UNIVERSIDAD NACIONAL HERMILIO VALDIZÁN

ESCUELA DE POST GRADO

MAESTRIA EN INGENIERÍA DE SISTEMAS

MENCION

TECNOLOGIAS DE INFORMACIÓN Y COMUNICACIÓN

PROYECTO DE TESIS

==========

METODOLOGIAS DE DESARROLLO DE SOFTWARE Y LOS DESARROLLADORES DE LAS MYPES EN TINGO MARÍA, 2017

==========

TESISTA:

Rannoverng YANAC MONTESINO

ASESOR:

MG. HUMBERTO FLORES FLORES

HUÁNUCO – PERÚ

2018

INDICE

INTRODUCCIÓN

La tarea de definir al proceso de desarrollo de software es una labor bastante compleja. Esto se debe a que existen diferentes puntos de vistas y concepciones. Esta problemática puede localizarse en las múltiples concepciones que se poseen acerca del software.

Para determinar las preferencias que tienen los de desarrolladores de software que trabajan en las MYPES de a ciudad de tingo maría es necesario realizar una investigación considerando a las 20 empresas que se dedican a este rubro. Con la finalidad de conocer: las razones y preferencias de utilizar una metodolgía que permita gestionar el proceso de construcción de un Sistema de Software de una manera uniforme. Sin embargo, las metodologías para software convencional no consideran adecuados para empresas que se dedican a llevar a cabo proyectos a medida

CAPITULO I

EL PROBLEMA DE INVESTIGACIÓN

1.1 DESCRIPCIÓN DEL PROBLEMA.

La tarea de definir al proceso de desarrollo de software es una labor bastante compleja. Esto se debe a que existen diferentes puntos de vistas y concepciones. Esta problemática puede localizarse en las múltiples concepciones que se poseen acerca del software (Vanzetti, 2006).

En todo desarrollo de sistemas de software, es de suma importancia el seguir alguna especificación que permita a los desarrolladores el tener una disciplina que haga que todas las etapas del desarrollo del sistema, desde la pesquisa inicial de requerimientos hasta las pruebas finales de los sistemas, sean no solo más coherentes sino también más formales. El desarrollo de software que este proyecto propone, al ser una herramienta que pretende tener aplicación dentro del contexto de un problema real, tiene que seguir un proceso de análisis y diseño que proporcione los cimientos bajo los cuales se va a desarrollar la aplicación conjuntamente. Los procesos de ingeniería de software, análisis y diseño que se involucran para el desarrollo de una aplicación de software que puede utilizarse como auxiliar al tratamiento del trastorno de lateralidad y ubicación espacial. (Fuentes, 2003)

El proceso de planificación de todo proyecto de software debe hacerse partiendo de una estimación del trabajo a realizar. Para obtener software de calidad es preciso medir el proceso de desarrollo, cuantificar lo que se ha hecho y lo que falta por hacer, estimar el tamaño del proyecto, costos, tiempo de desarrollo, control de calidad, mejora continua y otros parámetros. En este sentido, las métricas ayudan a entender tanto el proceso que se utiliza para desarrollar un producto, como el propio producto. Asimismo, tienen un papel decisivo en la obtención de un producto de alta calidad, porque determinan mediante estadísticas basadas en la experiencia, el avance del software y el cumplimiento de

parámetros requeridos. En la Ingeniería de Software, los proyectos de desarrollo tradicionales aplican una amplia diversidad de métricas e indicadores para especificar, predecir, evaluar y analizar distintos atributos y características de los productos y procesos que participan en el desarrollo y mantenimiento del software. La aplicación de un enfoque cuantificable es una tarea compleja que requiere disciplina, estudio y conocimiento de las métricas e indicadores adecuados para los distintos objetivos de medición y evaluación, con el fin de garantizar la calidad del software construido. (Martín, 2014).

Por esta razón ha sido necesario disponer de métodos eficientes, es decir, es necesaria una metodología que permita gestionar el proceso de construcción de un Sistema de Software de una manera uniforme. Sin embargo, las metodologías para software convencional no consideran adecuados para empresas que se dedican a llevar a cabo proyectos a medida. (Pytel, 2011).

1.2 FORMULACIÓN DEL PROBLEMA.

1.2.1 PROBLEMA GENERAL

¿Qué metodologías de desarrollo de software son las más usadas por los desarrolladores para la gestión del prceso de desarrollo en las MYPES en Tingo María, 2017?

1.2.2 PROBLEMAS ESPECIFICOS

- ¿Qué metodologías existen para el desarroolo de software?
- ¿Cuántas MYPES que se dedican al desarrollo de softare existen en la ciudad de Ting María?.
- ¿Cuáles son los factores que inciden en las preferencias para la selección y uso de las metodologías de desarrollo de software?

1.3 OBJETIVO GENERAL Y OBJETIVOS ESPECÍFICOS.

1.3.1 OBJETIVO GENERAL

Determinar las metodolgías más usadas por los desarrolladores de software para la gestión del proceso de desarrollo en las MYPES de la ciudad de Tingo María, 2017

1.3.2 OBJETIVOS ESPECÍFICOS

- Describir las metodolgías de desarrollo de software existente en la actualidad.
- Identificar las MYPES que se dedican al desarrollo de software.
- Determinar los factores que inciden en las preferencias para la seección y uso de las metodologías para la gestión del proceso de desarrollo.

1.4 HIPÓTESIS Y/O SISTEMA DE HIPÓTESIS.

1.4.1 HIPÓTESIS GENERAL

Se determinará las metodolgías más usadas por los desarrolladores de software para la gestión del proceso de desarrollo en las MYPES de la ciudad de Tingo María, 2017.

1.4.2 HIPÓTESIS ESPECÍFICOS

- Se describiár las metodolgías de desarrollo de software existente en la actualidad.
- Se identificará las MYPES que se dedican al desarrollo de software.
- Se determinará los factores que inciden en las preferencias para la seección y uso de las metodologías para la gestión del proceso de desarrollo.

1.5 VARIABLES

1.5.1 VARIABLE DEPENDIENTE

- Metodologías de Desarrollo de Software

1.5.2 VARIABLE INDEPENDIENTE

- Gestión del proceso en las MYPES de la ciudad de Tingo María

1.6 MATRIZ DE CONSISTENCIA DEL PROYECTO

<table>
<tr><th>PROBLEMA PRINCIPAL</th><th>OBJETIVO PRINCIPAL</th><th>JUSTIFICACIÓN</th><th>HIPÓTESIS PRINCIPAL</th><th colspan="3">VARIABLES</th><th>INDICADORES</th><th>TIPO Y DISEÑO</th></tr>
<tr><td rowspan="2">¿Qué metodología permitirá gestionar el proceso de construcción de un Sistema de Software?</td><td rowspan="2">Determinar la metodología que permitirá gestionar el proceso de construcción de un Sistema de Software.</td><td rowspan="8">La inmersión que ha tenido la tecnología en el día a día del ser humano, ha generado la necesidad de integrar procesos con componentes tecnológicos en la estructura de las metodologías de administración proyectos. Implementar una PMO (Oficina de Administración de Proyectos) en una empresa de outsourcing de tecnología, es un reto que para las empresas de este sector del mercado, requieren adoptar metodologías apropiadas para generar valor a su desarrollo, transformar sus procesos, ser innovadoras, incrementar su productividad y rentabilidad, y por supuesto, tener presente que la tecnología es sólo una herramienta y que los procesos son eficientes en la medida como se administren.</td><td rowspan="2">Existe una metodología que permite optimizar la gestión de proceso de construcción de un Sistema de Software.</td><td colspan="3">VARIABLE (Y): Sistema de Software.</td><td rowspan="10">El diseño de investigación es de tipo Descriptivo</td></tr>
<tr><td>Y1:</td><td>requerimientos</td><td>funcional
no funcional</td></tr>
<tr><td>Problemas Específicos</td><td>Objetivos Específicos</td><td>Hipótesis Específicas</td><td rowspan="4">Y2:</td><td rowspan="4">Diseño</td><td>funcionalidad
Relación
Adaptabilidad</td></tr>
<tr><td>¿Qué metodologías son empleadas para el desarrollo de software?</td><td>Seleccionar las metodologías que son empleadas para el desarrollo de software.</td><td>Las metodologías seleccionadas son empleadas para el desarrollo de software.</td></tr>
<tr><td rowspan="2">¿En qué se diferencian estas metodologías entre ellas mismas?</td><td rowspan="2">Elaborar un análisis comparativo entre las metodologías.</td><td rowspan="2">El análisis comparativo permite conocer las diferencias entre las metodologías.</td><td rowspan="3">Y3:</td><td rowspan="3">Codificación</td><td>Operatividad
Eficiencia
daptación</td></tr>
<tr><td colspan="3">VARIABLE (X): Metodología.</td></tr>
<tr><td colspan="2">X1:</td><td>Optimizar la gestión de proceso de construcción</td><td></td></tr>
</table>

