ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN SOFTWARE

PROYECTO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SOFTWARE

AUTORES: LUIS MIGUEL EGAS ROBALINO

JUAN XAVIER JÁTIVA ÁLVAREZ

TEMA: IMPLEMENTACIÓN DE UNA METODOLOGÍA DE DESARROLLO ÁGIL PARA EL ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DEL SISTEMA SOFTWARE Y APLICACIÓN DE UN CASO DE ESTUDIO PRÁCTICO, EN EL DEPARTAMENTO DE TECNOLOGÍAS DE INFORMACIÓN Y COMUNICACIONES DE LA ESCUELA TÉCNICA DE LA FUERZA AÉREA DE LA CIUDAD DE LATACUNGA, DURANTE EL PERÍODO ABRIL – DICIEMBRE DEL 2013

DIRECTOR: ING. GARCÉS, LUCAS

CODIRECTOR: ING. GUERRA, LUIS

LATACUNGA, JULIO DEL 2014

UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE
CARRERA DE INGENIERÍA EN SOFTWARE

CERTIFICADO

ING. LUCAS ROGERIO GARCÉS GUAYTA (DIRECTOR)
ING. LUIS ALBERTO GUERRA (CODIRECTOR)

CERTIFICAN:

Que el trabajo titulado "IMPLEMENTACIÓN DE UNA METODOLOGÍA DE DESARROLLO ÁGIL PARA EL ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DEL SISTEMA SOFTWARE Y APLICACIÓN DE UN CASO DE ESTUDIO PRÁCTICO, EN EL DEPARTAMENTO DE TECNOLOGÍAS DE INFORMACIÓN Y COMUNICACIONES DE LA ESCUELA TÉCNICA DE LA FUERZA AÉREA DE LA CIUDAD DE LATACUNGA, DURANTE EL PERÍODO ABRIL – DICIEMBRE DEL 2013" realizado por los señores**:** Luis Miguel Egas Robalino y Juan Xavier Játiva Álvarez, ha sido guiado y revisado periódicamente y cumple normas estatutarias establecidas por la ESPE, en el Reglamento de Estudiantes de la Universidad de Fuerzas Armadas-ESPE.

Debido a que constituye un trabajo de excelente contenido científico que coadyuvará a la aplicación de conocimientos y al desarrollo profesional, SI recomiendan su publicación.

El mencionado trabajo consta de un empastado y un disco compacto el cual contiene los archivos en formato portátil de Acrobat (pdf). Autorizan a los señores**:** Luis Miguel Egas Robalino y Juan Xavier Játiva Álvarez que lo entregue al Ing. Luis Guerra, en su calidad de Director de Carrera.

Latacunga, Julio de 2014.

----------	----------
ING. LUCAS ROGERIO GARCÉS DIRECTOR	ING. LUIS ALBERTO GUERRA CODIRECTOR

UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE
CARRERA DE INGENIERIA EN SOFTWARE

DECLARACIÓN DE RESPONSABILIDAD

NOSOTROS, LUIS MIGUEL EGAS ROBALINO Y JUAN XAVIER JÁTIVA ÁLVAREZ

DECLARAMOS QUE:

El proyecto de grado denominado "IMPLEMENTACIÓN DE UNA METODOLOGÍA DE DESARROLLO ÁGIL PARA EL ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DEL SISTEMA SOFTWARE Y APLICACIÓN DE UN CASO DE ESTUDIO PRÁCTICO, EN EL DEPARTAMENTO DE TECNOLOGÍAS DE INFORMACIÓN Y COMUNICACIONES DE LA ESCUELA TÉCNICA DE LA FUERZA AÉREA DE LA CIUDAD DE LATACUNGA, DURANTE EL PERÍODO ABRIL – DICIEMBRE DEL 2013" ha sido desarrollado con base a una investigación exhaustiva, respetando derechos intelectuales de terceros, conforme las citas que constan al pie de las páginas correspondientes, cuyas fuentes se incorporan en la bibliografía.

Consecuentemente este trabajo es de nuestra autoría.

En virtud de esta declaración, nos responsabilizamos del contenido, veracidad y alcance científico del proyecto de grado en mención.

Latacunga, Julio de 2014.

Luis Miguel Egas Robalino	Juan Xavier Játiva Álvarez
CI: 1716941412	CI: 1717538274

UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE
CARRERA DE INGENIERÍA EN SOFTWARE

AUTORIZACIÓN

NOSOTROS, LUIS MIGUEL EGAS ROBALINO Y JUAN XAVIER JÁTIVA ÁLVAREZ

Autorizamos a la Universidad de las Fuerzas Armadas - ESPE la publicación, en la biblioteca virtual de la Institución del trabajo "IMPLEMENTACIÓN DE UNA METODOLOGÍA DE DESARROLLO ÁGIL PARA EL ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DEL SISTEMA SOFTWARE Y APLICACIÓN DE UN CASO DE ESTUDIO PRÁCTICO, EN EL DEPARTAMENTO DE TECNOLOGÍAS DE INFORMACIÓN Y COMUNICACIONES DE LA ESCUELA TÉCNICA DE LA FUERZA AÉREA DE LA CIUDAD DE LATACUNGA, DURANTE EL PERÍODO ABRIL – DICIEMBRE DEL 2013" cuyo contenido, ideas y criterios son de nuestra exclusiva responsabilidad y autoría.

Latacunga, Julio de 2014.

Luis Miguel Egas Robalino	Juan Xavier Játiva Álvarez
CI: 1716941412	CI: 1717538274

DEDICATORIA

El presente proyecto de investigación está dedicado especialmente a mi esposa Jasmina Chambers y a mi hijo Luis Damián Egas, quienes me brindaron su amor, cariño, comprensión y ayuda constante durante todo el proceso hasta la culminación de mencionado proyecto.

A mis amados padres, Luis Egas y Jacqueline Robalino, fuente de inspiración, superación y pilares fundamentales en mi vida, quienes con su amor y bondad han estado incondicionalmente presentes, desde mis primeros pasos hasta la consecución de mis objetivos personales y profesionales alcanzados hoy en día. De igual manera, a mis Hermanos Estefanía, Andrés, Jésica y mi sobrino Leonardo Egas, quienes con su fortaleza e inteligencia, me han servido de ejemplo, admiración y constancia en el trabajo.

Finalmente, a Dios por darme el intelecto, la sabiduría y la luz necesaria para tomar las mejores decisiones en mi vida y afrontar las eventualidades del día a día.

LUIS MIGUEL EGAS ROBALINO

Dedico la realización de este proyecto a mi esposa Diana Fuel por todo su apoyo incondicional y amor brindado en el día a día para la consecución de este proyecto de investigación.

A mi madre Elizabeth Álvarez por ser una madre ejemplar, con su paciencia y amor inculcarme valores para ser un hombre de bien.

JUAN XAVIER JÁTIVA ÁLVAREZ

AGRADECIMIENTO

Uno de los valores más nobles de la expresión del ser humano y en especial del soldado, es el agradecimiento, como virtud y sentimiento que se arraiga en lo más profundo del corazón y que permite exaltarlo con cada tarea que se realiza con buena predisposición y honestidad, permitiendo que estos actos jamás se olviden y perduren en la eternidad.

En tal sentido, a través del presente proyecto de investigación, presentamos nuestra más sincera gratitud a la noble institución armada, la gloriosa Fuerza Aérea Ecuatoriana mediante el Comando de Educación y Doctrina, y al alma máter del profesionalismo en el país, la Universidad de Fuerzas Armadas ESPE Extensión Latacunga; instituciones que nos han abierto la puertas para nuestra preparación académica-militar, y que con plena confianza, han depositado en nuestras manos la excelencia del profesionalismo, para plasmarla en cada rincón de la Patria donde estemos entregando nuestro contingente.

Es necesario también el reconocimiento y agradecimiento a los señores Ing. Lucas Garcés e Ing. Luis Guerra, Director y Codirector de tesis respectivamente, quienes con tesón, sabiduría y comprensión, permitieron orientar el desarrollo de mencionado proyecto hasta su culminación con éxito.

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

El presente proyecto de investigación desarrolló la propuesta de implementación de una Metodología de Desarrollo Ágil de Sistemas Software para ser utilizada en el Departamento de Tecnologías de la Información y Comunicaciones de la Escuela Técnica de la Fuerza Aérea, ubicado en la ciudad de Latacunga. El proyecto surgió de la necesidad de optimizar los procesos de desarrollo software, generado por la carencia de estándares, procedimientos, técnicas, métodos o un marco de trabajo coherente y estructurado a seguir, causando graves inconvenientes en el manejo de las fases del ciclo de vida del software por parte del personal que labora en el mismo. En este sentido, la investigación inicia con un estudio de la evolución de las Metodologías de Desarrollo en el Proceso de Ingeniería de Sistemas Software, el cual permite dar a conocer la cronología desde el nacimiento de las primeras técnicas hasta el establecimiento de las Metodologías de Desarrollo Tradicionales y Ágiles como instrumentos válidos para el desarrollo software. Seguidamente se realiza un estudio de selección y comparación de las metodologías de desarrollo software a fin de identificar la o las metodologías que mejor se adapten al contexto de trabajo del Departamento y cumplir así con nuestro propósito. Finalmente, se generará e implementará la Metodología propuesta y se verificará la validez del mismo a través de la aplicación de un caso de estudio práctico.

Palabras Claves: Metodologías de Desarrollo Tradicionales y Ágiles, Sistemas Software, Procesos de Desarrollo Software, Fases del Ciclo de Vida del Software, Ingeniería de Sistemas Software, Caso de Estudio Práctico.

ABSTRACT

This research project developed the proposal for implementation of a methodology Agile Software Development Systems for use in the Department of Information Technology and Communications School of the Air Force, located in the city of Latacunga. The project arose from the need to optimize the software development processes, generated by lack of standards, procedures, techniques, methods or consistent framework and structured to follow, causing serious problems in the management of the phases of the life cycle Software by staff who work in it. In this sense, research begins with a study of the evolution of Development Methodologies in Process Systems Engineering Software, which allows to present the chronology since the birth of the first techniques to the establishment of the Development Methodologies Traditional and Agile as valid tools for software development. Following is a selection study and comparison of software development methodologies to identify or methodologies that best fit the context of the Department's work and fulfill our purpose is done. Finally, build and deploy the proposed methodology and its validity is verified through the application of a practical case study.

Keywords: Traditional methodologies and Agile Development, Software Systems, Software Development Process, Phases of the Software Life Cycle, Software Engineering Systems, Practical Case Study.

CAPÍTULO 1
DEFINICIÓN DEL PROBLEMA

1.1 Introducción del Capítulo

En el presente capítulo se describe parte de la investigación, donde se da a conocer los problemas que ha generado, el hecho de no utilizar metodologías para el desarrollo de un sistema software y cómo la oportuna aplicación de estas en el proceso de ingeniería del software nos sirve para efectivizar el propósito deseado, mediante la aplicación de un caso de estudio práctico, como es la satisfacción del cliente.

También se presenta el motivo por el cual el Departamento de Tecnologías de Información y Comunicaciones (TIC) debe involucrarse en la implementación de una metodología de desarrollo ágil, con el fin de optimizar el análisis, diseño e implementación del sistema software, apoyado con la aplicación de un caso de estudio práctico, que servirá de base para el posterior desarrollo de sistemas software en la Escuela Técnica de la Fuerza Aérea (ETFA).

1.2 Planteamiento del Problema

La Escuela Técnica de la Fuerza Aérea de la ciudad de Latacunga, posee el Departamento de TIC en donde se desarrolla Software, adaptado a las necesidades de cada departamento. Para gestionar el desarrollo correcto de estos productos software no utilizan metodologías y métodos durante el proceso, causando graves inconvenientes en las fases del desarrollo del sistema software.

En la ETFA una de las mayores deficiencias en la práctica de construcción de software es la poca atención que se presta al seguimiento de parámetros y prácticas específicas durante todas las etapas de desarrollo de software, causando así una de las problemáticas actuales de la Institución, que es consecuencia de la falta de aplicación de herramientas, normas, estándares y metodologías disponibles que proporcionen un soporte tecnológico, conceptual y humano en el Departamento de las TIC, ocasionando la adquisición de sistemas a otras empresas por la deficiente calidad de software diseñado y tiempos tardíos de entrega.

El Departamento de TIC desarrolla sistemas software para las diferentes dependencias y necesidades de la ETFA, los cuales se han visto afectados en su

desarrollo por la falencia de estándares, metodologías, métricas, procedimientos o prácticas adecuadas para la generación de proyectos de software causando económicamente un problema de operatividad de todo el departamento. La falta de cumplimiento de estos procesos dificulta el avance normal de las tareas comunes dentro del equipo de trabajo del departamento causando un deterioro de la imagen competitiva ante la ETFA.

