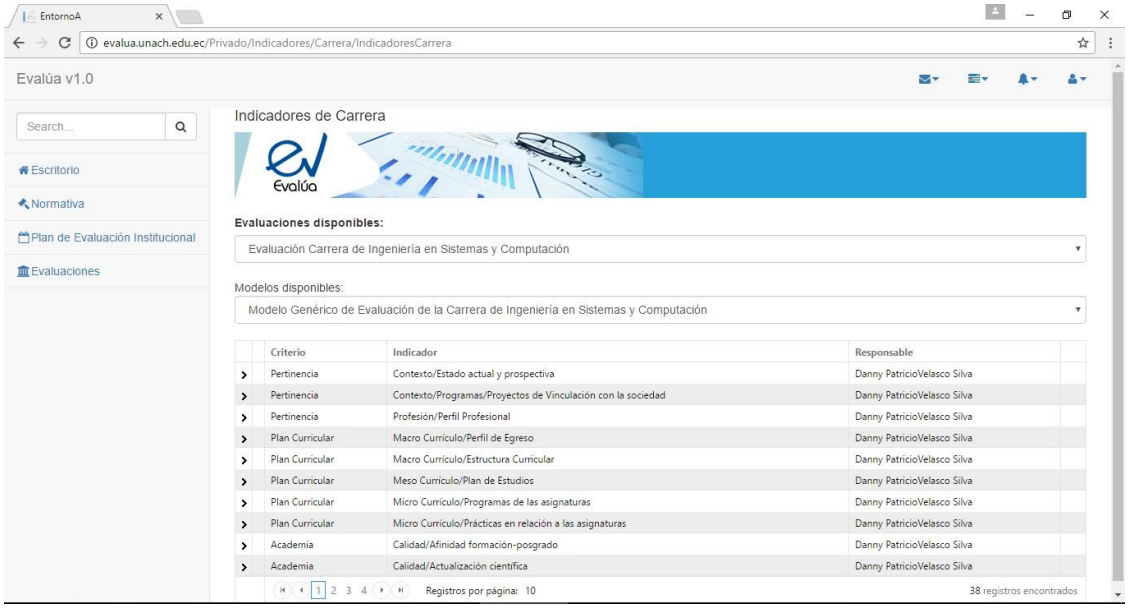
Thread Group (Grupo de hilos): Elemento que define el número de hilos (threads) de ejecución que definirán el número de usuarios a simular.

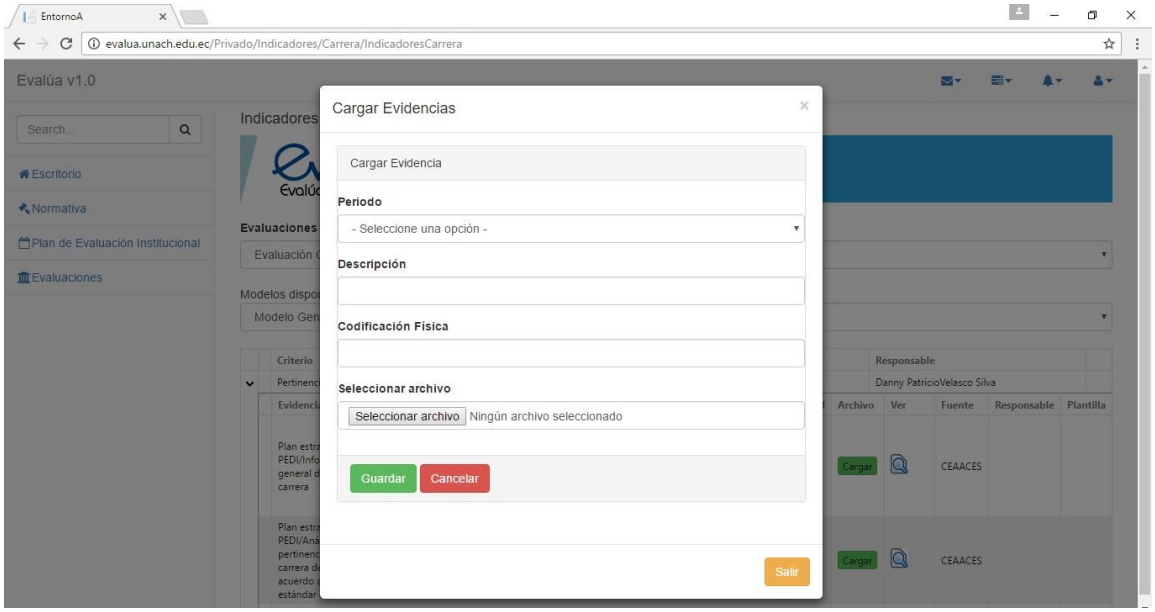
Listeners: Elementos que nos permiten tener acceso a los datos recopilados por las pruebas; estos pueden ser desde datos estadísticos hasta gráficas de evolución.

Summary Report: Elemento que nos permite obtener el resumen de los resultados de las mediciones realizadas.

Pre Medición: Sistema en producción del Departamento de Evaluación y Acreditación sin el módulo de Plan de Mejoras.

Vamos a ejecutar el plan de medición establecido en la siguiente URL: http://evalua.unach.edu.ec/Privado/Indicadores/Carrera/IndicadoresCarrera. Se ejecutarán pruebas de carga simulando peticiones de un número variable de usuarios.

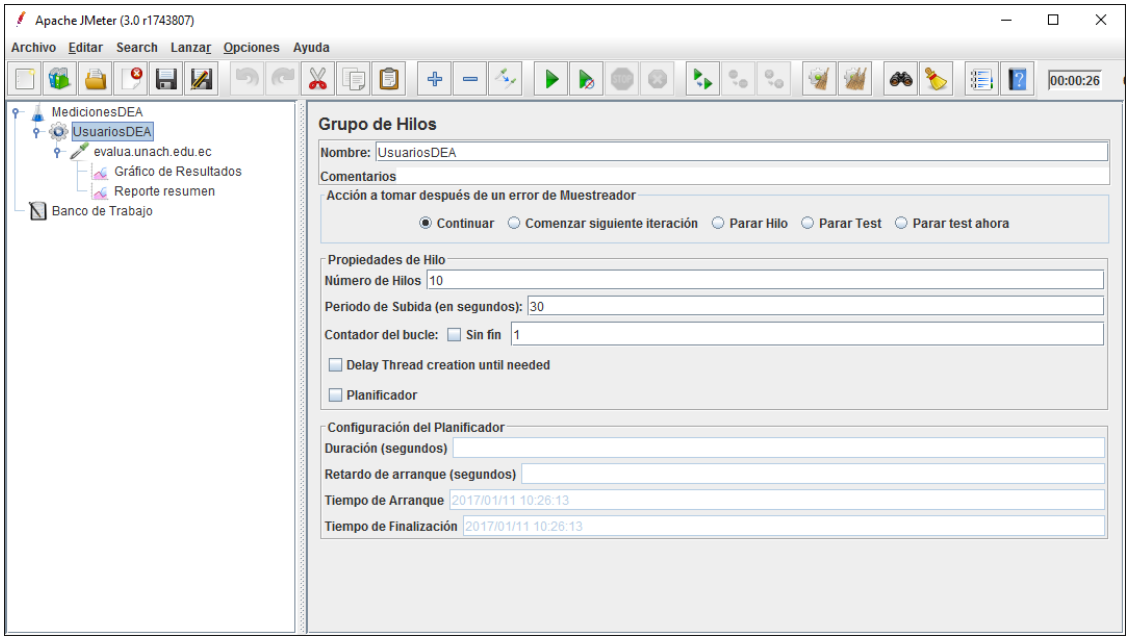




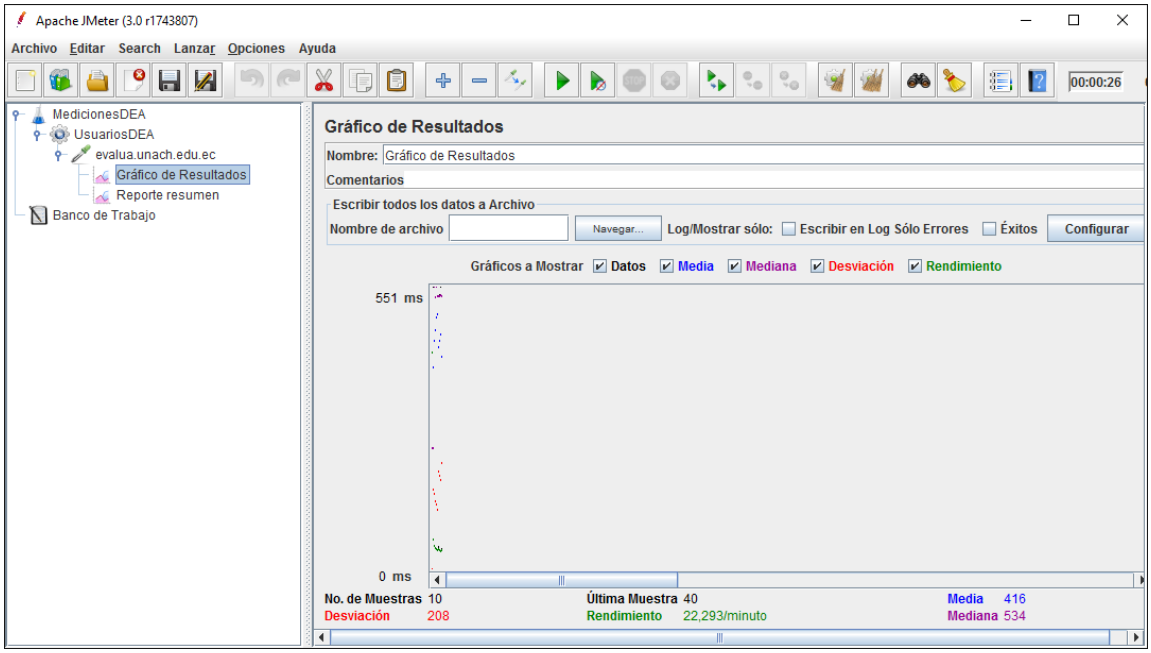
Prueba 1

Simulación de 10 usuarios con un periodo de subida de 30 segundos

Se considerarán 10 hilos (simulación de 10 usuarios) y un periodo de subida de 30 segundos (3 segundos entre el lanzamiento de cada hilo).



