ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA EN SISTEMAS

"ESTUDIO ESTADÍSTICO ENTRE UNA APLICACIÓN COMPUESTA CON PRISM Y OTRA MONOLÍTICA PARA EL ESTUDIO DE LA GEOGRAFÍA ECUATORIANA"

TESIS DE GRADO

Previa a la obtención del título de

INGENIERO EN SISTEMAS INFORMÁTICOS

Presentado por:

MAGALI MARICELA PÉREZ CARRANZA

HENRY JAVIER PACA QUINALUIZA

RIOBAMBA – ECUADOR

2012

Al culminar una etapa más de nuestra vida queremos agradecer profundamente a nuestros padres, quienes nos han apoyado y motivado durante nuestra formación académica, creyeron en nosotros en todo momento y no dudaron de nuestras habilidades.

A nuestros maestros y tutores Iván y Jorge, a quienes les debemos gran parte de nuestros conocimientos, gracias a su paciencia, enseñanza y dedicación en cada detalle necesario para alcanzar nuestros sueños, metas y culminar con éxito nuestra carrera.

Y finalmente un eterno agradecimiento a esta prestigiosa institución la cual abre sus puertas a jóvenes como nosotros, preparándonos para un futuro competitivo y formándonos como personas de bien.

Magali Maricela Pérez Carranza

Henry Javier Paca Quinaluiza

Con inmenso cariño dedico este trabajo a mi familia, en especial a mis papis, por todo su esfuerzo y sacrificio para ayudarme a cumplir mis sueños, por su comprensión y sobre todo su incondicional apoyo y enseñarme que una caída no es una derrota, a ser una persona que lucha y se esfuerza por ser cada día mejor. Va por ustedes, por lo que valen, porque admiro su fortaleza y por lo que han hecho de mí.

Magali

Dedico todo el esfuerzo involucrado en este trabajo a mi familia, que son un ejemplo de perseverancia y constancia. A mis padres por ser el pilar fundamental en todo cuanto soy, en toda mi educación, tanto académica, como de la vida, por su incondicional apoyo perfectamente mantenido a través del tiempo.

Henry

FIRMAS DE RESPONSABILIDAD Y NOTA DEL TRIBUNAL

NOMBRES	**FIRMAS**	**FECHA**
Ing. Iván Menes **DECANO DE LA FACULTAD** **INFORMÁTICA Y ELECTRÓNICA**	_____	_____
Ing. Raúl Rosero **DIRECTOR DE ESCUELA** **INGENIERÍA EN SISTEMAS**	_____	_____
Ing. Iván Menes **DIRECTOR DE TESIS**	_____	_____
Ing. Jorge Menéndez **MIEMBRO DE TESIS**	_____	_____
Tlgo. Carlos Rodríguez **DIRECTOR DEL CENTRO DE** **DOCUMENTACIÓN**	_____	_____

NOTA: _____

“Nosotros, Magali Maricela Pérez Carranza y Henry Javier Paca Quinaluiza, somos responsables de las ideas, doctrinas y resultados expuestos en esta Tesis de Grado, y el patrimonio intelectual de la misma pertenecen a la ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO”

Magali Maricela Pérez Carranza

Henry Javier Paca Quinaluiza

ÍNDICE DE ABREVIATURAS

API.- Application Programming Interface

CPU.- Central Process Unit

CRC.- Class Responsibility Collaboration

CRUD.- Create Update Delete

DI.- Dependency Inyector

MEF.- Managed Extensibility Framework

MVC.- Model View Controler

MVVM.- Model View ViewModel

RAM.- Random Access Memory

RIA.- Rich Internet Application

SDK.- Software Developer Kit

SOA.- Service Oriented Architecture

WCF.- Windows Communication Foundation

WPF.- Windows Presentation Foundation

XAML.- Extensible Application Markup Language

XML.- Extensible Markup Language

XP.- Extreme Programming

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La tecnología está presente en todo lo que nos rodea, desde nuestro trabajo, comunidad, familia, hasta nuestro hogar, en fin todo lo relacionado con la vida cotidiana. Otro de los campos que más se ha visto influenciado por los cambios de la tecnología es la educación, vemos que muchas escuelas hoy en día están integrando la tecnología en el ambiente del aprendizaje y empezando a explorar el potencial tan grande que ofrece para educar y aprender. Con el uso adecuado, la tecnología ayuda a los estudiantes a adquirir las habilidades necesarias para sobrevivir en una sociedad enfocada en el conocimiento tecnológico.

Es por esta razón que la Unidad Educativa "ANDES COLLEGE" que radica en la ciudad de Riobamba, requiere de la elaboración de una aplicación que ayude al estudio de la geografía ecuatoriana, con el uso de tecnología de vanguardia que ayude a tener un proceso enseñanza aprendizaje de mejor calidad y con más participación por parte del estudiante.

Para la elaboración de la aplicación primero se debe evaluar cuál es la arquitectura correcta que permita que la aplicación trabaje de mejor manera; por lo que se realizará el estudio estadístico entre una aplicación compuesta con PRISM y otra monolítica.

Para la implementación de la aplicación de estudio de la geografía ecuatoriana se desarrollará el estudio de los siguientes puntos:

- Estudiar el desarrollo de una Aplicación Compuesta con PRISM y una Aplicación Monolítica.
- Determinar los parámetros necesarios para realizar el análisis comparativo entre Aplicaciones Compuesta con PRISM y Aplicaciones Monolítica.
- Comparar estadísticamente el desarrollo de un prototipo de Aplicación Compuesta con PRISM y un prototipo de Aplicación Monolítica.
- Desarrollar una aplicación de escritorio para el estudio de la Geografía Ecuatoriana en nivel primario que ayude al enriquecimiento del aprendizaje.

La metodología con la que se desarrollará la aplicación es la metodología ágil XP la misma que fue utilizada por su rapidez y relativa facilidad para ser implementada.

Los resultados que se obtengan con la investigación entre las dos formas de implementar la aplicación de estudio de la geografía ecuatoriana (Aplicaciones monolíticas y aplicaciones compuestas) y los parámetros establecido para el estudio estadístico (Número de líneas de código escritas y tiempo de desarrollo) indicarán que forma de implementación es la mejor.

CAPÍTULO I

MARCO REFERENCIAL

1.1. Aspectos generales

TITULO DEL PROYECTO DE TESIS

ESTUDIO ESTADÍSTICO ENTRE UNA APLICACIÓN COMPUESTA CON PRISM Y OTRA MONOLÍTICA PARA EL ESTUDIO DE LA GEOGRAFÍA ECUATORIANA.

1.2. Formulación general del proyecto de tesis

1.2.1. Antecedentes

En la unidad educativa “Andes College” se dicta la materia Estudios Sociales, uno de los temas tratados en dicha materia es la Geografía Ecuatoriana, la enseñanza de esta materia se la realiza únicamente de forma teórica convirtiéndose en un problema al momento de captar la atención de los estudiantes, para lo cual se requiere de una aplicación que apoye al estudio de la misma mejorando el aprendizaje, ya que según estudios realizados el aprendizaje mediante el uso de la lúdica y los juegos benefician de gran manera a la adquisición del conocimiento [5].

En aplicaciones cuya experiencia de usuario debe ser enriquecida, el uso de dispositivos de entrada son parte esencial para lograr una mejor interacción con el usuario; tecnologías modernas como Kinect permite a los usuarios controlar e interactuar con aplicaciones sin necesidad de tener contacto físico, mediante el reconocimiento de gestos con el movimiento del cuerpo, comandos de voz, objetos e imágenes [11].

Si bien es cierto Kinect fue desarrollado inicialmente para el mundo de los videojuegos, hoy en día se lo puede utilizar en aplicaciones de propósito general brindando una mejor experiencia de usuario y aportando significativamente en el proceso enseñanza aprendizaje.

Normalmente estas aplicaciones requieren de una gran cantidad de componentes y controles de usuario que muy a menudo tienen un alto grado de acoplamiento con la

interfaz de usuario, generando un gran problema al momento de modificar, agregar, o eliminar algún componente o control de usuario, pues esto significaría una modificación completa de la interfaz y lógica de usuario de la aplicación.

Este problema ocurre cuando una aplicación es diseñada en base a una arquitectura monolítica la cual se caracteriza por tener un alto grado de acoplamiento entre sus controles y la interfaz de usuario lo que dificulta la reutilización de código, mantenimiento y flexibilidad [4].

Es razonable esperar que un desarrollador con una experiencia moderada pueda mantener y ampliar fácilmente este tipo de aplicaciones; no obstante, a medida que aumenta el número de partes móviles y personas que trabajan en dichas partes, mantener el proyecto controlado se vuelve exponencialmente más difícil.

Ante este problema existe una solución novedosa con la que se puede estructurar una aplicación de forma modular con un bajo grado de acoplamiento; a este tipo de soluciones se las conoce como aplicaciones compuestas con PRISM que constan de módulos de acoplamiento flexible que se descubren y componen dinámicamente en tiempo de ejecución. Los módulos contienen componentes visuales y no visuales que representan los distintos sectores del sistema. Los componentes visuales se agregan en un Shell común que actúa como host de todo el contenido de la aplicación [8] [16].

De esta manera a una aplicación compuesta se le pueden agregar nuevas funcionalidades en tiempo de ejecución sin la necesidad de un rediseño de arquitectura.

1.2.2. Lugar de aplicación

El Tema investigativo de esta tesis se la realizará en la Facultad de Informática y Electrónica de la Escuela Superior Politécnica de Chimborazo; el caso práctico se lo aplicará en la Unidad Educativa “Andes College” ambos localizados la Ciudad de Riobamba, Provincia de Chimborazo.

1.2.3. Formulación del problema

Entre un modelo de arquitectura de Aplicación Compuesta con PRISM y un modelo de arquitectura Monolítica, cuál es la mejor para desarrollar una aplicación que permita mejorar la enseñanza de la materia geografía en nivel primario para la Unidad Educativa “Andes College”.

1.3. Justificación del tema

1.3.1. Justificación teórica

El desarrollo y avance tecnológico en el campo de experiencia de usuario ha dado lugar al desarrollo de aplicaciones de propósito general usando nuevas tecnologías, originando que se desarrollen aplicaciones a gran escala y descuidando la arquitectura adecuada para que una aplicación sea flexible y fácil de mantener. Adicionalmente el uso de un dispositivo Kinect se proyecta en el campo educativo como una tecnología capaz de aportar transformaciones significativas en la forma en la que todos interactuamos con las aplicaciones

La tecnología Kinect inicialmente fue desarrollada para el uso en videojuegos de última generación, sin embargo organizaciones mundiales como la NASA aprovechan el potencial que esta tecnología brinda para crear aplicaciones que enriquecen la experiencia de usuario ayudando a entender los contenidos de aprendizaje, puesto que aporta nuevas formas de interacción entre lo virtual y lo real.

1.3.2. Justificación aplicativa

En la unidad educativa “Andes College” se dicta la materia Estudios Sociales, uno de los temas tratados en dicha materia es la Geografía Ecuatoriana, para lo cual se requiere de una aplicación que sirva de apoyo en el aprendizaje y estudio de la misma.

Mediante la investigación propuesta se desarrollara una aplicación para el aprendizaje y estudio de la geografía ecuatoriana mediante el uso de un dispositivo Kinect que permitirá enriquecer la experiencia de usuario y aprendizaje; ya que según estudios realizados los métodos lúdicos son las herramientas más efectivas para promover el aprendizaje y transferir el conocimiento gracias a su capacidad de simular la realidad ofreciendo un escenario para cometer errores y aprender de ellos en la práctica.

1.4. Objetivos

Objetivo General

Realizar el estudio estadístico comparativo entre el desarrollo de una Aplicación Compuesta con PRISM y una Aplicación Monolítica para el desarrollo de una aplicación de apoyo al estudio de la Geografía Ecuatoriana en la Unidad Educativa “Andes College”.

Objetivos Específicos

- Estudiar el desarrollo de una Aplicación Compuesta con PRISM y una Aplicación Monolítica.
- Determinar los parámetros necesarios para realizar el análisis comparativo entre Aplicaciones Compuesta con PRISM y Aplicaciones Monolítica.
- Comparar estadísticamente el desarrollo de un prototipo de Aplicación Compuesta con PRISM y un prototipo de Aplicación Monolítica
- Desarrollar una aplicación de escritorio para el estudio de la Geografía Ecuatoriana en nivel primario que ayude al enriquecimiento del aprendizaje.

1.5. Hipótesis

Realizar un estudio estadístico entre el desarrollo de una aplicación compuesta con PRISM y una aplicación monolítica permitirá escoger un modelo de arquitectura correcto

para crear una aplicación de estudio de la Geografía Ecuatoriana en la Unidad Educativa “Andes College”.

CAPÍTULO II

MARCO TEÓRICO

Casi todos hemos observado alguna vez la construcción de un edificio. Comienza por los cimientos, luego las columnas y vigas, las distintas plantas, hasta tener un esqueleto de soporte. Después se construyen paredes, suelos, puertas y ventanas, instalaciones eléctricas y de fontanería, bancadas, etc., nos es casi imposible imaginar el levantar una pared antes que las columnas. En resumen, primero se crea la estructura o esqueleto del edificio, y luego se ensamblan las distintas partes.

La primera sirve de soporte a las segundas, que aportan la mayoría de las funcionalidades básicas del inmueble. ¿Qué pasaría si se cometen errores en una u otra?

Si se olvida construir una columna y se detecta al finalizar el edificio, probablemente este no obtenga nunca el permiso de habitabilidad. Sin embargo, si lo que se olvida es instalar una bañera o un armario empotrado se pierde solo una funcionalidad del inmueble (para el usuario final), pero indudablemente será habitable y dicho fallo será probablemente subsanable con un esfuerzo relativamente pequeño.

Esta forma de proceder es una estrategia general de solución de problemas en multitud de disciplinas sociales y técnicas. Una de ellas es la creación de software. A diferencia de la construcción de un edificio “común”, el software no se rige por leyes físicas ni por procedimientos conocidos, sino que es inherentemente específico y experimental. Como indica su nombre, la característica principal del software es ser, flexible, elástico y, en general, muy específico para la solución de una tarea concreta, sobre sistemas de hardware concretos, por personal con actitudes, aptitudes, formación y experiencia concretas. Todo esto hace del diseño de un software particular una tarea generalmente única, creativa, con las incertidumbres y riesgos que ello conlleva.

Desde el punto de vista de qué debe hacer el software, la arquitectura se define a partir de un conjunto de requisitos críticos funcionales, de rendimiento, o de calidad [14]. Considerando cómo el software debe dar solución a tales objetivos, la arquitectura constituye el problema central de diseño, es decir, el conjunto de estructuras, clases y atributos principales del software y sus interfaces de comunicación. Desde otro punto de vista más tangible, la arquitectura se materializa en el conjunto de componentes de código fuente y ejecutables que implementan dicho esqueleto, lo que posibilita demostrar y evaluar en qué medida el diseño da solución a aquellos requisitos críticos.

Dado que no existe una teoría establecida sobre cómo proceder, el diseño, implementación y evaluación de la arquitectura de un software complejo puede realizarse a través de un proceso iterativo de prototipado, demostración y corrección. Este proceso permite atender y resolver en los inicios del proyecto los riesgos asociados a los requisitos más críticos y a las decisiones de diseño más difíciles, que son aquellos que más pueden comprometer el éxito del proyecto. Así, el equipo de desarrollo debe diseñar, construir y estabilizar primero la arquitectura del software antes de diseñar e implementar el conjunto de componentes elementales que se integran en la arquitectura y que aportan las funcionalidades finales de usuarios.

La esencia del principio de priorizar la arquitectura es dedicar los mínimos esfuerzos a garantizar la corrección de las partes más importantes, costosas e indefinidas del sistema, y cuyo prototipo permita una demostración tangible de la viabilidad del proyecto.

2.1. Software e ingeniería de software

Es común darse cuenta que la invención de una tecnología puede tener efectos profundos e inesperados con otras tecnologías con las que en apariencia no tienen ninguna relación, como en empresas comerciales, en personas y aún en la cultura en general.

En la actualidad, el software es la tecnología individual más importante en el ámbito mundial. Nadie en la década de los 50 podría haber predicho que el software se convertiría en una tecnología indispensable en los negocios, la ciencia y la ingeniería; que el software permitiría la creación de nuevas tecnologías, la expansión de tecnologías

antiguas como la industria de la impresión; o que gracias a ella se construiría una gran red llamada internet cubriendo cambiando todo; desde la investigación bibliográfica hasta las compras de los consumidores y los hábitos diarios de los jóvenes.

En la actualidad la industria del software se ha convertido en un factor dominante de la economía del mundo. El programador solitario de la era inicial ha sido sustituido por equipos especialistas en desarrollo de software, en los que cada uno se enfoca en una parte de la tecnologíarequerida para desarrollar una aplicación compleja o dar soporte a aplicaciones existentes.

Pese a que en la actualidad se trabajan con equipos especializados en el desarrollo de software aún las empresas se realizan las siguientes preguntas:

- ¿Por qué tarda tanto la obtención del software?
- ¿Por qué son tan altos los costos de desarrollo?
- ¿Por qué es imposible encontrar todos los errores en el software antes de entregarlo a los clientes?
- ¿Por qué se gasta tanto tiempo y esfuerzo en el mantenimiento de los programas existentes?
- ¿Por qué es difícil medir el progreso al desarrollar y darle mantenimiento al software?

Estas y muchas otras preguntas demuestran la preocupación de la industria por el software y por la manera en que este se desarrolla; una preocupación que ha conducido a la adopción de la práctica de la ingeniería de software.

La Ingeniería de Software es un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación, mantenimiento y retiro del software; la Ingeniería de Software es la rama de la ingeniería que aplica los principios de la computación y las matemáticas para lograr soluciones costo-efectivas, es decir, permite elaborar consistentemente productos correctos y utilizables.

El proceso de ingeniería de software se define como un conjunto de etapas parcialmente ordenadas con la intención de lograr un objetivo, en este caso, la obtención de un producto de software de calidad.El proceso de desarrollo de software es aquel en que las necesidades del usuario son traducidas en requerimientos del software, estos son transformados en diseño y el diseño implementado en código, este es probado, documentado y certificado para su uso operativo. Concretamente define quién está haciendo qué, cuándo hacerlo y cómo alcanzar un cierto objetivo.

El proceso de desarrollo de software requiere por un lado un conjunto de conceptos, una metodología y un lenguaje propio, a este proceso también se le llama el ciclo de vida del software [2].

2.1.1. Ciclo de vida del software

El ciclo de vida define el estado de las fases a través de las cuales se mueve un proyecto de desarrollo de software, determina el orden de las fases involucradas y los criterios de transición asociadas entre las mismas.

Un modelo de ciclo de vida del software presenta las siguientes características:

- Describe las fases principales de desarrollo de software.
- Define las fases primarias esperadas de ser ejecutadas durante esas fases.
- Ayuda a administrar el progreso del desarrollo, y
- Provee un espacio de trabajo para la definición de un detallado proceso de desarrollo de software.

Así, los modelos por una parte suministran una guía para los ingenieros de software con el fin de ordenar las diversas actividades técnicas en el proyecto, por otra parte suministran un marco para la administración del desarrollo y el mantenimiento, en el sentido en que permiten estimar recursos, definir puntos de control intermedios, monitorear el avance, etc.

Usualmente en el ciclo de vida de un sistema se consideran las siguientes etapas:

- **Análisis**: Define el alcance del proyecto y se desarrolla un caso de negocio; La etapa de análisis en el ciclo de vida del software corresponde al proceso mediante

el cual se intenta descubrir qué es lo que realmente se necesita es decir las características que el sistema debe poseer.

- **Diseño**: Define un plan del proyecto, especifica las características y fundamenta la arquitectura. La fase del diseño se divide en dos partes: *Diseño global* o arquitectónico y *diseño detallado*. En el diseño arquitectónico se transforma los requisitos en una arquitectura de alto nivel, y se crea una estructura modular y se representan las relaciones de control entre los módulos. Se trata de no centrarse en los detalles y código de los procedimientos sino en centrarse en la arquitectura del software que permita obtener una visión general del software. En el diseño detallado para cada módulo se refina el diseño, se definen los requisitos del módulo y su documentación.
- **Desarrollo**: Construye el software con las herramientas adecuadas, un entorno de desarrollo que facilite nuestro trabajo y un lenguaje de programación apropiado para el tipo de sistema que vayamos a construir [12].
- **Despliegue**: Es la puesta en producción del software.
- **Pruebas**: Se comprueba que se cumplen criterios de corrección y calidad.
- **Mantenimiento**: Es el proceso de mejora y optimización y adaptación a nuevos requisitos del software desplegado.

Evolución del software

El software evoluciona a través del tiempo, sin importar su dominio de aplicación, tamaño o complejidad. El cambio que con frecuencia es llamado mantenimiento de software conduce este proceso, y se presenta cuando se corrigen errores, cuando el software se

adapta a un nuevo ambiente, cuando el cliente solicita características o funciones nuevas, y cuando la aplicación experimenta una reingeniería para proporcionar beneficios en un contexto moderno.

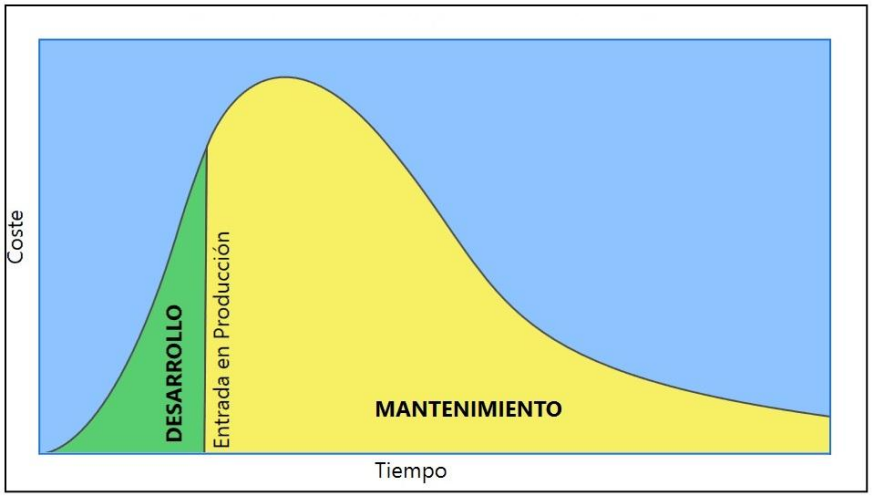


Figura II. 1.Curva de coste de aplicaciones informáticas

La etapa de mantenimiento consume típicamente del 40% al 80% de los recursos de una empresa de desarrollo de software. De hecho, con un 60% de media, es probablemente la etapa más importante del ciclo de vida del software. Dada la naturaleza del software, que ni se rompe ni se desgasta con el uso, su mantenimiento incluye tres facetas diferentes:

- Eliminar los defectos que se detecten durante su vida útil (mantenimiento correctivo).
- Adaptarlo a nuevas necesidades (mantenimiento adaptativo), cuando el sistema ha de funcionar sobre una nueva versión del sistema operativo o en un entorno hardware diferente.

- Añadirle nueva funcionalidad (mantenimiento perfectivo), cuando se proponen características deseables que supondrían una mejora del sistema ya existente.

De las distintas facetas del mantenimiento, la eliminación de defectos sólo supone el 17% del coste de mantenimiento de un sistema, mientras que el diseño e implementación de mejoras es responsable del 60% del coste de mantenimiento. Es decir, más de un tercio del coste total del software se emplea en añadirle características a software ya existente. La corrección de errores supone, en contraste, "sólo" en torno al 10% del coste total del software.

Se ha observado que, cuanto mejor sea el software, más tendremos que invertir en su mantenimiento, aun cuando se emplee menos esfuerzo en corregir defectos. Este hecho, que puede parecer paradójico, se debe simplemente, a que nuestro sistema se usará más veces, de las que no habíamos previsto. Por tanto, las propuestas de modificación y mejora serán numerosas.

Durante la etapa de mantenimiento, se repiten todas las etapas que ya hemos visto del ciclo de vida del software. Al tratar principalmente de cómo añadirle nueva funcionalidad a un sistema ya existente, el mantenimiento repite "en miniatura" el ciclo de vida del software completo, añadiendo una nueva tarea que es el comprender el sistema existente, por lo que se podría decir que el mantenimiento de un sistema es más difícil que su desarrollo.

Debido a que el mantenimiento de software demanda una gran cantidad de tiempo y recursos tanto para el desarrollo como recursos económicos, es de gran importancia diseñar una arquitectura robusta, escalable y que se adapte a futuros cambios por avances en la tecnología o nuevos requerimientos de negocio. Es aquí en donde cumple un papel muy importante la arquitectura de software pues es aquí en donde se define, de manera abstracta, los componentes que llevan a cabo alguna tarea, sus interfaces y la comunicación entre ellos.

2.2. Arquitectura de software

La arquitectura de softwarees el proceso por el cual se define una solución para los requisitos técnicos y operacionales del mismo. Este proceso define qué componentes forman el sistema, cómo se relacionan entre ellos, y cómo mediante su interacción llevan a cabo la funcionalidad especificada, cumpliendo con criterios de calidad, seguridad, disponibilidad, usabilidad, mantenibilidad [14].

Durante el diseño de la arquitectura se tratan los temas que pueden tener un impacto importante en el éxito o fracaso de nuestro sistema. Algunas preguntas que hay que hacerse al respecto son:

- ¿En qué entorno va a ser desplegado nuestro sistema?
- ¿Cómo va a ser nuestro sistema puesto en producción?
- ¿Cómo van a utilizar los usuarios nuestro sistema?