1.7 JUSTIFICACIÓN E IMPORTANCIA.

La inmersión que ha tenido la tecnología en el día a día del ser humano, ha generado la necesidad de integrar procesos con componentes tecnológicos en la estructura de las metodologías de administración proyectos. Implementar una PMO (Oficina de Administración de Proyectos) en una empresa de outsourcingde tecnología, es un reto que para las empresas de este sector del mercado, requieren adoptar metodologías apropiadas para generar valor a su desarrollo, transformar sus procesos, ser innovadoras, incrementar su productividad y rentabilidad, y por supuesto, tener presente que la tecnología es sólo una herramienta y que los procesos son eficientes en la medida como se administren

1.8 VIABILIDAD

- La viabilidad de este proyecto se hace efectiva debido a que existen diferentes metodologías para el desarrollo, de acuerdo a la necesidad del usuario, y a su medida; Además permite ser sostenible, rentable económicamente. Tiene la capacidad de lograr un buen desempeño financiero

1.9 LIMITACIONES

1.8.1 LIMITACIÓN DE RECURSOS

El factor limitante el presente trabajo de investigación, es el desconocimiento de las MYPES que se dedican al desarrollo de software.

1.10 DELIMITACION

1.10.1 DELIMITACIÓN DE TIEMPO

Se contara contará con un tiempo de 3 meses.

Tiempo con la que disponen los estudiantes para el estudio de la asignatura; es decir, pocas horas se destinan al estudio de las horas de práctica.

1.10.2 DELIMITACIÓN DE ESPACIO

La investigación se realizará a las empresas (MYPES) que se dedican al desarrollo de software de la ciudad de Tingo María en el año 2017.

CAPITULO II

MARCO TEÓRICO

2.1 ANTECEDENTES.

INTERNACIONALES

- (Rivera, 2008) Impacto de la evolución de las tic's en la administración de las agencias de viajes minoristas de México En ésta investigación descriptiva, se busca analizar la Revolución Tecnológica en Información y Comunicación con el objetivo de estudiar los impactos positivos y negativos que ha producido en la administración de las Agencias de Viajes Minoristas. Además, se revisará el desarrollo tecnológico en la industria turística, describiendo su utilidad y sus repercusiones en el área mediante un instrumento de medición.

  Las Agencias de Viajes Minoristas se han visto beneficiadas por las TIC`s al reducir procedimientos, acortar distancias, reducción de costos, implementación de programas de reservaciones (GDS), sistemas que facilitan la administración y contabilidad de la empresa. Por otro lado, el alcance por Internet como la creación de líneas aéreas de bajo costo le quitan mercado a las Agencias de Viajes Minoristas; las alianzas entre hoteles y líneas aéreas también reducen las ventas en éstas organizaciones.

(Asin Luque, 1013) Implementacion de las TICS en la mipymes colombianas con la actividad comercial

Este trabajo de investigación analizará el entorno actual de penetración de las TIC en las MiPyMEs de Colombia, a la par que mostrará las bondades de incluir dichas tecnologías en el ámbito de la empresa comercial, además de analizar las herramientas hardware y software disponibles para ello. Finalmente esbozará los mecanismos y estrategias que pueden permitir a las MiPyMEs comerciales hacer un uso eficiente de las TIC.

NACIONALES

- (Julca Rodrigues) Tecnologías de la información aplicables al sector turismo del Perú

  El presente estudio analiza la situación tecnológica de las empresas turísticas de Ayacucho, departamento del Perú. Las empresas consideradas para este estudio son las que realizan actividades características del Turismo (definidas por la OMT), y se agrupan en cuatro categorías: Alojamientos hoteleros y de otro tipo, Restauración, Transporte de Viajeros y Agencias de Viaje.

  A pesar de que la gran mayoría de las empresas turísticas de Ayacucho no cuenta aún con los recursos tecnológicos que estos tiempos demandan, lo cierto es que existe un marcado interés en poder revertir esta situación y hay amplias posibilidades de aumentar la dotación tecnológica de estas empresas. A la hora de valorar estas posibilidades, deben hacerse precisiones sobre qué tipo de tecnologías podrían utilizarse más y qué tipo de empresas son las más susceptibles de mejorar su nivel tecnológico.

2.2 BASES TEÓRICAS.

Las tecnologías de la información y de las comunicaciones

- 3.2.1 ¿QUÉ ES UN MÉTODO?

- Un Método se compone de diversos aspectos que nos permitirán conseguir una meta o lograr un objetivo. Se define más claramente como un conjunto de herramientas, las cuales utilizadas mediante las técnicas correctas, permiten la ejecución de procesos que nos llevarán a cumplir los objetivos que buscamos. En pocas palabras y aunque esto lo puedes encontrar como tal en internet, es un conjunto de herramientas, técnicas y procesos que facilitan la obtención de un objetivo.
- 3.2.2 ¿QUÉ ES UNA METODOLOGÍA?
- En el desarrollo de software, una metodología hace cierto énfasis al entorno en el cuál se plantea y estructura el desarrollo de un sistema. Como lo mencioné al principio, existen una gran cantidad de metodologías de la programación que se han utilizado desde los tiempos atrás y que con el paso del tiempo han ido evolucionando. Esto se debe principalmente a que no todos los sistemas de la información, son compatibles con todas las metodologías, pues el ciclo de vida del software puede ser variable. Por esta razón, es importante que dependiendo del tipo de software que se vaya a desarrollar, se identifique la metodología para el diseño de software idónea.
- 3.2.3 ¿EN QUÉ CONSISTEN LAS METODOLOGÍAS DE DESARROLLO DE SOFTWARE?
- Una Metodología de desarrollo de software, consiste principalmente en hacer uso de diversas herramientas, técnicas, métodos y modelos para el desarrollo. Regularmente este tipo de metodología, tienen la necesidad de venir documentadas, para que los programadores que estarán dentro de la planeación del proyecto, comprendan perfectamente la metodología y en algunos casos el ciclo de vida del software que se pretende seguir.
- Aunque actualmente existen mucha variedad en metodologías de programación. La realidad es que todas están basadas en ciertos enfoques generalistas que se crearon hace muchos años, algunos tipos de metodologías de desarrollo de software que se utilizaron e inventaron al principio de nuestra era tecnológica y son las que

veremos a continuación.

- 3.2.4 METODOLOGÍAS DE DESARROLLO DE SOFTWARE
- 1970
- Programación estructurada sol desde 1969
- Programación estructurada Jackson desde 1975
- 1980
- Structured Systems Analysis and Design Methodology (SSADM) desde 1980
- Structured Analysis and Design Technique (SADT) desde 1980
- Ingeniería de la información (IE/IEM) desde 1981
- 1990
- Rapid application development (RAD) desde 1991.
- Programación orientada a objetos (OOP) a lo largo de la década de los 90's
- Virtual finite state machine (VFSM) desde 1990s
- Dynamic Systems Development Method desarrolladoen UK desde 1995.
- Scrum (desarrollo), en la última parte de los 90's
- Rational Unified Process (RUP) desde 1999.
- Extreme Programming(XP) desde 1999
- Nuevo milenio
- Enterprise UnifiedProcess (EUP) extensiones RUP desde 2002
- Constructionist design methodology (CDM) desde 2004 por Kristinn R. Thórisson
- Agile Unified Process (AUP) desde 2005 por Scott Ambler
- 3.2.5 ENFOQUES DE DESARROLLO DE SOFTWARE
- Cada metodología de desarrollo de software tiene más o menos su propio enfoque para el desarrollo de software. Estos son los enfoques más generales, que se desarrollan en varias metodologías específicas. Estos enfoques son los siguientes:
- Modelo en cascada: Framework lineal.
- Prototipado: Framework iterativo.
- Incremental: Combinación de framework lineal e iterativo.
- Espiral: Combinación de framework lineal e iterativo.