Ahora inicializamos las pruebas. Para ello haremos uso de la opción de menú Lanzar > Arrancar. Y obtendremos los resultados en el “Gráfico de Resultados”:

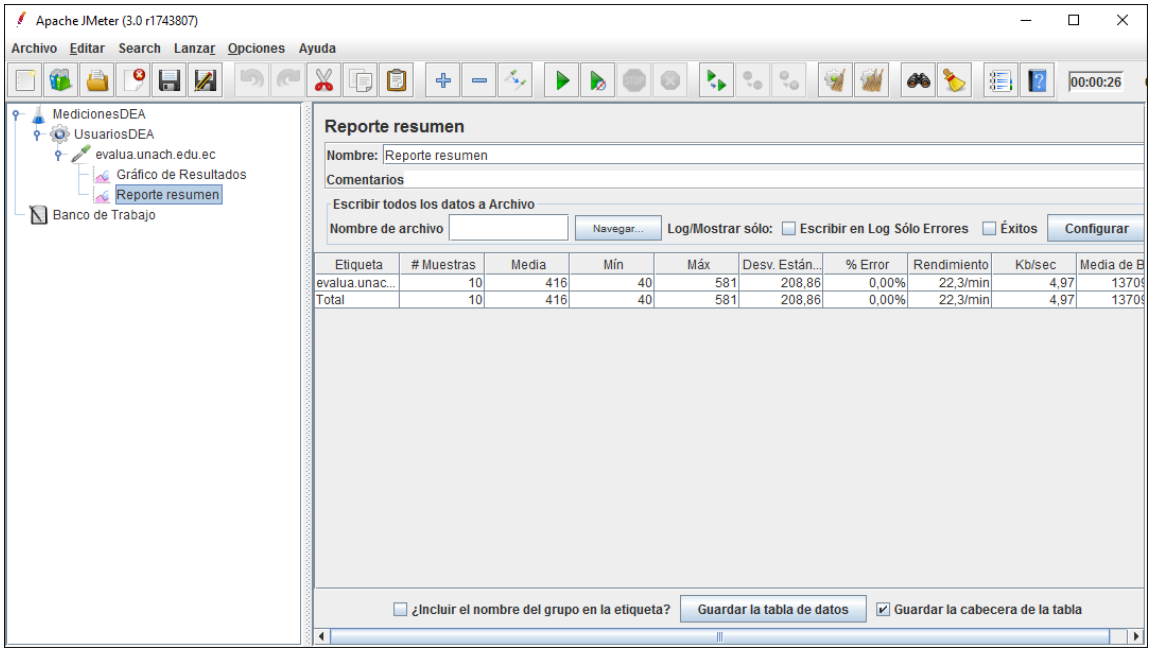


Una breve descripción de los elementos del gráfico puede ser la siguiente:

- Datos: muestra los valores actuales de los datos.
- Media: representa la Media.
- Mediana: dibuja la Mediana.
- Desviación: muestra la Desviación Estándar (una medida de la Variación).
- Rendimiento: representa el número de muestras por unidad de tiempo.

En la parte inferior de la ventana aparecen los valores actuales. “Última muestra” representa el tiempo transcurrido para la muestra en uso, valor mostrado en el gráfico como”Datos”.

El elemento “Summary Report” mostrará, a su vez, el siguiente aspecto:



El “Summary Report” (Informe Resumen) muestra la siguiente información:

- Etiqueta: El nombre de la muestra (conjunto de muestras).
- # Muestras: El número de muestras para cada URL.
- Media: El tiempo medio transcurrido para un conjunto de resultados.
- Mín: El mínimo tiempo transcurrido para las muestras de la URL dada.
- Máx: El máximo tiempo transcurrido para las muestras de la URL dada.
- Desv. Estándar: La desviación estándar para las muestras de la URL dada.
- Error %: Porcentaje de las peticiones con errores.
- Rendimiento: Rendimiento medido en base a peticiones por segundo/minuto/hora.
- Kb/sec: Rendimiento medido en Kilobytes por segundo.
- Media de Bytes: Tamaño medio de la respuesta de la muestra medido en bytes (erróneamente, en JMeter 2.2 muestra el valor en kB).

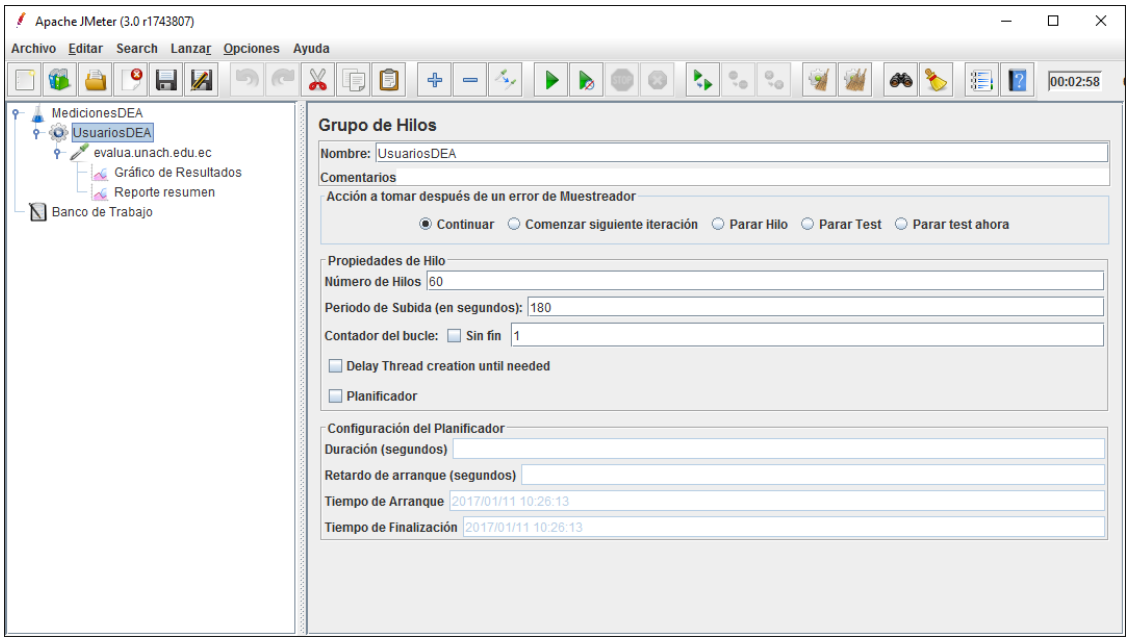
Podemos observar que las pruebas se han realizado sin errores. Esto se deduce de la columna representativa del tanto por ciento de errores para cada una de las peticiones asociadas a la muestra. El rendimiento nos muestra que para una simulación de 10 usuarios junto a un periodo de subida de 30 segundos el servidor es capaz de aceptar una media de 22,3 peticiones por segundo. La latencia (entendida como el tiempo de espera para la renderización de la página, el

tiempo en obtener respuesta del servidor) para cada conjunto de pruebas no supera el valor de 551 milisegundos (representado por el eje “y” de la gráfica).