Al momento del desarrollo del sistema software las prácticas que se utilizan en el proceso de construcción del mismo, presentan reajustes que se deben realizar para una mejor estimación de recursos en tiempo, herramientas prácticas, talento humano. Además, las falencias sobre la aplicación de estrategias que se disponen en la metodología a seguir, provoca la falta de comunicación entre los Departamentos de la ETFA y el Departamento de TIC.

La ausencia de coordinación entre los desarrolladores y el Jefe de Departamento, junto al pobre levantamiento de requisitos iniciales y la carencia de documentación formal que sirva de referencia para la Institución genera el retraso del desarrollo normal del producto software, la sobrecarga de trabajo, la baja calidad con la que se desarrolla y la entrega del producto software limita sus competencias de manera muy grave.

El Departamento de TIC al carecer de una metodología apropiada y la falta de modelos para planificar su alcance al momento del desarrollo de software de calidad, puede conllevar al prematuro declive de la entrega o la desorganización interna que afecta principalmente al núcleo central de la Escuela.

Basándonos en estos inconvenientes se formula el siguiente problema:

¿Cómo optimizar el análisis, desarrollo e implementación del sistema software en el Departamento de TIC de la Escuela Técnica de la Fuerza Aérea de la ciudad de Latacunga, durante el período abril – diciembre del 2013?

1.3 Antecedentes

El Departamento de Tecnologías de Información y Comunicaciones, es el encargado del desarrollo de sistemas software ajustado a las necesidades y requerimientos de la ETFA.

El personal con el que cuenta el Departamento de TIC, se encuentra incursionando en proyectos enmarcados en el desarrollo de software a medida, por lo que surge la

necesidad de establecer un marco metodológico que defina los pasos a seguir en futuros desarrollos y que le permita gestionar de manera óptima procesos basados en metodologías de desarrollo ágil.

La tendencia gubernamental es fortalecer la automatización de procesos, para concordar con las áreas administrativas en una verdadera cultura de procesos que conlleve una optimización de los recursos del Estado, razón por la cual la mayoría de entidades públicas se encuentran incursionando en el desarrollo de sistemas software basadas en metodologías de desarrollo ágil.

El propósito de implementar una metodología ágil es optimizar las etapas de análisis, diseño e implementación del sistema software en el Departamento de TIC, bajo un verdadero concepto de interoperabilidad que permita establecer los lineamientos para el desarrollo del software mediante herramientas, normas, estándares y metodologías disponibles que permitan obtener un producto software definido y conceptualizado, no solo entre soluciones legadas sino interactuadas con sistemas software externos, permitiendo disponer de información oportuna y en tiempo real.

El desarrollo de software no es una tarea fácil. Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo software. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. Otra aproximación es centrarse en otras dimensiones, como por ejemplo el factor humano o el producto software. Esta es la filosofía de las metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas.

Las metodologías ágiles están revolucionando la manera de producir software, por lo que se espera solucionar esta problemática a través del estudio de metodologías de desarrollo ágil para el análisis, diseño e implementación del sistema software, la aplicación de un caso práctico, la verificación de resultados que se han de obtener del proceso, y posterior la implementación de dicha metodología en el Departamento TIC de la ETFA.

1.4 Objetivo General

- Implementar una metodología de desarrollo ágil que optimice el análisis, diseño e implantación de sistemas software, con la aplicación del caso de

estudio práctico “Sistema Inteligente para el Control Disciplinario de los Aspirantes a Tropa de la ETFA”, en el Departamento de Tecnologías de la Información y Comunicaciones de la Escuela Técnica de la Fuerza Aérea.

1.5 Objetivos Específicos

- Determinar el marco teórico vinculado al proceso de análisis, diseño e implantación de sistemas software basados en una metodología de desarrollo ágil.
- Desarrollar la propuesta del modelo ágil en el desarrollo, análisis e implantación de sistemas software.
- Aplicar la propuesta en el Departamento de TIC de la ETFA incorporando una metodología ágil de desarrollo de software, a través del caso de estudio práctico “Sistema Inteligente para el Control Disciplinario de los Aspirantes a Tropa de la ETFA”.
- Validar los resultados obtenidos de la aplicación de la metodología de desarrollo ágil con el caso de estudio práctico.

1.6 Justificación

En el Departamento de TIC de la ETFA se desarrolla sistemas software ajustado a las necesidades institucionales y de cada uno de los departamentos de la misma, en el cual se ha detectado falencias para gestionar el desarrollo del producto software en relación a la utilización correcta de metodologías y métodos durante el proceso de construcción del software; razón por la cual se considera importante la necesidad de solventar los inconvenientes en las fases de desarrollo del sistema software con la aplicación de una metodología de desarrollo ágil.

En el Departamento de TIC, una de las mayores deficiencias en la práctica de construcción software es la poca atención que se da al seguimiento de parámetros y prácticas específicas durante todas las etapas de desarrollo software; razón por la cual es importante el uso de herramientas, normas, estándares y metodologías que proporcionen un soporte tecnológico, conceptual y humano coherente en el Departamento de TIC, y que además eviten la adquisición del sistema software a otras empresas y que optimicen la calidad del producto software diseñado a medida y que se ajuste a los tiempos de entrega.

Es importante técnicamente porque el desarrollo del sistema software mediante una metodología ágil, desde el Departamento de TIC de la ETFA para los distintos departamentos, facilita el proceso de desarrollo software de manera organizada y estructurada, obligando a generar una adecuada implementación que ayude a mejorar la gestión de la calidad del software.

Económicamente el proyecto es importante porque ayuda a minimizar los gastos que está incurriendo la ETFA en la adquisición de Software a otras empresas, además que la falencia en la aplicación de metodologías, estándares, métricas, procedimientos o prácticas para la generación del producto software han ocasionado económicamente un problema de operatividad, no sólo del Departamento TIC, sino de los demás departamentos que hacen uso del sistema software, acompañado de un deterioro de la imagen competitiva de la ETFA.

Socialmente es importante debido a que el proyecto incentivará a que todos los miembros del equipo de desarrollo, del Departamento de TIC, trabajen armónicamente en todas las fases de desarrollo y se evite producir demasiados tiempos de holgura, una mala administración de los recursos, demoras en la entrega del producto software y la desorganización en el proceso de ingeniería; además que alinea los objetivos del Departamento con los objetivos de la Dirección.

1.7 Hipótesis

Si se implementa una metodología de desarrollo ágil y se aplica a un caso de estudio práctico "Sistema Inteligente para el Control Disciplinario de los Aspirantes a Tropa", entonces se optimiza el análisis, diseño e implementación del sistema software, en el Departamento de Tecnologías de Información y Comunicaciones de la Escuela Técnica de la Fuerza Aérea de la ciudad de Latacunga, durante el período abril – diciembre del 2013.

1.8 Variables de la Investigación

- Variable Independiente: Se implementa una metodología de desarrollo ágil y se aplica un caso de estudio práctico "Sistema Inteligente para el Control Disciplinario de los Aspirantes a Tropa".
- Variable Dependiente: Se optimiza el análisis, diseño e implementación del sistema software, en el Departamento de Tecnologías de Información y

Comunicaciones de la Escuela Técnica de la Fuerza Aérea de la ciudad de Latacunga, durante el período abril – diciembre del 2013.

- Indicadores:
  - Entrega de productos del software a tiempo.
  - Tiempos de desarrollo de software ajustados al plan.
  - Esfuerzo del personal de acuerdo a la planificación.
  - Cumplimiento de los objetivos del Departamento de TIC del área de desarrollo software de la ETFA.
  - Cumplimiento de la planificación del proyecto software.
  - Optimización de la gestión de la calidad del software.
  - Ciclo de vida del sistema software organizado y estructurado.
  - Satisfacción del equipo de trabajo

1.9 Conclusión del Capítulo

El Departamento de Tecnologías de Información y Comunicaciones de la Escuela Técnica de la Fuerza Aérea, es el encargado del desarrollo de sistemas software ajustado a las necesidades y requerimientos de la Institución, que al momento se encuentra incursionando en proyectos enmarcados en el desarrollo de software a medida buscando la automatización de los procesos. Sin embargo, existen falencias al gestionar el desarrollo correcto de los productos software que se generan, debido a la carencia en el uso de metodologías o métodos durante el proceso, causando graves inconvenientes en las fases del desarrollo del sistema software.

En este sentido, se ha decido dar una solución por medio de la presente propuesta de investigación, al implementar una metodología que se adapte a los procesos de la ETFA y cuyo fin es la optimización del análisis, diseño e implementación del sistema software que será verificable mediante la aplicación de un caso de estudio práctico.

Para la implementación del presente proyecto se tomara como base fundamental normas, estándares y procedimientos existentes dentro de las Metodologías de Desarrollo Ágil del Software que han sido utilizados por entidades internacionales y se han probado en varios proyectos con muy buenos resultados, lo cual nos respalda para la generación de la propuesta.

CAPÍTULO 2
MARCO TEÓRICO

2.1 Introducción del Capítulo

Desde inicios de 1940, escribir software ha evolucionado hasta convertirse en una profesión que se ocupa de cómo crear software y maximizar su calidad. La calidad puede referirse a cuán mantenible es el software, su estabilidad, velocidad, usabilidad, comprobabilidad, legibilidad, tamaño, costo, seguridad y número de fallas o "bugs", así como, entre muchos otros atributos, a cualidades menos medibles como elegancia, concisión y satisfacción del cliente. La mejor manera de crear software de alta calidad es un problema separado y controvertido cubriendo el diseño de software, principios para escribir código, llamados "mejores prácticas", así como cuestiones más amplias de gestión como tamaño óptimo del equipo de trabajo, el proceso, la mejor manera de entregar el software a tiempo y tan rápidamente como sea posible, la "cultura" del lugar de trabajo, prácticas de contratación y así sucesivamente.

Pero para poder hablar de calidad, se tuvo que generar un gran proceso histórico, que hasta hoy en día sigue en constante evolución, es el caso de las Metodologías de Desarrollo de Software. Estas proponen como objetivo principal presentar un conjunto de técnicas tradicionales, modernas y ágiles de modelado de sistemas que permitirían desarrollar software con calidad, incluyendo heurísticas de construcción y criterios de comparación de modelos de sistemas.

En el presente escenario se encuentra el Departamento TIC, cuyas actividades van orientadas al cumplimiento de la misión de la ETFA, siendo una de las más importantes y a su vez vulnerables, el desarrollo del software; cuyo departamento parte con la idea de generar inicialmente mediante el análisis, diseño e implementación sistemas o soluciones informáticas a medida. Tomando la visión futurista de la Fuerza Aérea, el Departamento TIC se encuentra incursionando en el desarrollo de sistemas software de calidad en base a estándares, normas y métodos aplicables en todo el mundo, por lo que la presente investigación tratará de dar una solución a los problemas que atañe al mismo por la falta o carencia de un marco de trabajo a seguir.

2.2 Antecedentes Históricos

Evolución de las Metodologías de Desarrollo de la Ingeniería de Software en el Proceso la Ingeniería de Sistemas Software.

Desde el principio del uso de los ordenadores, al trabajar sobre el desarrollo de los primeros programas, se siguieron una serie de pautas o métodos para llevar a buen fin el proyecto. Bien es verdad que en esta situación la metodología era simple, era el típico proceso de abajo a arriba, con análisis insuficientes, ya que el problema era comprendido fácilmente en su totalidad. Además sólo preocupaba el proceso ya que los datos no solían ser importantes en sí y lo que realmente importaba era resolver algún tipo de problema físico cuya formulación matemática, y sobre todo su resolución, era complicada. Por lo tanto en el principio de la informática los métodos eran de tipo ascendente y orientado a procesos.

Al introducirse el ordenador en las empresas para la gestión de sus necesidades se partió de la mentalidad antes citada y se trató de resolver aquellos problemas cuya complejidad, en cuanto a procedimiento de resolución, era grande.

A finales de la década de los sesenta empezaron a aparecer ordenadores en las empresas para resolver problemas del tipo de cálculo de nómina, no de gestión de personal, debido a que era una labor que exige hacer muchos cálculos, en general repetitivos, y por tanto con alta probabilidad de error. Al resolverlos manualmente son trabajos en los cuales lo fundamental es comprender los procesos que se deben seguir para hacer correctamente el cálculo de los haberes y en donde los datos, aún no informatizados masivamente, se consideran secundarios.

En la década de los setenta empezó a tomar cuerpo la idea de que si bien los procesos son importantes, y una incorrección en su tratamiento puede causar notables problemas e incomodidades, más importantes son los datos. Los procesos son, similares en todas las organizaciones y, en algunos casos, son hasta relativamente fácil de transportar. Sin embargo los datos son algo propio de la organización, algo totalmente diferente de los de las demás organización, y que la caracterizan.

Ante esta situación no cabe otra alternativa que el análisis por partes, para lo cual hay que partir de la idea más general posible sobre el sistema a estudiar e ir descendiendo por pasos sucesivos hasta llegar a los detalles menores. Se está ante procesos de arriba a abajo, únicos, que permiten que haya coherencia entre todos los componentes del complicado sistema.