- RAD: Rapid Application Development, framework iterativo.
- 3.2.6 METODOLOGIAS TRADICIONALES
- 3.2.6.1 INTRODUCCION
- En febrero de 2001, tras una reunión celebrada en Utah-EEUU, nace el término ágil aplicado al desarrollo de software. En esta reunión participan un grupo de 17 expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías de software. Su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto.
- Se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.
- Tras esta reunión se creó The Agile Alliance, una organización, sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos. El punto de partida es fue el manifiesto Ágil, un documento que resume la filosofía ágil.
- Principales ideas de la metodología ágil:
- Se encarga de valorar al individuo y las iteraciones del equipo más que a las herramientas o los procesos utilizados.
- Se hace mucho más importante crear un producto software que funcione, que escribir mucha documentación.
- El cliente está en todo momento colaborando en el proyecto.
- Es más importante la capacidad de respuesta ante un cambio realizado que el seguimiento estricto de un plan.
- 3.2.6.2 VENTAJAS Y DESVENTAJAS
- Ventajas:
- La primera y más importante, es que estas metodologías ofrecen una rápida respuesta a cambios de requisitos a lo largo del desarrollo del proyecto gracias a su proceso iterativo, es tan importante realizar una buena recolecta de requisitos, como después poder modificarlos

evitando grandes pérdidas en cuanto a costes, motivación, tiempo.

- El cliente, si quiere colaborar, puede observar cómo va avanzando el proyecto, y por supuesto, opinar sobre su evolución gracias a las numerosas reuniones que realiza el equipo con el cliente. Esto le da tranquilidad.
- Uniendo las dos anteriores, se puede deducir que al utilizar estas metodologías, los cambios que quiera realizar el cliente van a tener un menor impacto, ya que se va a entregar en un pequeño intervalo de tiempo un pequeño "trozo" del proyecto al cliente, y si éste quiere cambiarlo, solo se habrá perdido unas semanas de trabajo. Con las metodologías tradicionales las entregas al cliente se realizaban tras la realización de una gran parte del proyecto, eso quiere decir que el equipo ha estado trabajando meses para que luego un mínimo cambio que quiera realizar el cliente, conlleve la pérdida de todo ese trabajo.
- Importancia de la simplicidad al eliminar trabajo innecesario.
- Otros beneficios de las metodologías ágiles.
- Simplificación de la sobrecarga de procesos.
- Los equipos que trabajan para crear productos regulados por estándares de la industria, deben demostrar el cumplimiento de esas normas. Estos equipos suelen adoptar importantes sobrecargas de trabajo para asegurarse de que cumplen con los estrictos mandatos de código. Las metodologías ágiles pueden ayudarles a cumplir los estándares de la industria con menos sobrecarga utilizando iteraciones más cortas y empaquetadas. El beneficio neto es un proceso que:
- Puede adaptarse a los cambios que, inevitablemente, surgirán
- Requiere menos sobrecarga en el proceso end-to-end
- Implica menos trabajo a medida que se acerca la fecha final
- Calidad mejorada
- Las prácticas de desarrollo ágil proporcionan la funcionalidad suficiente como para satisfacer las expectativas de los accionistas con una alta calidad. Piensa en lo que significa "ofrecer lo suficiente". Eso es, proporcionar la mínima funcionalidad con la máxima calidad.

La mínima funcionalidad no implica necesariamente una pobre funcionalidad, implica lo suficiente como para conseguir que el trabajo se realice. Los accionistas suelen pensar que saben lo que quieren cuando especifican altos niveles de requerimientos para un producto TI o de software. Sin embargo, la mayoría de las veces, cuando ven el producto final, éste no resuelve el problema. Simplemente, no se han imaginado de forma precisa el mismo, o el problema ha cambiado o, incluso, la tecnología no era tan buena como parecía. También puede ser que el producto no funcione realmente del modo en que las partes interesadas tenían intención, incluso aunque pensaran que habían descrito los requerimientos de manera clara. Desarrollar iteraciones en poco tiempo y demostrar a los accionistas los productos pronto y con frecuencia, permite tanto a los accionistas como a los equipos de desarrollo ponerse de acuerdo y coincidir en que el producto cumple con cada una de sus necesidades.

- Mejorar la previsibilidad a través de una mejor gestión del riesgo.
- Cuando los equipos de desarrollo no cumplen con sus fechas de lanzamiento, a menudo se debe a muchas razones completamente justificables. Puede ser que el equipo no hubiera entendido bien lo difícil que sería utilizar la nueva tecnología; los requerimientos podían no estar del todo claros o los clientes cambiaron de opinión cuando el trabajo estaba prácticamente finalizado. En cualquier caso, los negocios demandan productos que cumplan con las fechas de entrega, de modo que los planes de negocio directamente relacionados con ellos puedan cumplirse. Hay muchas formas en las que las metodologías ágiles pueden ayudar a que los proyectos TI cumplan con las fechas de lanzamiento previstas.
- Dar prioridad a los riegos. Las prácticas ágiles priorizan los aspectos de desarrollo de alto riesgo permitiendo así una reducción de los riesgos iniciales.
- Evaluación de riesgos en paralelo. Para áreas de riesgo donde debería haber múltiples soluciones y el equipo no se pone de acuerdo a la hora de tomar el camino más adecuado, se debe tener

en cuenta la posibilidad de optar por el desarrollo multi-set. Esto requiere que diferentes equipos trabajen en paralelo resolviendo el mismo problema con diferentes soluciones. La mayoría de los equipos no considerarán ni tan siquiera esta forma de trabajar, porque están convencidos de que el tiempo y el coste requeridos para hacer una evaluación paralela son demasiado grandes. De hecho, el desarrollo paralelo de alternativas es probable que traiga consigo las decisiones clave a seguir.

- Mejor perfil de productividad
- Los equipos ágiles son más productivos que aquellos que utilizan métodos tradicionales a lo largo de todo el ciclo de desarrollo. El desarrollo tradicional suele mostrar un patrón de trabajo parecido a un palo de hockey, empezando con un ciclo de diseño de abajo arriba, moviéndose hasta una fase de prototipo para pasar luego a un ciclo de desarrollo largo, seguido por otro ciclo totalmente impredecible en el que se integran las piezas para pasar, por último, a la fase de prueba final. A medida que el proyecto progresa, los equipos tienen que trabajar juntos de manera más coherente, confiando en que todas las piezas trabajen juntas tal y como esperan. Pero lo cierto es que esto raramente ocurre, de modo que la interacción entre los equipos aumenta a medida que lo hacen los problemas de integración. Por último, la fase de pruebas lleva todo esto al límite. Trabajando juntos como un único equipo en fases verticales del producto desde el principio del ciclo de producción, se evita el ciclo de productividad tradicional de palo de hockey. Los equipos ágiles tienden a ser muy productivos desde la primera iteración hasta su lanzamiento y su ritmo tiene que ser gestionado de modo que no se produzca agotamiento. Los equipos ágiles que mantienen este código de trabajo con cada iteración, permiten realizar pruebas de rendimiento y sistemas desde el principio, pudiendo empezar en las primeras iteraciones. De este modo, defectos críticos como problemas de integración se descubren antes, la calidad general del producto es mayor y el equipo funciona de manera más productiva durante todo el ciclo de desarrollo.