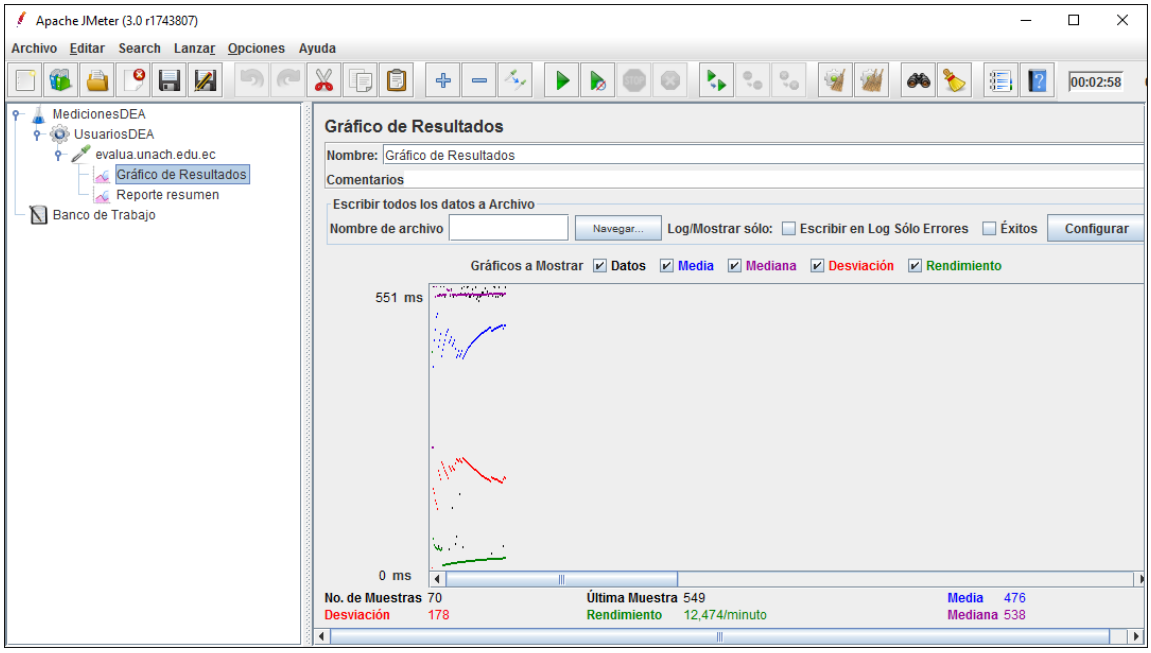
Prueba 2

Simulación de 60 usuarios con un periodo de subida de 180 segundos

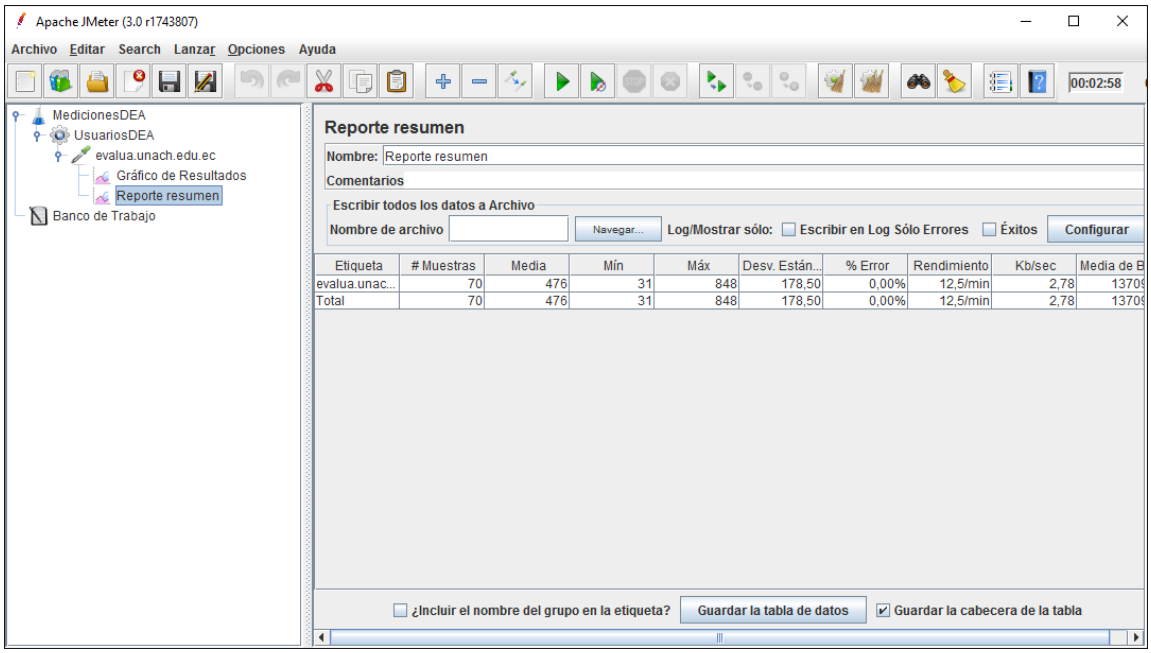
Si ahora realizamos la simulación con 60 usuarios considerando un periodo de subida de 180 segundos (de nuevo 3 segundos entre el lanzamiento de cada hilo) los resultados serán los siguientes, teniendo en cuenta que dichos resultandos se irán solapando a los ya obtenidos en la simulación anterior.



El elemento “Gráfico de Resultados” mostrará un aspecto como el siguiente:



Podemos observar que la ejecución se ha detenido. El “Summary Report” ofrece el siguiente aspecto:



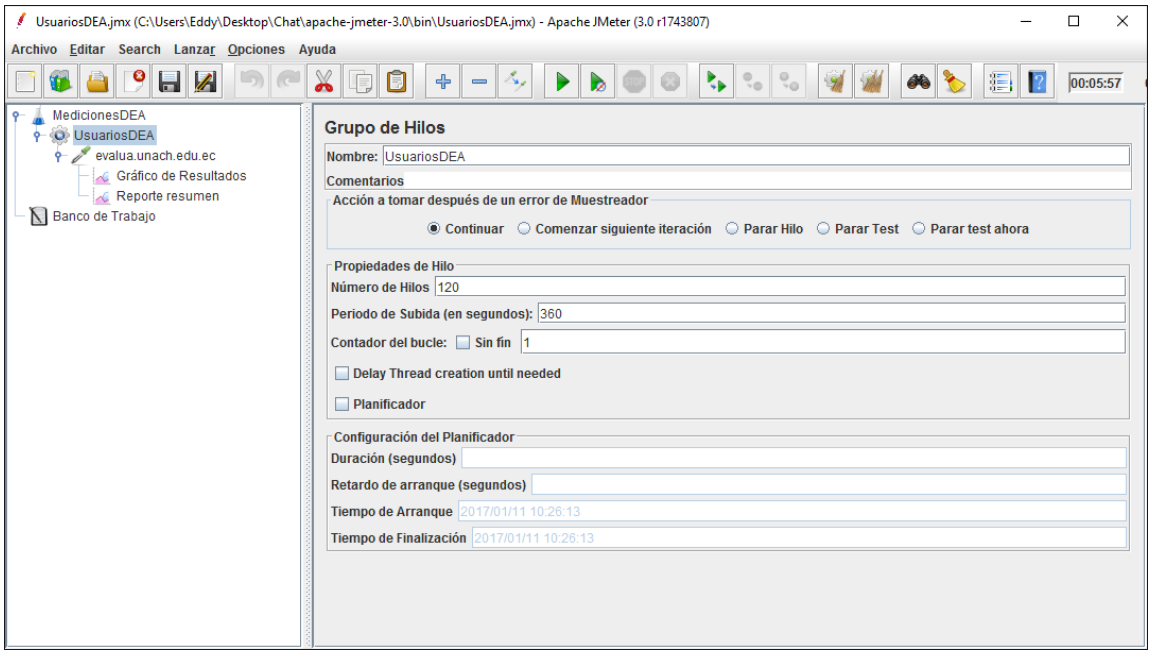
En este caso volvemos a observar que las pruebas se han realizado sin errores. El rendimiento nos muestra que para una simulación de 60 usuarios junto a un periodo de subida de 180

segundos el servidor es capaz de aceptar una media de 12,5 peticiones por minuto. La latencia se ve aumentada hasta un valor de 551ms.