La década de los ochenta es la época marcada por las metodologías dirigida a datos cuya importancia va tomando cuerpo en las organizaciones. A mediados de la década el estado de la técnica permite a considerar entidades más complejas y con personalidad más acusada. Se empiezan a estudiar los objetos en sí como unidades de información.

Por otra parte las metodologías de desarrollo comienzan a interesarse no sólo en lograr que el proyecto sea puesto en funcionamiento sino en minimizar costos durante su desarrollo y sobre todo durante su mantenimiento.

Los nuevos métodos van buscando minimizar riesgos y, puesto que los errores más perjudiciales se producen en los primeros pasos, se comienza ya desde la fase más general del estudio por analizar los riesgos que significa seguir con las siguientes fases del desarrollo. Si los riesgos son superiores a lo que se consideran permisibles, se vuelve al paso anterior o se abandona el desarrollo. No sólo se realizan desarrollos lineales, en cascada, sino también desarrollos y métodos en espiral que son notablemente más complejos.

Enseguida se muestra la evolución cronológica de las metodologías de desarrollo de software, desde las primeras prácticas de desarrollo, pasando por el modelo de procesos, la conceptualización de los modelos del ciclo de vida del software, la propuesta de los modelos de la ingeniería del software, hasta desembocar en la metodología misma de los procesos de la ingeniería de software actuales.

2.2.1 Primera Etapa Cronológica (1950-1960): “Programación o técnicas de codificación”

En los años 50 el desarrollo estaba a cargo de programadores, por lo que se vio la importancia del análisis y diseño en el desarrollo de los sistemas. Aparecen los analistas programadores y analistas de sistemas [2]. En esta misma época, no existían metodologías de desarrollo. Las personas que desarrollaban los sistemas eran programadores más enfocados en la tarea de codificar, que en la de recoger y comprender las necesidades de los usuarios.

A medida que la complejidad de las tareas que realizaban las computadoras aumentaba, se hizo necesario disponer de un método sencillo para programar. Entonces, se crearon los lenguajes de programación de tercera generación [47] que diferían de las generaciones anteriores (lenguaje ensamblador conocido como segunda

generación y lenguaje de máquina como primera generación) en que sus instrucciones o primitivas eran de alto nivel (comprensibles por el programador, como si fueran lenguajes naturales) e independientes de la máquina. Estos lenguajes se llamaron lenguajes de alto nivel. Los ejemplos más conocidos son FORTRAN (FORmula TRANslator) que fue desarrollado para aplicaciones científicas y de ingeniería, y COBOL (COmmon Business-Oriented Language), que fue desarrollado por la U.S. Navy de Estados Unidos, para aplicaciones de gestión o administración.

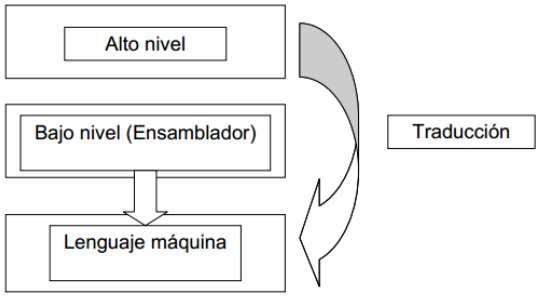


Figura 1. Generaciones de lenguajes de programación
Fuente: (ITI Gestión-Sistemas, Curso 2007-2008)

Aunque se disponía de las más recientes técnicas de codificación de la época, los usuarios a menudo, no quedaban satisfechos con el sistema software que se generaba, porque sus necesidades no estaban definidas con claridad en una fase de análisis previo. Ante esta perspectiva se vio la importancia del análisis y del diseño en el desarrollo de un sistema.

2.2.2 Segunda Etapa Cronológica (1960-1970): "Modelo de procesos"

De las técnicas de codificación de la primera etapa que desembocó en la inconformidad de los usuarios por la carencia de un análisis inicial, nace la necesidad de buscar alternativas para esquematizar de alguna manera la producción del software.

Desde que se empezó a trabajar sobre el desarrollo de programas, se siguieron ciertos métodos que permitían llevar a producir un buen proyecto, estas metodologías aplicadas eran simples, solo se preocupaban por los procesos mas no por los datos, por lo tanto los métodos eran desarrollados hacia los procesos.

El **modelo de procesos** predominaba para los años 60 y consistía en codificar y corregir (Code-and-Fix). Entonces, Codificar y corregir surge así como un modelo

poco útil, pero sin embargo es la respuesta para muchos programadores al carecer de una estructura formal a seguir. Realmente se la consideró aplicable, ya que de hecho la programación se la llevaba de forma intuitiva. A pesar que el desarrollo de software tomó una connotación de tarea unipersonal y donde el programador era el usuario de la aplicación, se lo consideró como una base inicial para la fabricación del software, en vista de que en este modelo se empieza a establecer una idea general de lo que se necesita construir, se utiliza cualquier combinación de diseño, código, depuración y métodos de prueba no formales que se los aplica hasta que se tiene el producto listo para entregarlo. Si al terminar se descubría que el diseño era incorrecto, la solución era desecharlo y volver a empezar [3]. Este modelo implementaba el código y luego se pensaba en los requisitos, diseño, validación y mantenimiento.

Paralelamente, aparece la **Crisis del Software**, llamada así por los problemas que surgieron a medida que se daba el desarrollo del software. Especialmente fue marcada por los excesos de costos, la escasa fiabilidad, la insatisfacción de los usuarios y el tiempo de creación de software que no finalizaba en el plazo establecido; todos estos aspectos conocidos como "síntomas" de la crisis de software.

Esto provocó grandes pérdidas en la década de los 70´s sobre el desarrollo de software, dando como resultado una nueva disciplina llamada "Ingeniería del Software" que abarcaría los aspectos técnicos del software y la gestión de datos. (RUBI, 2012).

2.2.3 Tercera Etapa Cronológica (1970-1985): “Proceso de Desarrollo Software y Modelos Tradicionales del Ciclo de Vida”

Esta tercera etapa está marcada por las soluciones que deben realizarse para resolver la llamada “Crisis del Software”. Mientras que el término **Ingeniería del Software** fue acuñado en la Conferencia de Ingeniería de Software de la OTAN en 1968 para discutir la crisis, los problemas que intentaba tratar empezaron mucho antes. Se volvió claro que los enfoques individuales al desarrollo de programas no escalaban hacia los grandes y complejos sistemas de software. Éstos no eran confiables, costaban más de lo esperado y se distribuían con demora. La historia de la ingeniería del software está entrelazada con las historias contrapuestas de hardware y software.

Para dar soluciones que enfrentaba el software, a inicios de la década de los setenta, se empezó a tomar la importancia de los datos, y para solucionar sistemas

complejos empezó el análisis por partes o etapas, se introducen la planeación y la administración.

El **ciclo de vida de desarrollo de software** o SDLC (Software Develop Life Cicle) empezó a aparecer, a mediados de la década, como un consenso para la construcción centralizada de software, y daría las pautas en la que se logra establecer, de manera general, los estados por los que pasa el producto software desde que nace a partir de una necesidad, hasta que muere.

Para formalizar la estructura del ciclo de vida se logra establecer el **proceso de desarrollo software**, como una ayuda al proceso de resolución a los problemas, intentando transformar la necesidad en una solución automatizada que satisface la misma. De esta manera, ambos conceptos se formarían para tratar de afrontar la crisis de la etapa anterior.

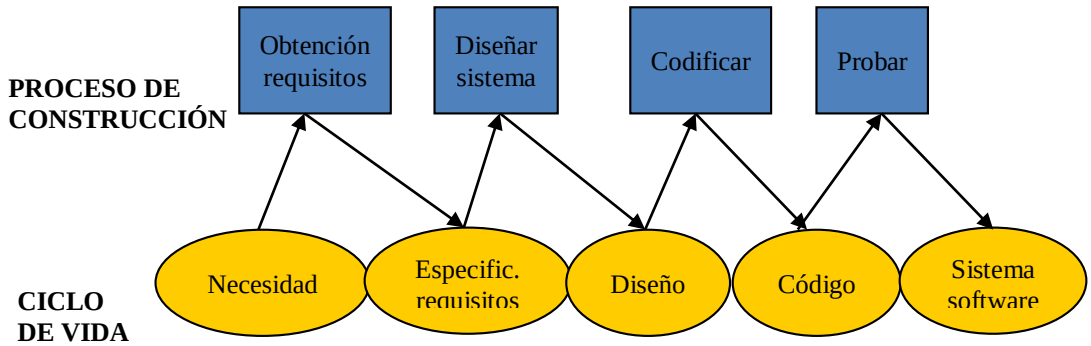


Figura 2. Proceso Software vs Ciclo de Vida

Fuente: (Los autores)

Es así, como la ingeniería del software se comienza a implantar y a valerse de una serie de modelos que establecen y muestran las distintas etapas y estados por los que pasa un producto software, desde su concepción inicial, pasando por su desarrollo, puesta en marcha y posterior mantenimiento, hasta la retirada del producto. A estos modelos se les denomina "**Modelos de ciclo de vida del software**", los cuales pretenden abarcar todo el proceso completo creando en cada paso normativas y parámetros describiendo el desarrollo de software desde la fase inicial hasta la final y orientarlo a ajustarse a las propias necesidades de cada empresa y que sean aceptados internacionalmente.

El aporte de dichos modelos al desarrollo del software se focaliza en el suministro de 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 administrar el progreso de desarrollo y el mantenimiento, en el sentido en que permitirían estimar recursos, definir puntos de control intermedios, monitorear el avance, etc.

El primer modelo publicado sobre el proceso de desarrollo software se derivó a partir de procesos más generales de ingeniería de sistemas. Debido al paso de una fase en cascada a otra, el modelo se conoce como **Modelo en Cascada**; definido por Winston Royce, el cual comenzó a diseñarse en 1966 y fue terminado alrededor de 1970 como respuesta al modelo de procesos. Este modelo sugiere un enfoque sistemático y secuencial para el desarrollo del software. Tiene más disciplina y se basa en el análisis, diseño, pruebas y mantenimientos [4]. Se define como una secuencia de fases, que en la etapa final de cada una de ellas se reúne la documentación para garantizar que cumple con las especificaciones y los requisitos antes de pasar a la siguiente fase.

La importancia de éste modelo para la época fue en convertirse en un ejemplo de un proceso dirigido por un plan; en principio, se planificaría y programaría todas las actividades del proceso, antes de comenzar a trabajar con ellas. Entonces, comienza a enseñar a los programadores que el proceso de software no es un simple modelo lineal, sino que implica retroalimentación de una fase a otra.

Por otro lado, se empieza a descubrir los errores y las omisiones en los requerimientos originales del software. Surgen los errores de programa y diseño, y se detecta la necesidad de nueva funcionalidad. Su principal problema detectado es la partición inflexible del proyecto en distintas etapas. Tienen que establecerse compromisos en una etapa temprana del proceso, por lo que dificulta responder a los requerimientos cambiantes del cliente. Por lo tanto, nace el sentido de evolucionar el sistema para mantenerse útil.

Posteriormente surgiría el **Modelo en V** propuesto por Alan Davis, desarrollado para terminar con algunos de los problemas que se vieron utilizando el enfoque de cascada tradicional. Los defectos estaban siendo encontrados demasiado tarde en el ciclo de vida, ya que las pruebas no se introducían hasta el final del proyecto. El modelo en V permitió hacer más explícita la tarea de la iteración de las actividades del proceso. Las pruebas que se implementarían en cada fase ayudarían a corregir posibles errores sin tener que esperar a que sean rectificados en la etapa final del proceso. Esto, sumado las pruebas unitarias y de integración se consigue obtener exactitud en los programas.

Este modelo proporcionó un avance significativo al desarrollo software. A pesar de estar relacionado con el modelo de cascada, el modelo V demostró ser más efectivo ya que pudo con las pruebas que se realizarían durante cada fase, se detectó menos errores por que estos se pudieron corregir al momento, lo contradictorio de estas pruebas es que generaron muchos costos y también se perdía tiempo para la entrega al usuario final.

Paralelamente derivaría, como respuesta de las debilidades del modelo en cascada, el **Modelo Iterativo**. Este modelo buscaría un mejor desempeño del desarrollo software al reducir el riesgo que surge entre las necesidades del usuario y el producto final por malos entendidos durante la etapa de recogida de requisitos. Propondría en la **iteración** de varios ciclos de vida en cascada que al final de cada iteración se le entregaría al cliente una versión mejorada o con mayores funcionalidades del producto. Por lo tanto, el cliente es quien después de cada iteración evalúa el producto y lo corrige o propone mejoras. Estas iteraciones se repetirían hasta obtener un producto que satisfaga las necesidades del cliente.