- Capacidad para aprovechar las inversiones realizadas.
- Adoptar las técnicas ágiles de manera satisfactoria precisa un importante soporte de herramientas. La mayoría de los equipos invierten fuertemente en buenas herramientas de software y necesitan elevar esa inversión y reducir la cantidad de cambios cuando empiezan a adoptar las metodologías ágiles. La inversión más crítica es una buena planificación y una herramienta de gestión del trabajo que proporcione una cartera de pedidos del equipo visible y que asocie el trabajo con cada elemento de trabajo en cartera. Idealmente, esa herramienta de planificación se integra con otras herramientas de desarrollo, lo que permite a los equipos mantener la trazabilidad de la cartera de pedidos en otros aspectos, incluyendo:
- Los requerimientos que la impulsan.
- La arquitectura bajo desarrollo.
- El software de la solución.
- Las pruebas que validan la solución.
- Las herramientas de Rational trabajan muy bien juntas y con otros productos para ayudar a entregar con éxito proyectos ágiles. IBM Rational Team Concert es el componente central de colaboración y flujo de trabajo de la solución IBM Rational para ingeniería de software y sistemas. Proporciona una interfaz Open Services for Lifecycle Collaboration (OSLC) que permite a las herramientas capacitadas OSLC integrarse sin problemas. Rational Team Concert proporciona al equipo planificación ágil, visible y probada. De hecho, su adopción para simplemente ese propósito ha sido viral en IBM Software Group. Rational Team Concert proporciona gestión de elementos, planificación de equipos asociada a cargas de trabajo y visibilidad de proyectos, todos ellos, elementos críticos en los proyectos ágiles. Los elementos de trabajo en Rational Team Concert son muy flexibles en su uso y pueden asociarse con lanzamientos, equipos y demás.
- Desventajas:
- Falta de documentación del diseño. Al no haber documentación es el código (junto con sus comentarios) lo que se toma como

documentación.

- Problemas derivados de la comunicación oral. No hace falta decir que algo que está escrito “no se puede borrar”, en cambio, algo dicho es muy fácil crear ambigüedad.
- Fuerte dependencia de las personas.
- Falta de reusabilidad derivada de la falta de documentación
- Restricciones en cuanto a tamaño de los proyectos
- Problemas derivados del fracaso de los proyectos ágiles. Si un proyecto ágil fracasa no hay documentación o hay muy poca; lo mismo ocurre con el diseño. La comprensión del sistema se queda en las mentes de los desarrolladores.
- 3.2.6.3 TIPOS DE METODOLOGIAS
- 3.2.6.3.1 WATERFALL (CASCADA)
- En Ingeniería de software el desarrollo en cascada, es denominado así por la posición de las fases en el desarrollo de esta, que parecen caer en cascada “por gravedad” hacia las siguientes fases. Es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior. Al final de cada etapa, el modelo está diseñado para llevar a cabo una revisión final, que se encarga de determinar si el proyecto está listo para avanzar a la siguiente fase. Este modelo fue el primero en originarse y es la base de todos los demás modelos de ciclo de vida.
- Este modelo comenzó a diseñarse en 1966 y se terminó alrededor de 1970. El principal problema de esta aproximación es el que no podemos esperar que las especificaciones iníciales sean correctas y completas y que el usuario puede cambiar de opinión sobre una u otra característica.
- Además los resultados no se pueden ver hasta muy avanzado el proyecto por lo que cualquier cambio debido a un error puede suponer un gran retraso además de un alto coste de desarrollo.
- Como es evidente esto es solo un modelo teórico, si el usuario cambia de opinión en algún aspecto tendremos que volver hacia atrás en el ciclo de vida.

-
- Figura Nº 1 – Modelo del ciclo de vida en Cascada (Waterfall).
- Ingeniería y Análisis del Sistema: debido que el software es siempre parte de un sistema mayor, el trabajo comienza estableciendo los requisitos de todos los elementos del sistema y luego asignando algún subconjunto de estos requisitos al software.
- Análisis de los requisitos del software: el proceso de recopilación de los requisitos se centra e intensifica especialmente en el software. El ingeniero de software (Analistas) debe comprender el ámbito de la información del software, así como la función, el rendimiento y las interfaces requeridas.
- Diseño: el diseño del software se enfoca en cuatro atributos distintos del programa: la estructura de los datos, la arquitectura del software, el detalle procedimental y la caracterización de la interfaz. El proceso de diseño traduce los requisitos en una representación del software con la calidad requerida antes de que comience la codificación.
- Codificación: el diseño debe traducirse en una forma legible para la máquina. El paso de codificación realiza esta tarea. Si el diseño se realiza de una manera detallada la codificación puede realizarse mecánicamente.
- Prueba: una vez que se ha generado el código comienza la prueba del programa. La prueba se centra en la lógica interna del software, y en las funciones externas, realizando pruebas que aseguren que la entrada definida produce los resultados que realmente se requieren.
- Mantenimiento: el software sufrirá cambios después de que se entrega al cliente. Los cambios ocurrirán debido a se han encontrado errores, a que el software deba adaptarse a cambios del entorno externo (sistema operativo o dispositivos periféricos), o debido a que el cliente requiera ampliaciones funcionales o del rendimiento.
- Desventajas:
- Los proyectos reales raramente siguen el flujo secuencial que propone el modelo, siempre hay iteraciones y se crean problemas en la aplicación del paradigma.
- Normalmente, es difícil para el cliente establecer explícitamente

al principio todos los requisitos. El ciclo de vida clásico lo requiere y tiene dificultades en acomodar posibles incertidumbres que pueden existir al comienzo de muchos productos.

- El cliente debe tener paciencia. Hasta llegar a las etapas finales del proyecto, no estará disponible una versión operativa del programa. Un error importante no detectado hasta que el programa esté funcionando puede ser desastroso.
- 3.2.6.3.2 PROTOTIPO
- Un prototipo es una versión preliminar de un sistema de información con fines de demostración o evaluación.
- El prototipo de requerimientos es la creación de una implementación parcial de un sistema, para el propósito explícito de aprender sobre los requerimientos del sistema. Un prototipo es construido de una manera rápida tal como sea posible.
- Esto es dado a los usuarios, clientes o representantes de ellos, posibilitando que ellos experimenten con el prototipo. Estos individuos luego proveen la retroalimentación sobre lo que a ellos les gustó y no les gustó acerca del prototipo proporcionado, quienes capturan en la documentación actual de la especificación de requerimientos la información entregada por los usuarios para el desarrollo del sistema real. El prototipo puede ser usado como parte de la fase de requerimientos (determinar requerimientos) o justo antes de la fase de requerimientos (como predecesor de requerimientos). En otro caso, el prototipo puede servir su papel inmediatamente antes de algún o todo el desarrollo incremental en modelos incremental o evolutivo.
- El prototipo ha sido usado frecuentemente en los 90, porque la especificación de requerimientos para sistemas complejos tiende a ser relativamente dificultoso de cursar. Muchos usuarios y clientes encuentran que es mucho más fácil proveer retroalimentación convenientemente basada en la manipulación, leer una especificación de requerimientos potencialmente ambigua y extensa.
- Diferente del modelo evolutivo donde los requerimientos mejor entendidos están incorporados, un prototipo generalmente se

construye con los requerimientos entendidos más pobremente.

- En caso que ustedes construyan requerimientos bien entendidos, el cliente podría responder con "sí, así es", y nada podría ser aprendido de la experiencia.
- Es un método menos formal de desarrollo.
- El prototipo es una técnica para comprender las especificaciones.
- Un prototipo puede ser eliminado.
- Un prototipo puede llegar a ser parte del producto final.
- Ventajas:
- Útiles cuando los requerimientos son cambiantes.
- Cuando no se conoce bien la aplicación.
- Cuando el usuario no se quiere comprometer con los requerimientos.
- Cuando se quiere probar una arquitectura o tecnología.
- Cuando se requiere rapidez en el desarrollo.
- Desventajas:
- No se conoce cuando se tendrá un producto aceptable.
- No se sabe cuántas iteraciones serán necesarias.
- Da una falsa ilusión al usuario sobre la velocidad del desarrollo.
- Se puede volver el producto aún y cuando no esté con los estándares.
-
- Figura Nº 2– Modelo del ciclo de vida Prototipo.
- 3.2.6.3.3 SPIRAL
- Toma las ventajas del modelo de desarrollo en cascada y el de prototipos añadiéndole el concepto de análisis de riesgo.
- Se definen cuatro actividades:
- Planificación: en la que se recolectan los requisitos iniciales o nuevos requisitos a añadir en esta iteración.
- Análisis de riesgo: basándonos en los requisitos decidimos si somos capaces o no de desarrollar el software y se toma la decisión de continuar o no continuar.
- Ingeniería: en el que se desarrolla un prototipo basado en los requisitos obtenidos en la fase de planificación.
- Evaluación del cliente: el cliente comenta el prototipo. Si está

conforme con él se acaba el proceso, si no se añaden los nuevos requisitos en la siguiente iteración.