- ¿Qué otros requisitos debe cumplir el sistema? (seguridad, rendimiento, concurrencia, configuración.)
- ¿Qué cambios en la arquitectura pueden impactar al sistema ahora o una vez desplegado?

Importancia

En el marco de la ingeniería del software, el proceso de diseño de la arquitectura juega un papel muy importante.

La diferencia entre un buen proceso de diseño arquitectural y uno malo puede suponer el fracaso o éxito de nuestro proyecto.

En el diseño de la arquitectura tratamos los temas más importantes a la hora de definir nuestro sistema, es decir, creamos un molde básico de nuestra aplicación dentro del cual se decide:

- Qué tipo de aplicación se va a construir. (Web, RIA, Rich Client)
- Qué estructura lógica va a tener la aplicación (N-Capas, Componentes)
- Qué estructura física va a tener la aplicación (Cliente/Servidor, N-Tier)
- Qué riesgos hay que afrontar y cómo hacerlo. (Seguridad, Rendimiento, Flexibilidad)
- Qué tecnologías vamos a usar (WCF,WF,WPF, Silverlight, Entity Framework)

La falta de un diseño solido de arquitectura o en su defecto no apreciar las consecuencias a largo plazo son decisiones clave que pueden poner en riesgo el desarrollo de un proyecto.

Las herramientas y plataformas modernas ayudan a simplificar la tarea de creación de aplicaciones, pero no remplazan la necesidad de diseñar una aplicación cuidadosamente, basada en los requerimientos y escenario especifico.

El riesgo expuesto por la falta de una arquitectura solida da como consecuencia software inestable, incapaz de soportar los requerimientos de negocio actuales o futuros dificultando la implementación o gestión en un entorno de producción.

EL diseño y desarrollo de sistemas deben tener en cuenta las consideraciones: para el usuario, el sistema (la infraestructura), y los objetivos de negocio. Para cada una de estas áreas se debe identificar los escenarios clave, atributos de calidad importante(ej. la fiabilidad o escalabilidad) y las áreas clave de la satisfacción e instalación.

Objetivos

La arquitectura de aplicaciones busca construir un puente entre los requerimientos de negocio ylos requerimientos técnicos mediante la comprensión de casos de uso, y luego buscar la manera de aplicar esos casos de uso dentro del software.

El objetivo de la arquitectura de software es identificar los requerimientos que afectan a la estructura de la aplicación. Un buen diseño de arquitectura reduce los riesgos de negocio asociados con la construcción técnica de la solución, siendo lo suficientemente flexible y capaz de soportar cambios que se producen con el tiempo en hardware y software de la tecnología, así como en escenarios de usuario y requisitos.

Un arquitecto debe tener en cuenta el efecto general de las decisiones de diseño, las ventajas y desventajas inherentes entre los atributos de calidad (como son el rendimiento y la seguridad), y las compensaciones necesarias para hacer frente a los usuarios, el sistema y los requerimientos del negocio.

Y sobre todo mantener en mente que la arquitectura puede:

- Exponer la estructura del sistema, pero ocultar los detalles de la implementación.
- Realizar todos los casos de uso y escenarios.
- Hacer frente a las exigencias de las diferentes partes interesadas.
- Manejar los requisitos funcionales y de calidad.

2.3. ¿Qué es un estilo de arquitectura?

Para diseñar la arquitectura de un sistema es importante tener en cuenta los intereses de los distintos agentes que participan. Estos agentes son los usuarios del sistema, el propio sistema y los objetivos del negocio. Cada uno de ellos impone requisitos y restricciones

que deben ser tenidos en cuenta en el diseño de la arquitectura y que pueden llegar a entrar en conflicto, por lo que se debe alcanzar un compromiso entre los intereses de cada participante.

Para los usuarios es importante que el sistema responda a la interacción de una forma fluida, mientras que para los objetivos del negocio es importante que el sistema cueste poco. Los usuarios pueden querer que se implemente primero una funcionalidad útil para su trabajo, mientras que el sistema puede tener prioridad en que se implemente la funcionalidad que permita definir su estructura.

El trabajo del arquitecto es delinear los escenarios y requisitos de calidad importantes para cada agente así como los puntos clave que debe cumplir y las acciones o situaciones que no deben ocurrir.

El objetivo final de la arquitectura es identificar los requisitos que producen un impacto en la estructura del sistema y reducir los riesgos asociados con la construcción del sistema. La arquitectura debe soportar los cambios futuros del software, del hardware y de funcionalidad demandada por los clientes. Del mismo modo, es responsabilidad del arquitecto analizar el impacto de sus decisiones de diseño y establecer un compromiso entre los diferentes requisitos de calidad así como entre los compromisos necesarios para satisfacer a los usuarios, al sistema y los objetivos del negocio.

En síntesis, la arquitectura debería:

- Mostrar la estructura del sistema pero ocultar los detalles.
- Realizar todos los casos de uso.
- Satisfacer en la medida de lo posible los intereses de los agentes.
- Ocuparse de los requisitos funcionales y de calidad.
- Determinar el tipo de sistema a desarrollar.
- Determinar los estilos arquitecturales que se usarán.
- Tratar las principales cuestiones transversales.

Una vez vistas las principales cuestiones que debe abordar el diseño de la arquitectura del sistema, ahora vamos a ver los pasos que deben seguirse para realizarlo. En una metodología ágil como Scrum, la fase de diseño de la arquitectura comienza durante en el pre-juego (Pre-game) o en la fase de Inicio (Inception) en RUP, en un punto donde ya hemos capturado la visión del sistema que queremos construir.

En el diseño de la arquitectura lo primero que se decide es el tipo de sistema o aplicación que vamos a construir.

Los principales tipos son: aplicaciones móviles, de escritorio, RIAs (Rich Internet Application), aplicaciones de servicios, aplicaciones web. Es importante entender que el tipo de aplicación viene determinado por la topología de despliegue y los requisitos y restricciones indicadas en los requisitos.

La selección de un tipo de aplicación determina en cierta medida el estilo arquitectural que se va a usar. El estilo arquitectural es en esencia la partición más básica del sistema en

bloques y la forma en que se relacionan estos bloques. Los principales estilos arquitecturales son: Cliente/Servidor, Sistemas de Componentes, Arquitectura en capas, MVC, N-Niveles, SOA. Como ya hemos dicho, el estilo arquitectural que elegimos depende del tipo de aplicación. Una aplicación que ofrece servicios lo normal es que se haga con un estilo arquitectural SOA.

Por otra parte, a la hora de diseñar la arquitectura tenemos que entender también que un tipo de aplicación suele responder a más de un estilo arquitectural. Por ejemplo, una página web hecha con ASP.NET MVC sigue un estilo Cliente/Servidor pero al mismo tiempo el servidor sigue un estilo Modelo Vista Controlador.

Tras haber seleccionado el tipo de aplicación y haber determinado los estilos arquitecturales que más se ajustan al tipo de sistema que vamos a construir, tenemos que decidir cómo vamos a construir los bloques que forman nuestro sistema. Por ello el siguiente paso es seleccionar las distintas tecnologías que vamos a usar. Estas tecnologías están limitadas por las restricciones de despliegue y las impuestas por el cliente. Hay que entender las tecnologías como los ladrillos que usamos para construir nuestro sistema. Por ejemplo, para hacer una aplicación web podemos usar la tecnología ASP.NET o para hacer un sistema que ofrece servicios podemos emplear WCF.

Cuando ya hemos analizado nuestro sistema y lo hemos fragmentado en partes más manejables, tenemos que pensar como implementamos todos los requisitos de calidad que tiene que satisfacer. Los requisitos de calidad son las propiedades no funcionales que debe tener el sistema, como por ejemplo la seguridad, la persistencia, la usabilidad, la

mantenibilidad, etc. Conseguir que nuestro sistema tenga estas propiedades va a traducirse en implementar funcionalidad extra, pero esta funcionalidad es ortogonal a la funcionalidad básica del sistema.

Para tratar los requisitos de calidad el primer paso es preguntarse ¿Qué requisitos de calidad requiere el sistema? Para averiguarlo tenemos que analizar los casos de uso.

Una vez hemos obtenido un listado de los requisitos de calidad las siguientes preguntas son: ¿Cómo consigo que mi sistema cumpla estos requisitos? ¿Se puede medir esto de alguna forma? ¿Qué criterios indican que mi sistema cumple dichos requisitos?

Los requisitos de calidad nos van a obligar a tomar decisiones transversales sobre nuestro sistema. Por ejemplo, cuando estamos tratando la seguridad de nuestro sistema tendremos que decidir cómo se autentican los usuarios, como se maneja la autorización entre las distintas capas, etc. De la misma forma tendremos que tratar otros temas como las comunicaciones, la gestión de excepciones, la instrumentación o el cacheo de datos.

Los procesos software actuales asumen que el sistema cambiará con el paso del tiempo y que no podemos saber todo a la hora de diseñar la arquitectura. El sistema tendrá que evolucionar a medida que se prueba la arquitectura contra los requisitos del mundo real. Por eso, no hay que tratar de formalizar absolutamente todo a la hora de definir la arquitectura del sistema. Lo mejor es no asumir nada que no se pueda comprobar y dejar abierta la opción de un cambio futuro. No obstante, sí que existirán algunos aspectos que podrán requerir un esfuerzo a la hora de realizar modificaciones. Para minimizar dichos

esfuerzos es especialmente importante el concepto de desacoplamiento entre componentes. Por ello es vital identificar esas partes de nuestro sistema y detenerse el tiempo suficiente para tomar la decisión correcta. En síntesis las claves son:

- Construir "hasta el cambio" más que "hasta el final".
- Utilizar herramientas de modelado para analizar y reducir los riesgos.
- Utilizar modelos visuales como herramienta de comunicación.
- Identificar las decisiones clave a tomar.

A la hora de crear la arquitectura de nuestro sistema de forma iterativa e incremental, las principales preguntas a responder son:

- ¿Qué partes clave de la arquitectura representan el mayor riesgo si las diseño mal?
- ¿Qué partes de la arquitectura son más susceptibles de cambiar?
- ¿Qué partes de la arquitectura puedo dejar para el final sin que ello impacte en el desarrollo del sistema?
- ¿Cuáles son las principales suposiciones que hago sobre la arquitectura y como las verifico?
- ¿Qué condiciones pueden provocar que tenga que cambiar el diseño?

Como ya hemos dicho, los procesos modernos se basan en adaptarse a los cambios en los requisitos del sistema y en ir desarrollando la funcionalidad poco a poco. En el plano del diseño de la arquitectura, esto se traduce en que definiremos la arquitectura del

sistema final poco a poco. Podemos entenderlo como un proceso de maduración, como el de un ser vivo. Primero tendremos una arquitectura a la que llamaremos línea base y que es una visión del sistema en el momento actual del proceso. Junto a esta línea base tendremos una serie de arquitecturas candidatas que serán el siguiente paso en la maduración de la arquitectura. Cada arquitectura candidata incluye el tipo de aplicación, la arquitectura de despliegue, el estilo arquitectural, las tecnologías seleccionadas, los requisitos de calidad y las decisiones transversales. Las preguntas que deben responder las arquitecturas candidatas son:

- ¿Qué suposiciones he realizado en esta arquitectura?
- ¿Qué requisitos explícitos o implícitos cumple esta arquitectura?
- ¿Cuáles son los riesgos tomados con esta evolución de la arquitectura?
- ¿Qué medidas puedo tomar para mitigar esos riesgos?
- ¿En qué medida esta arquitectura es una mejora sobre la línea base o las otras arquitecturas candidatas?

Dado que usamos una metodología iterativa e incremental para el desarrollo de nuestra arquitectura, la implementación de la misma debe seguir el mismo patrón. La forma de hacer esto es mediante pruebas arquitecturales. Estas pruebas son pequeños desarrollos de parte de la aplicación (Pruebas de Concepto) que se usan para mitigar riesgos rápidamente o probar posibles vías de maduración de la arquitectura. Una prueba arquitectural se convierte en una arquitectura candidata que se evalúa contra la línea base. Si es una mejora, se convierte en la nueva línea base frente a la cual crear y evaluar las nuevas arquitecturas candidatas. Las preguntas que debemos hacerle a una

arquitectura candidata que surge como resultado de desarrollar una prueba arquitectural son:

- ¿Introduce nuevos riesgos?
- ¿Soluciona algún riesgo conocido esta arquitectura?
- ¿Cumple con nuevos requisitos del sistema?
- ¿Realiza casos de uso arquitecturalmente significativos?
- ¿Se encarga de implementar algún requisito de calidad?
- ¿Se encarga de implementar alguna parte del sistema transversal?

Los casos de uso importantes son aquellos que son críticos para la aceptación de la aplicación o que desarrollan el diseño lo suficiente como para ser útiles en la evaluación de la arquitectura.

En resumen, el proceso de diseño de la arquitectura tiene que decidir qué funcionalidad es la más importante a desarrollar. A partir de esta decisión tiene que decidir el tipo de aplicación y el estilo arquitectural, y tomar las decisiones importantes sobre seguridad y, rendimiento… que afectan al conjunto del sistema. El diseño de la arquitectura decide cuales son los componentes más básicos del sistema y como se relacionan entre ellos para implementar la funcionalidad. Todo este proceso debe hacerse paso a paso, tomando solo las decisiones que se puedan comprobar y dejando abiertas las que no. Esto significa mitigar los riesgos rápidamente y explorar la implementación de casos de uso que definan la arquitectura.

2.4. Arquitecturasmonolíticas

El diseño de aplicaciones no es una tarea sencilla, es necesario tomar un gran número de decisiones a nivel de arquitectura, diseño e implementación. Estas decisiones tendrán un impacto en las capacidades de la aplicación en seguridad, escalabilidad, disponibilidad, mantenimiento, entre otras.

En el campo de la arquitectura contemporánea, se considera como arquitecturas monolíticasa aquellas que tienen componentes que son administrados y mostrados por una sola región de la aplicación y cuyo mantenimiento de los componentes puede llegar a afectar el comportamiento lógico y visual de la aplicación [8].

El diseño de una aplicación implica la toma de decisiones sobre su arquitectura lógica y física, así como sobre la tecnología e infraestructura que se emplearán para implementar su funcionalidad. Para tomar estas decisiones, debe tener un conocimiento claro de los procesos que realizará la aplicación (requisitos funcionales), así como los niveles de escalabilidad, disponibilidad, seguridad y mantenimiento necesarios (requisitos no funcionales).

El análisis de la mayoría de las soluciones empresariales basadas en modelos de componentes por capas muestra que existen varios tipos de componentes habituales. Aunque la lista que se muestra en la siguiente figura no es completa, representa los tipos de componentes de software más comunes encontrados en la mayoría de las soluciones desarrolladas.

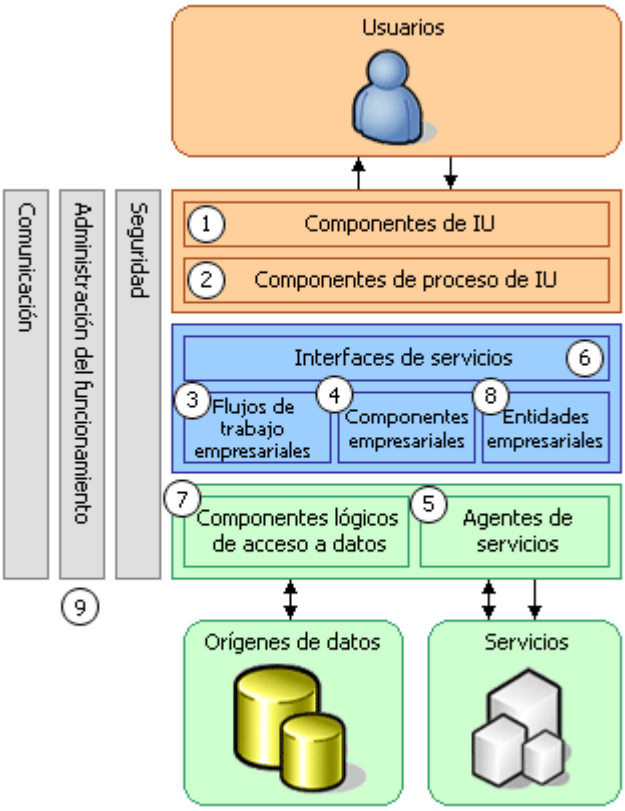


Figura II. 2. Tipos de componentes utilizados en un escenario de ejemplo

Los tipos de componentes identificados en el escenario de diseño de ejemplo son:

Componentes de interfaz de usuario. La mayor parte de las soluciones necesitan ofrecer al usuario un modo de interactuar con la aplicación. Las interfaces de usuario se implementan utilizando formularios de Windows, páginas web, controles u otro tipo de tecnología que permita procesar y dar formato a los datos de los usuarios, así como adquirir y validar los datos entrantes procedentes de éstos.

Componentes de proceso de usuario. En un gran número de casos, la interacción del usuario con el sistema se realiza de acuerdo a un proceso predecible. Por ejemplo, en la aplicación comercial, podríamos implementar un procedimiento que permita ver los datos del producto. De este modo, el usuario puede seleccionar una categoría de una lista de categorías de productos disponibles y, a continuación, elegir uno de los productos de la categoría seleccionada para ver los detalles correspondientes. Del mismo modo, cuando el usuario realiza una compra, la interacción sigue un proceso predecible de recolección de datos por parte del usuario, por el cual éste en primer lugar proporciona los detalles de los productos que desea adquirir, a continuación los detalles de pago y, por último, la información para el envío. Para facilitar la sincronización y organización de las interactuaciones con el usuario, resulta útil utilizar componentes de proceso de usuario individuales. De este modo, el flujo del proceso y la lógica de administración de estado no se incluyen en el código de los elementos de la interfaz de usuario, por lo que varias interfaces podrán utilizar el mismo "motor" de interacción básica.

Interfaces de servicios. Para exponer lógica empresarial como un servicio, es necesario crear interfaces de servicios que admitan los contratos de comunicación (comunicación basada en mensajes, formatos, protocolos, seguridad y excepciones, entre otros) que requieren los clientes. Por ejemplo, el servicio de autorización de tarjetas de crédito debe exponer una interfaz de servicios que describa la funcionalidad que ofrece el servicio, así como la semántica de comunicación requerida para llamar al mismo. Las interfaces de servicios también se denominan fachadas empresariales.

Flujos de trabajo empresariales. Una vez que el proceso de usuario ha recopilado los datos necesarios, éstos se pueden utilizar para realizar un proceso empresarial. Por ejemplo, tras enviar los detalles del producto, el pago y el envío a la aplicación comercial, puede comenzar el proceso de cobro del pago y preparación del envío. Gran parte de los procesos empresariales conllevan la realización de varios pasos, los cuales se deben organizar y llevar acabo en un orden determinado. Por ejemplo, el sistema empresarial necesita calcular el valor total del pedido, validar la información de la tarjeta de crédito, procesar el pago de la misma y preparar el envío del producto. El tiempo que este proceso puede tardar en completarse es indeterminado, por lo que sería preciso administrar las tareas necesarias, así como los datos requeridos para llevarlas a cabo. Los flujos de trabajo empresariales definen y coordinan los procesos empresariales de varios pasos de ejecución larga y se pueden implementar utilizando herramientas de administración de procesos empresariales, como BizTalk Server Orchestration u otras.

Componentes empresariales. Independientemente de si el proceso empresarial consta de un único paso o de un flujo de trabajo organizado, la aplicación requerirá probablemente el uso de componentes que implementen reglas empresariales y realicen tareas empresariales. Por ejemplo, en la aplicación se deberá implementar una funcionalidad que calcule el precio total del pedido y agregue el costo adicional correspondiente por el envío del mismo. Los componentes empresariales implementan la lógica empresarial de la aplicación.

Componentes de entidad empresarial. La mayoría de las aplicaciones requieren el paso de datos entre distintos componentes. Por ejemplo, en una aplicación comercial es

necesario pasar una lista de productos de los componentes lógicos de acceso a datos a los componentes de la interfaz de usuario para que éste pueda visualizar dicha lista. Los datos se utilizan para representar entidades empresariales del mundo real, como productos o pedidos. Las entidades empresariales que se utilizan de forma interna en la aplicación suelen ser estructuras de datos, como conjuntos de datos, DataReader o secuencias de lenguaje de marcado extensible (XML), aunque también se pueden implementar utilizando clases orientadas a objetos personalizadas que representan entidades del mundo real necesarias para la aplicación, como productos o pedidos.

Componentes lógicos de acceso a datos. La mayoría de las aplicaciones y servicios necesitan obtener acceso a un almacén de datos en un momento determinado del proceso empresarial. Por ejemplo, la aplicación empresarial necesita recuperar los datos de los productos de una base de datos para mostrar al usuario los detalles de los mismos, así como insertar dicha información en la base de datos cuando un usuario realiza un pedido.

Por tanto, es razonable abstraer la lógica necesaria para obtener acceso a los datos en una capa independiente de componentes lógicos de acceso a datos, ya que de este modo se centraliza la funcionalidad de acceso a datos y se facilita la configuración y el mantenimiento de la misma.

Agentes de servicios. Cuando un componente empresarial requiere el uso de la funcionalidad proporcionada por un servicio externo, tal vez sea necesario hacer uso de código para administrar la semántica de la comunicación con dicho servicio. Por ejemplo,

los componentes empresariales de la aplicación comercial descrita anteriormente podrían utilizar un agente de servicios para administrar la comunicación con el servicio de autorización de tarjetas de crédito y utilizar un segundo agente de servicios para controlar las conversaciones con el servicio de mensajería.

Los agentes de servicios permiten aislar las idiosincrasias de las llamadas a varios servicios desde la aplicación y pueden proporcionar servicios adicionales, como la asignación básica del formato de los datos que expone el servicio al formato que requiere la aplicación.

Componentes de seguridad, administración operativa y comunicación. La aplicación probablemente utilice también componentes para realizar la administración de excepciones, autorizar a los usuarios a que realicen tareas determinadas y comunicarse con otros servicios y aplicaciones.

2.4.1. Diseño de la capa de presentación

La capa de presentación como se muestra en la figura II.3 contiene los componentes necesarios para que el usuariointeractúe con la aplicación. Las capas de presentación más simples contienen componentes de interfaz, como formularios Windows o formularios Web. Las interacciones más complejas conllevan el diseño de componentes de proceso de usuario que permiten organizar los elementos de la interfaz y controlar la interacción con el usuario.

Los componentes de proceso de usuario resultan especialmente útiles cuando la interacción del usuario sigue una serie de pasos predecibles, como al utilizar un asistente para realizar una tarea determinada.

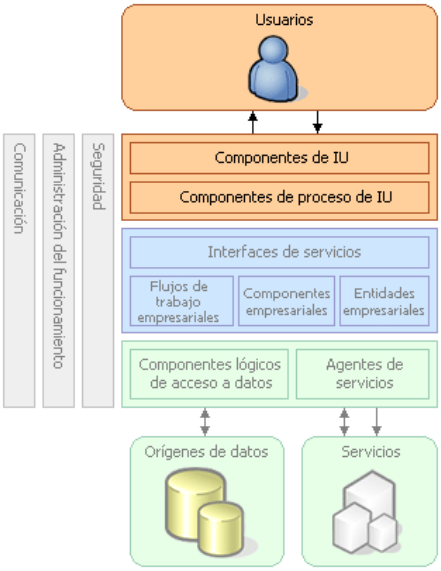


Figura II. 3. Componentes de la capa de presentación

2.4.1.1. Diseño de componentes de interfaz de usuario

La implementación de interfaces de usuario se puede llevar a cabo de varias formas. Por ejemplo, una interfaz Windows o Web, que incluyen procesamiento de voz, o interfaz para dispositivos móviles, entre otros. Los componentes de la interfaz de usuario administran la interacciónconcon el usuario, muestra, obtienen datos, e interpretan los eventos generados por el usuario.

Las interfaces de usuario como se muestra en la figura II.4 constan normalmente de varios elementos gráficos como controles de usuario, controles personalizados, formularios o páginas que permiten mostrar datos y aceptar la entrada del usuario. Por

ejemplo, una aplicación puede contener un control DataGrid que muestre una lista de categorías de productos y un control de botón de comando que indica que el usuario desea ver los productos de la categoría seleccionada.

Cuando un usuario interactúa con un elemento de la interfaz, se genera un evento que llama al código de una función de control. Ésta, a su vez, llama a componentes empresariales, componentes lógicos de acceso a datos o componentes de proceso de usuario para implementar la acción deseada y recuperar los datos que se han de mostrar.

Figura II. 4. Diseño de interfaz de usuario

Los componentes de la interfaz de usuario deben mostrar datos al usuario, obtener y validar los datos procedentes del mismo e interpretar las acciones de éste que indican que desea realizar una operación con los datos. Asimismo, la interfaz debe filtrar las acciones disponibles con el fin de permitir al usuario realizar sólo aquellas operaciones que le sean necesarias en un momento determinado.

2.4.1.2. Acceso a componentes lógicos de acceso a datos desde la interfaz de usuario

Las interfaces de usuario de determinadas aplicaciones necesitan procesar los datos disponibles como consultas expuestas por los componentes lógicos de acceso a datos. Independientemente de si los componentes de la interfaz de usuario invocan directamente a los componentes lógicos de acceso a datos.