Para inicios de 1980, fue propuesto el **Modelo de Desarrollo Incremental** por Harlan Mills, el cual combinaría elementos del modelo en cascada con la filosofía interactiva de construcción de prototipos. Surgió el enfoque incremental de desarrollo como una forma de reducir la repetición del trabajo en el proceso de desarrollo y dar oportunidad de retrasar la toma de decisiones en los requisitos hasta adquirir experiencia con el sistema. Se basaría en la filosofía de construir incrementando las funcionalidades del programa y aplicaría secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento del software.

Tanto el Modelo Iterativo e Incremental dieron un avance significativo al desarrollo del software, pues permitieron la resolución de problemas de alto riesgo en tiempos tempranos del proyecto, una visión de avance en el desarrollo desde las etapas iniciales, una obtención del feedback del usuario lo antes posible, el aprendizaje y experiencia del equipo iteración tras iteración, una menor tasa de fallo del proyecto, mejor productividad del equipo, y menor cantidad de defectos, según demuestran estudios realizados sobre proyectos que han aplicado esta técnica. Pero a su vez presentan problemas agudos para sistemas grandes, complejos y de larga duración, y donde además diversos equipos desarrollan diferentes partes del sistema [48].

A finales de esta etapa, aparece el **desarrollo en espiral** de Barry Boehm en 1985 [5], el cual sería utilizado de forma generalizada en la ingeniería del software. A diferencia de otros modelos de proceso que finalizan cuando se entrega el software, el modelo se adaptaría para aplicarse a lo largo de toda la vida del software de cómputo. El nuevo producto evoluciona a través de cierto número de iteraciones alrededor de la espiral, teniendo un carácter permanente operativo hasta que el software se retira. La espiral presentaría un enfoque realista para el desarrollo de sistemas software a gran escala; considerándose así, el remedio de los modelos anteriores.

2.2.4 Cuarta Etapa Cronológica (1985-1999): "Métodos Rápidos e inicios del Desarrollo Ágil de la Ingeniería de Software"

A mediados de los años 80, es la época marcada por la distinción y conceptualización de los métodos de la ingeniería de software cuya importancia va tomando cuerpo en las organizaciones. Se empiezan a estudiar los objetos en sí como unidades de información, dando como punta pie inicial para el proceso evolutivo de desarrollo software el **modelo en espiral** presenciado a finales de la etapa anterior [11].

Boehm, autor de diversos artículos de ingeniería del software; modelos de estimación de esfuerzos y tiempo que se consume en hacer productos software; y modelos de ciclo de vida, ideó y promulgó un modelo desde un enfoque distinto al tradicional en Cascada: el Modelo Evolutivo Espiral. Su modelo de ciclo de vida en espiral tiene en cuenta fuertemente el riesgo que aparece a la hora de desarrollar software. Para ello, se comienza mirando las posibles alternativas de desarrollo, se opta por la de riesgos más asumibles y se hace un ciclo de la espiral. Si el cliente quiere seguir haciendo mejoras en el software, se vuelven a evaluar las nuevas alternativas y riesgos y se realiza otra vuelta de la espiral, así hasta que llegue un momento en el que el producto software desarrollado sea aceptado y no necesite seguir mejorándose con otro nuevo ciclo. Este modelo no fue el primero en tratar el desarrollo iterativo, pero fue el primer modelo en explicar las iteraciones.

Este modelo, Boehm lo propone en 1988 en su artículo "A Spiral Model of Software Development and Enhancement" y básicamente consiste en una serie de ciclos que se repiten en forma de espiral, comenzando desde el centro. Se suele

interpretar como que dentro de cada ciclo de la espiral se sigue un modelo en cascada, pero no necesariamente ha de ser así.

Para los años 90 se quiere dar respuesta al entorno siempre cambiante y en rápida evolución en que se han de desarrollar los programas informáticos, lo cual da lugar a trabajar en ciclos cortos (como mini-proyectos) que implementan una parte de las funcionalidades, pero sin perder el rumbo general del proyecto global.

Surge entonces el desarrollo de software de "**métodos rápidos**" [13] (también denominado Modelo rápido o abreviado AG) los cuales reducirían el tiempo del ciclo de vida del software (por lo tanto, acelera el desarrollo) al desarrollar, en primera instancia, una versión prototipo y después integrar la funcionalidad de manera iterativa para satisfacer los requisitos del cliente y controlar todo el ciclo de desarrollo. Los métodos rápidos se originaron por la inestabilidad del entorno técnico y el hecho de que el cliente a veces es incapaz de definir cada uno de los requisitos al inicio del proyecto. El término "rápido" es una referencia a la capacidad de adaptarse a los cambios de contexto y a los cambios de especificaciones que ocurren durante el proceso de desarrollo.

Empezando con las ideas de Brian Gallagher, Alex Balchin, Barry Boehm y Scott Shultz, James Martin desarrolló el enfoque de desarrollo rápido de aplicaciones durante los 80 en IBM y lo formalizó finalmente en 1991, con la publicación del libro, “Desarrollo rápido de aplicaciones”. Aparece entonces el **Desarrollo rápido de aplicaciones (RAD)** [6], para responder a la necesidad de entregar sistemas muy rápido. El método tiene una lista de tareas y una estructura de desglose de trabajo diseñada para la rapidez; comprende el desarrollo iterativo, la construcción de prototipos y el uso de utilidades CASE (Computer Aided Software Engineering). Tradicionalmente, el desarrollo rápido de aplicaciones tendría la tendencia a englobar también la usabilidad, utilidad y rapidez de ejecución. El desarrollo rápido de aplicaciones fue una respuesta a los procesos no ágiles de desarrollo desarrollados en los 70 y 80, tales como el método de análisis y diseño de sistemas estructurados y otros modelos en cascada. El enfoque de RAD no es apropiado para todos los proyectos. El alcance, el tamaño y las circunstancias, todo ello determinaría el éxito de un enfoque RAD.

En 1994, nace el **Método de desarrollo de sistemas dinámicos (DSDM),** desarrollado como un proceso de entrenamiento de negocios en Inglaterra, se estableció para crear una metodología RAD unificada, el cual definiría el marco para

desarrollar un proceso de producción de software. El aporte como metodología radica en entregar sistemas software en tiempo y presupuesto ajustándose a los cambios de requisitos durante el proceso de desarrollo software, comprometiendo a la participación continua con el usuario como clave principal para llevar a cabo un proyecto eficiente y efectivo, de tal forma que las decisiones se tomarían de forma más exacta.

En 1995 Schwaber y Sutherland, durante el OOPSLA '95 [7], presentaron en paralelo una serie de artículos describiendo **Scrum**, siendo ésta la primera aparición pública del método. Durante los años siguientes, Schwaber y Sutherland, colaboraron para consolidar los artículos antes mencionados, así con sus experiencias y el conjunto de mejores prácticas de la industria que conformarían a lo que actualmente se le conoce como Scrum. Para 2001, Schwaber y Mike Beedle describirían la metodología en el libro Agile Software Development with Scrum. Este método definiría un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Ha contribuido como método a la generación de software de calidad, principalmente por apoyar a proyectos con un rápido cambio de requisitos y al presentar una guía para las actividades de desarrollo dentro de un proceso de análisis incorporando actividades estructurales como: requerimientos, análisis, diseño, evolución y entrega; demostrando así, ser eficaz para proyectos con plazos de entrega muy apretados, requerimientos cambiante y negocios críticos.

Para 1996, surge **Extreme Programming** (**XP**) o Programación Extrema fundada por Ken Beck [8], identificando qué era lo simple y difícil al momento de programar. Centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP ha contribuido como método a la generación efectiva de software, fundamentada en la realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP promueve una adecuada práctica para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

En junio del 1998 se lanza **Rational Unified Process RUP** (Proceso unificado racional), siendo un método de desarrollo iterativo promovido por la compañía Rational Software, que fue comprada por IBM. El método RUP especifica principalmente, la constitución del equipo y las escalas de tiempo, así como un número

de modelos de documento. El enfoque práctico de RUP describe las buenas prácticas de ingeniería de software que se recomiendan para su uso en desarrollo de sistemas software [49].

2.2.5 Quinta Etapa Cronológica (2000 al presente): "Metodologías del Proceso de la Ingeniería de Software"

Con la creciente demanda de software en muchas organizaciones pequeñas, la necesidad de soluciones de software de bajo costo llevó al crecimiento de métodos más simples y rápidos que desarrollaran software funcional, de los requisitos de implementación, más rápidos y más fáciles [1]. El uso de prototipos rápidos, procedimientos y normas evolucionó al uso del concepto de la Metodología del Proceso de la Ingeniería de Software, destacando la utilización de las metodologías ligeras completas como la programación extrema (XP), que intentó simplificar muchas las áreas de la ingeniería de software, incluyendo la recopilación de requerimientos y las pruebas de confiabilidad para el creciente y gran número de pequeños sistemas de software. Sistemas de software muy grandes todavía utilizan metodologías muy documentadas, con muchos volúmenes en el conjunto de documentación; sin embargo, sistemas más pequeños tenían un enfoque alternativo más simple y rápido para administrar el desarrollo y mantenimiento de cálculos y algoritmos de software, almacenamiento y recuperación de información y visualización.

En el año 2001, miembros prominentes de la comunidad de desarrollo software se reunieron en Snowbird, Utah, y adoptaron el nombre de "**métodos ágiles**". Poco después, algunas de estas personas formaron la "alianza ágil", una organización sin fines de lucro que promueve el desarrollo ágil de aplicaciones.

El desarrollo ágil de software guiaría a los proyectos de desarrollo de software que evolucionan rápidamente con cambiantes expectativas y mercados competitivos. Los proponentes de este método creen que procesos pesados, dirigidos por documentos (como TickIT, CMM e ISO 9000) están desapareciendo en importancia. Algunas personas creen que las empresas y agencias exportan muchos de los puestos de trabajo que pueden ser guiados por procesos pesados.

Es así, como se introduce el manejo de las **Metodologías de Desarrollo Ágil**, contemplando un conjunto de métodos de ingeniería del software, que se basan en el desarrollo iterativo e incremental, teniendo presente los cambios y respondiendo a

estos mediante la colaboración de un grupo de desarrolladores auto-organizados y multidisciplinares.

Muchos métodos similares al ágil fueron creados antes del 2000. Entre los más notables se encuentran: Scrum, Crystal Clear (cristal transparente), programación extrema (en inglés eXtreme Programming o XP), desarrollo de software adaptativo (ASD), feature driven development (FDD), Método de desarrollo de sistemas dinámicos (en inglés Dynamic Systems Development Methodo DSDM).

En las metodologías ágiles, los procesos se desarrollan de manera solapada, donde el ciclo de vida del proyecto, es cíclico. La diferencia en el ciclo de vida de un proyecto ágil, en comparación con uno tradicional, se debe a la forma en la que el agilismo, solapa los procesos de manera iterativa.

La tendencia del control de procesos para desarrollo de software ha traído como resultado que proyectos no resulten exitosos debido al cambiante contexto que existe, por lo cual las metodologías ágiles pretenden resolver este inconveniente, construyendo soluciones a la medida asegurando la calidad. Los métodos ágiles fueron pensados especialmente para equipos de desarrollo pequeños, con plazos reducidos, requisitos volátiles y nuevas tecnologías. La filosofía de las metodologías ágiles, pretenden dar mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad [9].

Se puede apreciar claramente la evolución paulatina que ha desembocado hasta conformar la concepción de las **Metodologías del Proceso de Ingeniería de Software**, la cual describe el conjunto de herramientas, técnicas, procedimientos y soporte documental para el diseño del **Sistema Software**.

Por otro lado, Sommerville (2002) define que "una metodología de ingeniería de software es un enfoque estructurado para el desarrollo de software cuyo propósito es facilitar la producción de software de alta calidad de una forma costeable", cabe destacar que para usar este enfoque se debe manejar conceptos fundamentales tales como; procesos, métodos, tareas, procedimientos, técnicas, herramientas, productos, entre otros.

Particularmente, una metodología se basa en una combinación de los modelos de proceso genéricos para obtener como beneficio un software que solucione un problema. Adicionalmente una metodología debería definir con precisión los

artefactos, roles y actividades, junto con prácticas, técnicas recomendadas y guías de adaptación de la metodología al proyecto. Sin embargo, la complejidad del proceso de creación de software es netamente dependiente de la naturaleza del proyecto mismo, por lo que el escogimiento de la metodología estará acorde al nivel de aporte del proyecto, ya sea pequeño, mediano o de gran nivel.

Esta etapa cronológica logra establecer metodologías que pueden involucrar prácticas tanto de **Metodologías Ágiles** como de **Metodologías Tradicionales o Convencionales**. Tener metodologías diferentes para aplicar de acuerdo con el proyecto que se desarrolle resulta una idea interesante. De esta manera podríamos tener una metodología para cada proyecto, la problemática sería definir cada una de las prácticas, y en el momento preciso definir parámetros para saber cuál usar. Es importante tener en cuenta que el uso de un método ágil no es para todos. Sin embargo, una de las principales ventajas de los métodos ágiles es su peso inicialmente ligero y por eso las personas que no estén acostumbradas a seguir procesos encuentran estas metodologías bastante agradables. Por otro lado, las metodologías tradicionales o convencionales permiten crear software de manera más segura ya que estas están más establecidas según por sus pasos.