-
- Figura Nº 3 – Modelo del ciclo de vida en Espiral.
- El esquema del ciclo de vida para estos casos puede representarse por un bucle en espiral, donde los cuadrantes son, habitualmente, fases de especificación, diseño, realización y evaluación (o conceptos y términos análogos).
- En cada fase el producto gana “madurez” (aproximación al final deseado) hasta que en una iteración se logre el objetivo deseado y este sea aprobado se termina las iteraciones.
- 3.2.6.3.4 INCREMENTAL
- Permite construir el proyecto en etapas incrementales en donde cada etapa agrega funcionalidad. Estas etapas, consisten en requerimientos, diseño, codificación, pruebas y entrega. Permite entregar al cliente un producto más rápido en comparación del modelo en cascada.
- Reduce los riegos ya que provee visibilidad sobre el progreso de las nuevas versiones.
- Provee retroalimentación a través de la funcionalidad mostrada.
- Permite atacar los mayores riesgos desde el inicio.
- Se pueden hacer implementaciones parciales si se cuenta con la suficiente funcionalidad.
- Las pruebas y la integración son constantes.
- El progreso se puede medir en periodo cortos de tiempo.
- Resulta más sencillo acomodar cambios al acortar el tamaño de los incrementos.
- Se puede planear en base a la funcionalidad que se quiere entregar primero.
- Por su versatilidad requiere de una planeación cuidadosa tanto a nivel administrativo como técnico.
- Ventajas:
- La solución se va mejorando en forma progresiva a través de las múltiples iteraciones, incrementa el entendimiento del problema y

de la solución por medio de los refinamientos sucesivos.

- Los clientes no esperan hasta el fin del desarrollo para utilizar el sistema. Pueden empezar a usarlo desde el primer incremento.
- Los clientes pueden aclarar los requisitos que no tengan claros, conforme ven las entregas del sistema.
- Se disminuye el riesgo de fracaso de todo el proyecto, ya que se puede distribuir en cada incremento.
- Las partes más importantes del sistema son entregadas primero, por lo cual se realizan más pruebas en estos módulos y se disminuye el riesgo de fallos.
- Desventaja:
- Requiere de mucha planeación, tanto administrativa como técnica
- Requiere de metas claras para conocer el estado del proyecto.
- Es un proceso de desarrollo de software, creado en respuesta a las debilidades del modelo tradicional de cascada.
- Para apoyar el desarrollo de proyectos por medio de este modelo se han creado Framework (entornos de trabajo), de los cuales los dos más famosos son el Rational Unified Process (RUP) y el Dynamic Systems Development Method.
- El desarrollo incremental e iterativo es también una parte esencial de un tipo de programación conocido como Extreme Programming y los demás frameworks de desarrollo rápido de software.
-
- Figura Nº 4 – Modelo del ciclo de vida Incremental.
- 3.2.6.3.5 RAD
- La metodología de desarrollo conocida como diseño rápido de aplicaciones RAD (rapid application development) ha tomado gran auge debido a la necesidad que tienen las instituciones de crear aplicaciones funcionales en un plazo de tiempo corto. RAD es un ciclo de desarrollo diseñado para crear aplicaciones de computadoras de alta calidad.
- El método comprende el desarrollo interactivo, la construcción de prototipos y el uso de utilidades CASE (Computer Aided Software Engineering). Tradicionalmente, el desarrollo rápido de

aplicaciones tiende a englobar también la usabilidad, utilidad y la rapidez de ejecución.

- Hoy en día se suele utilizar para referirnos al desarrollo rápido de interfaces gráficas de usuario tales como Glade, o entornos de desarrollo integrado completos. Algunas de las plataformas más conocidas son Visual Studio, Lazarus, Gambas, Delphi, Foxpro, Anjuta, Game Maker, Velneo o Clarion. En el área de la autoría multimedia, software como Neosoft Neoboo y MediaChance Multimedia Builder proveen plataformas de desarrollo rápido de aplicaciones, dentro de ciertos límites.
- Fases del RAD
- Modelado de gestión: el flujo de información entre las funciones de gestión se modela de forma que responda a las siguientes preguntas: ¿Qué información conduce el proceso de gestión? ¿Qué información se genera? ¿Quién la genera? ¿A dónde va la información? ¿Quién la procesó?
- Modelado de datos: el flujo de información definido como parte de la fase de modelado de gestión se refina como un conjunto de objetos de datos necesarios para apoyar la empresa. Se definen las características (llamadas atributos) de cada uno de los objetos y las relaciones entre estos objetos.
- Modelado de proceso: los objetos de datos definidos en la fase de modelado de datos quedan transformados para lograr el flujo de información necesario para implementar una función de gestión. Las descripciones del proceso se crean para añadir, modificar, suprimir, o recuperar un objeto de datos. Es la comunicación entre los objetos.
- Generación de aplicaciones: El DRA asume la utilización de técnicas de cuarta generación. En lugar de crear software con lenguajes de programación de tercera generación, el proceso DRA trabaja para volver a utilizar componentes de programas ya existentes (cuando es posible) o a crear componentes reutilizables (cuando sea necesario). En todos los casos se utilizan herramientas automáticas para facilitar la construcción del software.
- Pruebas de entrega: Como el proceso DRA enfatiza la reutilización,

ya se han comprobado muchos de los componentes de los programas. Esto reduce tiempo de pruebas. Sin embargo, se deben probar todos los componentes nuevos y se deben ejercitar todas las interfaces a fondo.

- Características de RAD:
- Equipos Híbridos:
- Equipos compuestos por alrededor de seis personas, incluyendo desarrolladores y usuarios de tiempo completo del sistema así como aquellas personas involucradas con los requisitos.
- Los desarrolladores de RAD deben ser "renacentistas": analistas, diseñadores y programadores en uno.
- Herramientas Especializadas:
- Desarrollo "visual"
- Creación de prototipos falsos (simulación pura)
- Creación de prototipos funcionales
- Múltiples lenguajes
- Calendario grupal
- Herramientas colaborativas y de trabajo en equipo
- Componentes reusables
- Interfaces estándares (API)
- Timeboxing:
- Las funciones secundarias son eliminadas como sea necesario para cumplir con el calendario.
- Prototipos iterativos y evolucionarios:
- Reunion JAD (Joint Application Development):
- Se reúnen los usuarios finales y los desarrolladores.
- Lluvia de ideas para obtener un borrador inicial de los requisitos.
- Iterar hasta acabar:
- Los desarrolladores construyen y depuran el prototipo basado en los requisitos actuales.
- Los diseñadores revisan el prototipo.
- Los clientes prueban el prototipo, depuran los requisitos.
- Los clientes y desarrolladores se reúnen para revisar juntos el producto, refinar los requisitos y generar solicitudes de cambios.

- Los cambios para los que no hay tiempo no se realizan. Los requisitos secundarios se eliminan si es necesario para cumplir el calendario.
- Ventajas:
- Comprar puede ahorrar dinero en comparación con construir.
- Los entregables pueden ser fácilmente trasladados a otra plataforma.
- El desarrollo se realiza a un nivel de abstracción mayor.
- Visibilidad temprana.
- Mayor flexibilidad.
- Menor codificación manual.
- Mayor involucramiento de los usuarios.
- Posiblemente menos fallas.
- Posiblemente menor costo.
- Ciclos de desarrollo más pequeños.
- Interfaz gráfica estándar.
- Desventajas:
- Comprar puede ser más caro que construir.
- Costo de herramientas integradas y equipo necesario.
- Progreso más difícil de medir.
- Menos eficiente.
- Menor precisión científica.
- Riesgo de revertirse a las prácticas sin control de antaño.
- Más fallas (por síndrome de “codificar a lo bestia”).
- Prototipos pueden no escalar, un problema mayúsculo.
- Funciones reducidas (por “timeboxing”).
- Dependencia en componentes de terceros: funcionalidad de más o de menos, problemas legales.
-
- Figura Nº 5 – Modelo del ciclo de vida RAD.
- 3.2.6.3.6 PROCESO UNIFICADO DE DESARROLLO SOFTWARE}
- Proceso Unificado de Desarrollo (RUP) es una metodología de desarrollo de software que está basado en componentes e interfaces bien definidas, y junto con el Lenguaje Unificado de Modelado (UML), constituye la metodología estándar más utilizada para el análisis,

implementación y documentación de sistemas orientados a objetos.