El acceso directo a los componentes lógicos de acceso a datos desde la interfaz de usuario parece contradecir el concepto de disposición en capas. No obstante, resulta útil en este caso considerar a la aplicación como un servicio homogéneo; se llama a la aplicación y ésta decide cuáles son los componentes internos más adecuados para responder a una solicitud determinada.

Se recomienda permitir a los componentes lógicos de acceso a datos el acceso directo a los componentes de la interfaz de usuario cuando:

- Está dispuesto a relacionar estrechamente los métodos y esquemas de acceso a datos con la semántica de la interfaz de usuario. Esta relación requiere el mantenimiento conjunto de los cambios de la interfaz de usuario y de los esquemas.
- La implementación física coloca juntos a los componentes lógicos de acceso a datos y a los componentes de interfaz de usuario, lo que permite obtener datos en formatos de secuencias (como DataReader) de los componentes lógicos de acceso a datos, que se pueden enlazar directamente a la salida de las interfaces de usuario para obtener un mayor rendimiento.

2.4.1.3. Diseño de componentes de proceso de usuario

La interacción del usuario con la aplicación puede seguir un proceso predecible. Para facilitar la coordinación del proceso de usuario y controlar el mantenimiento del estado requerido al visualizar varias páginas o formularios de la interfaz de usuario, puede crear componentes de proceso de usuario.

Los componentes de proceso de usuario se implementan normalmente como clases que exponen métodos a los cuales pueden llamar las interfaces de usuario. Cada método encapsula la lógica necesaria para realizar una acción específica en el proceso de usuario. La interfaz de usuario crea una instancia del componente del proceso de usuario y la utiliza para efectuar la transición a través de los pasos del proceso.

El diseño de componentes de proceso de usuario para su uso por parte de varias interfaces da lugar a una implementación más compleja, debido al aislamiento de los problemas específicos de los dispositivos. No obstante, puede facilitar la distribución del trabajo de desarrollo de la interfaz de usuario entre varios equipos, cada uno de ellos utilizando el mismo componente de proceso de usuario.

Los componentes de proceso de usuario coordinan la visualización de los elementos de la interfaz. Se abstraen de la funcionalidad de procesamiento y adquisición de datos proporcionada por los componentes de la interfaz de usuario.

La separación de la funcionalidad de interacción del usuario en componentes de interfaz y proceso de usuario conlleva las siguientes ventajas:

- El estado de la interacción de usuario de ejecución larga se mantiene más fácilmente, lo que permite el abandono y la reanudación de la interacción, probablemente incluso utilizando una interfaz de usuario diferente.
- Varias interfaces de usuario pueden utilizar el mismo proceso.

El uso de un enfoque sin estructura para diseñar la lógica de la interfaz de usuario puede dar lugar a situaciones no deseadas, debido al aumento del tamaño de la aplicación o la incorporación de nuevos requisitos. Si necesita agregar una interfaz de usuario específica para un dispositivo determinado, tal vez deba volver a diseñar el flujo de datos y la lógica de control.

En la figura II.5 se muestra el modo en que la interfaz y el proceso de usuario se pueden abstraer el uno del otro.

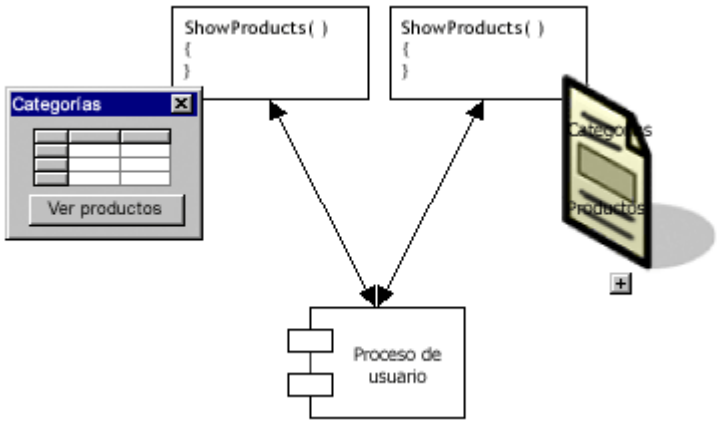


Figura II. 5. Interfaces de usuario y componentes de proceso de usuario

Los componentes de proceso de usuario ayudan a resolver los siguientes problemas de diseño de interfaces de usuario:

Control de actividades de usuarios concurrentes. Determinadas aplicaciones permiten a los usuarios realizar varias tareas a la vez poniendo a su disposición más de un elemento de interfaz.

Los componentes de proceso de usuario simplifican la administración del estado de varios procesos salientes encapsulando todo el estado necesario para el proceso en un solo componente.

Uso de varios paneles para una actividad. Si utiliza varias ventanas o paneles en una actividad de usuario determinada, es importante mantenerlos sincronizados, los componentes de proceso de usuario facilitan la implementación de este tipo de interfaz a través de la centralización del estado de todas las ventanas en una única ubicación.

2.4.2. Diseño de capas empresariales

La parte más importante de la aplicación es la funcionalidad que proporciona. Una aplicación realiza un proceso empresarial que consta de una o varias tareas. En los casos más simples, cada tarea se puede encapsular en un método de un componente y llamar de forma sincrónica o asincrónica. Para los procesos empresariales más complejos que requieren varios pasos y transacciones de ejecución larga, la aplicación necesita disponer de un modo de organizar las tareas empresariales y almacenar el estado hasta que el proceso se haya completado.

Puede diseñar la lógica en las capas empresariales para su uso directo por parte de componentes de presentación o su encapsulación como servicio y llamada a través de una interfaz de servicios, que coordina la conversación asincrónica con los llamadores del servicio e invoca el flujo de trabajo de los componentes empresariales, como se muestra en la figura II.6. La parte principal de la lógica empresarial se suele denominar lógica de negocio.

Los componentes empresariales también pueden realizar solicitudes de servicios externos, en cuyo caso tal vez sea preciso implementar agentes de servicios para administrar la conversación requerida para la tarea empresarial específica realizada por cada uno de los servicios que necesita utilizar.

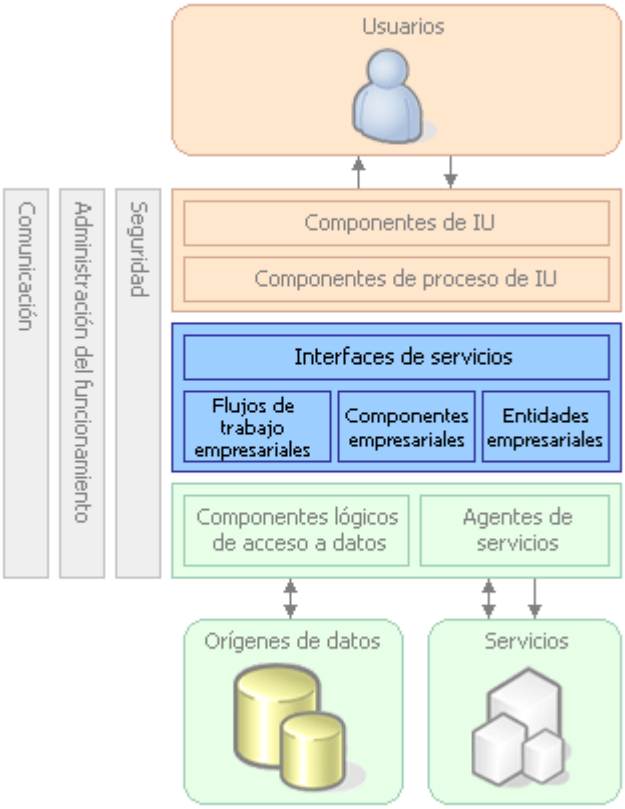


Figura II. 6. Capas de componentes empresariales

2.4.2.1. Componentes y flujos de trabajo empresariales

Al implementar la lógica empresarial, es necesario decidir si es preciso organizar o no el proceso empresarial, o si será suficiente con disponer de un conjunto de componentes empresariales.

Se recomienda el uso de flujos de trabajo empresariales para:

- Administrar un proceso que conlleve varios pasos y transacciones de ejecución larga.
- Exponer una interfaz que implemente un proceso empresarial que habilite la aplicación para establecer una conversación o un contrato con otros servicios.

Puede implementar el proceso empresarial utilizando sólo componentes empresariales cuando:

- No necesite mantener el estado de la conversación más allá de la actividad empresarial, y la funcionalidad empresarial se pueda implementar como una única transacción atómica.
- Necesite encapsular funcionalidad y lógica que se pueda volver a utilizar por parte de numerosos procesos empresariales.
- La lógica empresarial que se debe implementar necesita una gran cantidad de programación, o el control detallado de las estructuras de datos y API.

- Necesita disponer del control total de los datos y del flujo de lógica.

En el ejemplo comercial, el pedido conlleva varios pasos (la autorización de la tarjeta de crédito, el procesamiento del pago y la organización de la entrega, entre otros), los cuales se deben realizar en una secuencia concreta.

El enfoque de diseño más adecuado para este tipo de proceso empresarial es crear componentes empresariales para encapsular cada uno de los pasos individuales en el proceso y organizar dichos componentes utilizando un flujo empresarial.

2.4.2.2. Diseño de componentes empresariales

Los componentes empresariales pueden ser la raíz de las transacciones atómicas. Éstos implementan las reglas empresariales en diversos patrones y aceptan y devuelven estructuras de datos simples o complejos. Los componentes empresariales deben exponer funcionalidad de modo que sea independiente de los almacenes de datos y los servicios necesarios para realizar la tarea, y se deben componer de forma coherente desde el punto de vista del significado y transaccional.

Normalmente, la lógica empresarial evoluciona y aumenta, proporcionando lógica y operaciones de mayor nivel que encapsulan la lógica preexistente. En un gran número de casos, necesitará componer funcionalidad empresarial preexistente con el fin de realizar la lógica empresarial requerida. Al componer lógica empresarial, debe prestar especial atención a la evolución de las transacciones.

Si el proceso empresarial invocará a otros procesos empresariales en el contexto de una transacción atómica, todos los procesos invocados deben garantizar que sus operaciones participan en la transacción existente de modo que las operaciones se deshagan en caso de que la lógica empresarial que realiza las llamadas se interrumpa. Una técnica muy segura es volver a intentar una operación atómica si ésta da error, sin miedo a que los datos pierdan su coherencia. Considere el límite de una transacción determinada como un límite de reintento.

Si no puede implementar transacciones atómicas, necesitará ofrecer métodos y procesos de compensación. Tenga en cuenta que una acción de compensación no devuelve necesariamente todos los datos de la aplicación a su estado anterior, sino que restaura los datos empresariales a un estado coherente.

2.4.2.3. Uso de fachadas empresariales con interfaces de servicios

El canal o mecanismo de comunicaciones que utilice para exponer la lógica empresarial como un servicio puede tener una forma asociada de implementar el código de la interfaz de servicios. Por ejemplo, si decide crear servicios Web, la mayor parte de la lógica de la interfaz de servicios residirá en el propio servicio Web.

Si se pretende crear un sistema que se pueda invocar a través de mecanismos diferentes, debe agregar una fachada entre la lógica empresarial y la interfaz de servicios. Al implementar esta fachada, puede consolidar en una ubicación el código relacionado con las políticas (como la autorización, la auditoría y las validaciones, entre otros) de modo que se pueda utilizar por parte de varias interfaces de servicios que traten con diversos

canales. Esta fachada ofrece una mayor facilidad de mantenimiento debido a que aísla los cambios en los mecanismos de comunicación de la implementación de los componentes empresariales. A continuación el código de la interfaz de servicios trata con los detalles del mecanismo o el canal de comunicación y define el contexto adecuado para la invocación del componente de fachada empresarial.

2.4.2.4. Representar de datos y pasarlos a través de niveles

Cuando los componentes lógicos de acceso a datos devuelven datos, pueden hacerlo en varios formatos. Los formatos pueden variar desde un formato centrado en datos (por ejemplo, una cadena XML) hasta un formato más orientado a objetos (por ejemplo, un componente personalizado que encapsula una instancia de una entidad empresarial).

El formato de datos que elija dependerá del modo en que quiera trabajar con los mismos. Se recomienda evitar los diseños que requieran la transferencia de datos en un formato orientado a objetos personalizado, ya que ello requiere la implementación de serialización personalizada y puede repercutir negativamente en el rendimiento. Generalmente, debería utilizar un formato más centrado en datos, como conjunto de datos, para pasar los datos de los componentes lógicos de acceso a datos a las capas empresariales y, a continuación, utilizarlos para hacer uso de una entidad empresarial personalizada si desea trabajar con los datos de un modo orientado a objetos. En un gran número de casos, sin embargo, resultará más fácil trabajar sólo con los datos empresariales contenidos en un conjunto de datos.

2.4.2.5. Representación de datos con componentes de entidades empresariales personalizadas

En la mayoría de los casos, se recomienda que trabaje directamente con datos, utilizando los conjuntos de datos. Esto permite también pasar los datos estructurados entre las distintas capas de la aplicación sin tener que escribir código personalizado. No obstante, si desea encapsular todos los detalles del uso de un formato en particular o si desea agregar comportamientos a los datos, tal vez deba desarrollar componentes personalizados. De este modo, obtendrá control total sobre lo que los componentes de la aplicación pueden hacer con los datos, permitiéndole abstraer formatos internos de los esquemas de datos utilizados por la aplicación, así como agregar comportamiento a los datos.

Los componentes de entidad empresarial personalizados no son una parte obligatoria de todas las aplicaciones. Un gran número de soluciones no utilizan representaciones personalizadas de entidades empresariales, sino que en su lugar utilizan conjuntos de datos o documentos XML, ya que proporcionan toda la información necesaria y el modelo de desarrollo se basa más en tareas y documentos que el orientado a objetos.

2.4.3. Diseño de capas de datos

Casi todas las aplicaciones y servicios necesitan almacenar y obtener acceso a un determinado tipo de datos. Por ejemplo, almacenar datos de productos, clientes y pedidos.

Al trabajar con datos debe determinar:

- El almacén de datos que utiliza.
- El diseño de los componentes utilizados para obtener acceso al almacén de datos.
- El formato de los datos pasados entre componentes y el modelo de programación necesario para ello

La aplicación o servicio puede disponer de uno o varios orígenes de datos, los cuales pueden ser de tipos diferentes. La lógica utilizada para obtener acceso a los datos de un origen de datos se encapsulará en componentes lógicos de acceso a datos que proporcionan los métodos necesarios para la consulta y actualización de datos. Los datos con los que la lógica de la aplicación debe trabajar están relacionados con entidades del mundo empresarial que forman parte de la empresa. En determinados escenarios, puede disponer de componentes personalizados que representan estas entidades, mientras que en otros puede decidir trabajar con datos utilizando directamente conjuntos de datos.

En la figura II.7 se muestra cómo la capa de datos lógicos de una aplicación consta de uno o varios almacenes de datos y describe una capa de componentes lógicos de acceso a datos utilizados para recuperar y manipular los datos en dichos almacenes.

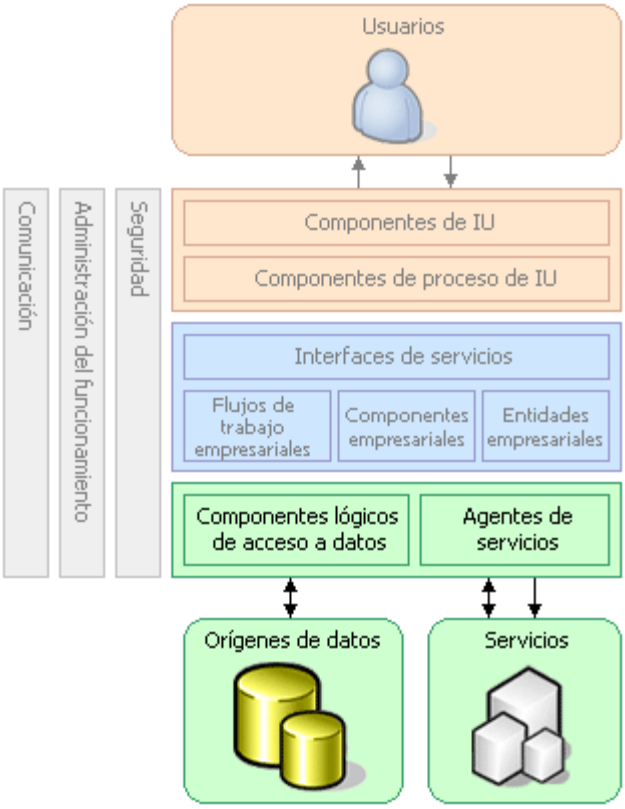


Figura II. 7. Componentes de datos

La mayoría de las aplicaciones utilizan una base de datos relacional como almacén principal de los datos de la aplicación. También se puede utilizar servicios web, bases de datos heredadas etc.

2.4.3.1. Almacenes de datos

Entre los tipos de almacenes habituales se encuentran:

- **Bases de datos relacionales**. Las bases de datos relacionales, como las bases de datos SQL Server, proporcionan funcionalidad de administración de un gran volumen de datos transaccionales de alto rendimiento con capacidades de seguridad, operaciones y transformación de datos. Las bases de datos relacionales también alojan instrucciones y funciones complejas de lógica de datos en forma de almacenamientos almacenados que se pueden utilizar como un entorno eficaz para los procesos empresariales con un gran volumen de datos.
- **Sistema de archivos**. Puede decidir almacenar los datos en sus propios archivos en el sistema de archivos. Estos archivos pueden presentar su propio formato o el formato XML con un esquema definido para los propósitos de la aplicación.

Hay un gran número de otros almacenes como bases de datos XML, servicios de procesamiento analítico en línea y bases de datos de almacenamiento de datos, entre otros.

2.4.3.2. Componentes lógicos de acceso a datos

Independientemente del almacén de datos utilizado, la aplicación o el servicio utilizarán componentes lógicos de acceso a datos para obtener acceso a los datos. Estos componentes abstraen la semántica del almacén de datos subyacente y la tecnología de acceso a datos (como Entity Framework) y proporcionan una interfaz simple de programación para la recuperación y realización de operaciones con datos.

Los componentes lógicos de acceso a datos suelen implementar un patrón de diseño sin estado que separa el procesamiento empresarial de la lógica de acceso a datos. Cada

uno de estos componentes suele proporcionar métodos para realizar operaciones CRUD relacionadas con una entidad empresarial determinada de la aplicación (por ejemplo, Order). Los procesos empresariales pueden utilizar estos métodos. La interfaz de usuario pueden utilizar las consultas específicas para procesar los datos de referencia (como una lista de tipos de tarjetas de crédito válidos).

Cuando la aplicación contiene varios componentes lógicos de acceso a datos, puede resultar útil utilizar un componente de ayuda de acceso a datos genéricos para administrar las conexiones de las bases de datos, ejecutar comandos y almacenar parámetros en caché, entre otros.

Los componentes lógicos de acceso a datos proporcionan la lógica necesaria para obtener acceso a datos empresariales específicos, mientras que el componente de ayuda para el acceso a datos centraliza el desarrollo de API de acceso a datos y la configuración de la conexión a éstos, permitiendo de esta forma la reducción de código duplicado.

En la figura II.8 se muestra el uso de componentes lógicos de acceso a datos para obtener acceso a datos.

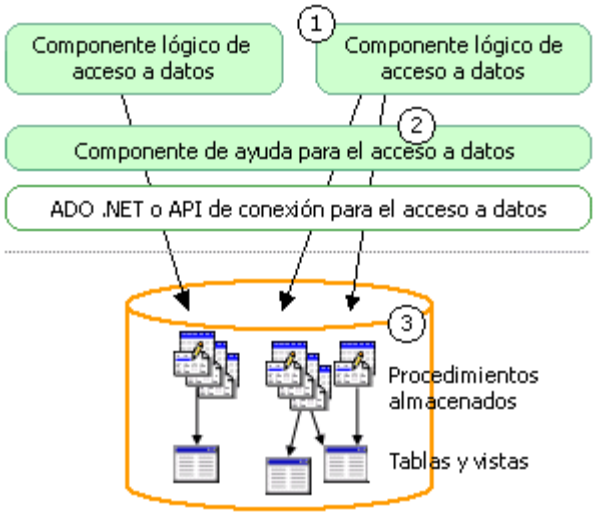


Figura II. 8. Componentes lógicos de acceso a datos

Observe los siguientes puntos en la figura anterior:

1. Los componentes lógicos de acceso a datos exponen métodos para insertar, eliminar, actualizar y recuperar datos, incluyendo la provisión de funcionalidad de paginación al recuperar grandes cantidades de datos.
2. Puede utilizar un componente de ayuda de acceso a datos para centralizar la administración de la conexión y todo el código relacionado con un origen de datos específico.
3. Se recomienda implementar las consultas y operaciones de datos como procedimientos almacenados (si es compatible con el origen de datos) para mejorar el rendimiento y la facilidad de mantenimiento.

El uso de componentes lógicos de acceso a datos se recomienda para todas las aplicaciones que requieren obtener acceso a datos empresariales (como productos y pedidos, entre otros). No obstante, otros productos y tecnologías pueden utilizar bases de datos para almacenar sus propios datos operacionales, sin tener que hacer uso de este tipo de componentes.

Los componentes lógicos de acceso a datos proporcionan acceso simple a funcionalidad de bases de datos (consultas y operaciones de datos), devolviendo estructuras de datos simples y complejas. La implementación de una lógica propia de acceso a datos en los componentes lógicos de acceso a datos permite encapsular toda la lógica de acceso a datos de la aplicación completa en una única ubicación central, lo que facilita el mantenimiento y la extensibilidad de la aplicación.

2.4.3.3. Funcionalidad de los componentes lógicos de acceso a datos

Cuando se llaman, los componentes lógicos de acceso a datos realizan lo siguiente:

- Llevan a cabo asignaciones y transformaciones simples de argumentos de entrada y salida.
- Obtienen acceso de un único origen.
- Actúan en una tabla principal y realizan operaciones en tablas relacionadas.

De forma opcional, pueden realizar las siguientes tareas:

- Utilizan un componente de utilidad personalizado para administrar y encapsular esquemas de bloqueo optimistas.
- Utilizan un componente de utilidad personalizado para implementar una estrategia de almacenamiento de datos en caché para los resultados de consultas no transaccionales.
- Implementan el enrutamiento dinámico de datos de sistemas de gran escala que proporcionan escalabilidad a través de la distribución de datos en varios servidores de bases de datos.

Los componentes lógicos de acceso a datos no deben:

- Invocar a otros componentes lógicos de acceso a datos. Un diseño en el que los componentes lógicos de acceso a datos no invocan a los otros componentes del mismo tipo facilita mantener la previsibilidad de los datos y, por tanto, aumenta la facilidad del mantenimiento de la aplicación.
- Inicializar transacciones heterogéneas. Debido a que cada uno de los componentes lógicos de acceso a datos sólo trata con un único origen de datos, no puede existir un escenario en el que uno de estos componentes constituya la raíz de una transacción heterogénea. En determinados casos, sin embargo, un componente lógico de acceso a datos puede controlar una transacción que conlleve varias actualizaciones en un único origen de datos.
- Mantener el estado entre llamadas a métodos.

El siguiente código en C# muestra un esquema parcial del esqueleto de un componente lógico de acceso a datos simple que se podría utilizar para el acceso a los datos del

pedido. Este código no se proporciona como plantilla, sino para ilustrar algunos de los conceptos descritos en este artículo.

```
public class OrderData
{
  private string conn_string;

public OrderData()
  {
    // obtener la cadena de conexión de una ubicación segura o
    // cifrada y asignarla a conn_string
  }
  public DataSet RetrieveOrders()
  {
    // Código para recuperar un conjunto de datos que contiene los datos de los pedidos
}
  public OrderDataSet RetrieveOrder(Guid OrderId)
{
    // Código para devolver un conjunto de datos introducido llamado OrderDataSet
    // que representa un pedido específico.
    // (OrderDataSet tendrá un esquema que se ha definido en Visual Studio)
}
  public void UpdateOrder(DataSet updatedOrder)
{
    // código para actualizar la base de datos en función de las propiedades
    // de los datos del pedido enviados como parámetro de tipo conjunto de datos
  }
}
```

2.4.3.4. Integración con servicios

Si en su proceso empresarial intervienen servicios externos, será necesario controlar la semántica de la comunicación con cada servicio al que sea necesario llamar. En concreto, deberá utilizar la API de comunicación adecuada para llamar al servicio y realizar las traducciones necesarias entre los formatos de datos utilizados por el servicio y los utilizados por el proceso empresarial. Si el contrato de servicio consta de una

conversación de ejecución larga, también deberá mantener el estado intermedio mientras espera una respuesta.

Se recomienda utilizar un componente de agente de servicios que encapsule la lógica necesaria para encapsular estas tareas e inicializar y administrar una conversación basada en mensajes para cada uno de los servicios que debe consumir la aplicación.

Los agentes de servicios se pueden considerar como los componentes lógicos de acceso a datos para los servicios distintos a los almacenes de datos, o como servidores proxy o emisarios para otros servicios. Determinados publicadores de servicios pueden proporcionar a los llamadores un agente de servicios incorporado. En otros casos, por el contrario, puede que sea necesario que desarrolle el suyo propio.

El objetivo de utilizar un agente de servicios es:

- Encapsular el acceso a un servicio.
- Aislar la implementación de los procesos empresariales de la implementación de formato de datos o cambios de esquema
- Proporcionar los formatos de datos de entrada y salida compatibles con los componentes empresariales que llaman al servicio.