Por otra parte las metodologías de desarrollo comienzan a interesarse no sólo en lograr que el proyecto sea puesto en funcionamiento sino en minimizar costos durante su desarrollo y sobre todo durante su mantenimiento.

Los nuevos métodos que paulatinamente van generándose, buscan minimizar riesgos y, puesto que los errores más perjudiciales se producen en los primeros pasos, se comienza ya desde la fase más general del estudio por analizar los riesgos que significa seguir con las siguientes fases del desarrollo. Si los riesgos son superiores a lo que se consideran permisibles, se vuelve al paso anterior o se abandona el desarrollo. No sólo se realizan desarrollos lineales, en cascada, sino también desarrollos y métodos en espiral que son notablemente más complejos, apoyándose también con el desarrollo ágil.

2.3 Antecedentes Conceptuales y Referenciales

2.3.1 Caracterización Tecnológica del Proceso de la Ingeniería del Software

a. Definición de Proceso

Proceso, según el diccionario de la Real Academia Española, describe "La acción de avanzar o ir para adelante, al paso del tiempo y al conjunto de etapas sucesivas

advertidas en un fenómeno natural o necesarias para concretar una operación artificial". (RAE, 2012)

Según la IEEE 9000 describe al proceso como "Conjunto de actividades mutuamente relacionadas o que interactúan, las cuales transforman elementos de entrada en resultados".

Según la enciclopedia libre Wikipedia, expresa que "Un proceso es un conjunto de actividades o eventos (coordinados u organizados) que se realizan o suceden (alternativa o simultáneamente) bajo ciertas circunstancias con un fin determinado. Este término tiene significados diferentes según la rama de la ciencia o la técnica en que se utilice." (Fundación Wikipedia Inc., 2013)

Según Sommerville, Ian, expresa que "Un proceso de software es un conjunto de actividades que conducen a la creación de un producto software. Estas actividades pueden consistir en el desarrollo de software desde cero en un lenguaje de programación estándar como Java o C. Sin embargo cada vez más se desarrolla un nuevo software ampliando y modificando los sistemas existentes y configurando e integrando software comercial o componentes del sistema". (Sommerville, 2005) [12].

La consideración asumida como definición de proceso, se encuentra escrita por parte de Sommerville, ya que cumple con las características específicas como conocimiento y procesos de beneficio múltiple fundamentales para el entorno en que se desarrolla la investigación.

b. Definición de Ingeniería

La ingeniería, según la enciclopedia libre Wikipedia, "Es el conjunto de conocimientos y técnicas científicas aplicadas a la creación, perfeccionamiento e implementación de estructuras (tanto físicas como teóricas) para la resolución de problemas que afectan la actividad cotidiana de la sociedad." (Fundación Wikipedia Inc., 2013)

Según la Real Academia Española, expone que "Se entiende por ingeniería toda aplicación de las ciencias físicas, químicas y matemáticas; de la técnica industrial y en general, del ingenio humano, a la utilización e invención sobre la materia." (RAE, 1994)

Según Cavero, expresa que "La ingeniería del software retoma conocimientos de diversas fuentes y aborda el desarrollo del software de calidad a nivel industrial. Así,

construye conocimiento en torno a las prácticas de desarrollo de software. Este conocimiento se concreta en la definición de procesos, métodos y modelos que pueden ser aplicados por los ingenieros en sus actividades profesionales.” (Cavero, 2005).

Se asume en la investigación la conceptualización de la ingeniería referida por Cavero, por cuanto contiene las características precisas y el uso adecuado de los recursos necesarias en el campo estructurado.

c. Definición de Software

Según la Real Academia Española El software es un conjunto de programas, instrucciones y reglas informáticas que permiten ejecutar distintas tareas en una computadora. Se considera que el software es el equipamiento lógico e intangible de un ordenador (RAE, 1993).

La terminación "software" fue usado por primera vez por John W. Tukey y lo describe como conjunto de instrucciones detalladas que controlan la operación de un sistema computacional. Está formado por una serie de instrucciones y datos, que permiten aprovechar todos los recursos que el computador tiene, de manera que pueda resolver gran cantidad de problemas (Tukey, 1958).

Se asume en la investigación la definición de software dada por John W. Tukey, ya que redacta adecuadamente las características como sistema computacional, uso de recursos, resolver problemas que son precisamente los puntos necesarios que aportan a esta investigación.

d. Proceso de Ingeniería del Software

El proceso de ingeniería del software consiste en un conjunto coherente de políticas, estructuras organizacionales, tecnologías, procedimientos y artefactos que son necesarios para concebir, desarrollar, instalar y mantener un producto software (Fugetta, 2000).

El proceso de Ingeniería del Software se basa en modelos, métodos y herramientas que sirven como una guía para los ingenieros del software durante el proceso de desarrollo, con la finalidad de mejorar la calidad de los proyectos, procesos y productos mediante la evaluación y medición de los mismos. El objetivo de las organizaciones desarrolladoras de estos modelos, procesos y metodologías es que en las empresas desarrolladoras de software se los ponga en práctica para ver las mejoras

en los procesos de cada una de las fases de desarrollo (Armando Cabrera, Procesos de la Ingeniería de Software, 2012).

2.3.2 Principios del Proceso de Ingeniería del Software

Según Pressmann Roger (2005) [14], los procesos de desarrollo de software definitivamente no se pueden dejar de lado, en lugar de ello se deben fortalecer, es decir se debe armar un cuerpo disciplinar soportado en conjunto de principios, que permitan ajustar los problemas de manera formal, sin ambigüedades, dejando de lado la especulación y las prácticas ad hoc.

Según Patton Ron (2006), los defectos más comunes de los procesos están en considerar una serie de prácticas que funcionaron en determinadas situaciones. Pero no pasan de ser prácticas, que tarde o temprano no tendrán sentido en escenarios diferentes. Las tendencias sobre todo de los métodos heterodoxos están muy ligadas a la importancia que se da a las personas, el mismo manifiesto ágil propone: "Los individuos y la interacción por encima de los procesos y herramientas". Y no es que las personas no sean importantes en la búsqueda del conocimiento, lo que es un punto débil es que el método de llegar al conocimiento dependa de la persona, la disciplina por si sola debe brindar los mecanismos formales que permitan seguir la senda al conocimiento, en este caso la senda al desarrollo. En otras palabras los fenómenos suceden para su naturaleza intrínseca y no por la interpretación subjetiva de las personas que lo experimentan.

Apuntando en la dirección de dar una directriz a la disciplina de software se formulan cinco principios fundamentales que restringen de manera formal el que hacer en esta área del conocimiento, así:

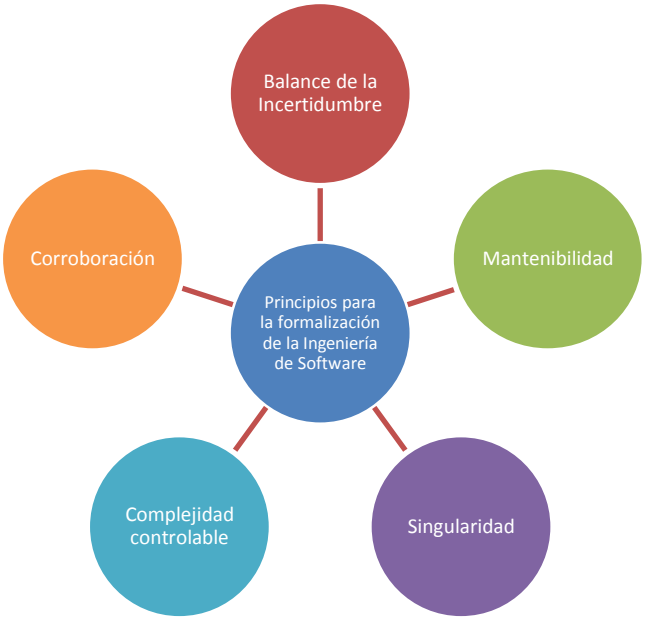


Figura 3. Principios para la formalización de la Ingeniería de Software
Fuente: (Facultad de Ingeniería, Universidad Distrital Francisco José de Caldas de Colombia)

a. Balance de la Incertidumbre

Según Shaw Mary and Garlan David., en su libro "Software Architecture, Perspectives on an Emerging Discipline", el desarrollo de software es un ejercicio creativo en el que se parte de un dominio del problema con gran incertidumbre para llegar a un dominio de la solución con gran certidumbre. En el dominio del problema hay más preguntas que respuestas, mientras en el dominio de la solución hay más respuesta que preguntas, por lo menos con respecto al problema resuelto. En este orden de ideas es de suponer que el ejercicio de desarrollar software esta en ir eliminando paulatinamente la incertidumbre para acercarse a la certidumbre permitida que nos dé la solución. En este camino es de esperarse que se busquen mecanismos que faciliten el entendimiento del problema.

Aparecen dos constantes importantes las preguntas y las repuestas. Por un lado las preguntas se originan de las posibilidades que puede tomar un problema mientras las respuestas se originan de las soluciones que se pueden adoptar ante un problema. Las preguntas son necesarias y pertinentes para resolver un problema, de no ser así tendríamos un problema predecible en donde no hay más posibilidades que la obtenida en un primer razonamiento.

El desarrollo de un problema basado en un esquema lógico de programación sugiere que se realicen preguntas y respuestas, las cuales posibilitan que la transición de estados contenga la lógica necesaria para mostrar una solución. Ante estas evidencias vale la pena cuestionarse sobre la correlación que deben tener las preguntas y las respuestas, y producto de esta correlación podrían originarse los siguientes planteamientos:

- ¿Una pregunta debería originar una sola respuesta?
- ¿Una pregunta debería originar varias respuestas?
- ¿Varias preguntas deberían originar una sola respuesta?

b. Singularidad

El debate sobre las metodologías y procesos de desarrollo puede tener tanto de ancho como de profundo, quizá el problema en si no lo constituya el proceso sino el producto. Querer desarrollar un producto de software por medio de un proceso atado a ideas convencionales puede ser la fuente del problema. El software es un producto con propiedades particulares, contemplando las siguientes:

Producto de software Producto convencional

Figura 4. Software como producto
Fuente: (Facultad de Ingeniería, Universidad Distrital Francisco José de Caldas de Colombia)

- No es perceptible a los sentidos: es decir no se puede oler, degustar, tocar ver u oír, es una representación que existe en otros medios de almacenamiento y reproducción.
- Vive en el mundo binario y no se ajusta en su naturaleza misma a leyes fácticas.
- Es replicable fácilmente por tanto es fácil de reproducir.
- No se degrada aunque si se pueda desactualizar.
- Puede ser integrado y crear complejidades mayores.
- Es consecuencia de la lógica y el raciocinio.

Al no degradarse el software no se le puede hacer un conteo de depreciación como sucede con un producto convencional esto es una gran ventaja pues este efecto negativo se puede evitar, el concepto más parecido es el de desactualización que por su puesto sucede con cualquier otro producto. Al ser este, producto del raciocinio y la lógica también es posible pensarlo como el producto de la creatividad y la innovación características que lo hacen relevante frente a otros productos.

c. Complejidad Controlable

Desarrollar software implica un grado de complejidad considerable que se encuentra presente en todas y cada una de las actividades de proceso que se realicen para alcanzar este objetivo.

Los retos fundamentales de la complejidad de realizar la ingeniería de requerimientos están centrados en la comunicación y la representación. Por un lado la comunicación implica un proceso complejo de interacción de seres humanos y sistemas. No se pueden asumir que la obtención de requerimientos se reduce a la aplicación de métodos convencionales como: encuestas, entrevistas, cuestionarios, entre otras técnicas comunes. Cuando se ha resuelto el problema de comunicación es necesario resolver el otro problema fundamental, el cual se encuentra en la necesidad de la representación. A pesar de tener frameworks y métodos de análisis y modelado de requerimientos, como por ejemplo los casos de uso [15], las historias de usuario [16], entre otras. Seria optimista pensar que estas herramientas son suficientes para resolver el problema de representación. La representación exige que sea fiel a lo representado y que a su vez sea simple.

Por su lado el diseño tiene una complejidad en la arquitectura y el detalle. La arquitectura como disciplina emergente tiene sus propios retos, pero quizá se deba prestar atención a la necesidad de contar con modelos estandarizados que sirvan como mapas de navegación, al respecto hay importantes aportes, Shaw y Garlan [17] apunta a una clasificación de alto nivel de posibles modelos arquitectónicos, el trabajo de la pandilla de los cuatro GoF [18], es otra importante aproximación al problema. Los lenguajes MIL y ADL [19] también son un importante esfuerzo al respecto incluso UML 2.0 [20] se ha preocupado por dar soporte al problema de MDA [21] y el estándar MOF. Afortunadamente se es consciente de la necesidad de aproximar el modelo

expresado en lenguajes arquitectónicos a los modelos expresados en lenguajes de programación.