- Es un proceso que puede especializarse para una gran variedad de sistemas de software, en diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto.
- RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización.
- Es el resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo, a través del UML, y trabajo de muchas metodologías utilizadas por los clientes. La versión que se ha estandarizado vio la luz en 1998 y se conoció en sus inicios como Proceso Unificado de Rational 5.0, de ahí las siglas con las que se identifica a este proceso de desarrollo.
- Principales Elementos
- Como RUP es un proceso, en su modelación define como sus principales elementos:
- Trabajadores: define el comportamiento y responsabilidades (rol) de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo. Ellos realizan las actividades y son propietarios de elementos.
- Actividades: es una tarea que tiene un propósito claro, es realizada por un trabajador y manipula elementos.
- Artefactos: productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.
- Flujo de actividades: secuencia de actividades realizadas por trabajadores y que produce un resultado de valor observable.
- Características Principales de RUP
- Unifica los mejores elementos de metodologías anteriores.
- Preparado para desarrollar grandes y complejos proyectos.
- Orientado a Objetos.
- Utiliza el UML como lenguaje de representación visual.
- Principales ventajas:

- Coste del riesgo a un solo incremento.
- Reduce el riesgo de no sacar el producto en el calendario previsto.
- Acelera el ritmo de desarrollo.
- Se adapta mejor a las necesidades del cliente. Características del ciclo de vida de RUP.
- Dirigido por casos de uso
- Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso (cómo se llevan a cabo).
- Centrado en la arquitectura
- La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. RUP se desarrolla mediante iteraciones, comenzando por los casos de uso relevantes desde el punto de vista de la arquitectura. El modelo de arquitectura se representa a través de vistas en las que se incluyen los diagramas de UML.
- Iterativo e Incremental
- Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros.
- Por ejemplo, una iteración de elaboración centra su atención en el análisis y diseño, aunque refina los requerimientos y obtiene un producto con un determinado nivel, pero que irá creciendo incrementalmente en cada iteración.
- Es práctico dividir el trabajo en partes más pequeñas o miniproyectos. Cada miniproyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en los flujos de

trabajo, y los incrementos, al crecimiento del producto. Cada iteración se realiza de forma planificada es por eso que se dice que son miniproyectos.

- Flujo y fases de trabajo de RUP
- Cada fase representa un ciclo de desarrollo en la vida de un producto de software.
- La fase de concepción o inicio tiene por finalidad definir la visión, los objetivos y el alcance del proyecto, tanto desde el punto de vista funcional como del técnico, obteniéndose como uno de los principales resultados una lista de los casos de uso y una lista de los factores de riesgo del proyecto.
- El principal esfuerzo está radicado en el Modelamiento del Negocio y el Análisis de Requerimientos. Es la única fase que no necesariamente culmina con una versión ejecutable.
- La fase de elaboración tiene como principal finalidad completar el análisis de los casos de uso y definir la arquitectura del sistema, además se obtiene una aplicación ejecutable que responde a los casos de uso que la comprometen. A pesar de que se desarrolla a profundidad una parte del sistema, las decisiones sobre la arquitectura se hacen sobre la base de la comprensión del sistema completo y los requerimientos (funcionales y no funcionales) identificados de acuerdo al alcance definido. La fase de construcción está compuesta por un ciclo de varias iteraciones, en las cuales se van incorporando sucesivamente los casos de uso, de acuerdo a los factores de riesgo del proyecto.
- Este enfoque permite por ejemplo contar en forma temprana con versiones el sistema que satisfacen los principales casos de uso. Los cambios en los requerimientos no se incorporan hasta el inicio de la próxima iteración.
- La fase de transición se inicia con una versión de prueba (beta) del sistema y culmina con el sistema en fase de producción.
- En RUP se han agrupado las actividades en grupos lógicos definiéndose 9 flujos de trabajo principales, los 6 primeros son conocidos como flujos de ingeniería y los tres últimos como flujos de

apoyo.

- Modelo del Negocio: describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.
- Requerimiento: define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.
- Análisis y Diseño: describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requerimientos), por lo que indica con precisión lo que se debe programar.
- Implementación: define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.
- Prueba (Testing): busca los defectos a lo largo del ciclo de vida.
- Instalación: el sistema se pone en marcha en producción (se libera al cliente y usuario final) y se realizan actividades (empaque, instalación, asistencia a usuarios, etc.) para entregar el software a los usuarios finales.
- Administración del proyecto: involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.
- Administración de configuración y cambios: describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a utilización/actualización concurrente de elementos, control de versiones, etc.
- Ambiente: contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto; así como el procedimiento para implementar el proceso en una organización.
- Diferencias de RUP con las demás metodologías
- Algunos aspectos que diferencian a RUP de las demás metodologías y lo que lo hace único es que en RUP, los casos de uso no son sólo una herramienta para especificar los requisitos del sistema, sino que también guían su diseño, implementación y prueba. Los casos de

uso constituyen un elemento integrador y una guía del trabajo.

- Además de utilizar los casos de uso para guiar el proceso, se presta especial atención al establecimiento temprano de una buena arquitectura que no se vea fuertemente impactada ante cambios posteriores durante la construcción y el mantenimiento. También este propone que cada fase se desarrolle en iteraciones.
- 3.2.7 METODOLOGIAS AGILES
- 3.2.7.1 INTRODUCCION
- En las dos últimas décadas las notaciones de modelado y posteriormente las herramientas pretendieron ser la clave para el éxito en el desarrollo de software, no obstante, las expectativas no fueron satisfechas del todo. Esto se debe en gran parte a que otro importante elemento, la metodología de desarrollo, había sido postergado. De nada sirven buenas notaciones y herramientas si no se proveen directivas eficientes para su aplicación.
- Hasta hace poco el proceso de desarrollo llevaba asociada un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema tradicional para abordar el desarrollo de software ha demostrado ser efectivo y necesario en proyectos de gran tamaño donde por lo general se exige un alto grado de ceremonia en el proceso. Sin embargo, este enfoque no resulta ser el más adecuado y eficiente para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Ante las dificultades para utilizar metodologías tradicionales con estas restricciones de tiempo y flexibilidad, muchos equipos de desarrollo se resignan a prescindir del buen hacer de la ingeniería del software, asumiendo el riesgo que esto conlleva. Esto puede dar la impresión de que se van a hacer mal las cosas o de dejarlo a medias pero lo cierto es que ser “ágiles" no significa renunciar a formalismos ni dejar de ser estrictos, rigurosos y metódicos.
- En esta profesión y sobre todo en el área del desarrollo de software,

la teoría nos orienta a ser extremadamente estrictos al momento de hacer el análisis y documentación de nuestro sistema, pero en la práctica actual no se puede perder tanto tiempo realizando estas tareas porque cuando se finaliza el sistema, éste ya ha cambiado y el cliente se ha aburrido. Ser abducido y caer en esta nueva dinámica, es muy fácil, cuando la presión nos cae encima y los plazos de entrega se acercan y cada vez se acortan más.

- La alta competitividad actual hace que los sistemas de información se tengan que desarrollar de forma rápida para adaptarse a la organización. Las prisas hacen que lo primero que se deseche sea un análisis exhaustivo y se sustituye por uno superficial o simplemente se elimina. Éste es sin duda el gran error, ya que deseamos tener un sistema desarrollado rápidamente pero con lo que realmente nos encontramos es con un sistema lleno de errores, inmanejable y que no se puede mantener.
- Es difícil cambiar las reglas del mercado mundial, así que lo que se ha pensado es adaptar las metodologías de especificación y desarrollo a este entorno cambiante y lleno de presiones, en el que obtener un resultado rápido, algo que se pueda ver, mostrar y sobre todo utilizar, se ha vuelto crucial para el éxito de las organizaciones. La metodología necesariamente ha de ser ágil, debe tener un ciclo corto de desarrollo y debe incrementar las funcionalidades en cada iteración del mismo preservando las existentes, ayudando al negocio en lugar de darle la espalda.
- Es para este entorno para el que han nacido las metodologías ágiles y como consecuencia se creó el Manifiesto Ágil.
- El Manifiesto Ágil describe, básicamente, cuales son los principios sobre los que se basan los métodos reconocidos como ágiles.
- Éste manifiesto sugiere un enfoque orientado a la participación de los usuarios y clientes, más que hacia los procesos y herramientas, trabajando más en el software y menos en la documentación, colaborando más con los clientes en vez de estar negociando y respondiendo a los cambios sacrificando el plan de trabajo si es necesario.