Los agentes de servicios también puedan realizar los siguientes tipos de tareas habituales si es necesario:

- Llevar a cabo la validación básica de los datos intercambiados con el servicio.

- Almacenar en caché los datos necesarios para realizar consultas habituales.
- Autorizar el acceso al servicio, proporcionando una forma granular de comprobar la seguridad antes de obtener acceso al servicio desde el punto de vista de la aplicación que realiza la llamada. Normalmente, el servicio también suele autenticar y autorizar las solicitudes.
- Definir la seguridad adecuada u ofrecer las credenciales necesarias al servicio para la autenticación. Por ejemplo, para definir las credenciales para un servicio Web XML que se está invocando, puede utilizar HTTPCredentialCache.
- Asegurarse de que las partes adecuadas del mensaje están cifradas o de que se puede establecer un canal de seguridad si es necesario.
- Proporcionar información de supervisión que posibilite la interacción con el servicio que se va a implementar, lo que permite determinar si sus socios cumplen con sus contratos de nivel de servicio.

2.4.4. Infraestructura transversal

La mayoría de aplicaciones contienen funcionalidad común que se utiliza en los las diferentes Capas tradicionales e incluso en diferentes Niveles físicos (Tiers) como se muestra en la figura II.9. Este tipo de funcionalidad normalmente abarca operaciones como autenticación, autorización, cache, gestión de excepciones, logging/registros, trazas, instrumentalización y validación.

A este tipo de funcionalidad normalmente se le denomina Aspectos Transversales o Aspectos Horizontales, porque afectan a la aplicación entera, y deben estar por lo tanto centralizados en una localización central si es posible, para favorecer la reutilización de

componentes entre las diferentes capas. Por ejemplo, el código que genera trazas en los ficheros de log de una aplicación, probablemente se utilice desde muchos puntos diferentes de la aplicación (desde diferentes capas y niveles físicos). Si centralizamos el código encargado de realizar las tareas específicas de generar trazas (u otra acción), seremos capaces en el futuro de cambiar el comportamiento de dicho aspecto cambiando el código solamente en un área concreta.

Hay diferentes aproximaciones para diseñar e implementar estas funcionalidades, desde librerías de clases tipo “bloques de construcción” a utilizar desde mis capas tradicionales, hasta AOP donde se utilizan metadatos para bien insertar/inyectar código de aspectos transversales directamente en la compilación o bien durante el tiempo de ejecución (con intercepción de llamadas a objetos).

Las siguientes áreas o aspectos son los más habituales a considerar como parte de estas capas de infraestructura transversal:

- Seguridad
    - Identidad
    - Autenticación
    - Autorización
    - Arquitectura de Seguridad orientada a Claims
- Cache
- Gestión de Configuración
- Gestión de Excepciones

- Instrumentalización
- Logging
- Gestión de Estados
- Validación de datos de entrada

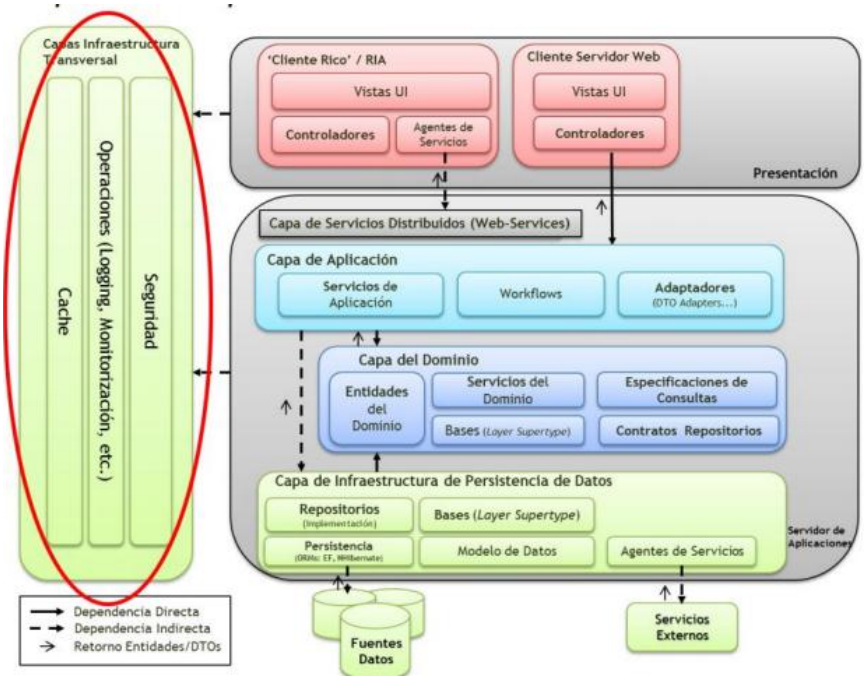


Figura II. 9. Aspectos Transversales

2.5. Arquitecturas compuestas

Una aplicación compuestaconsta de módulos de acoplamiento flexible que se descubren y componen dinámicamente en tiempo de ejecución [8]. Los módulos contienen

componentes visuales y no visuales que representan los distintos sectores verticales del sistema como se muestra en la figura II.10.

Los componentes visuales o vistas se componen en un Shell común que actúa como host de todo el contenido de la aplicación [9]. Los compuestos proporcionan servicios que vinculan estos componentes de nivel de módulo entre sí. Los módulos pueden ofrecer servicios adicionales que se relacionan con la funcionalidad específica de la aplicación.

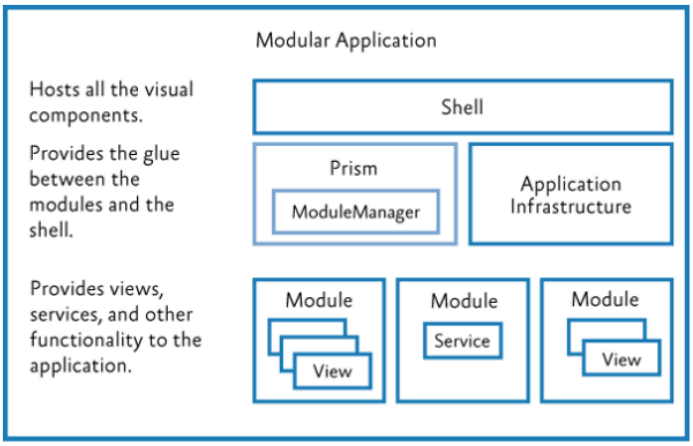


Figura II. 10. Vista General de una aplicación compuesta

En un nivel superior, una aplicación compuesta es una implementación del patrón de diseño Composite View, que describe una estructura de interfaz de usuario recursiva de vistas que contienen elementos secundarios que son vistas. A continuación, las vistas se componen mediante un mecanismo, normalmente en tiempo de ejecución, a diferencia de la composición estática en tiempo de diseño.

Para ilustrar las ventajas del patrón, piense en un sistema de entrada de pedidos en el que haya múltiples instancias de un pedido. Cada instancia puede implicar una complejidad considerable para mostrar el encabezado, los detalles, el envío y los recibos. A medida que evoluciona el sistema, puede necesitar mostrar más información. Imagine también que las partes del pedido se muestran de forma distinta según el tipo de pedido.

Si una pantalla de ese tipo se crea de forma estática, se podría acabar con una gran cantidad de lógica condicional para mostrar las distintas partes del pedido. Además, la incorporación de nueva funcionalidad aumenta la probabilidad de que la lógica existente deje de funcionar. No obstante, si se implementa como una vista compuesta, la pantalla de pedido se compone dinámicamente sólo con los elementos relevantes. Esto significa que se puede eliminar la lógica de presentación condicional y agregar nuevas pantallas secundarias sin modificar la vista de pedido.

Los módulos contribuyen a las vistas a partir de las que se crea la vista compuesta principal también denominada shell. Los módulos nunca se hacen referencia directa entre sí ni tampoco hacen referencia directa al shell. En su lugar, usan servicios para comunicarse entre sí y con el shell a fin de responder a las acciones del usuario.

Disponer de un sistema compuesto de módulos compuestos proporciona las siguientes ventajas:

- Los módulos pueden agregar los datos procedentes de distintos sistemas back-end en la misma aplicación.

- El sistema puede evolucionar más fácilmente con el tiempo.
- A medida que cambien los requisitos del sistema, se pueden agregar nuevos módulos al mismo con menos dificultades que en un sistema no modular.
- Los módulos existentes pueden evolucionar de forma más independiente, lo que mejora la capacidad de efectuar pruebas.
- Distintos equipos pueden desarrollar, probar y mantener los módulos.

2.5.1. Aplicaciones compuestas con PRISM

PRISM es un conjunto de bibliotecas destinadas al diseño y la construcción de aplicaciones de escritorio con Windows Presentation Foundation o Web con el uso de Silverlight, ricas en experiencia de usuario, flexibles y fáciles de mantener; parte de la implementación de patrones de diseño que se consideran claves en el desarrollo de este tipo de aplicaciones con una clara separación de conceptos y un escaso acoplamiento entre ellos [16].

PRISM está diseñado para poder hacer uso de todas las capacidades de los patrones de diseño de forma individual o todos juntos, dependiendo de los requerimientos y el escenario de la aplicación.

Las aplicaciones pueden usar el patrón de diseño MVVM, modularidad, regiones, comandos, eventos o la combinación de todos estos; PRISM reúne todas estas características que hacen que una aplicación tenga una flexibilidad para soportar cambios en el tiempo.

La figura II.11 ejemplifica una típica arquitectura compuesta en la cual se muestran todas las capacidades de trabajo en conjunto en una aplicación compuesta.

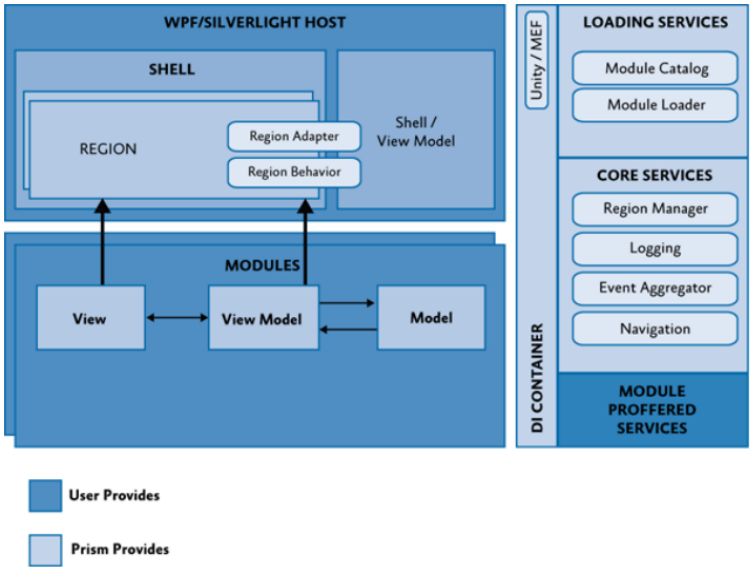


Figura II. 11. Arquitectura de una aplicación compuesta con PRISM

Las aplicaciones desarrolladas con PRISM consisten de un Shell que define las regiones sobre la cual se muestran las vistas o servicios compartidos que son cargados mediante módulos desarrollados de forma individual, y ser cargados al momento de ejecutar la aplicación o bajo demanda.

2.5.2. ¿Por qué usar PRISM?

El diseño y desarrollo de aplicaciones enriquecidas con WPF que sean flexibles y fáciles de mantener puede convertirse en un gran reto ya que el realizar cambios en el tiempo relacionados con los requerimientos de usuarios, interfaz gráfica o cambios en la tecnología puede afectar significativamente a la aplicación. Por lo tanto es muy importante

construir aplicaciones que sean flexibles y fáciles de modificar o extender sus funcionalidades en el tiempo; el diseño de este tipo de aplicaciones puede ser difícil de realizar ya que se requiere de una arquitectura que permita crear la aplicación por partes de tal forma que cada una de ellas pueda ser desarrollada, probada y acoplada a la aplicación global de forma sencilla, permitiendo a futuro un fácil mantenimiento.

El desarrollo de aplicaciones empresariales es mucho más complejo de realizar ya que requieren de un equipo de varios desarrolladores, diseñadores de interfaz de usuario y testers, esto puede convertirse en un gran reto para que equipos multidisciplinarios puedan trabajar efectivamente en diferentes partes de la aplicación de forma independiente garantizando que las partes desarrolladas se integren correctamente en la aplicación.

Una solución eficaz para afrontar los retos antes mencionados es la partición de la aplicación en una serie de componentes, débilmente acoplados, semi-independientes, componentes que pueden ser fácilmente integrados en una aplicación para formar una solución coherente.

Desarrollar una aplicación compuesta con PRISM provee muchos beneficios entre los más importantes:

- Los módulos pueden ser desarrollados, probados y puestos en producción de forma individual.

- Provee un shell común sobre el cual se componen varios módulos bajamente acoplados.
- Promueve la reutilización y una clara separación entre las diferentes capas horizontales de la aplicación como puede ser registro y autenticación, y las capas verticales como las funcionalidades de negocio específicas de la aplicación.
- Permite gestionar fácilmente las dependencias e interacciones entre los componentes de la aplicación.
- Ayuda a mantener una separación de las funcionalidades al permitir que los diferentes miembros de un equipo de desarrollo puedan concentrarse en un atarea especifica o en una funcionalidad de acuerdo a la experiencia. En particular, proporciona una separación limpia entre la interfaz de usuario y la lógica de negocio.

Las aplicaciones compuestas son muy apropiadas para una amplia gama de escenarios de aplicaciones cliente. Por ejemplo, es ideal para crear una rica experiencia de usuario final usando diferentes sistemas back-end, como lo muestra la figura II.12.

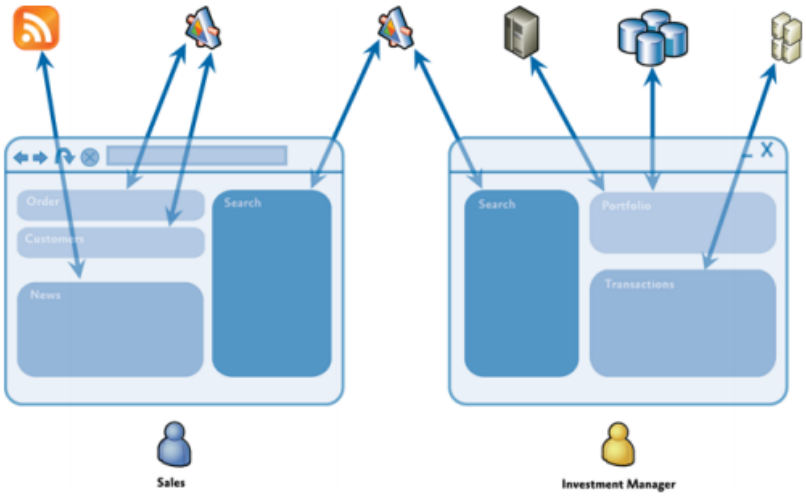


Figura II. 12. Aplicación compuesta con múltiples sistemas back-end

2.5.3. Desarrollando aplicaciones modulares con PRISM

Como se ha mencionado anteriormente una aplicación modular está dividida en un conjunto de unidades funcionales llamadas módulos que pueden ser integradas dentro de una aplicación completa.

Un módulo encapsula una parte funcional de la aplicación global, el mismo que incluye componentes relacionados como interfaz de usuario y lógica de negocio, o partes de la infraestructura de aplicación como niveles de autenticación para usuarios.

Estos módulos son independientes unos de otros y pueden ser desarrollados, probados, puestos en producción de forma independiente sobre la aplicación global.

En lo sucesivo, veremos cómo PRISM hace posible la creación de aplicaciones compuestas y los patrones que intervienen en ello.

Una aplicación PRISM se compone de tres elementos principales [9]:

- El bootstrapper que es el encargado de poner en marcha la aplicación, carga los módulos iniciales e inicializa el shell. También lanza ciertos servicios adicionales, como por ejemplo el servicio de logging.

- Uno o más módulos independientes que implementan las distintas funcionalidades de la aplicación.

- El Shell que es el contenedor (ventana) principal de la aplicación. Contiene aquellos elementos de la interfaz de usuario que están visibles en todo momento. Por ejemplo, una barra de menú o un ribbon son candidatos a estar en el shell, ya que son elementos que siempre están visibles en la interfaz.

La figura II.13 muestra el flujo del proceso que realiza PRISM para la carga de módulos.

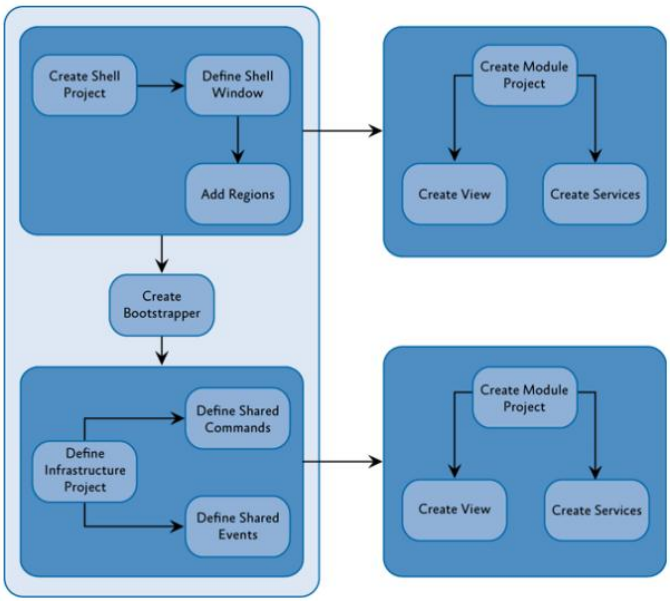


Figura II. 13. Flujo de procesos en una aplicación compuesta con PRISM

2.5.3.1. El Bootstrapper y contenedores

Al crear aplicaciones compuestas con PRISM, primero debe inicializar varios servicios de composición principales, es aquí donde interviene el Bootstrapper el cual ocupa de todas las funciones necesarias para que se produzca la composición tal y como se ilustra en la figura II.14. En muchos sentidos se trata del método main de una aplicación compuesta.

Figura II.14. Tareas de inicialización del bootstrapper

En primer lugar, se inicializa el contenedor de inyección de dependencia o simplemente conocido como contendor, este desempeña una función clave en una aplicación compuesta [10]. El contenedor contiene todos los servicios de aplicación usados en la composición, se encarga de insertar estos servicios donde sea necesario.

De forma predeterminada, PRISM usa el contenedor de inyección de dependencia Unity que implementa una clase UnityBootstrapper dentro de su framework, no obstante, PRISM fue diseñado para funcionar con otros contenedores, como MEF, Windsor, Structure Map y Sprint.NET [6] [13].

Ninguna de las clases de PRISM(que no sean las extensiones Unity) depende de un contenedor específico, es decir que con un mínimo de esfuerzo, se puede usar distintos contenedores de inyección dependencia, el uso de distintos contenedores está en función de los requerimientos de la aplicación y el escenario sobre el cual será implementado.

A medida que se configura el contenedor, varios servicios principales usados para la composición se registran automáticamente, incluido un logger y un event aggregator. El

bootstrapper permite sobrescribir cualquiera de estos. Si se sobrescribe el método que administra el contenedor en el bootstrapper, podríamos registrar nuestro propio cargador de módulos.

Si se requiere que los servicios no se registren de forma predeterminada, también se puede desactivar esta característica, sobrecargando el método Run en el bootsptrapper y poniendo en falso al parámetro useDefaultConfiguration.

Luego de configurar el contenedor, se configuran los adaptadores de región. Una región es una ubicación con nombre (normalmente un contenedor, como, por ejemplo, un panel) en la interfaz de usuario donde los módulos pueden insertar elementos de interfaz de usuario. Los adaptadores de región administran la conexión de los distintos tipos de región a los que se tendrá acceso, estos se asignan en una instancia singleton de la clase RegionAdapterMappings en el contenedor.

En este punto se crea el Shell, que es la ventana de nivel superior donde se definen las regiones. En vez de declararlo en un archivo App.xaml, se crea mediante el método CreateShell desde el bootstrapper específico de la aplicación, de este modo se garantiza que la inicialización del bootstrapper se lleva a cabo antes de que se muestre el shell. Debido a que la clase UnityBootstrapper es abstracta, se puede sobrescribirla para extender la funcionalidad de varios métodos virtuales.

Luego de haber creado el shell, es necesario registrarlos sobrescribiendo el ConfigureContainer, los servicios adicionales también se pueden registrar

programándolos en este punto si es necesario. Finalmente, el método CreateShell se sobrescribe con la lógica específica para crear el Shell, se puede usar el patrón Model View Presenter, de modo que el shell tenga un presentador asociado sin embargo no es necesario.

2.5.4. Partiendo la aplicación en módulos

Un módulo en una aplicación compuesta es una unidad de separación dentro de un compuesto que se puede implementar como un ensamblado independiente, el módulo es lo que contiene la mayor parte de la funcionalidad.

Cuando se desarrolla una aplicación compuesta, la estructura debe estar separada en módulos, cada uno de estos debe encapsular una porción de la aplicación general.

Durante el diseño de arquitectura del sistema se debe dividir la funcionalidad de la aplicación en módulos separados, estos deben representar las distintas capas verticales de nuestro sistema o una capa de servicio horizontal como se muestra en la figura II.15 y figura II.16 respectivamente.

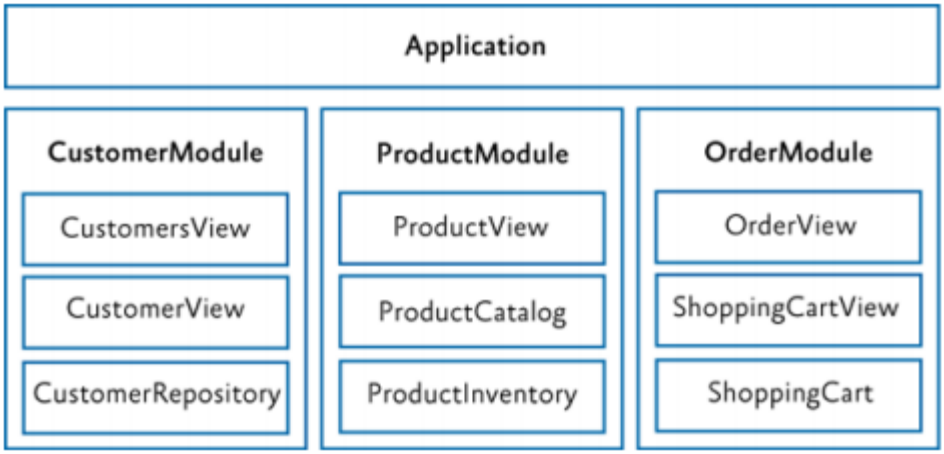


Figura II.15. Aplicación con módulos organizados en capas verticales

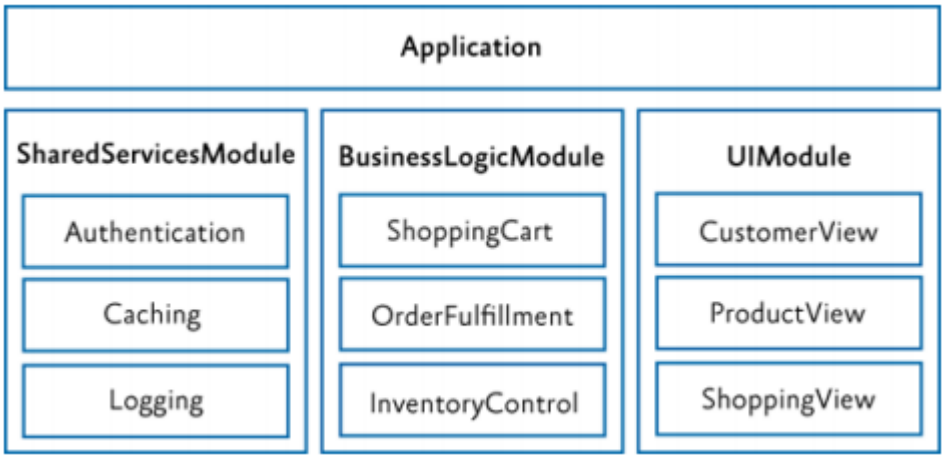


Figura II.16. Aplicación con módulos organizados en capas horizontales

Una aplicación grande puede tener módulos organizados con capas verticales y horizontales, algunos ejemplos de módulos incluyen lo siguiente:

- Un módulo que contenga características específicas de la aplicación, como un módulo de noticias.

- Un módulo que contenga un subsistema o la funcionalidad de un conjunto de casos de uso como un módulo de compras, facturación y contabilidad general.
- Un módulo que contenga la infraestructura de servicios como logging, caché, autorización de servicios o web services.

Un módulo debe ser lo más independiente posible de otros módulos; cuando un módulo tiene dependencia con otro, este puede ser acoplado usando interfaces definidas en una librería compartida instanciando tipos concretos, o usando el EventAggregator para comunicarse con otros módulos.

El objetivo de la modularidad es dividir la aplicación en módulos de tal forma que sea flexible, mantenible y estable y que soporte a futuro cambios debidos a tecnología, incremento o eliminación de funcionalidades.

2.5.4.1. Inicialización de módulos

Luego de dividir la aplicación en módulos independientes, la carga de módulos es un proceso en el que se usa un enumerador de módulos y un cargador de módulos. El enumerador se encarga de encontrar los módulos disponibles, devuelve varias colecciones de objetos que contienen metadatos acerca de un módulo; UnityBootstrapper contiene un GetModuleEnumerator que se debe sobrescribir para devolver el enumerador correcto; de lo contrario, se producirá una excepción en tiempo de ejecución. PRISM incluye enumeradores para buscar módulos de forma estática, a partir de una búsqueda en el directorio y dentro de un archivo de configuración.