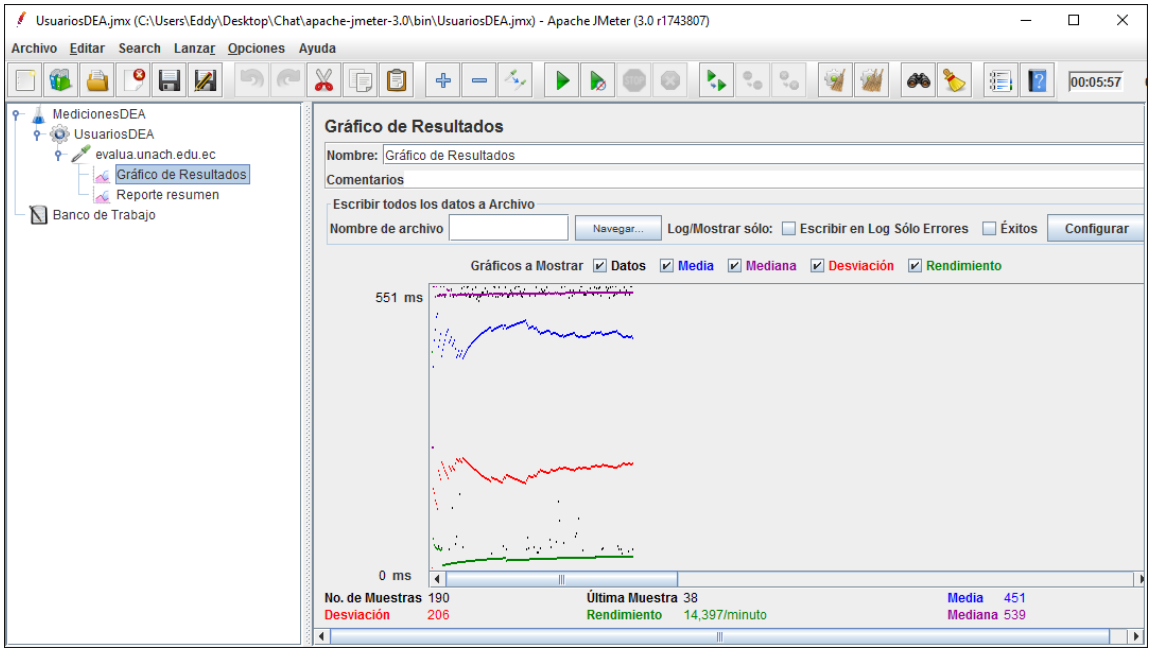
Prueba 3

Simulación de 120 usuarios con un periodo de subida de 360 segundos

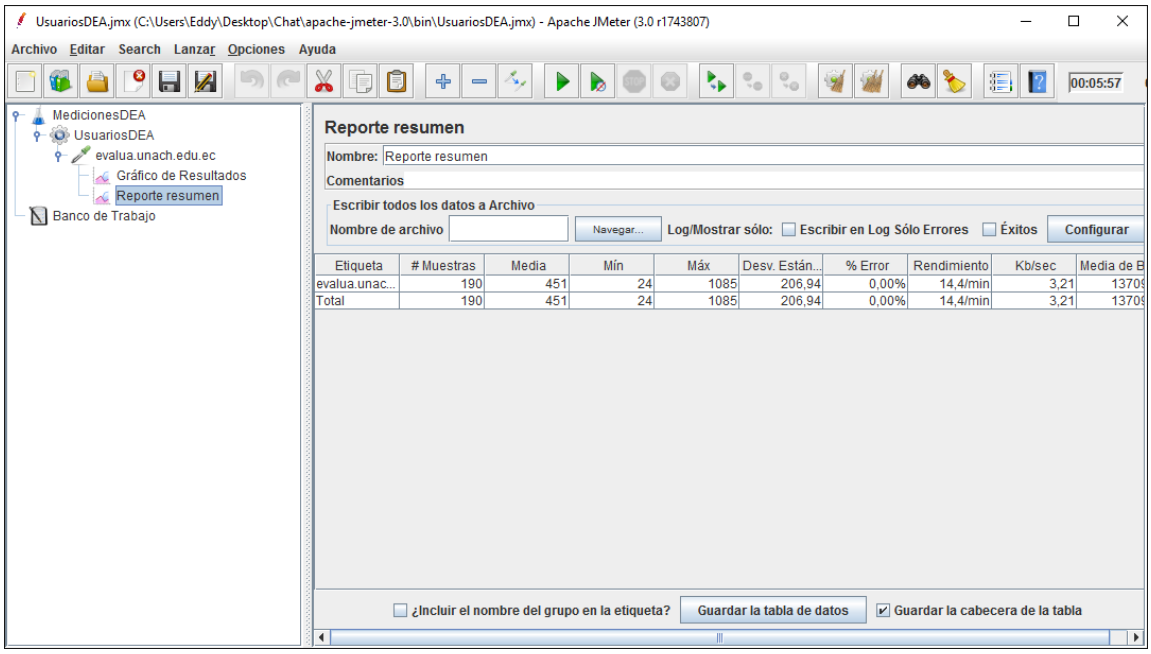
Una vez obtenidos e interpretados estos resultados lanzaremos de nuevo simulando 120 usuarios: Se cambia el número de hilos a 120 con un periodo de subida de 360 segundos (por lo tanto, se sigue manteniendo el tiempo entre hilos) y se estudian los resultados acumulados.



El elemento “Gráfico de Resultados” mostrará un aspecto como el siguiente:



Podemos observar que la ejecución se ha detenido. El “Summary Report” ofrece el siguiente aspecto:

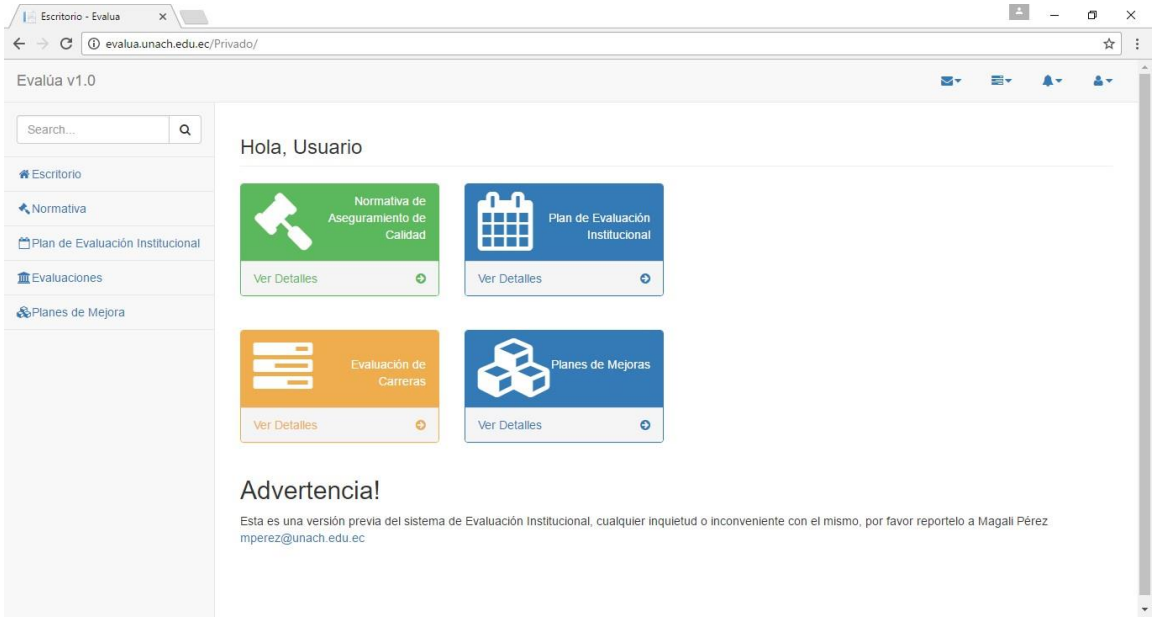


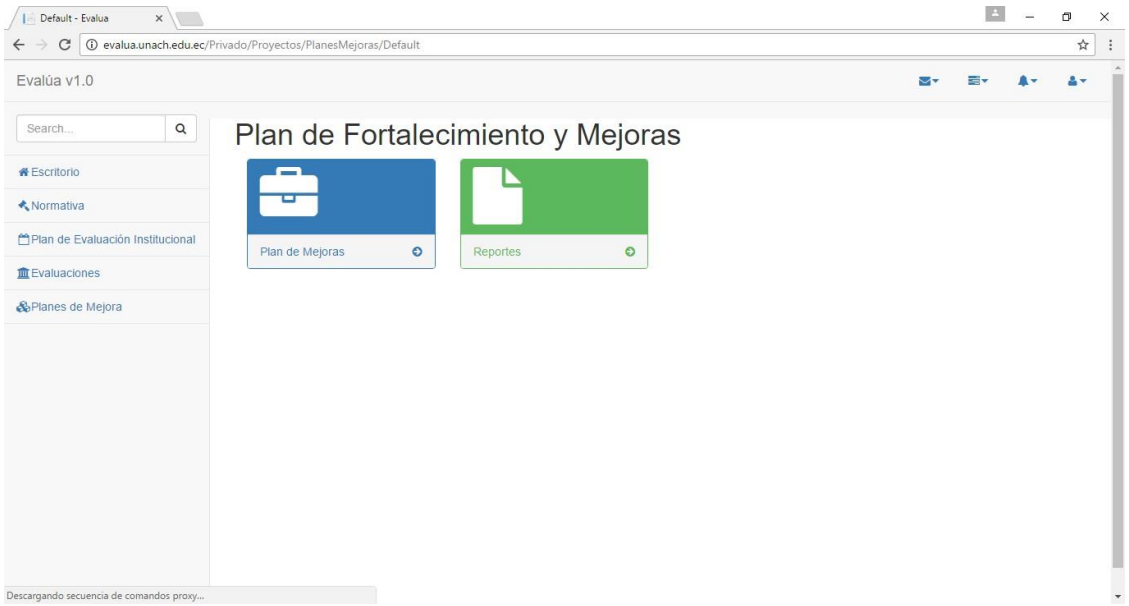
En este caso volvemos a observar que las pruebas se han realizado sin errores. El rendimiento nos muestra que para una simulación de 120 usuarios junto a un periodo de subida de 360 segundos el servidor es capaz de aceptar una media de 14,4 peticiones por minuto. La latencia se ve aumentada hasta un valor de 551ms.

Post Medición: Sistema en producción del Departamento de Evaluación y Acreditación con el módulo de Plan de Mejoras.

Vamos a ejecutar el plan de medición establecido en la siguiente URL:

http://evalua.unach.edu.ec/Privado/EvidenciaProyectos/Default. Se ejecutarán pruebas de carga simulando peticiones de un número variable de usuarios.