- En el "Manifiesto ágil" se definen los cuatro valores principales por las que se deberían guiar las metodologías ágiles.
- Al individuo y sus interacciones más que al proceso y las herramientas.
- Desarrollar software que funciona más que obtener una documentación exhaustiva.
- La colaboración con el cliente más que la negociación de un contrato.
- Responder a los cambios más que seguir una planificación.
- El significado de cada uno de estos valores son los siguientes:
- 1. Al individuo y sus interacciones más que al proceso y las herramientas.
- Sin duda, la herramienta fundamental de la ingeniería del software y del desarrollo de aplicaciones es el cerebro humano y nuestra capacidad para desarrollar y desenvolvernos con nuestro entorno.
- Una persona sola no realiza un proyecto, necesita de un entorno en el que desarrollar su trabajo y de un equipo con el que colaborar. Estas interacciones deben también cuidarse. Un factor clave para el éxito es construir un buen equipo, que se lleven bien entre ellos, y que además sepan cómo construir su propio entorno de desarrollo. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte a él. Esto nos resta eficiencia, es mejor que el equipo se configure sobre la base de sus necesidades y sus características personales.
- Además, las interacciones que haga el equipo con el usuario final deberían ser igual de fluidas siendo este usuario un miembro más del equipo, con un objetivo común, que es conseguir que el proyecto funcione y sea útil para él.
- Si todo esto funciona bien, la elección de las herramientas y el proceso mismo de desarrollo pasan a estar en un plano totalmente secundario en la ecuación del éxito del proyecto.
- 2. Desarrollar software que funciona más que obtener una documentación exhaustiva.
- Uno de los objetivos de una buena documentación es poder ir a

consultarla cuando haya que modificar algo del sistema o surja alguna incertidumbre. Sin duda es un arma de doble filo, una buena documentación actualizada en cada modificación y bien mantenida al día permite saber el estado de la aplicación y cómo realizar las modificaciones pertinentes, pero son pocos los que con las presiones externas de tiempo y dinero acaban actualizando la documentación. Generalmente, cuando se tiene que arreglar un programa porque algo falla o surge un cambio de alcance, nos concentramos en que funcione ya que es muy posible que haya usuarios parados y que la empresa o la organización esté perdiendo dinero, en estos casos, ni nos paramos a mirar detenidamente la documentación ni, cuando se acaba de arreglar, nos ponemos a actualizarla.

- En la filosofía ágil, lo primordial es evitar estos fallos, centrar nuestro tiempo en asegurar que el software funciona y que se ha testeado exhaustivamente, e intentar reducir la creación de documentos poco útiles, de éstos que acaban no manteniéndose y ni siquiera consultándose. La regla no escrita es no producir documentos superfluos, y sólo producir aquellos que sean necesarios de forma inmediata para tomar una decisión importante durante el proceso de desarrollo. Estos documentos deben ser cortos y centrarse en lo fundamental, olvidándonos de las ambigüedades que no aportan nada a la comprensión del problema. No obstante, los documentos son soporte de documentación, permiten la transferencia del conocimiento, registran información histórica, y en muchas cuestiones legales o normativas son obligatorios, pero se resalta que son menos importantes que los productos que funcionan.
- 3. La colaboración con el cliente más que la negociación de un contrato.
- La consultoría informática de los últimos años se ha convertido en una lucha a muerte entre el proveedor del servicio y el cliente que lo contrata. Por una parte, el cliente intenta que se hagan el mayor número de funcionalidades con el mismo dinero, por otra parte, el consultor intenta que por ese dinero sólo se realicen las funcionalidades contratadas inicialmente. En las reuniones de

seguimiento de los proyectos es fácil oír frases del tipo "esta modificación no entra en el contrato" respondida generalmente por la tan típica "pues yo ya no tengo más presupuesto y así no podemos trabajar". Al final, este tipo de proyectos hacen que el consultor informático sea un híbrido entre analista y abogado, que desarrolla habilidades legales para salvaguardarse en caso de conflicto jurídico.

- Sin embargo, para que un proyecto tenga éxito es fundamental la complicidad y el contacto continuo entre el cliente y el equipo de desarrollo. El cliente debe ser y sentirse parte del equipo. De esta manera, ambos entenderán las dificultades del otro y trabajarán de forma conjunta para solucionarlo.
- 4. Responder a los cambios más que seguir una planificación.
- Una organización cambia constantemente, se adapta a las necesidades del mercado y reorganiza sus flujos de trabajo para ser más eficiente. Es difícil, pues, que en el desarrollo de un proyecto, éste no sufra ningún cambio, ya que es seguro que las necesidades de información de la empresa habrán cambiado. Y no sólo esto, sino que las posibilidades económicas de la misma pueden influir sobre nuestro proyecto, bien sea agrandándolo o reduciéndolo. Son muchos los factores que alterarán nuestra planificación inicial del proyecto.
- Si no la adaptamos a estos cambios, corremos el riesgo de que, cuando acabemos, nuestro sistema no cumpla con las necesidades pre-establecidas y el cliente se haya gastado el dinero en vano. La habilidad de responder a los cambios de requisitos, de tecnología, presupuestarios o de estrategia, marcan sin duda el camino del éxito del proyecto.
- Como consecuencia de estos cuatro valores, el Manifiesto ágil también enuncia los doce principios que caracterizan un proceso ágil diferenciándolo de otro tradicional donde este enfoque no se había aplicado lo suficiente, siempre se había dejado implícito pero sin hacer hincapié en ellos.
- La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.

- Dar la bienvenida a los cambios incluso al final del desarrollo. Los cambios le darán una ventaja competitiva a nuestro cliente.
- Hacer entregas frecuentes de software que funcione, desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
- Las personas del negocio y los desarrolladores deben trabajar juntos diariamente a lo largo de todo el proyecto.
- Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos.
- El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
- El software que funciona es la principal medida del progreso.
- Los procesos ágiles promueven un desarrollo sostenido. Los promotores, usuarios y desarrolladores deben poder mantener un ritmo de trabajo constante de forma indefinida.
- La atención continua a la calidad técnica y al buen diseño mejoran la agilidad.
- La simplicidad es esencial. Se ha de saber maximizar el trabajo que no se debe realizar.
- Las mejores arquitecturas, requisitos y diseños surgen de los equipos que se han organizado ellos mismos.
- En intervalos regulares, el equipo debe reflexionar con respecto a cómo llegar a ser más efectivo, y ajustar su comportamiento para conseguirlo.
- Llegados a este punto, podemos intuir que las formas de hacer las cosas en la ingeniería del software están cambiando, adaptándose más a las personas y a las organizaciones en las que han de funcionar las aplicaciones.
- Las metodologías ágiles no son la gran solución a todos los problemas del desarrollo de aplicaciones, ni tan siquiera se pueden aplicar en todos los casos, pero sí que nos aportan otro punto de vista de cómo se pueden llegar a hacer las cosas, de forma más rápida, más adaptable y sin tener que perder la rigurosidad de las metodologías clásicas

2.3 DEFINICIONES CONCEPTUALES.

Las tecnologías de la información y de las comunicaciones (TIC) : Es un término que se utiliza actualmente para hacer referencia a una gama amplia de servicios, aplicaciones, y tecnologías, que utilizan diversos tipos de equipos y de programas informáticos, y que a menudo se transmiten a través de las redes de telecomunicaciones.

Las TIC se convierten en un gran soporte para la comunicación fluida y la coordinación permanente al interior de la estructura de la organización, así como entre las empresas que conforman la cadena de valor del bien o servicio ofrecido.