Para la carga de módulos, PRISM incluye un cargador de módulos que UnityBootstrapper usa de forma predeterminada, esta carga cada uno de los ensamblados de módulo (si no se han cargado) los inicializa; estos módulos pueden especificar dependencias en otros módulos, mediante un árbol de dependencias que inicializa los módulos en el orden correcto según estas especificaciones.

2.5.4.2. Módulos y servicios

Como se mencionó anteriormente, en una aplicación compuesta creada con la PRISM, la gran mayoría de la lógica de la aplicación se encuentra en los módulos. El código anterior muestra la implementación de una aplicación ejemplo que incluye cuatro módulos con la siguiente funcionalidad.

- NewsModule.- proporciona fuentes de noticias relacionadas para cada fondo de inversión que se selecciona.
- MarketModule.- ofrece datos de tendencia, así como datos de mercado en tiempo real, para el fondo de inversión seleccionado.
- WatchModule.- proporciona una lista de observación que muestra una lista de los fondos de inversión que se están supervisando.
- PositionModule.- muestra la lista de los fondos en que ha invertido y le permite efectuar transacciones de compra y venta.

En PRISM, un módulo es una clase que implementa la interfaz de IModule, esta interfaz tiene un único método, denominado Initialize, sí el bootstrapper equivale al método Main

de la aplicación, el método Initialize es el método Main de cada módulo. Cuando se carga un módulo, se resuelve a partir del contenedor, que también inserta las dependencias especificadas en el constructor del módulo; al conceder a los módulos acceso directo al contenedor se permite que el módulo registre y resuelva dependencias a partir del contenedor de un modo imperativo.

2.5.5. Regiones y regionmanager

Los módulos no son interesantes en sí mismos a menos que puedan representar contenido en la interfaz de usuario. En la sección anterior se mostró que el módulo WatchModule usa una región para agregar sus dos vistas. El uso de una región evita que el módulo deba contar con una referencia a la interfaz de usuario o que deba conocer cuál será la disposición y la presentación de las vistas insertadas. Para ofrecer un ejemplo de esto, la figura II.17 representa las secciones en las que el módulo WatchModule realiza la inserción.

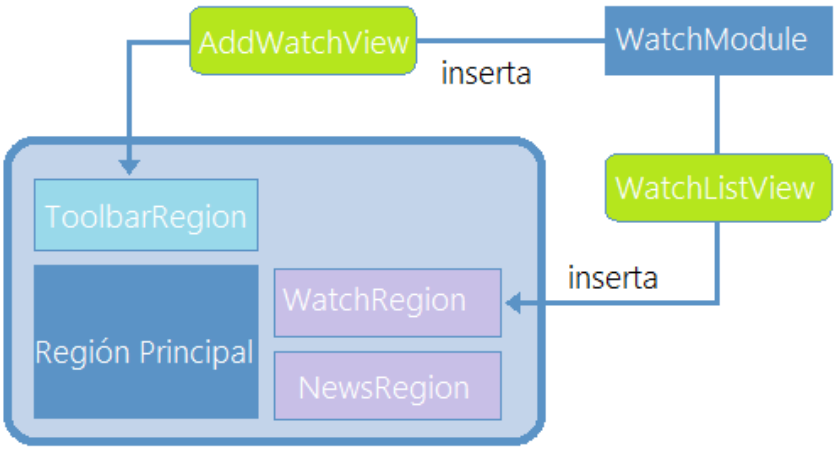


Figura II.17. Inserción de módulos en la aplicación

PRISM incluye una clase region que es básicamente un identificador que ajusta estas ubicaciones. La clase region contiene una propiedad views que es una colección de sólo lectura de las vistas que se mostrarán dentro de la región. Las vistas se agregan a la región llamando al método add de la región.

La propiedad views realmente contiene una colección genérica de objetos; no está limitada únicamente a objetos UIElement, esta colección implementa INotifyPropertyCollectionChanged de modo que el UIElement asociado a la región pueda enlazarse a él y observar los cambios.

Las regiones se pueden registrar definiéndolas dentro de un archivo XAML mediante el uso de un UIElement con una propiedad RegionName, como se puede apreciar en el siguiente código que ilustra la forma en la que se registraría una vista.

```
<ItemsControlName="MainToolbar"prism:RegionManager.RegionName="MainToolbarRegion"/>
```

Después de que una región se ha definido mediante XAML, se registrará automáticamente en tiempo de ejecución mediante el RegionManager, uno de los servicios de composición registrados por el bootstrapper.

El RegionManager es esencialmente, un objeto del tipo Dictionary donde la clave es el nombre de la región y el valor es una instancia de la interfaz IRegion, la propiedad RegionManager adjunta usa un RegionAdapter para crear una instancia; no obstante, hay que tener en cuenta que si el uso de propiedades adjuntas no sirve no se necesita registrar regiones adicionales de forma dinámica, puede crear manualmente una instancia

de la clase Region o una clase derivada y agregarla a la colección Regions de RegionManager.

En el fragmento de código XAML anterior, MainToolbarRegion es un ItemsControl, PRISM incluye tres adaptadores de región que el bootstrapper registra: SelectorRegionAdapter, ContentControlRegionAdapter y ItemsControlRegionAdapter. Los adaptadores se registran con una clase RegionAdapterMappings, todos los adaptadores heredan de RegionAdapterBase, que implementa la interfaz IRegionAdapter.

La figura II.18 muestra la implementación de ItemsControlRegionAdapter, y como RegionManager busca el adaptador para el control a usar, crea la región y lo asocia a ella para mostrar el contenido deseado.

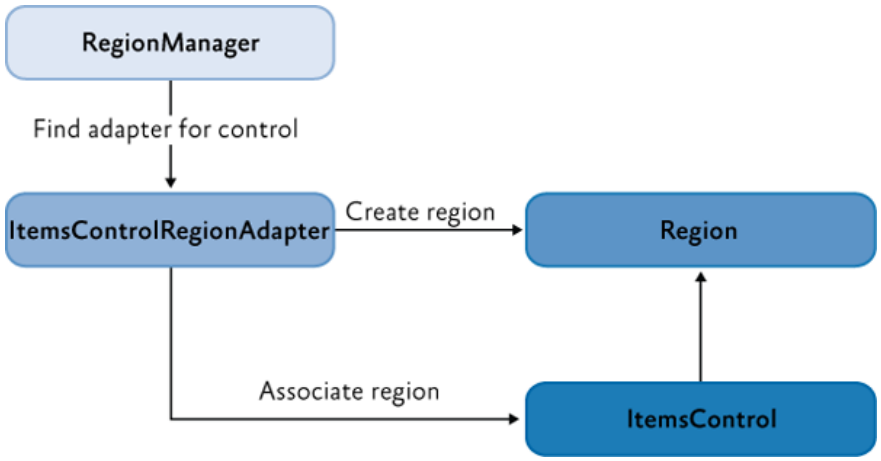


Figura II.18. Flujo de un adaptador de región

Las regiones pueden contener vistas que son activas o inactivas, en el caso de ItemsControl, todos sus elementos están activos constantemente porque no tiene una

noción de selección; sin embargo, en el caso de otros tipos de regiones, como Selector, sólo un elemento está seleccionado cada vez. Una vista puede implementar la interfaz IActiveAware para que su región le notifique que ha sido seleccionada, siempre que se selecciona la vista, tendrá su propiedad IsSelected establecida en true. Durante el desarrollo de una aplicación compuesta, puede ser necesario crear regiones adicionales y adaptadores de región, como uno que adapte un control de otro proveedor.

Una vez definida la región, se puede obtener acceso a ella desde cualquier clase de la aplicación si se retiene el servicio RegionManager, la forma habitual de llevar a cabo esto en una aplicación desarrollada con PRISM es hacer que el contenedor de inyección de dependencias inserte un RegionManager en el constructor de la clase que lo necesite.

Para agregar una vista o modelo a una región, sólo se tiene que llamar al método Add de la región, cuándo se agrega una vista, se puede pasar un nombre opcional:

```
_regionManager.Regions["MainRegion"].Add(  somePresentationModel, "SomeView");
```

Posteriormente se puede usar dicho nombre para recuperar la vista de la región mediante el uso del método GetView de la región.

2.5.6. Presentación independiente

Anteriormente se ha mencionado que una de las ventajas de crear una aplicación compuesta es que permite que el código tenga más capacidad de mantenimiento y se puedan realizar pruebas.

Existen varios patrones de presentación establecidos que se puede aplicar a las vistas para alcanzar este objetivo el más usado en el desarrollo de aplicaciones compuestas con WPF el patrón Model View ViewModel que es una variante cercana del patrón Presentation Model, optimizado para aprovechar algunas de las principales capacidades de WPF y Silverlight, tal como el enlace de datos, plantillas de datos, comandos y otros.

En el patrón MVVM, la vista encapsula la interfaz y cualquier lógica de interfaz de usuario, el modelo de vista [1]encapsula la lógica de presentación y el estado, y el modelo encapsula la lógica de negocio y datos. La vista interactúa con el modelo de vista a través de enlace de datos, comandos y eventos de notificaciones de cambio. El modelo de vista, observa y coordina las actualizaciones del modelo[2], convirtiendo tipos, validando y agregando de datos según sea necesario para su visualización en la vista, la figura II.19 muestra la interacción entre los distintos componentes del patrón MVVM.

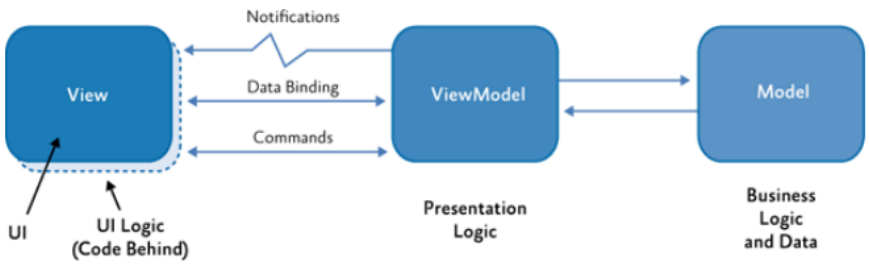


Figura II.19. Componentes de MVVM y su interacción

[1]Modelo de vista.- Es una traducción literal del inglés ViewModel

[2]Modelo.- Es una traducción literal del inglés Model

Al igual que con todos los patrones de presentación, la clave para usar el patrón MVVM radica en la comprensión de la forma adecuada para refactorizar [3]el código de su aplicación en las clases correctas, y en la comprensión de las formas en que estas clases interactúan en diversos escenarios.

Los principales componentes y características del patrón MVVM son las siguientes:

La vista (view)

La responsabilidad de la vista es el de definir la estructura y el aspecto de lo que el usuario ve en la pantalla. Lo ideal sería que el código subyacente de una vista sólo contiene un constructor que llama al InitializeComponentMethod. En algunos casos, el código subyacente puede contener código de la lógica de interfaz de usuario que implementa el comportamiento visual que es difícil o ineficiente de expresar en XAML, como animaciones complejas, o cuando el código tiene que manipular directamente los elementos visuales que son parte de la vista. No debe poner ningún código de la lógica en la vista, por lo general, el código de la lógica en la vista de código subyacente se pondrá a prueba a través de un enfoque de pruebas de interfaz de usuario automatizadas.

[3]Refactorizar.- Del inglés refactoring, es una técnica de la ingeniería de software para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.

En WPF, las expresiones de enlace de datos en la vista se evalúan según su contexto de datos. En MVVM, el contexto de la vista de datos se establece en el modelo de vista el mismo que implementa propiedades y comandos para que la vista se pueden unir y notificarla de cualquier cambio en el estado a través de eventos. La relación entre la Vista y el Modelo de Vista normalmente e de uno a uno.

Por lo general, las vistas son clases derivadas de un UserControl o CustomCOntrol, sin embargo en algunos casos, una vista puede ser representada por una plantilla de datos que especifica los elementos de interfaz de usuario que se utilizan para representar visualmente un objeto.

El Modelo de vista (viewmodel)

El modelo de vista encapsula la lógica de presentación y datos para la vista. No tiene ninguna referencia directa a la Vista o conocimiento alguno acerca de la aplicación específica de la vista o el tipo.

El modelo de vista implementa propiedades y comandos para que la vista puede enlazar datos y notificar a la vista de cualquier cambio de estado a través de eventos de notificación de cambio.

Las propiedades y comandos que el modelo de vista proporciona define la funcionalidad que ofrece la interfaz de usuario, pero la vista determina la forma en que la funcionalidad se va a representar.

El modelo de vista es responsable de coordinar la interacción de la vista con cualquier clase del Modelo que se requiere.

La relación entre el modelo de vista y el modelo es de un uno-a-. El modelo de vista puede optar por exponer las clases del modelo directamente a la vista, de modo que los controles en la vista de datos pueden unirse directamente a ellos. En este caso, el modelo deberá ser diseñado para soportar el enlace a datos y los eventos de cambio de notificación pertinentes.

El modelo de vista puede convertir o manipular los datos del modelo de manera que pueda ser fácilmente consumidos por la vista, además puede definir propiedades adicionales para apoyar específicamente la vista; estas propiedades no sería normalmente parte de (o no se puede añadir a) el modelo. Por ejemplo, el modelo de vista puede combinar el valor de dos campos para que sea más fácil para el fin de presentar, o se puede calcular el número de caracteres que quedan para la entrada para los campos con una longitud máxima.

El modelo de vista también puede implementar la lógica de validación de datos para asegurar la consistencia de los datos.

El modelo de vista también puede definir estados lógicos de la vista se puede utilizar para proporcionar cambios visuales en la interfaz de usuario. La vista puede definir cambios de diseño o estilo que reflejan el estado del modelo de vista. Por ejemplo, el modelo de vista puede definir un estado que indica que los datos se presentan de forma asincrónica a un

servicio web. La vista puede mostrar una animación durante este estado para proporcionar información visual al usuario.

Por lo general, el modelo de vista va a definir comandos o acciones que se pueden representar en la interfaz de usuario y que el usuario puede invocar. Un ejemplo común es cuando el modelo de vista proporciona un comando Enviar que permite al usuario enviar datos a un servicio web o en un repositorio de datos. La vista puede elegir para representar a ese comando con un botón para que el usuario puede hacer clic y enviar los datos. Por lo general, cuando el comando no está disponible, su representación de IU pasará a estar deshabilitada.

El modelo (model)

El modelo encapsula la lógica de negocio y datos, la lógica de negocio se define como cualquier lógica de aplicación que se ocupa de la recuperación y gestión de datos de la aplicación y de asegurar la coherencia y la validez de los datos.

Típicamente, el modelo representa el dominio del lado del cliente en la aplicación, también puede incluir el código para apoyar el acceso y el almacenamiento en caché de datos, aunque normalmente se emplea un repositorio de datos o servicio separado.

A menudo, el modelo y la capa de acceso a datos se generan como parte de una estrategia de acceso a datos o servicio, como ADO.NET Entity Framework, Servicios de datos de WCF o servicios WCF RIA, el modelotambién puede apoyar la validación de

datos y la administración de errores a través de las interfaces IDataErrorInfo (o INotifyDataErrorInfo), estas interfaces permiten que los datos de WPF sean notificados cuando los valores cambian para que la interfaz de usuario se pueden actualizar, también habilita el soporte para la validación de datos y presentación de informes de error en la capa de interfaz de usuario.

CAPÍTULO III

ANÁLISIS ESTADÍSTICO COMPARATIVO

En el contenido previo de este documento investigativo se ha proporcionado información relevante acerca de cómo los patrones de arquitectura ayudan a resolver problemas de diseño. Se ha estudiado cada uno de estos patrones demostrando la manera como deben ser construidos, las proyecciones de flexibilidad y mantenibilidad que ofrecen.

En este capítulo se implementa un análisis estadístico comparativo entre una aplicación compuesta y una aplicación monolítica; se detalla: los parámetros a tomarse en cuenta en el análisis comparativo, el desarrollo de los prototipos, la explicación de los puntajes

alcanzados y su interpretación; finalmente se realiza la comprobación de la Hipótesis para determinar la mejor opción e implementar de la aplicación final.

Para la comprobación de la hipótesis se ha decidido utilizar una técnica estadística, misma que ha permitido la obtención de resultados de cada uno de los requisitos desarrollados en los prototipos (Monolítico y Compuesto) que han sido estudiados y expresados en valores numéricos, permitiendo determinar la mejor opción para implementar la aplicación de estudio de la Geografía Ecuatoriana.

3.1. Definición de los parámetros a comparar

Como se ha visto en el capítulo anterior, los patrones de diseño de arquitectura de software ofrecen una cantidad de ventajas en la implementación de aplicaciones. Pero tal vez los mayores beneficios no son los que se obtienen al momento mismo de la implementación de sistemas, sino que son alcanzados en etapas posteriores a la implementación, en donde es necesario un mantenimiento de la aplicación, corrección o reorganización de partes de código, aumento de nuevas funcionalidades para cumplir con características o requerimientos adicionales, reutilización de componentes en nuevos sistemas, etc.

Sin embargo el uso de patrones de diseño de arquitectura de software pueden acelerar o afectar el tiempo de desarrollo, escribir mucho código, los tiempos de respuesta y consumo de recursos.

Así pues de todos los parámetros mensurables dentro del uso de los patrones de diseño de arquitectura de software hemos escogido los siguientes que serán medidos en el desarrollo de los prototipos monolítico y compuesto, siendo los siguientes:

- **Número de líneas de código escritas.-** El número de líneas de código escritas es un parámetro que se medirá durante el desarrollo de cada uno de los requerimientos. Se contabilizará la cantidad de líneas de código escritas para llevar a cabo un requerimiento en los prototipos.
- **Tiempo de desarrollo.-** Se medirá el tiempo que toma desarrollar cada uno de los requerimientos planteados en los prototipos.
- **Tiempo de respuesta.-** Es el tiempo en el que el prototipo tarda en responder a una petición. Este parámetro se medirá al ejecutar los requerimientos delos prototipos.
- **Uso de CPU.-** Es el uso del CPU al ejecutar los requerimientos implementados sobre los prototipos, los datos se obtendrán usando el administrador de tareas de Windows 7.
- **Uso de Memoria RAM**.-Es la cantidad de memoria RAM que consume los requerimientos implementadossobre los prototipos, los datos se obtendrán usando el administrador de tareas del Windows 7.
- **Tiempo de Mantenimiento.-** Se medirá el tiempo que toma desarrollar un nuevo requisito y adaptarlo a los prototiposdesarrollados; este parámetro se lo analizará en la etapa de mantenimiento.

La tabla III.I, detalla los parámetros y sus valores a usar dentro de la presente investigación.

Tabla III. I. Valores de los parámetros a usar

Parámetro	**Valor porcentual**
Número de líneas de código escritas	8%
Tiempo de desarrollo	12%
Tiempo de respuesta	10%
Uso de CPU	10%
Uso de memoria RAM	10%
Tiempo de Mantenimiento	50%
Total	100%

El valor que se le da a cada parámetro para la puntuación global está tomado de acuerdo a la importancia dentro del ciclo de vida de una aplicación (Figura III. 20), como se mencionó en el capítulo anterior la fase de mantenimiento y adaptación de nuevas funcionalidades a un producto software es la etapa que más recursos utiliza, y por lo tanto implica un mayor costo para los desarrolladores.

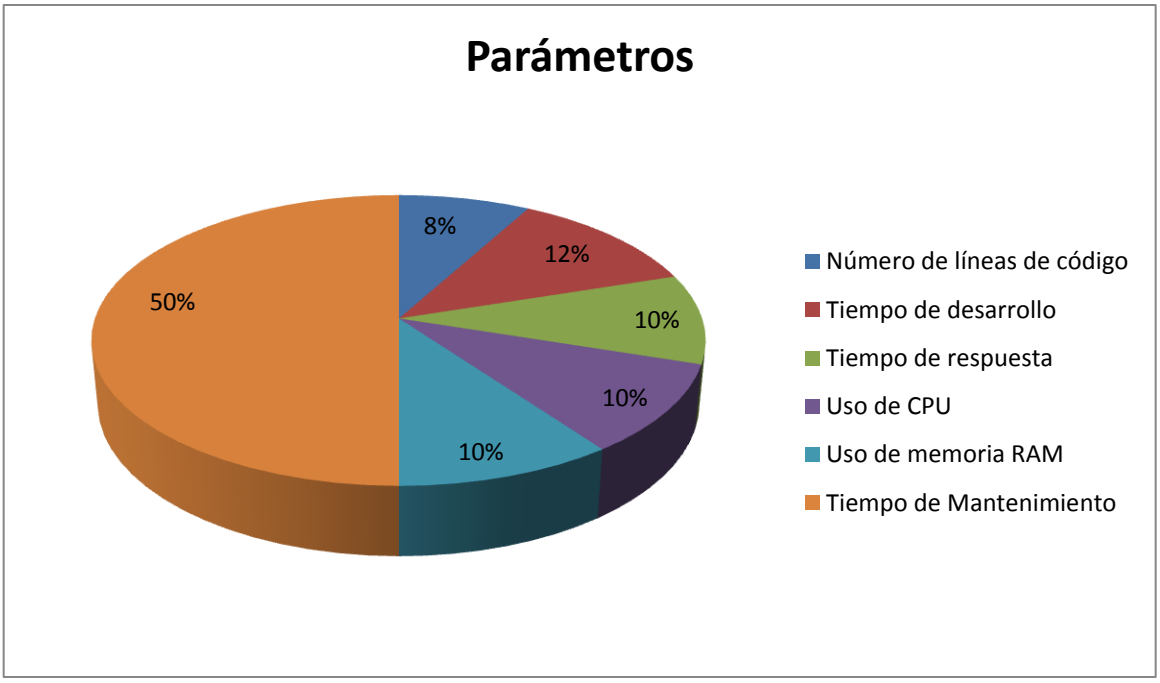


Figura III. 20. Valores porcentuales de los parámetros a usar

Una vez que se tiene la escala definida y con los parámetros evaluados en cada una las arquitecturas se obtendrá como resultado la arquitectura más adecuada con la que se implementará la aplicación de estudio de la Geografía Ecuatoriana.

3.2. Métodos técnicas y procedimientos

Para la realización del presente proyecto previamente han sido definidos un conjunto de métodos, técnicas y procedimientos que son la base para la ejecución de la investigación. Los elementos mencionados se detallan a continuación.

3.2.1. Métodos

Esta investigación tiene como marco principal el método científico experimental; de forma general, este método:

- Plantea el problema.
- Describe una hipótesis.
- Recopila información suficiente para comprobar la hipótesis con el uso de experimentos.
- Analiza y difunde resultados.

3.2.2. Técnicas y procedimientos

Para el desarrollo de las siguientes secciones de este capítulo, es necesario recordar la hipótesis planteada previamente a la ejecución de la aplicación.

"Realizar un estudio estadístico entre el desarrollo de una aplicación compuesta con PRISM y una aplicación monolítica permitirá escoger un modelo de arquitectura correcto para crear una aplicación de estudio de la Geografía Ecuatoriana en la Unidad Educativa Andes College".

Para la comprobación de esta hipótesis, se especifica a continuación la población y la muestra que serán el objeto de estudio. Así mismo se utilizará la estadística descriptiva e inferencial para alcanzar este objetivo [3].

Población.

Como población se ha escogido todos los requerimientos de la aplicación de estudio de la geografía ecuatoriana, los cuales se detallan el Capítulo IV de este documento.

Método de muestreo

Para la demostración de la hipótesis de esta investigación se ha seleccionado una muestra a través del método probabilístico. Para el cálculo de la muestra se trabajó con el nivel de confianza del 95%.

Muestra

Para realizar el estudio estadístico comparativo se procederá al desarrollo de dos prototipos, uno con arquitectura monolítica y otro con arquitectura compuesta.

Para esto se ha realizado el análisis de los requisitos de la aplicación de estudio de la geografía ecuatoriana, y del total de estos se ha procedido al cálculo de la muestra (Anexo I).

3.3. Análisis comparativo

Para proceder con el análisis comparativo el desarrollo de los prototipos se utilizó el lenguaje de programación y convenciones de codificación C# por ser uno de los lenguajes de programación más conocidos y fáciles de aprender, el desarrollo de la interfaz de usuario se la realizó con Windows Presentation Foundation mismo que enriquece la experiencia de usuario; y el almacenamiento de datos se lo realizó en Microsoft SQL Server 2008 Express Edition [7] [12] [15].

Los requerimientos de los prototipos (Compuesta y Monolítica) se desarrollaron con el uso de la metodología ágil XP, permitiendo obtener los valores de los parámetros mencionados anteriormente [18].

A continuación se detallan los recursos, parámetros y técnicas utilizadas, en las pruebas efectuadas para la selección de la arquitectura que mejores resultados obtenga para su completo desarrollo

Las características físicas del computador en las que se desarrollaron los prototipos y las pruebas correspondientes son las siguientes:

- CPU: Intel Core i5 M450 2.4 GHz.
- Memoria RAM: 4 GB.

Para obtener los resultados se tomaron las medidas de los prototipos desarrollados.

Se utilizó para estos fines una distribución de probabilidad T de Student con la que se determinó las diferencias entre dos medias muestralesy para la construcción del intervalo de confianza el cual mostró la diferencia entre las medias de las dos poblaciones.El valor de desconfianza con el que se trabajó en la distribución T de Student es del 5%.