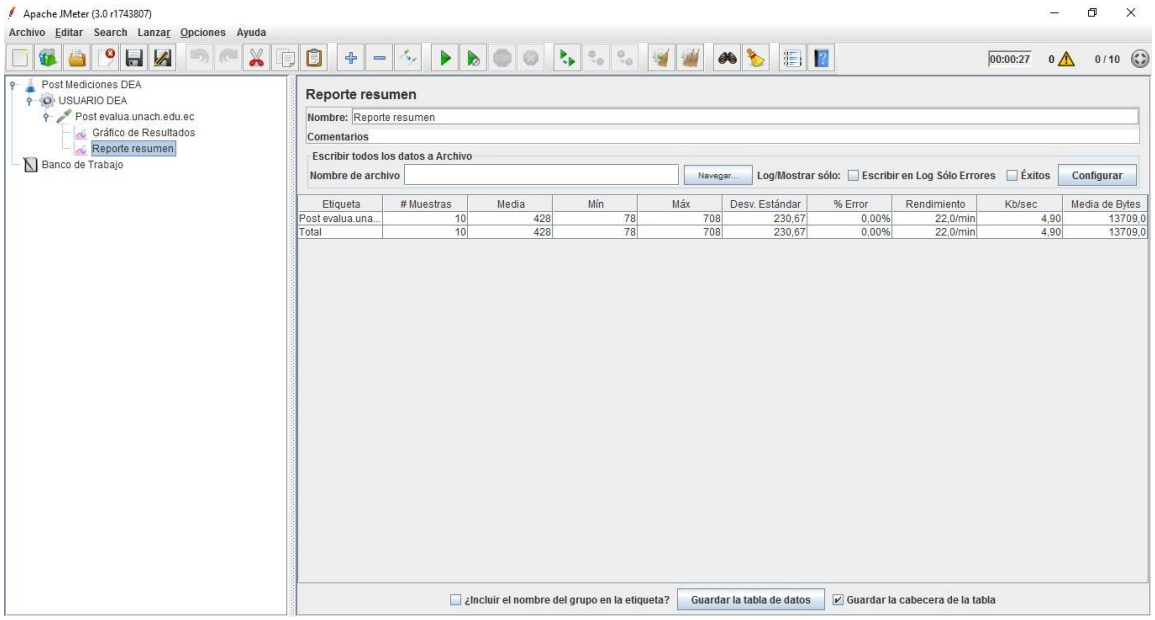


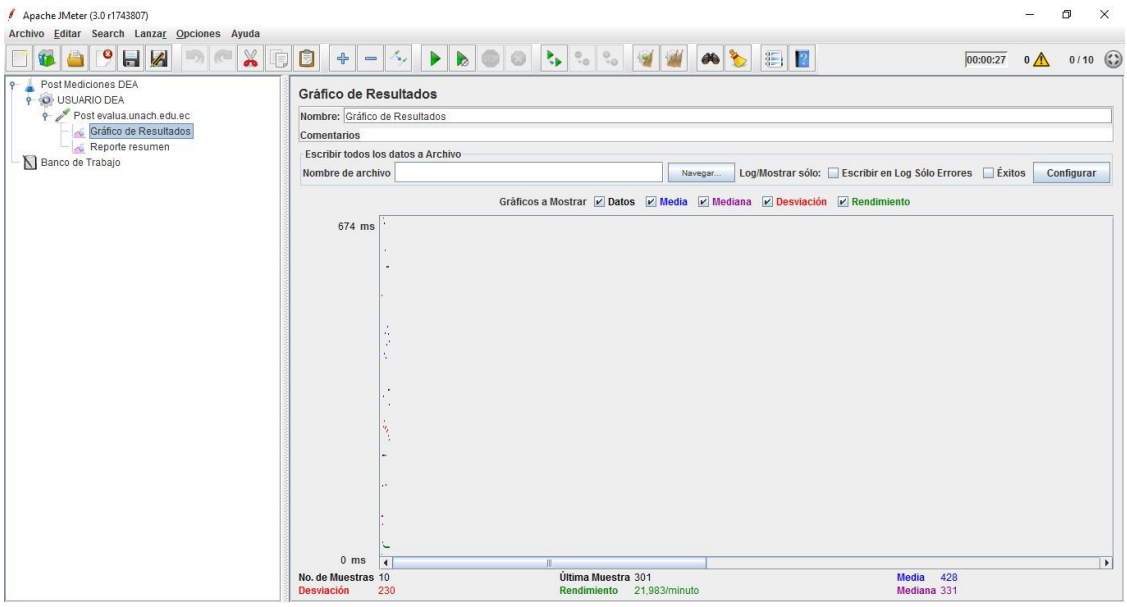
A continuación, se va a ejecutar las pruebas a la misma URL anterior pero considerando que ahora ya cuenta con el módulo de Plan de Mejoras.

Prueba 1

Simulación de 10 usuarios con un periodo de subida de 30 segundos

Se considerarán 10 hilos (simulación de 10 usuarios) y un periodo de subida de 30 segundos (3 segundos entre el lanzamiento de cada hilo).

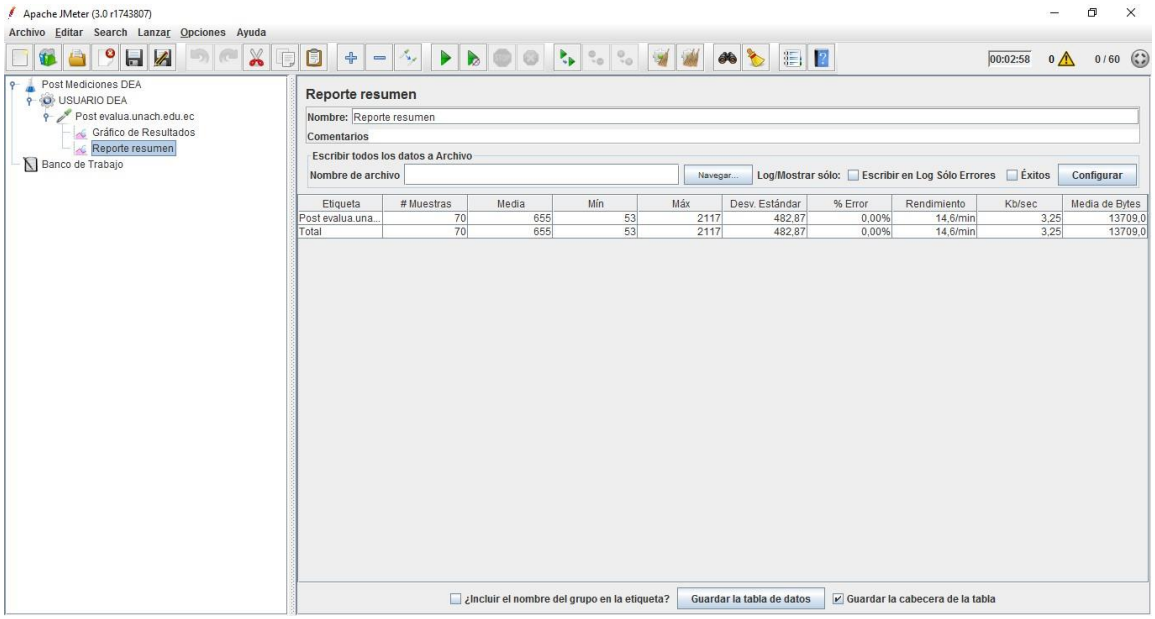


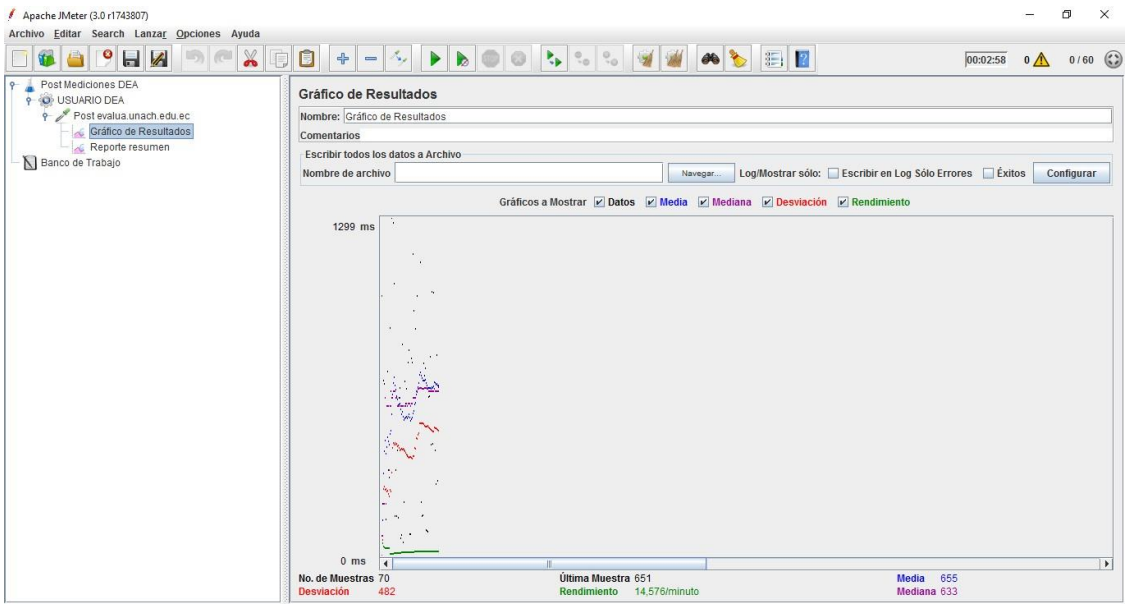


Prueba 2

Simulación de 60 usuarios con un periodo de subida de 180 segundos

Si ahora realizamos la simulación con 60 usuarios considerando un periodo de subida de 180 segundos (de nuevo 3 segundos entre el lanzamiento de cada hilo) los resultados serán los siguientes, teniendo en cuenta que dichos resultandos se irán solapando a los ya obtenidos en la simulación anterior.