La clasificación de las tecnologías de información y comunicaciones se presenta de la siguiente forma:

- **Hardware:** Componentes físicos asociados a las TIC tales como servidores, PC's, impresoras, etc.
- **Software:** Programas electrónicos que complementan el uso del hardware tales como bases de datos, lenguajes de programación, etc.
- **Telecomunicaciones:** Infraestructura necesaria para el envío y recepción de información electrónica (voz, datos e imagen).
- **Estrategia tecnológica del negocio**: Conjunto de acciones encaminadas a la adquisición e implantación de tecnologías que aseguren el logro de ventajas competitivas.

CAPITULO III

ASPECTOS METODOLÓGICOS

3.1 TIPO DE INVESTIGACIÓN.

Aplicada: Debido a que se aplicará el conocimiento para la investigación.

Enfoque: cuantitativo

Gómez (2006:121) señala que bajo la perspectiva cuantitativa, la recolección de datos es equivalente a medir.

De acuerdo con la definición clásica del término, medir significa asignar números a objetos y eventos de acuerdo a ciertas reglas. Muchas veces el concepto se hace observable a través de referentes empíricos asociados a él. Por ejemplo si deseamos medir la violencia (concepto) en cierto grupo de individuos, deberíamos observar agresiones verbales y/o físicas, como gritos, insultos, empujones, golpes de puño, etc. (los referentes empíricos).

Nivel: descriptivo explicativo

Describe situaciones y eventos. Esto es, decir cómo es y se manifiesta determinado fenómeno. Los estudios descriptivos buscan especificar las propiedades importantes de personas, grupos, comunidades o cualquier otro fenómeno que sea sometido a análisis Miden o evalúan diversos aspectos, dimensiones o componentes del fenómeno o fenómenos a investigar. Desde el punto de vista científico, describir es medir.

3.2 DISEÑO Y ESQUEMA DE LA INVESTIGACIÓN.

No Experimental: El presente diseño es no experimental debido a que se realiza sin manipular deliberadamente variables. Se basa fundamentalmente en la observación de fenómenos tal y como se dan en su contexto natural para analizarlos con posterioridad. En este tipo

de investigación no hay condiciones ni estímulos a los cuales se expongan los sujetos del estudio. Los sujetos son observados en su ambiente natural.

3.3 POBLACIÓN Y MUESTRA.

Población:

Está conformado por todos los desarrolladores de software de las MYPES que gestionan el proceso de desarrollo.

Muestra:

Se realizará a través del muestreo de conveniencia o por selección intencionada, por todos los desarrolladores de software de las MYPES que gestionan el proceso de desarrollo.

Dónde: **n**= 36

3.4 DEFINICIÓN OPERATIVA DEL INSTRUMENTOS DE RECOLECCIÓN DE DATOS.

TECNICAS	INSTRUMENTO
Análisis Bibliográfico	Bibliografía y material impresos de interés
Recopilación de Información	Observación, cuestionario, Documentación, Indagación.

3.5 TÉCNICAS DE RECOJO, PROCESAMIENTO Y PRESENTACIÓN DE DATOS.

- Ordenamiento y clasificación.
- Registro y procesamiento computarizado con Excel.

CAPITULO IV

ASPECTOS ADMINISTRATIVOS

4.1 CRONOGRAMA DE ACTIVIDADES

AÑO	2017									
ACTIVIDADES/MES	JN	JN	JL	JL	AG	AG	ST	ST	OC	OC
1. Determinación del problema y elaboración del proyecto de tesis	■									
2. Presentación del Proyecto de Tesis	■									
3. Designación del Asesor de Proyecto de tesis con Acto Resolutivo		■								
4.Designacion de Jurado Revisor de Proyecto de Tesis			■							
5.Revision de Proyecto de Tesis por el Jurado Revisor			■							
6.Levantamiento de Observaciones al Proyecto de Tesis				■						
7.Aprobacion del Proyecto de Tesis con Acto Resolutivo				■						
8.Elaboracion de Instrumentos de Investigación					■					
9. Validación del instrumento de investigación					■					
10.Aplicacion y Recolección de datos						■				
11. Sistematización de la información						■				
12.Presentacion del Borrador de Tesis							■			
13.Designación del Docente Asesor de Borrador de Tesis								■		
14.Designacion de Jurado Examinador del Borrador de Tesis								■		
15.Revision de Borrador de Tesis por el Jurado Examinador								■		
16.Levantamiento de Observaciones al borrador de Tesis									■	
17. Aprobación al borrador de tesis y autorización para defensa de la misma mediante acto resolutivo.									■	
18. Sustentación de Tesis										■

CAPITULO V

PRESUPUESTO

5.1. POTENCIAL HUMANO

RECURSOS HUMANOS	CANT	(S/.)
Asesor	1	1500.00
Asesor asignado por la universidad	1	350.00
Consultores	2	450.00
Revisor Especialista	1	200.00
SUB TOTAL		**2500.00**

5.2. RECURSOS MATERIALES

A. BIENES

GASTOS	(S/.)
Útiles de escritorio	300.00
Bibliografía especializada	400.00
Otros	300.00
SUB TOTAL	**1000.00**

B. SERVICIOS

GASTOS	(S/.)
Movilidad y Viáticos	300.00
Tipeo e impresión	100.00
Internet	100.00
Encuadernados, Espiralados	100.00
Honorarios Profesionales	500.00
Otros	100.00
SUB TOTAL	**1200.00**

5.3. RECURSOS FINANCIEROS

El presupuesto asignado a esta investigación será autofinanciado íntegramente por el investigador en los siguientes rubros:

RUBRO	(S/.)
RECURSOS HUMANOS	2 500.00
RECURSO MATERIAL Y SERVICIO	2 200.00
TOTAL GENERAL	**4 700.00**

UNIVERSIDAD NACIONAL
HUANUCO
HERMILIO VALDIZÁN
LEY 14915 1964

BIBLIOGRAFÍA

- ABRIL, David. Parte I Metodología de Desarrollo de Software. Especialización Tecnológica en Desarrollo de Aplicaciones para Dispositivos Móviles 2011. Análisis Documental. Pag. 3

- ARBOLEYA, Hernán. Propuesta de Ciclo de Vida y Mapa de Actividades para Proyectos de Explotación de Información. Universidad Nacional de Lanus (UNLA), Remedios de Escalada (Argentina) 2013. Trabajo de Grado (Licenciado en Sistemas). Pag. 9-11.

- CARDINALES, Cristina. Cuadro Comparativo de las Metodologías. República Bolivariana de Venezuela 2012. Documentación. Ministerio del Poder para la Educación Universitaria, Fundación Misión Sucre. Pag 3.

- CELIS Zaida. Facultad de filosofía y letras, Universidad Nacional Autónoma de México, XI Congreso Nacional de Investigación Educativa, Mexico DF.

- GALICIA Bic. Centro Europeo de Empresas e Innovación. Documentación. Madrid 2008.

- GIL, Gustavo. Herramienta para implementar LEL y escenarios (TILS), ingeniería de requerimientos. Universidad Nacional de la Plata, Argentina 2002.

- Instituto Nacional de Tecnologías de la comunicación. Gobierno de España. Ingeniería del Software: Metodologías y Ciclos de Vida. Madrid (España) 2009. Documentación (Laboratorio Nacional de Calidad del Software). Pag. 46-49

- MEZA, Daniel. Fundamentos de Desarrollo de Sistemas. Instituto Tecnológico de Comitán. Chiapas (México) 2010. Documentación (Ingeniería en Sistemas Computacionales). Pag. 21-24.

- PEDREIRA, Oscar. Revista española de innovación-Calidad del software. España. 2007

- PRESSMAN , Roger S. Ingeniería del Software Un Enfoque Práctico. McGraw- Hill. México 2005.

- RODRIGUEZ Rosangélica, GONZÁLEZ Henyeliz, ROSAS Victor. Metodología de Desarrollo Rápido de Aplicación. Universidad Nacional Experimental Politécnica de la Fuerza Armada Bolivariana. Núcleo Sucre-Carúpano (Venezuela) 2013. Documentación (Ingeniería de Sistemas).

- RUIZ, Yadira. Principales metodologías pesadas y orientadas a objetos en el desarrollo de software. 14 convención científica de ingeniería y arquitectura, México 2008. Investigación documental.