3.4. Puntajes alcanzados

Para la obtención de los valores de los parámetros se implementó 7 de los 9 requerimientos obtenidos como muestra; los dos últimos requerimientos se implementaron como funcionalidades adicionales y por ende considerados como etapa de mantenimiento para obtener los valores necesarios del parámetro “Tiempo de Mantenimiento”, y comprobar la característica principal (Mantenimiento) que tienen ambas arquitecturas estudiadas.

Sin embargo durante el desarrollo del prototipo surgió la actualización del SDK de Kinect por parte del fabricante lo que obligo a realizar una actualización a los requerimientos antes desarrollados, y ayudando a estudiar el comportamiento de las arquitecturas durante la fase de mantenimiento.

Los datos correspondientes a líneas de código escritas fueron contabilizados en el desarrollo de cada uno de los requisitos de la aplicación. Para las mediciones correspondientes a los tiempos de desarrollo y de mantenimiento,estos fueron medidos con un cronómetro y es el tiempo que se tardó en el desarrollo de cada requisito.

Para medir el uso del CPUy memoria RAM se utilizó el administrador de tareas del sistema operativo Windows 7, para esto se ejecutó la aplicacióny cada uno de los requisitos. Para el tiempo de respuesta se lo midió con la ayuda de un cronómetro y corresponde al tiempo que se tarde en responder a determinada funcionalidad o requisito.

Luego de culminar el desarrollo de los dos prototipos de la aplicación se obtuvieron los datosque se adjuntan en el anexo 1.

Con los puntajes alcanzados se procede a efectuar el estudio estadístico descriptivo de los mismos, se procede al cálculo de las medias de cada uno de los parámetros evaluados, así como también de su respectivo intervalo de confianza, la tabla III.II muestra los valores obtenidos; los cálculos se adjunta en el anexo 1.

Tabla III. II. Valores de las medias de los prototipos

Prototipo Monolítico						
	Líneas de código escritas	Tiempo de desarrollo	Uso de CPU	Uso de RAM	Tiempo de respuesta	Tiempo de Mantenimiento
Media	397.222	16.267	4.973	82.513	2.034	3.600
Intervalo de confianza	316.767 477.678	11.750 20.783	4.276 5.670	50.598 114.429	1.035 3.033	0.025 7.175
Prototipo Compuesto						
	Líneas de código escritas	Tiempo de desarrollo	Uso de CPU	Uso de RAM	Tiempo de respuesta	Tiempo de Mantenimiento
Media	411.444	18.094	4.332	86.080	2.400	0.733

Intervalo de confianza	320.091 502.798	13.026 23.163	3.453 5.212	60.061 112.099	1.371 3.429	0.154 1.312

Para la comparación se toma como referencia el intervalo de confianza de aquel valor de la media que sea menor, y en referencia a este se determinara por cada parámetro si existe diferencia significativa entre los valores.

3.5. Interpretación

Conclusión 1:

La figura III. 21 muestra la comparación de los valores obtenidos para el parámetro líneas de código escritas.

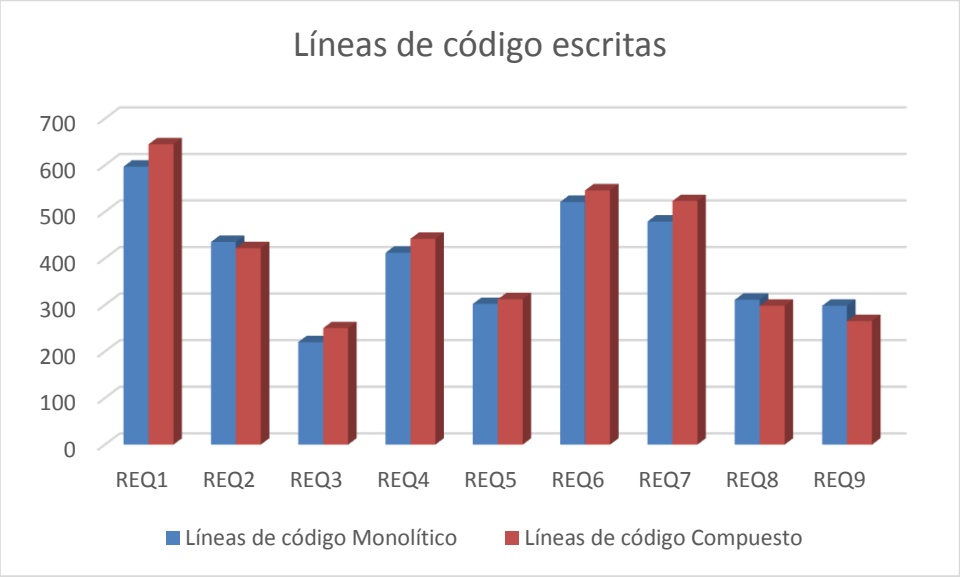


Figura III.21. Comparación del parámetro Líneas de código escritas

Estadísticamente se ha llegado a la conclusión que no existe diferencia significativa en el número de líneas de código escritas entre las dos formas de implementar una aplicación ya que los valores hallados (medias de líneas de código escritas)arquitectura monolítica 397.222 y arquitectura compuesta 411.444 se encuentran dentro del intervalo de confianza[316.767; 477.678].

Conclusión 2:

La figura III.22 muestra la comparación entre los valores obtenidos en el parámetro tiempo de desarrollo delos prototipos.

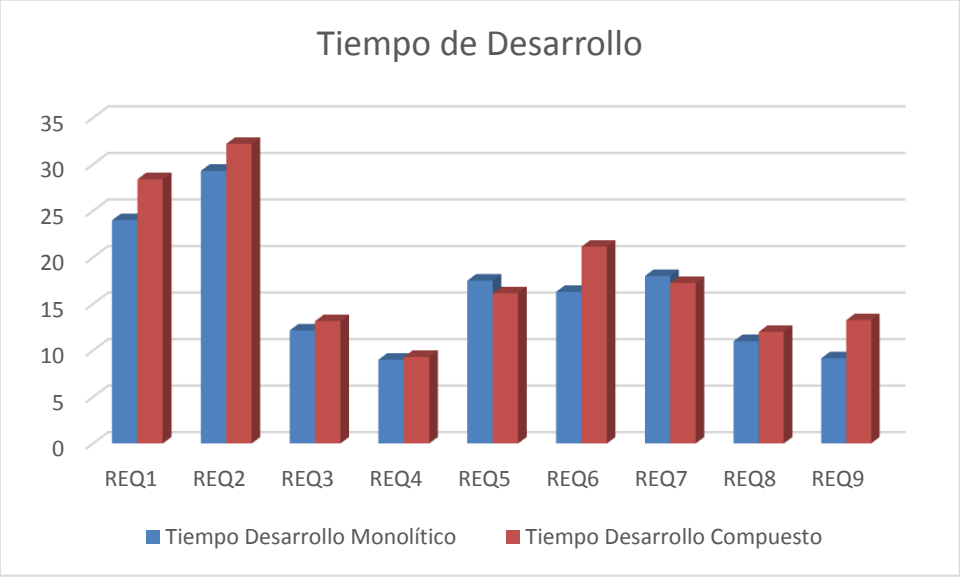


Figura III. 22.Comparación del parámetro Tiempo de Desarrollo

Estadísticamente se ha llegado a la conclusión que no existe diferencia significativa en el tiempo de desarrollo entre las dos formas de implementar una aplicación ya que los valores hallados de las medias de: arquitectura monolítica 16.267 y arquitectura compuesta 18.094 se encuentran dentro del intervalo de confianza [11.750; 20.783].

Conclusión 3:

La figura III. 23 muestra la comparación entre el uso de CPU delos prototipos.

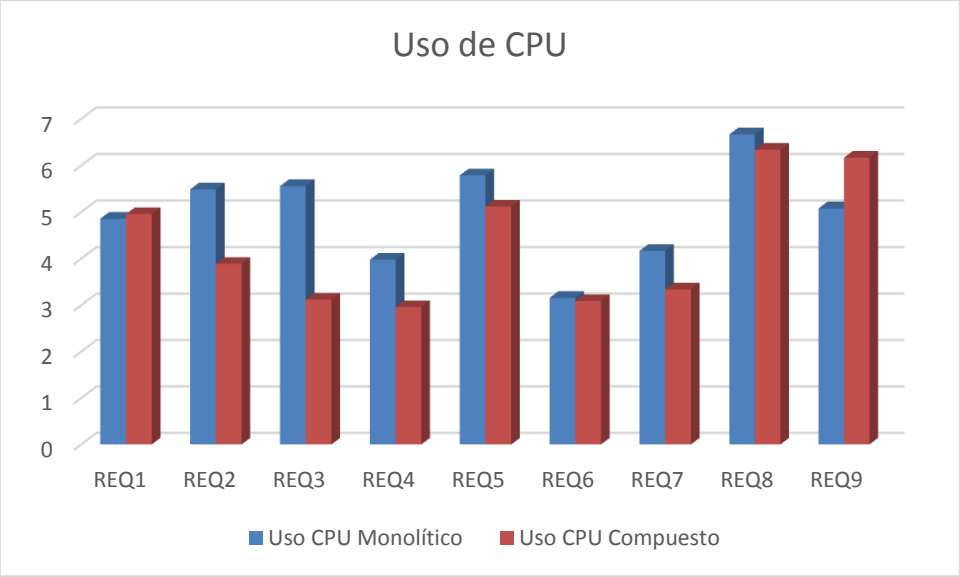


Figura III. 23Comparación del parámetro uso de CPU

Estadísticamente se ha llegado a determinar que no existe diferencia significativa en el uso de CPU al ejecutar los requerimientos implementados sobre los prototipos ya que los

valores hallados,media de los prototipos monolítico 4.973 y compuesto 4.332 se encuentran dentro del intervalo de confianza[3.453; 5.212].

Conclusión 4:

En la figura III.24 se muestra la comparación de los valores correspondientes al parámetro uso de memoria RAM de los prototipos.

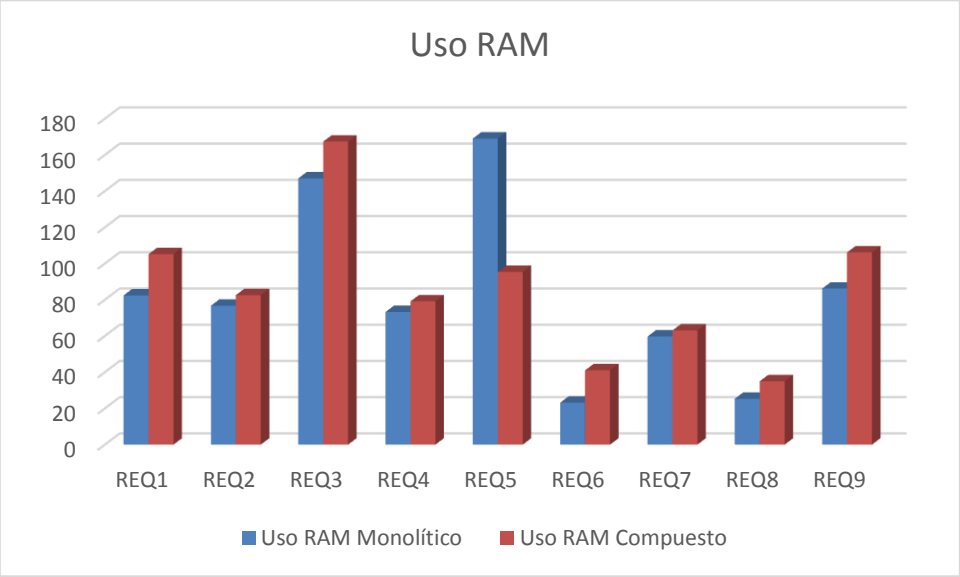


Figura III. 24.Comparación del parámetro uso de RAM

Estadísticamente se ha llegado a la conclusión que no existe diferencia significativa en el uso de memoria RAM que consume los requerimientos implementados en los prototipos ya que los valores hallados correspondientes a las medias correspondientes a prototipo

monolítico 82.513 y compuesto 86.080 se encuentran dentro del intervalo de confianza [50.598; 114.429].

Conclusión 5:

La figura III.25 muestra la comparación de los valores obtenidos correspondientes al tiempo de respuesta delos prototipos.

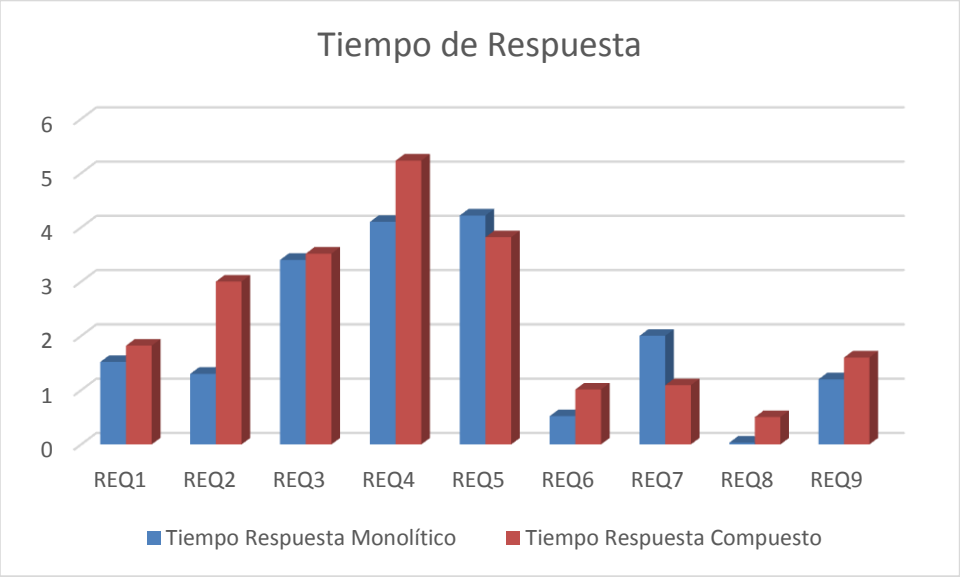


Figura III. 25.Comparación del parámetro tiempo de respuesta

Estadísticamente se ha llegado a la conclusión que no existe diferencia significativa en el tiempo de respuesta al ejecutar una funcionalidad de los prototipos ya que los valores hallados correspondientes a las medias del prototipo monolítico 2.034 y compuesto 2.400 se encuentran dentro del intervalo de confianza [1.035; 3.033].

Conclusión 6:

La figura III. 26 muestra la comparación entre los valores correspondientes al tiempo de mantenimiento delos prototipos.

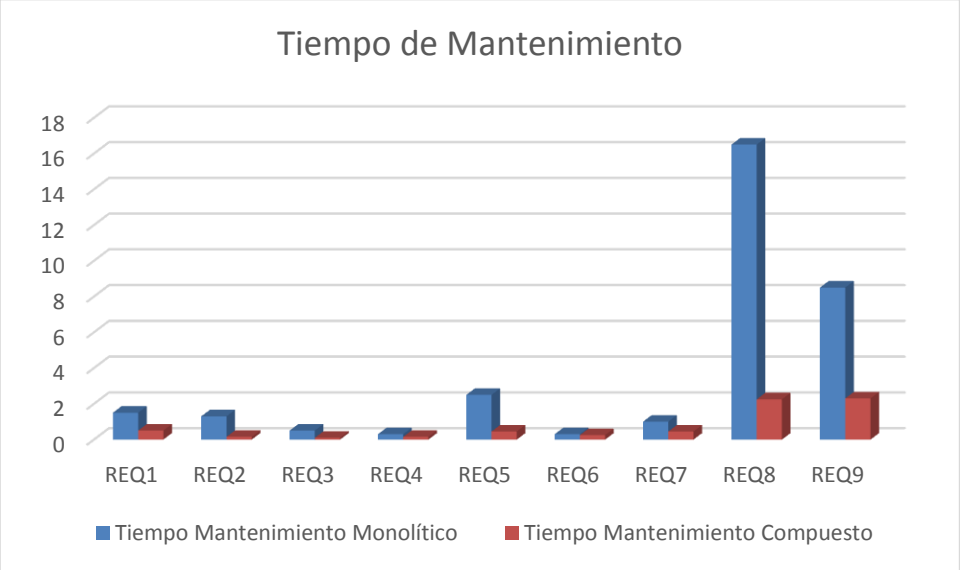


Figura III. 25Comparación del parámetro Tiempo de Mantenimiento

Estadísticamente se ha llegado a la conclusión que si existe diferencia significativa en el tiempo de mantenimiento ya que los valores hallados de la media de los prototipos monolítico 3.600 y compuesto 0.733, que representan el tiempo que toma desarrollar un nuevo requisito y adaptarlo al prototipo existente no se encuentran dentro del intervalo de confianza[0.154; 1.312].

Para obtener un puntaje total y determinar cuál es la mejor arquitectura para el desarrollo y mantenimiento de una aplicación se aplicó un modelo estadístico Anexo I, en el cual

seusó los pesos detallados en la tabla III. I, de acuerdo a la importancia dentro del ciclo de vida de desarrollo de software.

Al parámetro con menor valor se le asigna el máximo valor porcentual y para el otro caso se procede al cálculo de su correspondiente valor como se muestra en el Anexo I. Luego de realizar estas operaciones se tomará como arquitectura ganadora aquella que tenga el mayor valor porcentual.

Los valores obtenidos se muestran a continuación en la tablaIII.III.

Tabla III. III. Valores para la comparación final

	Prototipo Monolítico		**Prototipo Compuesto**	
	Media	**Peso ganador**	**Media**	**Peso ganador**
Líneas de código escritas	397.222	0.080	411.444	0.077
Tiempo de desarrollo	16.267	0.120	18.094	0.108
Uso de CPU	4.973	0.087	4.332	0.100
Uso de RAM	82.513	0.100	86.080	0.096
Tiempo de respuesta	2.034	0.100	2.400	0.085
Tiempo de mantenimiento	3.600	0.102	0.733	0.500
Totales		0.589		0.966
Total (%)	**58.9%**		**96.6%**	

La figura III. 27 muestra la comparación entre los puntajes alcanzados por los prototipos Monolítico y compuesto con PRISM.

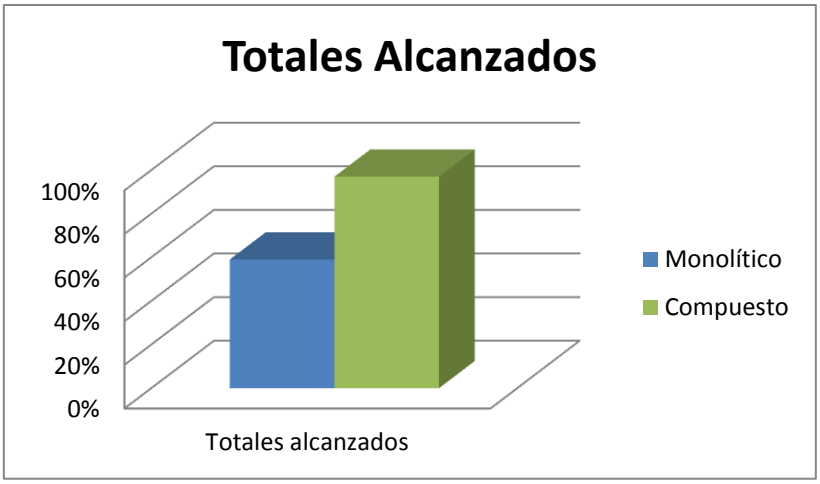


Figura III. 26 Totales alcanzados en la comparación

3.6. Resultados del análisis

Conclusión final: Se ha utilizado estadística descriptiva y estadística inferencial para poder hacer el análisis y escoger la mejor arquitectura para desarrollar la aplicación final; en el caso del número de líneas de código, tiempo de desarrollo, uso de CUP, uso de memoria RAM y tiempo de respuesta no existe diferencia significativa, pero los datos obtenidos con el parámetro tiempo de mantenimiento demuestra que la arquitectura compuesta es la más adecuada para la realización de la aplicación de estudio de la geografía ecuatoriana de acuerdo a las características que se tomó en cuenta en el presente estudio.

3.7. Comprobación de la hipótesis

Hipótesis: Realizar un estudio estadístico entre el desarrollo de una aplicación compuesta con PRISM y una aplicación monolítica permitirá escoger un modelo de arquitectura correcto para crear una aplicación de estudio de la Geografía Ecuatoriana en la unidad educativa “Andes College”.

Causa: Estudio estadístico entre el desarrollo de una aplicación compuesta con PRISM y una aplicación monolítica.

Efecto: Permitirá escoger un modelo de arquitectura correcto para crear una aplicación de estudio de la Geografía en la unidad educativa “Andes College”.

Para determinar la mejor opción de forma general se procedió al uso de un modelo estadístico que permitió conocer cuál es la arquitectura optima y por tanto representa la mejor opción para el desarrollo de la aplicación de estudio de la Geografía Ecuatoriana, resultando como ganador la arquitectura compuesta con el 96.6% sobre un 58.9% obtenido por la arquitectura monolítica.

CAPÍTULO IV

DESARROLLO DE LA APLICACIÓN PARA EL ESTUDIO DE LA GEOGRAFÍA ECUATORIANA

Para el desarrollo de la aplicación para el Estudio de la Geografía Ecuatoriana, se utilizó la metodología ágil XP la cual consta de las siguientes fases:

- Planificación
- Diseño
- Codificación
- Pruebas

4.1. Planificación

En la planificación se desarrolla una breve descripción del sistema, se detallan los requerimientos funcionales y no funcionales, se escriben las historias de usuario, iteraciones y la velocidad del proyecto.

4.1.1. Descripción del sistema

La tesis a implementar es el estudio estadístico entre el desarrollo de una aplicación compuesta con PRISM y una aplicación monolítica la cual permitirá crear una aplicación de estudio de la Geografía Ecuatoriana aplicada en la unidad educativa Andes College".

La aplicación consta de un módulo que contiene parte del contenido educativo de la materia de Geografía Ecuatoriana, la aplicación será manipulada a través del dispositivo Kinect mediante la voz y también con movimientos y gestos corporales. Esto permitirá que el estudiante interactuar con el contenido educativo de la asignatura.

La aplicación se desarrolló utilizando el lenguaje de programación C# y XAML sobre el sistema operativo Windows 7, para el almacenamiento de la información se utilizó el motor de base de datos Microsoft SQL Server 2008 Express por su poco consumo de recurso y ser gratuito brindando la posibilidad de almacenar hasta 10 GB de información.

4.1.2. Requerimientos

Para el desarrollo de la aplicación de Estudio de la Geografía, se necesita cumplir con los siguientes requerimientos:

Requerimientos funcionales:

Los requerimientos funcionales son una descripción de lo que un sistema debe hacer [17]. Los requerimiento denotan algo que el sistema entregado debe ser capaz de realizar. La tabla IV. IVdetalla los requerimientos de la aplicación a implementar y que fueron recopilados de las entrevistas con el cliente.

Tabla IV. IV. Requerimientos funcionales de la aplicación

Código	Requerimiento
REQ1	El sistema debe contener un módulo de estudio de la geografía ecuatoriana.
REQ2	Implementar un módulo de visualización de actividades a realizar en clase.
REQ3	El sistema debe permitir realizar las búsquedas por comandos de voz
REQ4	El sistema debe permitir actualizar el contenido de la materia desde internet.
REQ 5	El sistema debe permitir la interacción con diferentes entradas voz/movimientos corporales.
REQ 6	El sistema debe permitir visualizar el avance de las actividades por tema de clase.
REQ 7	El sistema debe permitir configurar la aplicación.
REQ 8	El sistema debe integrarse al contenido multimedia de la institución.
REQ 9	El sistema debe permitir pruebas de evaluación de actividades a realizar dentro de una clase.

Requerimientos no funcionales

Un requisito no funcional o atributo de calidad es un requisito que especifica criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos, ya que éstos corresponden a los requisitos funcionales.

La tabla IV.V lista los requerimientos no funcionales especificados por el cliente.

Tabla IV. V. Requerimientos no funcionales de la aplicación

Requerimiento	Descripción
Mantenibilidad	La aplicación debe permitir modificar las funcionalidades existentes y agregar nuevas funcionalidades.
Accesibilidad	Se debe tener métodos de acceder a la aplicación para personas que no puedan usar un dispositivo de entrada como teclado y mouse.
Escalabilidad	La aplicación debe tener la habilidad para reaccionar y adaptarse sin perder calidad, o bien manejar el crecimiento continuo de la misma.
Interfaz amigable	El sistema debe contar con una interfaz comprensible y de fácil manejo para el usuario.

4.1.3. Historias de usuario.

El objetivo de las historias de usuario es describir un caso de uso en dos o tres líneas con terminología del cliente.

En el Anexo II se describen las historias de usuario que fueron implementadas en el desarrollo de la aplicación de estudio de la geografía ecuatoriana.

La tabla IV. VI. muestra una de las historias de usuario implementadas dentro de la aplicación.

Tabla IV. VI. Historia de usuario implementada en la aplicación

Historia de usuario	
Número: 02	**Nombre**:Búsqueda por voz
Usuario: Docente y estudiante	
Modificación de historia número:	**Iteración asignada:** 2
Prioridad de negocio: Alta	**Puntos estimados:** 40 PE
Riesgo en desarrollo: Bajo	**Puntos reales:** 20 días
Descripción : Como usuario quiero realizar búsquedas en internet por medio de la voz para poder enriquecer el contenido de clase dictada	
Pruebas de aceptación: El sistema debe interactuar con los comandos de voz registrados para que el usuario pueda hacer uso de la aplicación. En caso de que el comando de voz sea erróneo debe mostrar sugerencias de los mismos.	