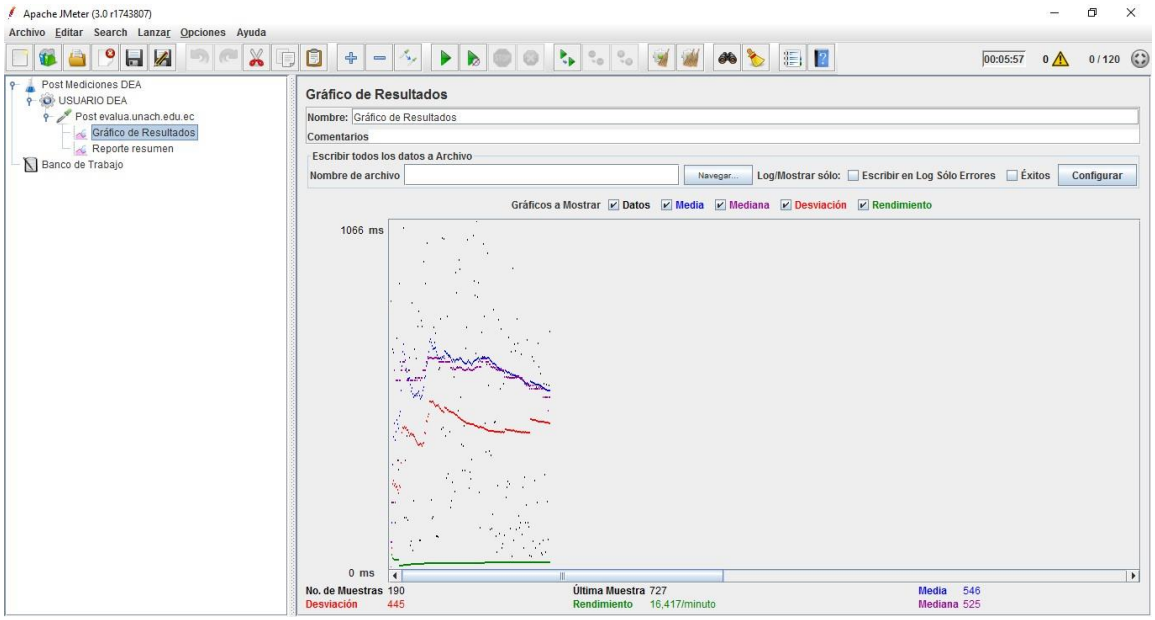


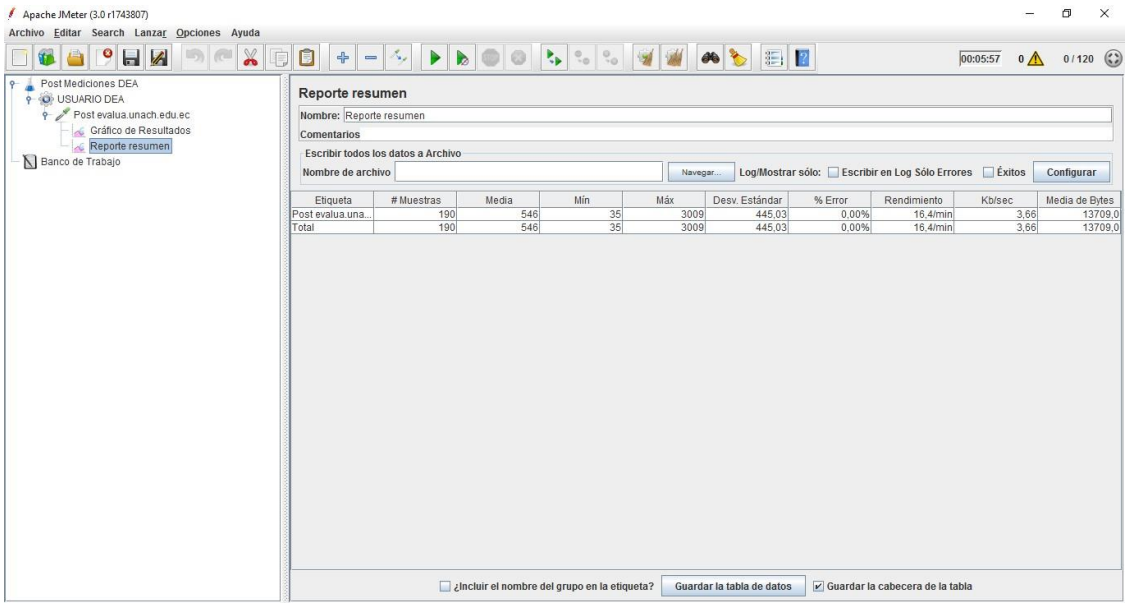


Prueba 3

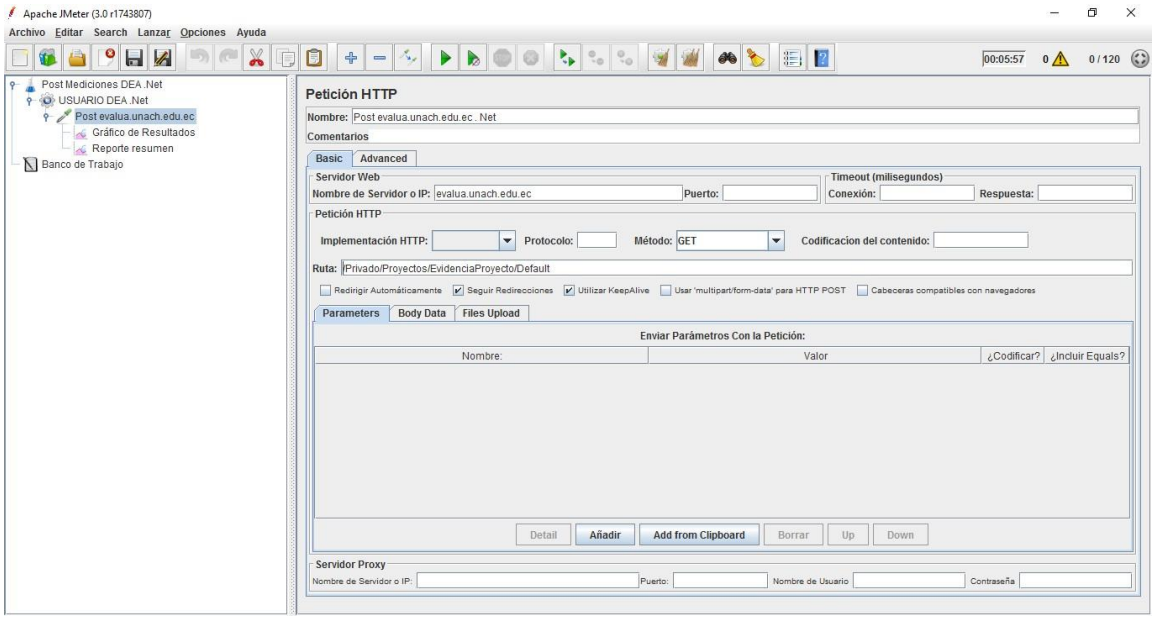
Simulación de 120 usuarios con un periodo de subida de 360 segundos

Una vez obtenidos e interpretados estos resultados lanzaremos de nuevo simulando 120 usuarios: Se cambia el número de hilos a 120 con un periodo de subida de 360 segundos (por tanto, se sigue manteniendo el tiempo entre hilos) y se estudian los resultados acumulados.