La complejidad de la implementación tiene su fuente en los leguajes de programación los paradigmas y los algoritmos. Un lenguaje es complejo desde la misma sintaxis, pasando por los mecanismos de gestión de memoria hasta la interpretación de los paradigmas. Los paradigmas de desarrollo similarmente a lo planteado por Tomas Kuhn [10], recurre a un conjunto de teorías, métodos y prácticas para la representación de un problema en términos de un lenguaje de alto nivel.

La prueba es una actividad de proceso que consume gran parte de los recursos del desarrollo, incluso invirtiendo grandes recursos es imposible realizar todo el conjunto de pruebas posible [22].

d. Corroboración

La corroboración, es la propiedad asociada a la capacidad de aprobación de los diferentes modelos del proceso de desarrollo. La corroboración es un concepto ampliado de prueba, necesario precisamente para obtener confianza en los modelos que se realizan a lo largo del proceso de desarrollo. Gran parte de la problemática ocasionada por la poca fiabilidad de una implementación es producida por la sobrecarga que tiene el modelo de implementación "código en un lenguaje de programación", esta carga es la carga de defectos altamente probables de los modelos anteriores a la implementación y que no so corregidos al momento de implementar.

Un modelo de análisis arroja errores de análisis, un modelo de diseño y arquitectura arroja errores propios del diseño y arquitectura, estos errores no se perciben hasta que se plantea el modelo de implementación que es sobre el que concretamente se puede probar. Un modelo de corroboración puede verse como en la Figura 5.

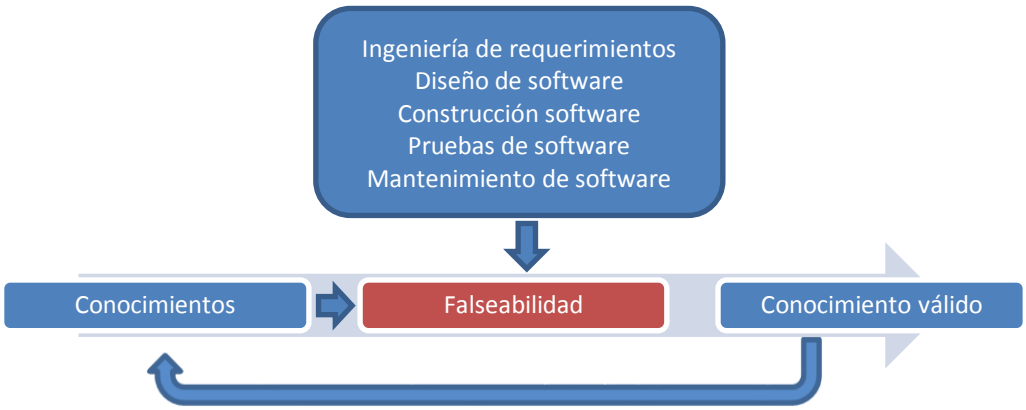


Figura 5. Modelo de Corroboración

Fuente: (Facultada de Ingeniería, Universidad Distrital Francisco José de Caldas de Colombia)

En el modelo de corroboración se identifican dos entradas por un lado el conocimiento y por otro lado las actividades de proceso: ingeniería de requerimientos, diseño de software, construcción de software, pruebas de software y mantenimiento de software. La combinación del conocimiento necesario pera la actividades de procesos entran en una máquina de falseación, cuyo objetivo fundamental es poner a prueba las hipótesis que se formulen de la entradas, tratando de demostrar que no es correcta, después del procesos de falseación se produce una clase de conocimiento que se considera validado, ya sea que en este sea pudo falsear o acertar. La misma prueba puede ser falseada, sentido en el cual la corroboración puede ser considerada incluso como una meta prueba.

e. Rastreabilidad

La rastreabilidad es otra propiedad que se debe conseguir en un software, esta podría confundirse con la trazabilidad pero al igual de la corroboración y prueba, la rastreabilidad y trazabilidad tienen una correspondencia similar. La rastreabilidad involucra un camino doble de construcción y de construcción de un concepto. Mientras en la construcción de un concepto se involucran un conocimiento ordenado y disciplinado ascendente, en la deconstrucción se involucra una revisión del conocimiento con la posibilidad de seguir ese conocimiento de manera no solo sencilla sino constructiva de nuevo conocimiento (ver Figura 6).

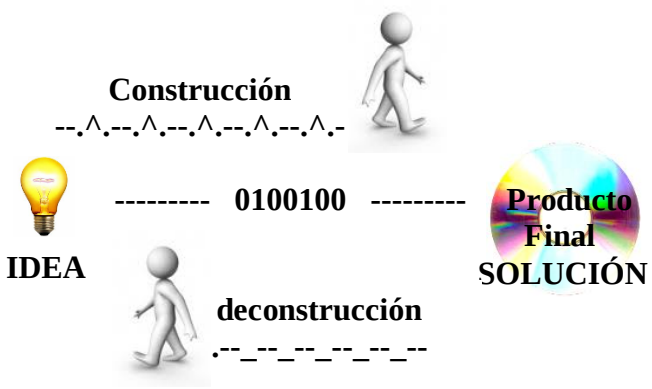


Figura 6. Rastreabilidad

Fuente: (Facultada de Ingeniería, Universidad Distrital Francisco José de Caldas de Colombia)

Los procesos de la rastreabilidad, construcción y deconstrucción permiten la elaboración de conocimiento en ambas direcciones, ya sea de la ida hacia el producto o tomando el mismo producto hasta reconstruir la idea que lo origino. En estos dos caminos es posible no solo hacer seguimiento sino construcción de conocimiento. La rastreabilidad permite sumergirse en el desarrollo construyendo conocimiento y regresar deconstruyendo, con la ganancia adicional de evolucionar en la concepción del producto.

2.3.3 Etapas de Procesos de Ingeniería del Software

Pressman, Roger S. (2003), en su libro "Ingeniería del Software, un enfoque Práctico" [30], establece que la ingeniería de software requiere llevar a cabo numerosas tareas agrupadas en etapas, al conjunto de estas etapas se le denomina ciclo de vida. Las etapas [24] comunes a casi todos los modelos de ciclo de vida son las siguientes:

a. Análisis de requisitos

Extraer los requisitos de un producto de software es la primera etapa para crearlo. Mientras que los clientes piensan que ellos saben lo que el software tiene que hacer, se requiere habilidad y experiencia para reconocer requisitos incompletos, ambiguos o contradictorios. El resultado del análisis de requisitos con el cliente se plasma en el documento ERS, Especificación de Requisitos del Sistema, cuya estructura puede

venir definida por varios estándares, tales como CMMI. Asimismo, se define un diagrama de Entidad/Relación, en el que se plasman las principales entidades que participarán en el desarrollo del software.

La captura, análisis y especificación de requisitos (incluso pruebas de ellos), es una parte crucial; de esta etapa depende en gran medida el logro de los objetivos finales. Se han ideado modelos y diversos procesos de trabajo para estos fines. Aunque aún no está formalizada, ya se habla de la Ingeniería de requisitos.

No siempre en la etapa de "análisis de requisitos" las distintas metodologías de desarrollo llevan asociado un estudio de viabilidad y/o estimación de costes. El más conocido de los modelos de estimación de coste del software es el modelo COCOMO.

b. Especificación

La especificación de requisitos describe el comportamiento esperado en el software una vez desarrollado. Gran parte del éxito de un proyecto de software radicará en la identificación de las necesidades del negocio (definidas por la alta dirección), así como la interacción con los usuarios funcionales para la recolección, clasificación, identificación, priorización y especificación de los requisitos del software.

c. Arquitectura

La integración de infraestructura, desarrollo de aplicaciones, bases de datos y herramientas gerenciales, requieren de capacidad y liderazgo para poder ser conceptualizados y proyectados a futuro, solucionando los problemas de hoy. El rol en el cual se delegan todas estas actividades es al Arquitecto de Software.

El arquitecto de software es la persona que añade valor a los procesos de negocios gracias a su valioso aporte de soluciones tecnológicas. La arquitectura de sistemas en general, es una actividad de planeación, ya sea a nivel de infraestructura de red y hardware, o de software. La arquitectura de software consiste en el diseño de componentes de una aplicación (entidades del negocio), generalmente utilizando patrones de arquitectura. El diseño arquitectónico debe permitir visualizar la interacción entre las entidades del negocio y además poder ser validado, por ejemplo por medio de diagramas de secuencia. Un diseño arquitectónico describe en general el cómo se construirá una aplicación de software.

d. Programación

Reducir un diseño a código puede ser la parte más obvia del trabajo de ingeniería de software, pero no necesariamente es la que demanda mayor trabajo y ni la más complicada. La complejidad y la duración de esta etapa está íntimamente relacionada al o a los lenguajes de programación utilizados, así como al diseño previamente realizado.

e. Prueba

Consiste en comprobar que el software realice correctamente las tareas indicadas en la especificación del problema. Una técnica de prueba es probar por separado cada módulo del software, y luego probarlo de forma integral, para así llegar al objetivo. Se considera una buena práctica el que las pruebas sean efectuadas por alguien distinto al desarrollador que la programó, idealmente un área de pruebas; sin perjuicio de lo anterior el programador debe hacer sus propias pruebas. En general hay dos grandes formas de organizar un área de pruebas, la primera es que esté compuesta por personal inexperto y que desconozca el tema de pruebas, de esta forma se evalúa que la documentación entregada sea de calidad, que los procesos descritos son tan claros que cualquiera puede entenderlos y el software hace las cosas tal y como están descritas. El segundo enfoque es tener un área de pruebas conformada por programadores con experiencia, personas que saben sin mayores indicaciones en qué condiciones puede fallar una aplicación y que pueden poner atención en detalles que personal inexperto no consideraría.

f. Documentación

Todo lo concerniente a la documentación del propio desarrollo del software y de la gestión del proyecto, pasando por modelaciones (UML),diagramas de casos de uso, pruebas, manuales de usuario, manuales técnicos, etc.; todo con el propósito de eventuales correcciones, usabilidad, mantenimiento futuro y ampliaciones al sistema.

g. Mantenimiento

Fase dedicada a mantener y mejorar el software para corregir errores descubiertos e incorporar nuevos requisitos. Esto puede llevar más tiempo incluso que el desarrollo del software inicial. Alrededor de 2/3 del tiempo de ciclo de vida de un proyecto [25] está dedicado a su mantenimiento. Una pequeña parte de este trabajo consiste eliminar

errores (*bugs*); siendo que la mayor parte reside en extender el sistema para incorporarle nuevas funcionalidades y hacer frente a su evolución.

2.3.4 Tipos de Procesos de Ingeniería del Software

Según Sommer Ville, Ian [26], expresa que "Existen procesos de software en todas las organizaciones, desde empresas unipersonales hasta grandes multinacionales. Estos procesos son de diferentes tipos dependiendo del grado de formalización del proceso, los tipos de productos desarrollados y el tamaño de la organización, entre otros. Hay cuatro clases de procesos de software.".

a. Procesos informales

Son procesos en los que no existe un modelo de proceso definido de forma estricta. El proceso utilizado es elegido por el equipo de desarrollo. Los procesos informales podrían utilizar procedimientos formales, como la gestión de configuraciones, pero los procedimientos a utilizar y sus relaciones son definidos por el equipo de desarrollo.

b. Procesos gestionados

Se utiliza un modelo de proceso para dirigir el proceso de desarrollo. El modelo de proceso define los procedimientos, su agenda y las relaciones entre los procedimientos.

c. Procesos metodológicos

Se utiliza algún o algunos métodos de desarrollo definidos como los métodos sistemáticos para diseño orientado a objetos. Estos procesos se benefician de la existencia de herramientas CASE para el diseño y el análisis.

d. Procesos de mejora

Son procesos que tienen inherentemente objetivos de mejora. Existe un presupuesto específico para estos procesos de mejora, y de procedimientos para

introducir tales mejoras. Como parte de estas mejoras, se introducen mediciones cuantitativas del proceso.

Esta clasificación es útil debido a que sirve como base para la mejora de procesos multidimensional. Ayuda a las organizaciones a elegir un proceso de desarrollo apropiado para los diferentes productos.

2.3.5 Entidades Reguladoras del Proceso de Ingeniería del Software

Estándares IEEE.- una asociación mundial dedicada a la estandarización principalmente, cuyo trabajo es promover la creatividad, el desarrollo y la integración, compartir y aplicar los avances en las tecnologías de la información, electrónica y ciencias en genera.