4.1.4. Plan de publicaciones.

Después de tener ya definidas las historias de usuario es necesario crear un plan de publicaciones, donde se indiquen las historias de usuario que se crearán para cada versión del programa y las fechas en las que se publicarán estas versiones. Un plan de publicaciones es una planificación donde los desarrolladores y clientes establecen los tiempos de implementación ideales de las historias de usuario, la prioridad con la que

serán implementadas y las historias que serán implementadas en cada versión del programa. Después de un plan de publicaciones tienen que estar claros estos cuatro factores: los objetivos que se deben cumplir (que son principalmente las historias que se deben desarrollar en cada versión), el tiempo que tardarán en desarrollarse y publicarse las versiones del programa, el número de personas que trabajarán en el desarrollo y cómo se evaluará la calidad del trabajo realizado.

Todas las iteraciones corresponden a un módulo funcional y entregable de la aplicación, es así que aproximadamente cada cuatro semanas la aplicación de estudio de la geografía constará con una funcionalidad adicional.

La primera iteración corresponde al Módulo de la geografía ecuatoriana, este módulo abarca parte del contenido de estudio de la materia de geografía ecuatoriana.

En una segunda iteración se permitirá que el usuario pueda manipular e interactuar con la aplicación mediante movimientos corporales gracias al uso de Kinect.

En la tercera iteración se permitirá que el usuario pueda visualizar el avance de la clase dictada, dándole la posibilidad de una mejor planificación al maestro del contenido en el tiempo de la clase. Si se desea realizar una actualización del contenido, ya sea por reforma curricular o reajuste de contenidos, la aplicación permite realizarlo desde internet, esta actividad ser la podrá realizar después de la cuarta iteración.

En la quinta iteración se cuenta con varias actividades para realizar durante el desarrollo de la clase.

En la sexta iteración la aplicación permite realizar búsquedas mediante el uso de la voz.

4.1.5. Iteraciones

El desarrollo se divide en iteraciones, en cada una de las cuales se eligen las historias de usuario a desarrollar y las tareas de desarrollo.

Se eligieron seis historias de usuario para su desarrollo las mismas que tienen un tiempo de implementación de 40 puntos estimados. Un punto estimado tiene la duración de 4 horas. El plan de iteraciones se describe en el Anexo III.

4.1.6. Velocidad del proyecto.

La velocidad de desarrollo del proyecto se dio en aproximadamente 5 tareas en 2 puntos estimados.

4.2. Diseño

A continuación se detalla el contenido que se desarrolló según la metodología XP en cada una de las subetapas del diseño.

4.2.1. Diseño de la arquitectura

La arquitectura de software es de especial importancia ya que la manera en que se estructura un sistema tiene un impacto directo sobre la capacidad de este para satisfacer lo que se conoce como los atributos de calidad del sistema La arquitectura define los componentes, interfaces y comportamientos de un sistema, este diseño se lo construye teniendo en cuenta los requerimientos ya que se debe construir exactamente lo que se ha solicitado, aunque hay margen para la innovación y la flexibilidad.

Para el desarrollo de la aplicación de estudio de la geografía ecuatoriana se construyó la arquitectura que se muestra en la figura IV.26.

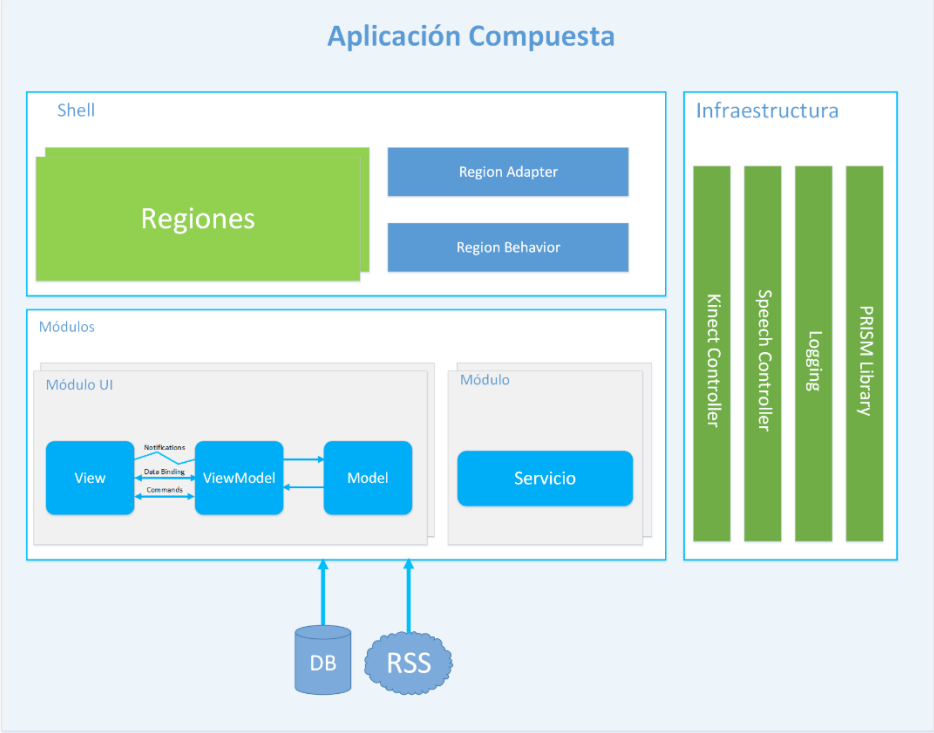


Figura IV.27Diseño de la arquitecturaimplementada en la aplicación

4.2.2. Diseño de la base de datos

Una de las tareas importantes a la hora de construir una aplicación es la base de datos, ya que aquí se guardan información relevante sobre la misma. Un buen diseño de la base de datos ayuda a un correcto manejo de la información requerida por la aplicación.

Para la aplicación de estudio de la geografía se tiene el siguiente esquema de base de datos que se muestra en la figura IV.27

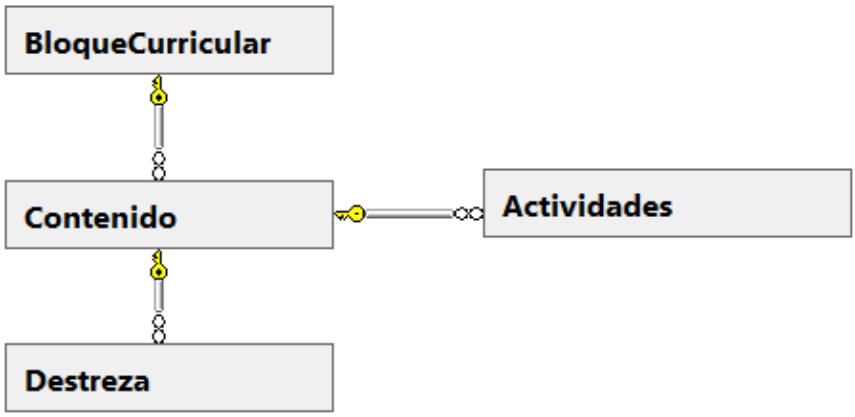


Figura IV. 28. Esquema de la base de datos usada en la aplicación

4.2.3. Riesgos

Teniendo en cuenta que la explotación de un riesgo causa daños o pérdidas financieras o administrativas a una empresa u organización, se tiene la necesidad de poder estimar la magnitud del impacto del riesgo a que se encuentra expuesta mediante la aplicación de controles.

Por lo que para este proyecto se ha realizado una gestión de riesgos lo cual consta de la identificación de los mismos.

Una vez identificados los riesgos se realizó su descripción, categorización, análisis de sus posibles consecuencias, se los ubicó en un rango de probabilidad de ocurrencia, determinación del impacto, determinación de la exposición al riesgo, determinación de la prioridad del riesgo y finalmente la hoja de gestión de riesgos en la cual constan los valores hallados anteriormente y una posible solución a ellos, esto se realizó de los riesgos cuya exposición se cataloga entre media y alta.

La descripción de los riesgos que se encontraron se detalla en el Anexo IV

4.2.4. Refactorizar

En los métodos que hacen uso de Speech Recognition se rescribieron ciertas partes del código para aumentar su legibilidad y mantenibilidad pero no se modificó su comportamiento.

4.2.5. Tarjetas C.R.C

Para el desarrollo de la aplicación de estudio de la geografía ecuatoriana se utilizó las tarjetas C.R.C que permiten al desarrollador centrarse y apreciar el desarrollo orientado a objetos.

Las tarjetas C.R.C representan objetos, la clase a la que pertenece el objeto se puede escribir en la parte de arriba de la tarjeta, en una columna a la izquierda se pueden escribir las responsabilidades u objetivos que debe cumplir el objeto y a la derecha, las clases que colaboran con cada responsabilidad. Las tarjetas C.R.C se encuentran detalladas en el Anexo V.

4.3. Codificación

La codificación debe hacerse ateniendo a estándares de codificación ya creados. Programar bajo estándares mantiene el código consistente y facilita su comprensión y escalabilidad.

Para el desarrollo de la aplicación de estudio de la geografía ecuatoriana se siguió el formato de la convención Pascal se refiere al nombramiento de elementos donde cada nueva palabra comienza con mayúscula y sigue con minúscula. Ejemplo: Color, UsuarioWeb, CreditoCartera, para el nombre las clases se usa sustantivos. No se debe usar el carácter '_'.

Para los métodos se usó el esquema de nombres <Verbo><Objeto>[<Predicado>], el verbo debe ser en infinitivo que indique la acción que ejecuta el método, el objeto es el nombre de la entidad sobre la que se ejecuta la acción, el predicado representa las opciones o restricciones de la acción.

Para las propiedades y variables se usa nombres descriptivos que tengan significado dentro del dominio del problema que está solucionando.

4.4. Pruebas

Uno de los pilares de la metodología XP es el uso de test para comprobar el funcionamiento de los códigos que vayamos implementando;para esto se usó las pruebas unitarias de Visual Studio, para determinar el correcto funcionamiento de los módulos creados. Algunas de las pruebas realizadas son: consumo de servicios, comunicación entre módulos, acceso a datos.

4.4.1. Pruebas de aceptación

Los test mencionados anteriormente sirven para evaluar las distintas tareas en las que ha sido dividida una historia de usuario. Para asegurar el funcionamiento final de una determinada historia de usuario se deben crear "Pruebas de aceptación".

Las pruebas se realizaron para las funcionalidades generales que debe cumplir la aplicación, los cuales se describieron en los requisitos.

Las pruebas de aceptación que se realizaron fueron en función a las características que debe cumplir el sistema y son las que se describen a continuación:

- El sistema debe poseer un módulo de estudio de la geografía ecuatoriana, que contenga actividades a realizar en clase, y permita visualizar el avance de las mismas, esto se puede probar con la manipulación del contenido de la aplicación.
- Se manipuló la aplicación con movimientos corporales, y comandos de voz para las búsquedas de contenido.
- El sistema permite la actualización de contenido de estudio desde internet.
- El sistema se integra al contenido multimedia que la institución posee, para hacer más interactiva la clase.

CONCLUSIONES

1. En el análisis comparativo realizado, los parámetros seleccionados son parte de las principales características de las arquitecturas estudiadas, esto permitió determinar cuál es la arquitectura más apropiada para el desarrollo de la aplicación final.

2. En el estudio estadístico que se realizó entre las arquitecturas monolítica y compuesta no existió diferencias significativas en el número de líneas de código escritas, tiempo de desarrollo, uso de CPU, uso de memoria RAM, tiempo de respuesta;pero si en el tiempo de mantenimiento con 58.9.% para el prototipo monolítico y 96,6% para el prototipo compuesto con PRISM con un margen de error del 5%, concluyéndose que por el menor tiempo de mantenimiento las aplicaciones compuestas con PRISM son más fáciles de mantener en el tiempo y hacen más productivo al equipo de desarrollo.

3. Desarrollar una aplicación compuesta con PRISM no tiene como objetivo la reducción del tiempo de desarrollo y su uso no reduce el número de líneas de código escritas, sino que en su mayor parte producen un efecto contrario; sin embargo este es un costo mínimo a pagarse frente a las múltiples ventajas que proporciona.

4. El desarrollo de una arquitectura compuesta con PRISM brinda múltiples ventajas, hacen un sistema flexible al cambio, crean un punto de partida para la reorganización del código y brindan un vocabulario de diseño de alto nivel.

RECOMENDACIONES

1. Se recomienda hacer uso de arquitecturas compuestas para el desarrollo de aplicaciones que vayan a modificar o incrementar sus funcionalidades en el tiempo.

2. Se recomienda determinar los parámetros de comparación en base a las características determinantes de cada una de las arquitecturas y al contexto en el que será aplicado.

3. Mantener una constante revisión sobre los nuevos patrones de arquitectura de software que se desarrollan en base a buenas prácticas de equipos de desarrollo especializados.

4. Realizar un estudio similar al presente comparando estas arquitecturas en plataformas web y con dispositivos móviles.

RESUMEN

El estudio estadístico entre una aplicación compuesta con PRISM y otra monolítica para el estudio de la geografía ecuatoriana, se realizó con el fin de determinar la arquitectura más óptima para el desarrollo de la misma, la cual se aplicó en la unidad educativa Andes College de la ciudad de Riobamba

Se aplicó el método estadístico con la distribución t de Student para la comparación del número de líneas de código escritas, tiempo de desarrollo, tiempo de respuesta, uso de CPU , uso de Memoria RAM y tiempo de mantenimiento de los prototipos desarrollados con arquitectura monolítica y compuesta, los cuales se implementaron con tecnología .NET en C# bajo el sistema operativo Windows, en los que se realizó pruebas de desarrollo de los requerimientos de cada prototipo obteniendo mediciones correspondientes a cada uno de los parámetro antes mencionados y comparándose las medias obtenidas, existiendo diferencia significativa únicamente en el tiempo de mantenimiento, con 58,9% para el prototipo monolítico y 96,6% para el prototipo compuesto con PRISM, con un margen de error del 5%.

Concluyendo que el prototipo compuesto con PRISM es el más adecuado, eligiéndose éste para desarrollar en su totalidad la aplicación de estudio de la geografía ecuatoriana, utilizando la metodología ágil de desarrollo XP.

Se recomienda, que los parámetros de comparación deben ser definidos en base a las características principales de cada arquitectura, tomando en cuenta el uso que se les va a dar en lo posterior permitiendo continuar con un estudio más profundo del tema.

SUMARY

The statistical study between a compound application with PRISM and other monolithic for the study of Ecuadorian geography, it was done with the aim of determining the most suitable architecture for its own development, which was applied at Andes College institute in Riobamba.

The statistical method was applied with t student distribution for the line number comparison of the written codes, time of development, time of response, use of CPU, RAM memory use, maintaining time of the prototypes developed with compou8nd and monolithic architecture, which were implemented with technology .NET in C# under Windows software, in which there were developed some development test for the requirements of each prototype, getting this way measurements corresponding to each one of the parameters mentioned before and compared with the actual measurements, with 58.9% for the monolithic prototype and 96.6% for the compound prototype with PRISM, with an error margin of 5%.

It is concluded that the compound prototype with PRISM is the most adequate, choosing this one to completely develop the application study of Ecuadorian geography, using the agile methodology of development XP.

It is recommended, that the comparative parameters should be defined based on the main characteristics of each architectures, taking into account the future use, allowing to continue with a deeper study of the topic.

GLOSARIO

Media

En matemáticas y estadística, la media aritmética (también llamada promedio o simplemente media) de un conjunto finito de números es igual a la suma de todos sus valores dividida entre el número de sumandos. Cuando el conjunto es una muestra aleatoria recibe el nombre de media muestral siendo uno de los principales estadísticos muestrales.

Expresada de forma más intuitiva, podemos decir que la media (aritmética) es la cantidad total de la variable distribuida a partes iguales entre cada observación.

Desviación Estándar

La desviación estándar o desviación típica (σ) es una medida de centralización o dispersión para variables de razón (ratio o cociente) y de intervalo, de gran utilidad en la estadística descriptiva.

Se define como la raíz cuadrada de la varianza. Junto con este valor, la desviación típica es una medida (cuadrática) que informa de la media de distancias que tienen los datos respecto de su media aritmética, expresada en las mismas unidades que la variable.

Intervalo de Confianza

Se llama intervalo de confianza en estadística a un par de números entre los cuales se estima que estará cierto valor desconocido con una determinada probabilidad de acierto. Formalmente, estos números determinan un intervalo, que se calcula a partir de datos de una muestra, y el valor desconocido es un parámetro poblacional. La probabilidad de éxito en la estimación se representa por 1 -α y se denomina nivel de confianza. En estas circunstancias, α es el llamado error aleatorio o nivel de significación, esto es, una medida de las posibilidades de fallar en la estimación mediante tal intervalo.

El nivel de confianza y la amplitud del intervalo varían conjuntamente, de forma que un intervalo más amplio tendrá más posibilidades de acierto (mayor nivel de confianza), mientras que para un intervalo más pequeño, que ofrece una estimación más precisa, aumentan sus posibilidades de error. Su fórmula es:

$$IC = \bar{x} \pm z(S\bar{x}) \Rightarrow \ donde \quad S\bar{x} = \frac{S}{\sqrt{n}}$$

$$Z = Valor\ de\ desconfianza\ (5\%) de\ la\ districución\ normal.$$

Varianza

En teoría de probabilidad, la varianza (σ2) de una variable aleatoria es una medida de su dispersión definida como la esperanza del cuadrado de la desviación de dicha variable respecto a su media.

Está medida en unidades distintas de las de la variable. Por ejemplo, si la variable mide una distancia en metros, la varianza se expresa en metros al cuadrado. La desviación estándar, la raíz cuadrada de la varianza, es una medida de dispersión alternativa expresada en las mismas unidades. Su fórmula es:

$$S^2 = \frac{\sum(x_i - \bar{x})^2}{n-1}$$

Distribución t de Student

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

Aparece de manera natural al realizar la prueba t de Student para la determinación de las diferencias entre dos medias muestrales y para la construcción del intervalo de confianza para la diferencia entre las medias de dos poblaciones cuando se desconoce la desviación típica de una población y ésta debe ser estimada a partir de los datos de una muestra

Metodología XP

La programación extrema es una metodología de desarrollo ligera (o ágil) se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Se puede considerar la programación extrema

como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software.

C#

Es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

Microsoft SQL Server

Es un sistema para la gestión de bases de datos producido por Microsoft y basado en el modelo relacional. Sus lenguajes para consultas son T-SQL y ANSI SQL.

XAML

XAML es el acrónimo de Xtensible Application Markup Language, Lenguaje Extensible de Formato para Aplicaciones en español. Es un lenguaje declarativo basado en XML,

optimizado para describir gráficamente interfaces de usuarios visuales ricas desde el punto de vista gráfico.

Windows Presentation Foundation (**WPF**)

Es una tecnología de Microsoft, permite el desarrollo de interfaces de interacción en Windows tomando características de aplicaciones Windows y de aplicaciones web.

WPF ofrece una amplia infraestructura y potencia gráfica con la que es posible desarrollar aplicaciones visualmente atractivas, con facilidades de interacción que incluyen animación, vídeo, audio, documentos, navegación o gráficos 3D. Separa, con el lenguaje declarativo XAML y los lenguajes de programación de .NET, la interfaz de interacción de la lógica del negocio, propiciando una arquitectura Modelo Vista Controlador para el desarrollo de las aplicaciones.

Memoria RAM

La memoria de acceso aleatorio (random-access memory), se utiliza como memoria de trabajo para el sistema operativo, los programas y la mayoría del software. Es allí donde se cargan todas las instrucciones que ejecutan el procesador y otras unidades de cómputo. Se denominan "de acceso aleatorio" porque se puede leer o escribir en una posición de memoria con un tiempo de espera igual para cualquier posición, no siendo necesario seguir un orden para acceder a la información de la manera más rápida posible.

CPU

La Unidad Central de Procesamiento (Central Processing Unit), es el componente del computador y otros dispositivos programables, que interpreta las instrucciones contenidas en los programas y procesa los datos.

BIBLIOGRAFÍA

1. **JARRETT, W., JAMES, A**., Beginning Kinect Programming with the Microsoft Kinect SDK., 1ª ed., Washington DC., Editorial Apress United States 2012., Pp 167-221.

2. **KENDALL, K., KENDALL, J.,** Análisis y diseño de sistemas., 2ª ed., DistritoFederal – México., Editorial Pearson Educación México, 2005., Pp 321-322.

3. **MONTGOMERY C., RUNGER C.,** Probabilidad y Estadística Aplicadas a la Ingeniería., 6ª ed., Distrito Federal – México., Editorial McGraw-Hill México, 1996., Pp 321-322.

BIBLIOGRAFÍA INTERNET

4. **ARQUITECTURA MONOLÍTICA**

   http://msdn.microsoft.com/es-es/magazine/cc785479.aspx

   2011/05/24

5. **EL JUEGO DE LAS HERRAMIENTAS EN LA EDUCACIÓN**

   http://www.galeon.com/serespontaneo/aficiones1433579.html

   2011/07/27

6. **BUILDING COMPOSABLE APPS IN .NET 4 WITH THE MANAGED EXTENSIBILITY FRAMEWORK**

   http://msdn.microsoft.com/en-us/magazine/ee291628.aspx

   2011/07/23

7. **C# CODING CONVENTIONS**

   http://msdn.microsoft.com/en-us/library/vstudio/ff926074.aspx

   http://en.wikipedia.org/wiki/Coding_conventions

   2012/05/10

8. **COMPOSITE APPLICATION GUIDANCE ASSETS**

   http://msdn.microsoft.com/en-us/library/ff921093(v=pandp.20).aspx

   2012/04/11

9. **COMPOSITE APPLICATION LIBRARY**

   http://msdn.microsoft.com/en-us/library/ff647752.aspx

   2012/04/17

10. **DEPENDENCY INJECTION**

    http://www.blackwasp.co.uk/DependencyInjection.aspx

    2012/06/18

11. KINECT

http://en.wikipedia.org/wiki/Kinect

2011/07/16

12. LENGUAJE DE PROGRAMACIÓN

http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n

http://www.iec.csic.es/criptonomicon/java/quesjava.html

http://en.wikipedia.org/wiki/C_Sharp_(programming_language)

2012/06/15

13. MANAGED EXTENSIBILITY FRAMEWORK (MEF)

http://msdn.microsoft.com/en-us/library/dd460648.aspx

2012/05/26

14. MICROSOFT APPLICATION ARCHITECTURE GUIDE, 2ND EDITION

http://msdn.microsoft.com/en-us/library/ff650706.aspx

2012/05/15

15. MICROSOFT SQL SERVER

http://en.wikipedia.org/wiki/Microsoft_SQL_Server

2012/08/25

16. PRISM 4.1 - DEVELOPER'S GUIDE TO MICROSOFT PRISM

http://msdn.microsoft.com/en-us/library/gg406140.aspx

2012/06/12

17. SOFTWARE REQUIREMENTS SPECIFICATION

http://en.wikipedia.org/wiki/Software_requirements_specification

http://www.math.uaa.alaska.edu/~afkjm/cs401/IEEE830.pdf

2012/07/23

18. THE NEW METHODOLOGY

http://www.extremeprogramming.org/

http://www.extremeprogramming.org/rules.html

http://martinfowler.com/articles/newMethodology.html

2012/07/25

ANEXOS

ANEXO I

Desarrollo de Cálculos probabilísticos

En este anexo se muestra las fórmulas aplicadas y los valores que se establecieron para tomar la decisión de cuál de las dos arquitecturas se escoge para implementar la aplicación de estudio de ;a geografía ecuatoriana.

Los valores que se encontraron son la media, la varianza, desviación estándar e intervalo de confianza de cada una de las formas de desarrollo (monolítico, compuesto) así como de los parámetros a comparar: líneas de código escritas, tiempo de desarrollo, uso de CPU, uso de memoria RAM,tiempo de respuesta, tiempo de mantenimiento.

En resumen con los valores que se realizó los cálculos son los que se muestran en las siguientes tablas.

CÁLCULO DE LA MUESTRA

Para el cálculo de la muestra se tiene los siguientes datos:

N= población total, los 9 requisitos de la aplicación.

σ= Desviación estándar de la población tomaremos el valor constante de 0,5

z = Valor de confianza, 95% de confianza equivale a 1,96

e = Límite aceptable de error muestral con 0.05.

$$n = \frac{N * \sigma^2 * z^2}{(N-1) * e^2 + \sigma^2 * z^2}$$

$$n = \frac{9 * 0.5^2 * 1.96^2}{(9-1) * 0.05^2 + 0.5^2 * 3.84}$$

$$n = \frac{8.64}{0.98}$$

$$n = 9$$

Toma de mediciones

Una vez desarrollados los prototipos, se procedió a tomar las medidas de los parámetros, a continuación se muestra en las siguientes tablas los resultados obtenidos para los prototipos monolítico y compuesto con PRISM.