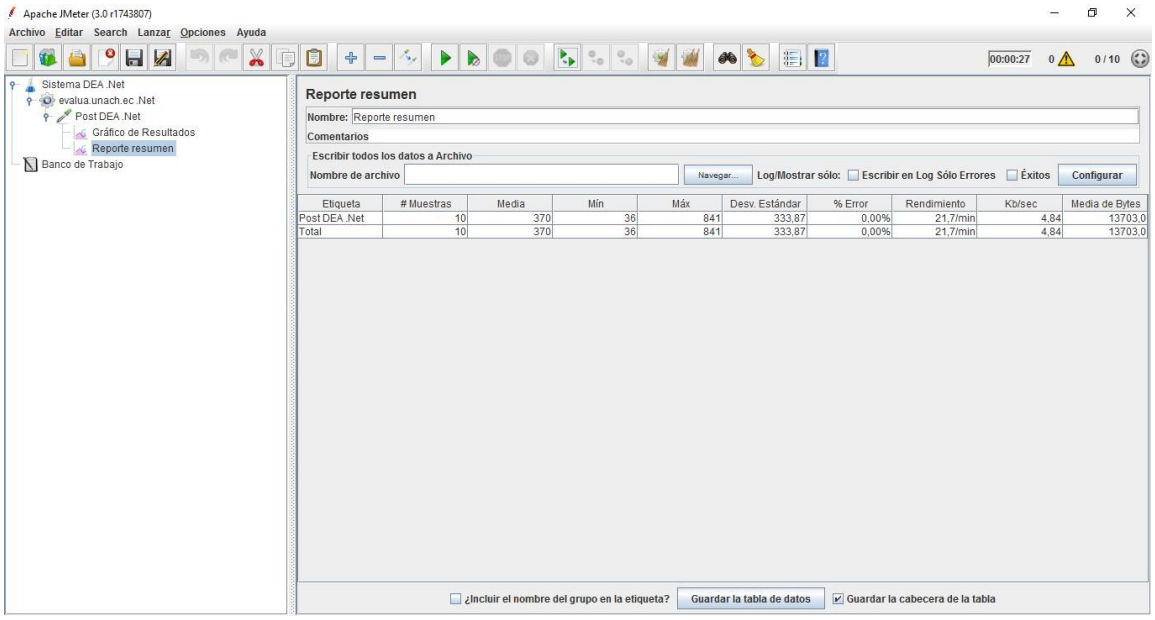
A continuación, se va a ejecutar las pruebas al módulo de Plan de Mejoras.

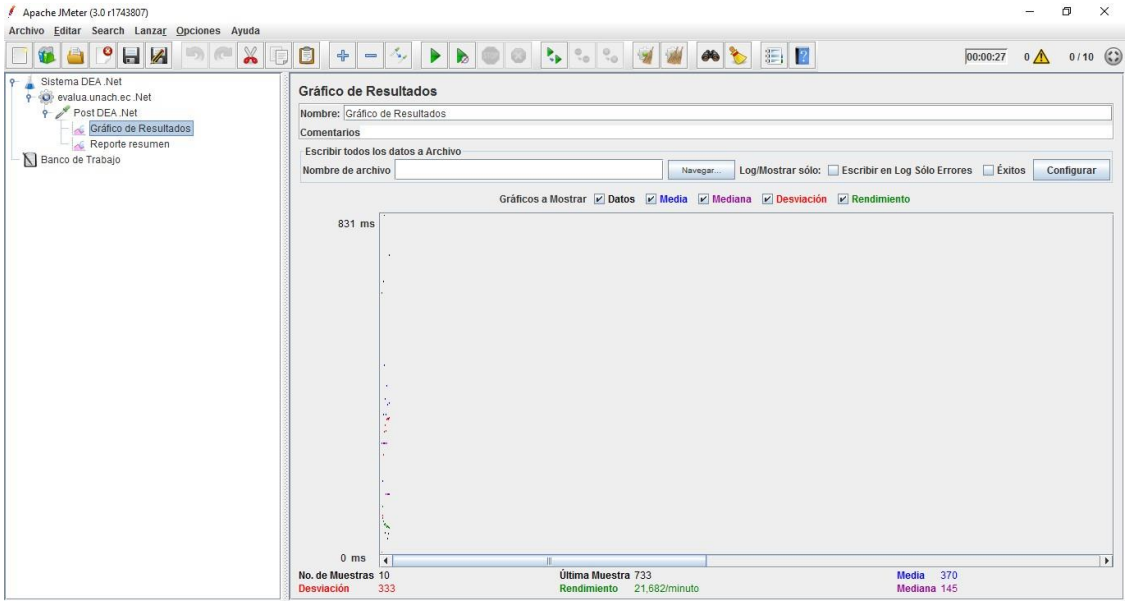


Prueba 1

Simulación de 10 usuarios con un periodo de subida de 30 segundos

Se considerarán 10 hilos (simulación de 10 usuarios) y un periodo de subida de 30 segundos (3 segundos entre el lanzamiento de cada hilo).

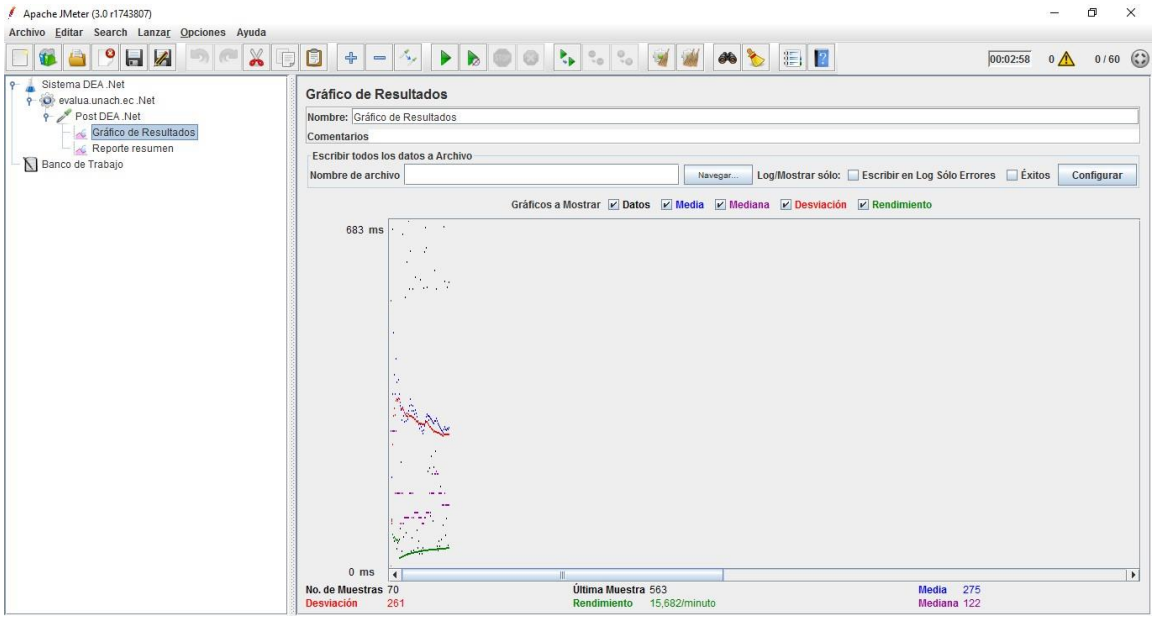

Apache JMeter (3.0 r1743807)
Archivo Editar Search Lanzar Opciones Ayuda
00:00:27 0 0 / 10
Sistema DEA .Net
evalua.unach.ec .Net
Post DEA .Net
Gráfico de Resultados
Reporte resumen
Banco de Trabajo
Reporte resumen
Nombre: Reporte resumen
Comentarios
Escribir todos los datos a Archivo
Nombre de archivo
Navegar...
Log/Mostrar sólo: Escribir en Log Sólo Errores Éxitos
Configurar
Etiqueta # Muestras Media Mín Máx Desv. Estándar % Error Rendimiento Kb/sec Media de Bytes
Post DEA .Net 10 370 36 841 333,87 0,00% 21,7/min 4,84 13703,0
Total 10 370 36 841 333,87 0,00% 21,7/min 4,84 13703,0
¿Incluir el nombre del grupo en la etiqueta?
Guardar la tabla de datos
Guardar la cabecera de la tabla

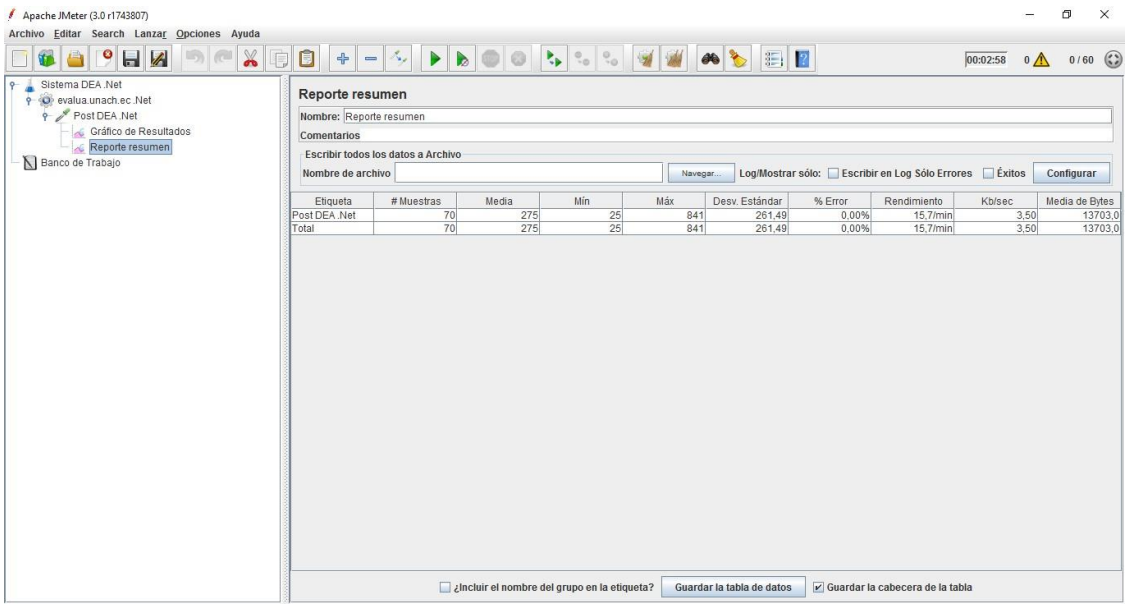

Apache JMeter (3.0 r1743807)
Archivo Editar Search Lanzar Opciones Ayuda
00:00:27 0 0 / 10
Sistema DEA .Net
evalua.unach.ec .Net
Post DEA .Net
Gráfico de Resultados
Reporte resumen
Banco de Trabajo
Gráfico de Resultados
Nombre: Gráfico de Resultados
Comentarios
Escribir todos los datos a Archivo
Nombre de archivo
Navegar...
Log/Mostrar sólo: Escribir en Log Sólo Errores Éxitos
Configurar
Gráficos a Mostrar Datos Media Mediana Desviación Rendimiento
831 ms
0 ms
No. de Muestras 10
Última Muestra 733
Media 370
Desviación 333
Rendimiento 21,682/minuto
Mediana 145

Prueba 2

Simulación de 60 usuarios con un periodo de subida de 180 segundos

Si ahora realizamos la simulación con 60 usuarios considerando un periodo de subida de 180 segundos (de nuevo 3 segundos entre el lanzamiento de cada hilo) los resultados serán los siguientes, teniendo en cuenta que dichos resultandos se irán solapando a los ya obtenidos en la simulación anterior.