Entre sus normas tenemos: Norma IEEE 1058.1 (específica el formato y contenidos que se debe utilizar para desarrollar los planes para proyectos software); IEEE 1074 (determina el conjunto de actividades esenciales que deben ser incorporadas en el desarrollo de un producto software); IEEE 830 (caracterización de una buena especificación de requisitos de software) (IEEE, ieee.org, 1998).

Normas ISO/IEC.- Este consorcio formado por las normas ISO en conjunto con las normas IEC presenta estándares que se han forjado para ser un solo estándar que globaliza los parámetros especializados aplicados a los procesos que cada empresa maneja. Es aliada importante en EEUU de los **Estándares ANSI**, con la diferencia que se encuentra ubicada en un lugar estratégico para su funcionalidad. (ANSI, 1998). Entre ellos tenemos las normas:

- ISO/IEC 17799 (un estándar para la seguridad de la información);
- La serie ISO/IEC 20000 (para la Gestión de Servicios);
- Norma ISO/IEC 9126 (estándar internacional para la evaluación de la calidad del software);
- La Norma ISO 9241 (norma enfocada a la calidad en usabilidad y ergonomía tanto de hardware como de software). (IEEE, 2007)

El propósito de la Normalización surge en diferentes aspectos de la gestión pero tienen que ver en general con la aplicación de leyes que salvaguarden en bienestar del

ser humano y garantice calidad tanto al lado que aplica y al lado que consume productos como servicios software. Cuando se aplican modelos de calidad de desarrollo, finalmente lo que se logra es que mejoren el proceso de ingeniería software, así como la calidad del software que se desarrolla.

- Las normas y estándares son voluntarios y no tienen obligación legal. Tratan mayormente sobre documentación de procesos e informes de control.
- Han sido diseñadas para ayudar a organizaciones privadas y gubernamentales a establecer y evaluar objetivamente sus procesos para desarrollo.
- Proporcionan, además, una guía para la certificación del sistema por una entidad externa acreditada.
- No establecen objetivos ambientales cuantitativos ni límites en cuanto a emisión de contaminantes.
- No fijan metas para la prevención de la contaminación ni se involucran en el desempeño ambiental a nivel mundial, sino que establecen herramientas y sistemas enfocados a los procesos de producción de una empresa u otra organización, y de las externalidades que de ellos deriven al medio ambiente.
- Los requerimientos de las normas son flexibles y, por lo tanto, pueden ser aplicadas a organizaciones de distinto tamaño y naturaleza.

Los modelos de calidad para el desarrollo del software dicen qué se debe hacer, no cómo se debe hacerlo. Debido a que estos factores dependen a menudo de las metodologías que usen las empresas de software y de sus propios objetivos en común.

Existen una variedad de modelos de calidad para el desarrollo del proceso de ingeniería software; a continuación se presentan algunos de ellos.

a. Ciclo de vida del producto software

Los estándares y normas que se describirán a continuación, se enfocan al ciclo de vida del producto software afectando así a todas las etapas incluidas dentro, sin necesidad de especificar un ciclo de vida específico, sino dejando a elección de los usuarios el mismo ya que se acomoda a cualquier ciclo de vida.

Estándar IEEE 1074.- Este estándar ha sido desarrollado por la IEEE para determinar el conjunto de actividades esenciales que deben ser incorporadas en el

desarrollo de un producto software, sin recomendar un ciclo de vida específico. Cabe mencionar que el IEEE 1074 requiere adaptarse a cada proyecto.

Las actividades que no se incluyan deben justificarse. Esta norma requiere la definición de un ciclo de vida del software del usuario y muestra la cartografía en típico ciclos de vida del software. No es la intención de definir o implica un ciclo de vida del software propio. (IEEE, arantxa, 1998)

El IEEE 1074 contempla 17 grupos de actividades y 65 actividades en total. Los grupos de actividades son:

- De Gestión del Proyecto (17 actividades)
  - Iniciación (4 actividades)
  - Planificación (8)
  - Monitoreo y control (5)

- De pre-desarrollo (11)
  - Exploración de conceptos (4)
  - Asignación al Sistema (3)
  - Importación al software (4)

- De desarrollo (10)
  - Requisitos (3)
  - Diseño (4)
  - Implementación (3)

- De post-desarrollo (12)
  - Instalación (3)
  - Operación y soporte (3)
  - Mantenimiento (3)
  - Retiro (3)
  - Integrales (15)
  - Evaluación (7)
  - Gestión de configuración (3)
  - Desarrollo de documentación (2)
  - Capacitación (3)

Estándar ISO 12207.- Este marco de referencia cubre el ciclo de vida del software desde la conceptualización de ideas hasta su retirada y consta de procesos para adquirir y suministrar productos y servicios software. Cubre además el control y la mejora de estos procesos. (NTP-ISO/IEC, 2006).

Contiene procesos, actividades y tareas para aplicar durante la adquisición de un sistema que contiene software, un producto software puro o un servicio software y durante el suministro, desarrollo, operación y mantenimiento de productos software.

La norma incorpora otras sub normas para el desarrollo de cada etapa del ciclo de vida entre las cuales están:

- NTP-ISO 9000:2001 Sistema de gestión de la calidad. Fundamentos y vocabularios, requisitos.
- NTP-ISO/IEC 9126.1 Ingeniería de software – Calidad de Producto, Modelo de calidad.
- NTP-ISO/IEC 12119 Tecnología de la información Paquetes, Software – Requerimientos de calidad y pruebas.
- NTP-ISO/IEC 14598.1 Tecnología de la información, Evaluación del producto software. Vista general
- NTP-ISO/IEC TR 9126.2Ingeniería de software, Calidad de producto, Métricas externas.
- NTP-ISO/IEC TR 9126.3Ingeniería de software, Calidad de producto, Métricas internas.

Se define aquí los principales procesos del ciclo de vida para la aplicación de esta norma:

- Proceso de adquisición.
  - Inicio.
  - Preparación de la solicitud de propuestas.
  - Preparación y actualización del contrato.
  - Seguimiento del proveedor.
  - Aceptación y finalización.

- Proceso de suministro
  - Inicio.

- Preparación de la respuesta.
- Contrato.
- Planificación.
- Ejecución y control.
- Revisión y evaluación.
- Entrega y finalización.

- Proceso de desarrollo.
  - Implementación del proceso.
  - Análisis de los requerimientos del sistema.
  - Diseño de la arquitectura del sistema.
  - Análisis de los requerimientos software.
  - Diseño de la arquitectura del software.
  - Diseño detallado del software.
  - Codificación y pruebas del software.
  - Integración del software.
  - Pruebas de calificación del software.
  - Integración del sistema.
  - Pruebas de calificación del sistema.
  - Instalación del software.
  - Apoyo a la aceptación del software.

- Proceso de operación.
  - Implementación del proceso.
  - Pruebas de operación.
  - Operación del sistema.
  - Soporte al usuario.

- Proceso de mantenimiento
  - Implementación del proceso.
  - Análisis de problemas y modificaciones.
  - Implementación de las modificaciones.
  - Revisión/aceptación del mantenimiento.
  - Migración.

- Retirada del software.

b. Etapas del desarrollo del software

En general, existen modelos de calidad establecidos para los procesos de ingeniería de software, los cuales definen a ésta de forma jerárquica; es decir la calidad se produce como consecuencia de la evaluación de un conjunto de indicadores o métricas en diferentes etapas.

Norma IEEE 1058.1.- Este estándar especifica el formato y contenidos de los planes para la gestión de proyectos software. No especifica las técnicas exactas que pueden ser usadas en el desarrollo de los planes de proyectos, ni ofrece ejemplos de los planes de gestión de proyectos. Cada organización que usa este estándar debería desarrollar un conjunto de prácticas y procedimientos para proporcionar una guía detallada para la preparación y actualización de los planes de gestión de los proyectos software basada en este estándar. Estas prácticas detalladas y procedimientos deberían tener en cuenta los factores del entorno, organizacionales y políticos que pueden influenciar en la aplicación de este estándar. Incorpora en ella la utilización de estos estándares:

- Glosario Estándar IEEE de Terminología de Ingeniería del Software (1983)
- Estándar IEEE para Planes de Gestión de la Configuración (1983)
- Estándar IEEE para la Documentación de Pruebas de Software (1983)
- Estándar IEEE de Planes para el Aseguramiento de Calidad del Software (1984)
- Guía IEEE para la Planificación de Aseguramiento de Calidad Software (1986)
- Estándar IEEE para la Planificación de Verificación y Validación de Software (1986)

Este estándar identifica el conjunto mínimo de elementos que debería aparecer en todos los planes para la gestión de productos software. A fin de seguir el estándar los planes para la gestión de proyectos software deben ajustarse al formato especificado en este estándar. El detalle de este plan, nos dice que podemos hacer durante el inicio del proyecto y se escribe a continuación:

- Introducción.
  - Visión General del Proyecto.
  - Entregables del Proyecto.
  - Evolución del PGPS.
  - Material de Referencia.
  - Definiciones y Acrónimos.

- Organización del Proyecto.
  - Modelo de Procesos.
  - Estructura Organizativa
  - Límites e Interfaces organizativos
  - Responsabilidades

- Procesos de Gestión.
  - Objetivos y Prioridades de gestión.
  - Supuestos, dependencias y restricciones
  - Gestión de Riesgos.
  - Mecanismos de supervisión y control
  - Plan del Personal

- Proceso Técnico
  - Métodos, herramientas y técnicas
  - Documentación del Software
  - Funciones de soporte a proyectos

- Paquetes de trabajo, Calendario y Presupuestos
  - Paquetes de trabajo
  - Dependencias
  - Requerimientos de recursos
  - Presupuesto y distribución de recursos
  - Calendario o Agenda

- Plan de Desarrollo.

Norma IEEE 830.- Un enfoque recomendado de cómo especificar los requisitos del software, con el fin de conseguir un documento completo y sin ambigüedades que ayude:

1. a los clientes o compradores a describir con precisión lo que quieren obtener;
2. a los suministradores a comprender exactamente lo que el cliente quiere:
3. a los individuos al cargo de desarrollar una especificación de requisitos del software (ERS) normalizada para su organización, de definir el formato y contenido de esas especificaciones, o bien de comprobar su calidad.

Esta "práctica recomendada" describe el proceso de creación de un producto (la especificación de requisitos del software o ERS) y el contenido del producto en sí. Puede utilizarse directamente o como modelo para un norma más específica.

- Introducción y alcance
- Referencias
- Definiciones
- Consideraciones en la producción de una ERS
- Partes de una ERS
- Plantillas/modelos.

c. Aseguramiento de la calidad del producto software final

La utilización de metodologías o procedimientos estándares para el análisis, diseño, programación y prueba del software que permitirán uniformar la filosofía de trabajo, en aras de lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que elevarán la productividad, tanto para la labor de desarrollo como para el control de la calidad del software.

Gestión de la calidad de software (ISO 9000).- Es la norma que establece una serie de requisitos para la normalización de un sistema de gestión de calidad.

Pone a disposición de un auditor o certificador los procesos internos, de forma que este indique si cumple o no la normativa al 100%, audita el sistema; Si los resultados son positivos se emiten la certificación y cada cierto tiempo se tiene que renovar. Su implantación en las organizaciones supone una gran cantidad de beneficios como:

1. Reducción de rechazos e incidencias en la producción o prestación de servicios
2. Aumento de la productividad

3. Mayor compromiso con los requisitos del cliente
4. Mejora continua

En el modelo propuesto por la ISO consiste en ocho principios estos son en si el cuerpo de la norma, que reflejan las mejores prácticas de para obtener calidad del proceso de desarrollo software.

- Organización enfocada al cliente. Las organizaciones dependen de sus clientes, y por lo tanto deberían comprender sus necesidades actuales y futuras, satisfacer sus requisitos y esforzarse por superar sus expectativas
- Liderazgo de la dirección. Los líderes establecen la unidad de propósito y la orientación de la organización. Ellos también deberían crear y mantener un ambiente interno, en el cual el personal pueda implicarse completamente en el logro de los objetivos de la organización
- Participación del personal. El personal, a todos los niveles, es la esencia de una organización y su total compromiso posibilita que sus habilidades sean usadas para el beneficio de la organización.
- Enfoque basado en procesos. Un resultado deseado se alcanza más eficientemente cuando los recursos y las actividades relacionados se gestionan como un proceso.
- Enfoque de sistema para la gestión. Identificar, entender y gestionar los procesos interrelacionados como un sistema contribuye a la eficacia y la eficiencia de una organización en el logro de sus objetivos.
- Mejora continua. La mejora continua del desempeño global de la organización debería ser un objetivo permanente de ésta.
- Enfoque (objetivo) basado en hechos para la toma de decisiones. Las decisiones eficaces se basan en el análisis de datos y la información.
- Relaciones mutuamente beneficiosas con el proveedor. Una organización y sus proveedores son interdependientes, y una relación mutuamente beneficiosa aumenta la capacidad de ambos para crear valor

Finalmente para cada uno de los criterios de calidad se definen un conjunto de métricas o medidas cuantitativas de ciertas características del producto que indican el grado en que dicho producto posee un determinado atributo de calidad. De esta manera, a través de un modelo de calidad para el desarrollo software se concretan los

aspectos relacionados con ella de tal manera que se puede definir, medir y planificar. Además el empleo de un modelo de calidad permite comprender las relaciones que existen entre diferentes características de un producto software.

d. Herramientas del Proceso de Ingeniería del Software

Según Giraldo, L. and Y. Zapata (2005) [31], las **herramientas** del proceso de ingeniería del software proporcionan un soporte automático o semi-automático para el proceso y los métodos, a estas herramientas se les llama herramientas CASE (C*omputer-Aided Software Engineering* o Ingeniería de Software Asistida por Computadora). Se puede definir a las Herramientas CASE como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un Software.