Monolítico						
	Líneas de código escritas	**Tiempo de Desarrollo (h)**	**Uso de CPU (%)**	**Uso de RAM (Mb)**	**Tiempo de Respuesta (s)**	**Tiempo de Mantenimiento (h)**
REQ1	597	24	4.860	82.400	1.521	1.5
REQ2	435	29.3	5.490	76.700	1.301	1.3
REQ3	220	12.15	5.560	147.010	3.401	0.5
REQ4	412	9	3.980	73.240	4.102	0.3
REQ5	302	17.5	5.790	169.000	4.221	2.5
REQ6	521	16.3	3.160	23.130	0.521	0.3
REQ7	479	18	4.170	59.670	2.002	1
REQ8	311	11	6.670	25.330	0.034	16.5
REQ9	298	9.15	5.080	86.140	1.201	8.5

Compuesto						
	Líneas de código escritas	**Tiempo de Desarrollo (h)**	**Uso de CPU (%)**	**Uso de RAM (Mb)**	**Tiempo de Respuesta (s)**	**Tiempo de mantenimiento (h)**
REQ1	645	28.4	4.960	105.200	1.821	0.5
REQ2	422	32.2	3.890	82.540	3.002	0.15
REQ3	250	13.15	3.120	167.340	3.517	0.1
REQ4	442	9.3	2.960	79.120	5.231	0.15
REQ5	312	16.15	5.120	95.320	3.819	0.45
REQ6	546	21.15	3.090	41.010	1.012	0.25
REQ7	523	17.25	3.340	63.050	1.092	0.45
REQ8	298	12	6.340	35.000	0.504	2.25
REQ9	265	13.25	6.170	106.140	1.601	2.3

Cálculo de la media poblacional

Para el cálculo de la media se tomaron las medidas de los valores de cada uno de los parámetros considerados dentro del análisis estadístico comparativo.

PROTOTIPO MONOLÍTICO

NÚMERO DE LÍNEAS DE CÓDIGO

Media

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

$$\bar{x} = 397{,}222$$

Varianza

$$S^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}$$

$S^2 = 15164{,}944$

Desviación estándar

$S = \sqrt{S^2}$

$S = 123{,}146$

Intervalo de confianza

$IC = \bar{x} \pm z(S\bar{x})$ $\qquad$ $S\bar{x} = \frac{S}{\sqrt{n}}$

$IC = \bar{x} \pm z(S\bar{x})$

$IC = [316{,}767\ ;\ 477{,}678$

TIEMPO DE DESARROLLO

Media

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

$\bar{x} = 16{,}267$

Varianza

$$S^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}$$

$S^2 = 47{,}792$

Desviación estándar

$S = \sqrt{S^2}$

$S = 6{,}913$

Intervalo de confianza

$\boldsymbol{IC = \overline{x} \pm\ z(S\overline{x})}$ $\boldsymbol{S\overline{x} = \frac{S}{\sqrt{n}}}$

$\boldsymbol{IC = \overline{x} \pm\ z(S\overline{x})}$

$\boldsymbol{IC} = [11{,}750\,;20{,}783]$

USO DE CPU

Media

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

$\bar{x}$ =4,973

Varianza

$$S^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}$$

$S^2 = 1{,}138$

Desviación estándar

$S = \sqrt{S^2}$

$S = 1{,}067$

Intervalo de confianza

$IC = \bar{x} \pm z(S\bar{x})$ $S\bar{x} = \frac{S}{\sqrt{n}}$

$IC = \bar{x} \pm z(S\bar{x})$

$IC = [4{,}276\,;5{,}670]$

USO DE MEMORIA RAM

Media

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

$\bar{x} = 82{,}513$

Varianza

$$S^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}$$

$S^2 = 2386{,}356$

Desviación estándar

$S = \sqrt{S^2}$

$S = 48{,}850$

Intervalo de confianza

$IC = \bar{x} \pm z(S\bar{x})$ $S\bar{x} = \frac{S}{\sqrt{n}}$

$IC = \bar{x} \pm z(S\bar{x})$

$$IC = [50{,}598\ ;\ 114{,}429]$$

TIEMPO DE RESPUESTA

Media

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

$\bar{x} = 2{,}034$

Varianza

$$S^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}$$

$S^2 = 2{,}339$

Desviación estándar

$S = \sqrt{S^2}$

$S = 1{,}529$

Intervalo de confianza

$IC = \bar{x} \pm z(S\bar{x})$ $\qquad$ $S\bar{x} = \frac{S}{\sqrt{n}}$

$IC = \bar{x} \pm z(S\bar{x})$

$IC = [1{,}035\ ; 3{,}033]$

TIEMPO DE MANTENIMIENTO

Media

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

$\bar{x} = 3{,}600$

Varianza

$$S^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}$$

$S^2 = 29{,}935$

Desviación estándar

$S = \sqrt{S^2}$

$S = 5{,}471$

Intervalo de confianza

$IC = \bar{x} \pm z(S\bar{x})$ $\qquad$ $S\bar{x} = \frac{S}{\sqrt{n}}$

$IC = \bar{x} \pm z(S\bar{x})$

$IC = [0{,}025\,;\, 7{,}175]$

PROTOTIPO COMPUESTO

NÚMERO DE LÍNEAS DE CÓDIGO ESCRITAS

Media

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

$\bar{x} = 411{,}444$

Varianza

$$S^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}$$

$S^2 = 19551{,}528$

Desviación estándar

$S = \sqrt{S^2}$

$S = 139{,}827$

Intervalo de confianza

$IC = \bar{x} \pm z(S\bar{x})$ $\qquad$ $S\bar{x} = \frac{S}{\sqrt{n}}$

$IC = \bar{x} \pm z(S\bar{x})$

$IC = [320{,}091\ ;\ 502{,}798]$

TIEMPO DE DESARROLLO

Media

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

$\bar{x} = 18{,}094$

Varianza

$$S^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}$$

$S^2 = 60{,}175$

Desviación estándar

$S = \sqrt{S^2}$

$S = 7{,}757$

Intervalo de confianza

$IC = \bar{x} \pm z(S\bar{x})$ $\qquad$ $S\bar{x} = \frac{S}{\sqrt{n}}$

$IC = \bar{x} \pm z(S\bar{x})$

$IC = [13{,}026\,;\, 23{,}163]$

USO DE CPU

Media

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

$\bar{x} = 4.332$

Varianza

$$S^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}$$

$S^2 = 1{,}812$

Desviación estándar

$S = \sqrt{S^2}$

$S = 1{,}346$

Intervalo de confianza

$IC = \bar{x} \pm z(S\bar{x})$ $\quad$ $S\bar{x} = \frac{S}{\sqrt{n}}$

$IC = \bar{x} \pm z(S\bar{x})$

$IC = [3{,}453\,;\, 5{,}212]$

USO DE MEMORIA RAM

Media

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

$\bar{x} = 86{,}080$

Varianza

$$S^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}$$

$S^2 = 1586{,}046$

Desviación estándar

$S = \sqrt{S^2}$

$S = 39{,}825$

Intervalo de confianza

$IC = \bar{x} \pm z(S\bar{x})$ $S\bar{x} = \frac{S}{\sqrt{n}}$

$IC = \bar{x} \pm z(S\bar{x})$

$IC = [60{,}061\,;\,112{,}099]$

TIEMPO DE RESPUESTA

Media

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

$\bar{x} = 2{,}400$

Varianza

$$S^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}$$

$S^2 = 2{,}481$

Desviación estándar

$S = \sqrt{S^2}$

$S = 1{,}575$

Intervalo de confianza

$IC = \bar{x} \pm z(S\bar{x})$ $S\bar{x} = \frac{S}{\sqrt{n}}$

$IC = \bar{x} \pm z(S\bar{x})$

$IC = 1{,}371\ ;3{,}429$

TIEMPO DE MANTENIMIENTO

Media

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

$\bar{x} = 0{,}733$

Varianza

$$S^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}$$

$$S^2 = 0{,}786$$

Desviación estándar

$S = \sqrt{S^2}$

$S = 0{,}886$

Intervalo de confianza

$IC = \bar{x} \pm\ z(S\bar{x})$ $\qquad$ $S\bar{x} = \frac{S}{\sqrt{n}}$

$IC = \bar{x} \pm\ z(S\bar{x})$

$IC = [0{,}154\ ;1{,}312]$

A continuación se muestra en las siguientes tablas los resultados obtenidos para los prototipos monolítico y compuesto con PRISM.

Monolítico						
	Líneas de código escritas	**Tiempo de Desarrollo (h)**	**Uso de CPU (%)**	**Uso de RAM (Mb)**	**Tiempo de Respuesta (s)**	**Tiempo de Mantenimiento (h)**
REQ1	597	24	4.860	82.400	1.521	1.5
REQ2	435	29.3	5.490	76.700	1.301	1.3
REQ3	220	12.15	5.560	147.010	3.401	0.5
REQ4	412	9	3.980	73.240	4.102	0.3
REQ5	302	17.5	5.790	169.000	4.221	2.5
REQ6	521	16.3	3.160	23.130	0.521	0.3
REQ7	479	18	4.170	59.670	2.002	1
REQ8	311	11	6.670	25.330	0.034	16.5
REQ9	298	9.15	5.080	86.140	1.201	8.5

Compuesto						
	Líneas de código escritas	**Tiempo de Desarrollo (h)**	**Uso de CPU (%)**	**Uso de RAM (Mb)**	**Tiempo de Respuesta (s)**	**Tiempo de mantenimiento (h)**
REQ1	645	28.4	4.960	105.200	1.821	0.5
REQ2	422	32.2	3.890	82.540	3.002	0.15
REQ3	250	13.15	3.120	167.340	3.517	0.1
REQ4	442	9.3	2.960	79.120	5.231	0.15
REQ5	312	16.15	5.120	95.320	3.819	0.45
REQ6	546	21.15	3.090	41.010	1.012	0.25
REQ7	523	17.25	3.340	63.050	1.092	0.45
REQ8	298	12	6.340	35.000	0.504	2.25
REQ9	265	13.25	6.170	106.140	1.601	2.3

ANEXO II

Historias de Usuario

Las historias de usuario representan los requisitos del software, son descritos en una o dos frases utilizando el lenguaje común del usuario; a continuación se detallan algunas historias de usuario para la aplicación de estudio de la geografía ecuatoriana.

Historia de usuario	
Número:01	**Nombre**: Módulo de la geografía ecuatoriana
Usuario: Docente y estudiante	
Modificación de Historia Número:	**Iteración Asignada**: 1
Prioridad de Negocio: Alta	**Puntos Estimados:**40 PE
Riesgo en Desarrollo: Bajo	**Puntos Reales:**20 días
Descripción : Como usuario quiero tener el contenido importante de la clase para poder apoyar en el proceso de enseñanza aprendizaje y lograr captar la atención de los estudiantes.	
Prueba de Aceptación: Que el contenido requerido se muestre categorizado.	

Historia de usuario	
Número: 02	**Nombre**: Búsqueda por voz
Usuario: Docente y estudiante	
Modificación de Historia Número:	**Iteración Asignada:** 2
Prioridad de Negocio: Alta	**Puntos Estimados:**40 PE
Riesgo en Desarrollo: Bajo	**Puntos Reales:**20 días
Descripción : Como usuario quiero realizar búsquedas en internet por medio de la voz para poder enriquecer el contenido de clase dictada	
Pruebas de Aceptación: El sistema debe interactuar con los comandos de voz registrados para que el usuario pueda hacer uso de la aplicación. En caso de que el comando de voz sea erróneo debe mostrar sugerencias de los mismos.	

Historia de usuario	
Número: 03	**Nombre**: Interacción mediante movimientos corporales
Usuario: Docente y estudiante	
Modificación de Historia Número:	**Iteración Asignada:** 3
Prioridad de Negocio: Alta	**Puntos Estimados:** 40 PE
Riesgo en Desarrollo: Bajo	**Puntos Reales:**20 días
Descripción : Como usuario quiero manipular la aplicación con movimientos corporales y con voz, para poder amenizar la clase.	
Pruebas de Aceptación: El sistema debe permitir interactuar con movimientos corporales, para que el usuario pueda hacer uso de la aplicación. En caso de que el usuario no sea detectado se mostrara una pantalla que le indique que debe acercarse al dispositivo Kinect.	

Historia de usuario	
Número: 04	**Nombre:**Visualización del avance de la clase
Usuario: Docente y estudiante	
Modificación de Historia Número:	**Iteración Asignada**: 4
Prioridad de Negocio: Alta	**Puntos Estimados:**40 PE
Riesgo en Desarrollo: Bajo	**Puntos Reales**: 20 días
Descripción : Como usuario deseo visualizar el avance de la clase dictada por medio de gráficos, para poder planificar las actividades posteriores.	
Pruebas de Aceptación: El usuario debe poder visualizar en la aplicación el avance de la clase dictada.	

Historia de usuario	
Número: 05	**Nombre:**Actualización del contenido desde internet
Usuario: Docente y estudiante	
Modificación de Historia Número:	**Iteración Asignada**: 5
Prioridad de Negocio: Alta	**Puntos Estimados:**40 PE
Riesgo en Desarrollo: Bajo	**Puntos Reales:**20 días
Descripción : Como usuario quiero actualizar el contenido de la aplicación desde internet para poder mantener el contenido avalado por el ministerio de educación para dictar la clase.	
Pruebas de Aceptación: El usuario debe poder actualizar el contenido de la aplicación desde internet, se mostrará un mensaje para que el usuario verifique la conexión a internet.	

Historia de usuario	
Número: 06	**Nombre:**Actividades a realizar
Usuario: Docente y estudiante	
Modificación de Historia Número:	**Iteración Asignada**: 6
Prioridad de Negocio: Alta	**Puntos Estimados:** 40 PE
Riesgo en Desarrollo: Bajo	**Puntos Reales:** 20 días
Descripción : Como usuario quiero tener en la aplicación varias tareas a realizar durante el dictado de la clase, para poder captar la atención de los estudiantes.	
Pruebas de Aceptación: El usuario debe poder visualizar en la aplicación las actividades a realizar en la clase dictada.	
Historia de usuario	

Número: 07	**Nombre:** Configuraciones
Usuario: Docente y estudiante	
Modificación de Historia Número:	**Iteración Asignada**: 7
Prioridad de Negocio: Alta	**Puntos Estimados:** 40 PE
Riesgo en Desarrollo: Bajo	**Puntos Reales:** 20 días
Descripción : Como usuario quiero tener en la aplicación la posibilidad de realizar configuraciones para activar o desactivar sugerencias de comandos y visualización del avatar, para lograr mejorar la experiencia de uso de la aplicación.	
Pruebas de Aceptación: El usuario debe poder realizar configuraciones en la aplicación.	

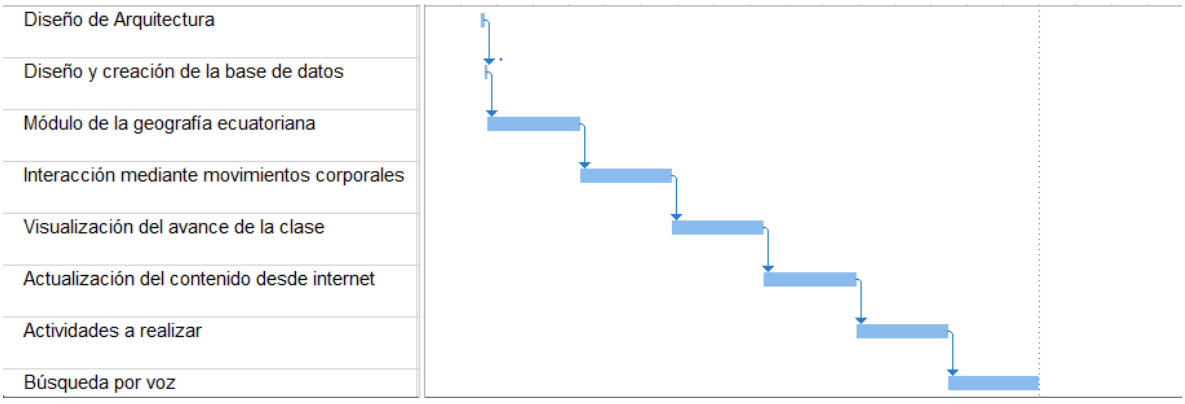
Historia de usuario	
Número: 08	**Nombre:** Diseño de Arquitectura
Usuario: Desarrollador	
Modificación de Historia Número:	**Iteración Asignada**: 7
Prioridad de Negocio: Alta	**Puntos Estimados:** 1 PE
Riesgo en Desarrollo: Bajo	**Puntos Reales:** 0.5días
Descripción : Se realizará el diseño de la arquitectura de la solución.	
Pruebas de Aceptación:	

Historia de usuario	
Número: 09	**Nombre:** Diseño y creación de la base de datos
Usuario: Desarrollador	
Modificación de Historia Número:	**Iteración Asignada**: 7
Prioridad de Negocio: Alta	**Puntos Estimados:** 1 PE
Riesgo en Desarrollo: Bajo	**Puntos Reales:**0.5 días
Descripción : Se realizará el diseño de la base de datos del sistema.	
Pruebas de Aceptación:	

ANEXO III

Plan de Iteraciones

En cada una de las iteraciones se muestra el número de tareas que se va a llevar a cabo y el tiempo en el que se desarrollará el cual está dado en semanas. El tiempo para el desarrollo de cada una de las iteraciones es de 4 semanas por iteración.



ANEXO IV

Gestión de Riesgos

DESCRIPCIÓN DE LOS RIESGOS

Se detalla los riesgos existentes en el desarrollo de la aplicación de estudio de la geografía ecuatoriana.

ID	Descripción del riesgo	Categoría	Consecuencia
R1	Error en la conexión con la base de datos.	R. Proyecto	Los datos no son almacenados u obtenidos.
R2	Los usuarios modifican los requerimientos del proyecto durante el desarrollo del software.	R. Proyecto	Retraso del proyecto, trabajo baldío del Software.
R3	La interfaz de usuario resulta compleja para el usuario.	R. Técnico	Pérdida de tiempo en la capacitación al personal que va a utilizar la aplicación.
R4	La aplicación excede en el consumo de recursos (memoria RAM y CPU) del computador.	R. Técnico	Lentitud en las tareas a realizar.
R5	Actualización de las herramientas de desarrollo	R. Técnico	Retraso en el proyecto.
R6	Costos altos de las herramientas de desarrollo y dispositivos a usar.	R. Proyecto	No realización del proyecto.
R7	Falta de documentación técnica en la tecnología a usar	R. Técnico	Retraso en el proyecto.
R8	Cambio de los directivos de la institución	R. Proyecto	Retraso del proyecto.

Rango de probabilidad	Descripción	Valor
1% - 33%	Baja	1
34% - 67%	Media	2
68% - 99%	Alta	3

Identificación	Probabilidad		
	%	Valor	Probabilidad
R1	70	3	Alta
R2	50	2	Media
R3	30	1	Baja
R4	50	2	Media
R5	20	1	Baja
R6	20	1	Baja
R7	30	1	Baja
R8	70	3	Alta

DETERMINACIÓN DEL IMPACTO

Impacto	Retraso	Impacto técnico	Impacto del costo	Valor
Bajo	1 Semana	Ligero efecto en el desarrollo del proyecto	< 1%	1
Moderado	2 Semana	Moderado efecto en el desarrollo del proyecto	< 5%	2
Alto	3 Semana	Severo efecto en el desarrollo del proyecto	< 10%	3
Critico	1 Mes	Proyecto no puede ser culminado	>= 10%	4

Identificación	Impacto	
	Valor	Impacto
R1	3	Alto
R2	2	Moderado
R3	1	Bajo
R4	2	Moderado
R5	1	Bajo
R6	1	Bajo
R7	1	Bajo
R8	3	Alto

DETERMINACIÓN DE LA EXPOSICIÓN AL RIESGO

Exposición	Valor	Color
Baja	1 o 2	Verde
Media	3 o 4	Amarillo
Alta	> 4	Rojo

Impacto / Probabilidad	Bajo = 1	Moderado = 2	Alto = 3	Crítico = 4
Alta =3	3	6	9	12
Media = 2	2	4	6	8
Baja = 1	1	2	3	4

Identificación	Exposición al riesgo	
	Valor	Exposición
R1	9	Alta
R2	4	Media
R3	1	Baja
R4	2	Baja
R5	1	Baja
R6	1	Baja
R7	1	Baja
R8	9	Alta

DETERMINACIÓN DE LA PRIORIDAD DEL RIESGO

ID	Descripción	Exposición	Prioridad
R1	Error en la conexión con la base de datos.	9	Alta
R8	Cambio de los directivos de la institución	9	Alta
R2	Los usuarios modifican los requerimientos del proyecto durante el desarrollo del software.	4	Media
	La aplicación excede en el consumo de recursos (memoria RAM y CPU) del computador.	2	Baja
R3	La interfaz de usuario resulta compleja para el usuario.	1	Baja
R5	Actualización de las herramientas de desarrollo	1	Baja
R6	Costos altos de las herramientas de desarrollo y dispositivos a usar.	1	Baja
R7	Falta de documentación técnica en la tecnología a usar	1	Baja

Línea de corte

HOJA DE GESTIÓN DEL RIESGO			
Id Riesgo: R1		**Fecha**: 09/06/2012	
Probabilidad: Alta **Valor:** 3	**Impacto**: Alto **Valor**: 3	**Exposición**: Alta **Valor**: 9	**Prioridad**: Alta
DESCRIPCIÓN: Error en la conexión con la base de datos.			
REFINAMIENTO: **Causa**: Los servicios de Microsoft SQL Server se detuvieron. **Consecuencia**: Los datos no son almacenados u obtenidos.			
REDUCCIÓN: Verificar la configuración y autorización demás permisos en el motor de base de datos y sistema operativo.			
SUPERVISIÓN: Consultar el archivo de logs para revisar posibles causas.			
GESTIÓN: Reiniciar los servicios.			
ESTADO ACTUAL: **Fase de reducción iniciada: X** **Fase de supervisión iniciada:** **Gestionando el riesgo:**			
RESPONSABLES: Desarrollador			

HOJA DE GESTIÓN DEL RIESGO			
Id Riesgo: R8		**Fecha**: 09/10/2012	
Probabilidad: Alta **Valor:** 3	**Impacto**: Alto **Valor**: 3	**Exposición**: Alta **Valor**: 9	**Prioridad**: Alta
DESCRIPCIÓN: Cambio de los directivos de la institución			
REFINAMIENTO: **Causa**: El rector de la institución y personal técnico encargado cambia constantemente. **Consecuencia**: Retraso del proyecto.			
REDUCCIÓN: Firma de acuerdos necesarios para garantizar el desarrollo de la aplicación.			
SUPERVISIÓN: Planificar reuniones con los directivos para mostrar los avances realizados en el desarrollo de la aplicación.			
GESTIÓN: Planificar reuniones periódicas durante el desarrollo del proyecto.			
ESTADO ACTUAL: **Fase de reducción iniciada: X** **Fase de supervisión iniciada:** **Gestionando el riesgo:**			
RESPONSABLES: Desarrollador			

HOJA DE GESTIÓN DEL RIESGO			
Id Riesgo: R2		**Fecha**: 09/10/2012	
Probabilidad: Media **Valor:** 2	**Impacto**: Medio **Valor**: 2	**Exposición**: Media **Valor**: 4	**Prioridad**: Media
DESCRIPCIÓN: Los usuarios modifican los requerimientos del proyecto durante el desarrollo del software.			
REFINAMIENTO: **Causa**: El usuario no pone en claro todos los requerimientos del software. **Consecuencia**: Retraso del proyecto, trabajo baldío del Software.			
REDUCCIÓN: Aplicar correctamente las técnicas de recopilación de información.			
SUPERVISIÓN: Planificar reuniones continuas con usuarios y desarrolladores antes de la implementación del proyecto hasta que los requerimientos estén totalmente definidos.			
GESTIÓN: Planificar reuniones periódicas durante el desarrollo del proyecto.			
ESTADO ACTUAL: **Fase de reducción iniciada: X** **Fase de supervisión iniciada:** **Gestionando el riesgo:**			
RESPONSABLES: Desarrollador			

ANEXO V

Tarjetas C.R.C

Bootstrapper

Bootstrapper	
Métodos CreateShell() InitialiazeShell() ConfigureContainer() ConfigureModuleCatalog() CreateLogger()	UnityBoostrapper

La tarjeta CRC pertenece a la clase Bootstrapper que es la encargada de poner en marcha la aplicación, carga los módulos iniciales e inicializa el shell, lanza ciertos servicios adicionales como el Logging para esto cuenta con los métodos CreateShell(), InitialiazeShell(), ConfigureContainer(), ConfigureModuleCatalog(), CreateLogger(). Esta clase hereda de la clase UnityBoostrapper.

SkeletalTracking

SkeletalTracking	
Propiedades IsRequired Kinect Messagge ShowRetry **Métodos** AppConfligOcurred() DiscoverSensor() UpdateStatus() KinectPropertyChanged() DrawStickMan()	

La tarjeta CRC de la clase SkeletalTracking muestra los atributos con los que cuenta, estos son: IsRequired, Kinect, Messagge, ShowRetry las mismas que son inicializadas en el constructor. La clase es la encargada de hacer uso del sensor kinect y detectar los movimientos corporales del usuario que se encuentre frente al dispositivo, para esto hace uso de los métodos detallados anteriormente. Esta clase no necesita de la colaboración de ninguna clase adicional.

SpeechRecognition

SpeechRecognition	
Propiedades Kinect **Métodos** SpeechEngineRecognitionRejected() SpeechEngineRecognized() InitializeSpeechGrammar()	

La tarjeta CRC pertenece a la clase SpeechRecognition que es la encargada de receptar la voz del usuario que se encuentre cerca del dispositivo kinect, con el cual se adquiere el sonido y se lo usa como comandos de voz para interactuar con la aplicación. Esta clase no necesita de la colaboración deninguna clase adicional.