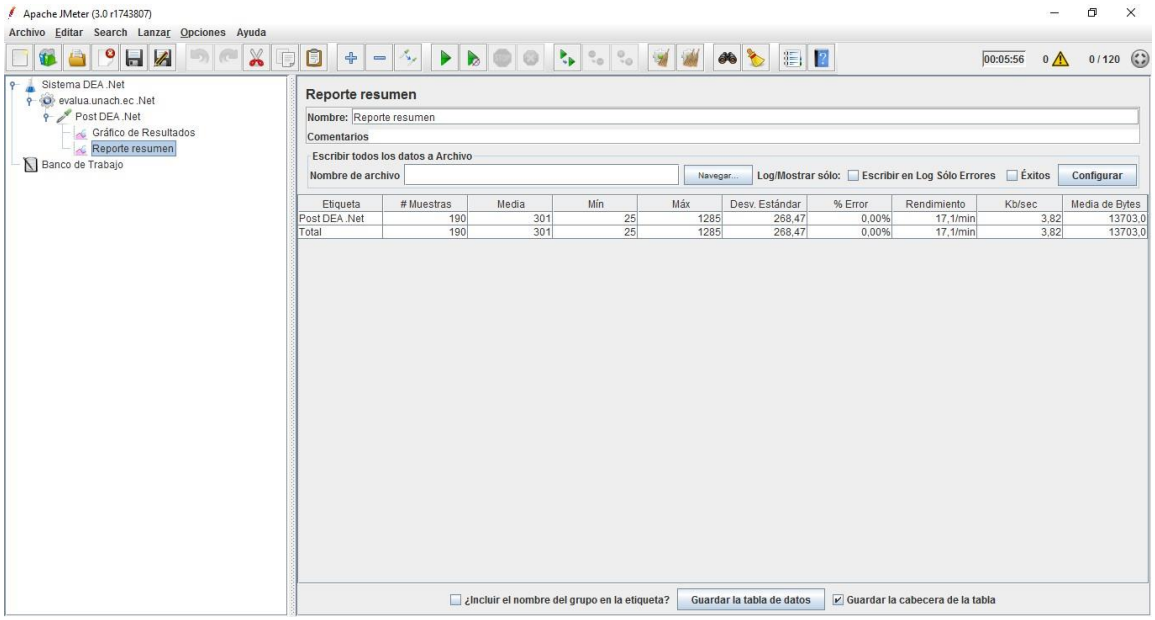


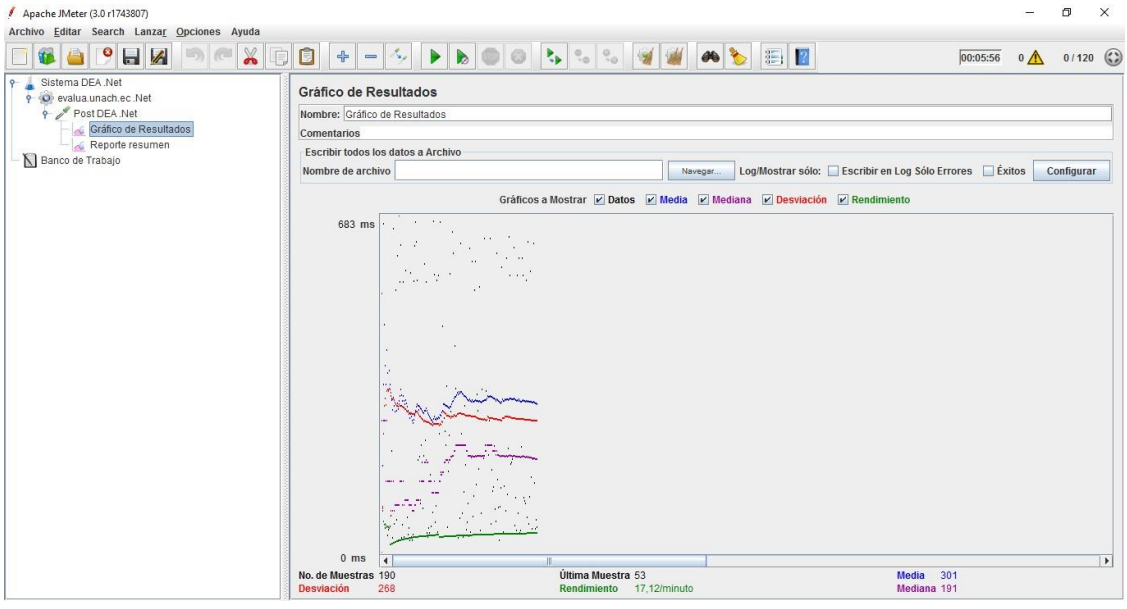


Prueba 3

Simulación de 120 usuarios con un periodo de subida de 360 segundos

Una vez obtenidos e interpretados estos resultados lanzaremos de nuevo simulando 120 usuarios: Se cambia el número de hilos a 120 con un periodo de subida de 360 segundos (por tanto, se sigue manteniendo el tiempo entre hilos) y se estudian los resultados acumulados.





Cuadro de Mediciones entre la Arquitectura N – Capas y la Arquitectura N – capas Orientada al Dominio con .NET

Nª	Usuarios	Segundos	Indicador	Sistema DEA (sin el modulo Plan de Mejoras)	Sistema DEA (con el modulo Plan de Mejoras)	Sistema DEA (con el modulo Plan de Mejoras)Arquitectura N – Capas Orientada al Dominio con .Net
1	10	30s	Máximo de tiempo invertido por una petición	581 ms	708 ms	841 ms
			Mínimo de tiempo invertido por una petición.	40 ms	78 ms	36 ms
			Media de tiempo invertido por una petición.	416 ms	428 ms	370 ms
			Mediana de tiempo invertido por una petición: significa que el 50% de las muestras tardaron menos del valor reflejado.	534 ms	331 ms	145 ms
			Tanto por ciento de respuestas con error.	0 %	0 %	0 %

			El rendimiento (thoughput): número de peticiones procesadas en una unidad de tiempo, que puede ser segundos, minutos y horas.	22.3/min	22.0 /min	21.7 /min
			El rendimiento en Kb/segundo: igual que la anterior pero con cantidad de datos en lugar de peticiones	4.97 kb/s	4.90 kb/s	4.84 Kb/s
2	60	180s	Máximo de tiempo invertido por una petición	848 ms	2117 ms	841 ms
			Mínimo de tiempo invertido por una petición.	31 ms	53 ms	25 ms
			Media de tiempo invertido por una petición.	476 ms	655 ms	275 ms
			Mediana de tiempo invertido por una petición: significa que el 50% de las muestras tardaron menos del valor reflejado.	538 ms	633 ms	122 ms
			Tanto por ciento de respuestas con error.	0 %	0 %	0 %

			El rendimiento (thoughput): número de peticiones procesadas en una unidad de tiempo, que puede ser segundos, minutos y horas.	12.5/min	14.6 /min	15.7 /min
			El rendimiento en Kb/segundo: igual que la anterior pero con cantidad de datos en lugar de peticiones	2.78 kb/s	3.25 kb/s	3.50 kb/s
3	120	360s	Máximo de tiempo invertido por una petición	1085 ms	3009 ms	1285 ms
			Mínimo de tiempo invertido por una petición.	24 ms	35 ms	25 ms
			Media de tiempo invertido por una petición.	451 ms	546 ms	301 ms
			Mediana de tiempo invertido por una petición: significa que el 50% de las muestras tardaron menos del valor reflejado.	539 ms	525 ms	191 ms
			Tanto por ciento de respuestas con error.	0%	0 %	0 %

			El rendimiento (thoughput): número de peticiones procesadas en una unidad de tiempo, que puede ser segundos, minutos y horas.	14.4/min	16.4 /min	17.1 /min
			El rendimiento en Kb/segundo: igual que la anterior pero con cantidad de datos en lugar de peticiones	3.21 kb/s	3.66 kb/s	3.82 kb/s