Otras definiciones:

- Las Herramientas CASE son un conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases.
- La sigla genérica para una serie de programas y una filosofía de desarrollo de software que ayuda a automatizar el ciclo de vida de desarrollo de los sistemas.
- Una innovación en la organización, un concepto avanzado en la evolución de tecnología con un potencial efecto profundo en la organización. Se puede ver al CASE como la unión de las herramientas automáticas de software y las metodologías de desarrollo de software formales.

Según Kendall & Kendall [32], el empleo de herramientas Case permiten integrar el proceso de ciclo de vida:

- Análisis de datos y procesos integrados mediante un repositorio.
- Generación de interfaces entre el análisis y el diseño.
- Generación del código a partir del diseño.
- Control de mantenimiento.
- **Tipos de Herramientas CASE**: No existe una única clasificación de herramientas CASE, es difícil incluirlas en una clase determinada. Podrían clasificarse atendiendo a:

- Las plataformas que soportan.
- Las fases del ciclo de vida del desarrollo de sistemas que abarca.
- La arquitectura de las aplicaciones que produce.
- Su funcionalidad.

Según Fuster, G. G., J. M. F. Torres, et al. (2006) [33], las herramientas CASE, en función de las fases del ciclo de vida abarcadas, se pueden agrupar de la forma siguiente:

Herramientas integradas, I-CASE (Integrated CASE, CASE integrado): abarcan todas las fases del ciclo de vida del desarrollo de sistemas. Son llamadas también CASE workbench. Las herramientas I-CASE se basan en una metodología. Tienen un repositorio y aportan técnicas estructuradas para todas las fases del ciclo de vida. Estas son las características que les confieren su mayor ventaja: una mejora de la calidad de los desarrollos. Sin embargo, no todas ellas son modernas en el sentido de aprovechar la potencia de las estaciones de trabajo o la utilización de lenguajes de alto nivel o técnicas de prototipo.

Herramientas de alto nivel, U-CASE (Upper CASE - CASE superior) o front-end, orientadas a la automatización y soporte de las actividades desarrolladas durante las primeras fases del desarrollo: análisis y diseño. Una estrategia posible es utilizar una U-CASE para análisis y diseño, combinada con otras herramientas más modernas para las fases de construcción y pruebas. En este caso, habría que vigilar cuidadosamente la integración entre las distintas herramientas.

Herramientas de bajo nivel, L-CASE (Lower CASE - CASE inferior) o back-end, dirigidas a las últimas fases del desarrollo: construcción e implantación.

Juegos de herramientas o toolkits, son el tipo más simple de herramientas CASE. Automatizan una fase dentro del ciclo de vida. Dentro de este grupo se encontrarían las herramientas de reingeniería, orientadas a la fase de mantenimiento.

Otra posible clasificación, utilizando la funcionalidad como criterio principal, es la siguiente:

- Herramientas de gestión de proyectos
- Herramientas de gestión y configuración de software (SCM)
- Herramientas de calidad y seguridad de software
- Herramientas de análisis y diseño
- Herramientas de desarrollo de interfaz de usuarios

- Herramientas para la Ingeniería de Software Orientada a Objetos
- Herramientas de integración y prueba
- Herramientas de métodos formales
- Herramientas Cliente/Servidor
- Herramientas de Ingeniería WEB
- Herramientas de Reingeniería

e. Ejemplos de Herramientas CASE

Para el Proceso de Análisis:

- **DFD:** El DFD es una herramienta gráfica muy valiosa para el análisis de requisitos del software. Sin embargo muchas veces el analista confunde los propósitos del diagrama, al elaborarlo e interpretar su función como un diagrama de flujo.

- **Microsoft Project:** Microsoft Project [37] es un software de administración de proyectos diseñado, desarrollado y comercializado por Microsoft para asistir a administradores de proyectos en el desarrollo de planes, asignación de recursos a tareas, dar seguimiento al progreso, administrar presupuesto y analizar cargas de trabajo. Permite el aprendizaje rápido con el planeamiento y la administración guiados, organización y seguimiento de las tareas y recursos, comparar versiones de planes de proyectos, evaluar los cambios, realizar un seguimiento del rendimiento, generar informes predefinidos, compartir planes de proyecto, colaboración entre grupos de trabajo, presenta diagramas como: Diagrama de Grant y Diagrama de Pert (diagrama de red).

- **Magic Draw:** MagicDraw [40], es una herramienta de modelaje con completas características UML. Es desarrollada por No Magic, Inc. Implementada totalmente en JAVA. Diseñada para los analistas del negocio, los analistas del software, los programadores, los ingenieros de software, y los escritores de la documentación, esta herramienta de desarrollo dinámica y versátil facilita análisis y el diseño de los sistemas y de las bases de datos orientados objeto.

- **System Architect:** System Architect [34] posee un repositorio único que integra todas las herramientas, y metodologías usadas. System Architect es considerado un Upper Case, que puede ser integrado a la mayoría de los generadores de código. Traduce modelos de entidades, a partir de la enciclopedia, en esquemas para Sybase, DB2, Oracle, Ingress, SQL Server, RDB, XDB, Progress, Paradox, SQL Base, AS400, Interbase, OS/2, DBMS, Dbase 111, Informix, entre otros. Genera también Windows DDL y definiciones de datos para lenguaje C/C++. Posibilita a través de ODBC, la creación de bases de datos a partir del modelo de entidades, para los diversos manejadores de bases de datos arriba mencionados.

Para el Proceso de Diseño:

- **Racional Rose:** Según Zhao, J. and D. Thomas (2005) [38], Rational Rose es una herramienta de producción y comercialización establecidas por Rational Software Corporation (actualmente parte de IBM). Rose es un instrumento operativo conjunto que utiliza el Lenguaje Unificado (UML) como medio para facilitar la captura de dominio de la semántica, la arquitectura y el diseño.

- **CASE Studio:** Herramienta con potente utilidad de modelado para varias bases de datos. CASE Studio es una herramienta profesional con la que pueden diseñarse bases de datos, incluye facilidades para la creación de diagramas de relación, modelado de datos y gestión de estructuras. Tiene soporte para trabajar con una amplia variedad de formatos de base de datos (Oracle, SQL, MySQL, PostgreSQL, Access) y permite además generar xcripts SQL, aplicar procesos de ingeniería inversa, usar plantillas de diseño personalizables y crear detallados informes en HTML y RTF.

- **Power Designer:** Power Designer [35] es una suite de aplicaciones de Powersoft para la construcción, diseño y modelado de datos a través de diversas aplicaciones. Es una herramienta para el análisis, diseño inteligente y construcción sólida de una base de datos y un desarrollo orientado a modelos de datos a nivel físico y conceptual.

- **ERwin:** PLATINUM ERwin es una herramienta de diseño de base de datos. Brinda productividad en diseño, generación, y mantenimiento de aplicaciones. Desde un modelo lógico de los requerimientos de información, hasta el modelo físico perfeccionado para las características específicas de la base de datos diseñada, ERwin permite visualizar la estructura, los elementos importantes, y optimizar el diseño de la base de datos.

Para el Proceso de Codificación:

- **JDeveloper:** Para Pressmann [39], este magnífico entorno integrado desarrollado por Oracle trabaja con la ingeniería inversa, es decir primero se crea el código y después el diagrama. Es un software propietario pero gratuito desde 2005. Las primeras versiones de 1998 estaban basadas en el entorno JBuilder de Borland, pero desde la versión 9i de 2001 está basado en Java, no estando ya relacionado con el código anterior de JBuilder.

Para todo el Proceso de Ciclo de Vida:

- **Visual Paradigm:** Visual Paradigm [41] es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. También proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Presenta licencia gratuita y comercial.

- **Oracle Designer:** Oracle Designer [36] es un juego de herramientas para guardar las definiciones que necesita el usuario y automatizar la construcción rápida de aplicaciones cliente/servidor. Integrado con Oracle Developer, Oracle Designer provee una solución para desarrollar sistemas empresariales cliente/servidor. Sofisticadas aplicaciones cliente/servidor pueden ser 100% generadas usando la lógica de la aplicación y el módulo de componentes reusables. Oracle Designer también habilita la captura del diseño de sistemas existentes, salvaguardando la versión actual. Todos los datos ingresados por cualquier herramienta de Oracle Designer, en cualquier fase de desarrollo, se

guardan en un repositorio central, habilitando el trabajo fácil del equipo y la dirección del proyecto. Oracle Designer no fuerza al uso de alguna metodología específica, pero en cambio proporciona un juego de herramientas que le permiten que use la metodología de desarrollo que elija. Oracle Designer soporta las siguientes metodologías: Desarrollo Rápido de Aplicaciones (RAD), Ingeniería de la Información (IE), Modelado Asistido de Procesos, Captura de Diseño Asistido.

2.3.6 Caracterización Tecnológica de las Metodologías de Desarrollo de la Ingeniería de Software

a. Método

Según el diccionario de la Real Academia Española, el método se define como "El modo de decir o hacer con orden; y filosóficamente se refiere a el procedimiento que se sigue en las ciencias para hallar la verdad y enseñarla". (RAE, 1993)

Según Hernández, Christen, Jaramillo, Villaseñor, Roca y Zamudio (1990), el método es un "Procedimiento concreto que se emplea, de acuerdo con el objeto y con los fines de la investigación, para organizar los pasos de ésta y propiciar resultados coherentes".

Vélez S. (2001), afirma que es el "Camino para alcanzar una meta. Sistema de principios (identidad, contradicción, exclusión) y normas (inducción, deducción) de razonamiento para establecer conclusiones en forma objetiva". Bajo esta óptica, Carles Tomás (s.f), afirma que "ningún método puede considerarse como definitivo y menos aún como universal. La universalidad no es sinónimo de objetividad".

La definición tomada de Vélez S. se considera acertada, debido a un concepto muy bien estructurado definido precisamente para el propósito de esta investigación.

b. Metodología

Según la Real Academia Española, la metodología se define como "Ciencia del método" o "Conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal". (RAE, 1997)

Según Herrman, C. S. (2009), "La metodología hace referencia al conjunto de procedimientos racionales utilizados para alcanzar una gama de objetivos que rigen en un a investigación científica, una exposición doctrinal o tareas que requieran habilidades, conocimientos o cuidadas específicos".

Según el Software Engineering Institute, "La metodología es un conjunto de métodos integrados para alcanzar una meta común".

La definición tomada de Software Engineering Institute se considera acertada, debido a que es un concepto preciso para el propósito de esta investigación.

c. Desarrollo

Según el diccionario de la Real Academia Española, el desarrollo "Está vinculado a la acción de desarrollar o a las consecuencias de este accionar" (RAE, 1997). Es necesario, por lo tanto, rastrear el significado del verbo desarrollar: se trata de incrementar, agrandar, extender, ampliar o aumentar alguna característica de algo físico (concreto) o intelectual (abstracto).

Según la Universidad El Bosque de Colombia Consiste en: "Trabajos sistemáticos basados en los conocimientos existentes, derivados de la investigación y/o la experiencia práctica, dirigidos a la producción de nuevos materiales, productos o dispositivos; al establecimiento de nuevos procesos, sistemas y servicios, o a la mejora sustancial de los ya existentes". (UBC, Bogotá 2009).

Según el Diccionario Definición ABC, el término desarrollo tiene varias acepciones: "En primer lugar, el término puede ser entendido como el proceso de evolución, crecimiento y cambio de un objeto, persona o situación específica en determinadas condiciones. El desarrollo es la condición de evolución que siempre tiene una connotación positiva ya que implica un crecimiento o paso hacia etapas o estadíos superiores".

La definición acertada de desarrollo propuesta por la Universidad El Bosque, caracteriza un detalle adecuado para el campo de acción que se dará en la investigación.

2.3.7 Metodologías de Desarrollo Software

Para Duval, Paul M. [42], el desarrollo de software no es una tarea fácil. Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas

dimensiones del proceso de desarrollo. Por una parte tenemos aquellas propuestas más **tradicionales** que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en muchos otros. Una posible mejora es incluir en los procesos de desarrollo más actividades, más artefactos y más restricciones, basándose en los puntos débiles detectados. Sin embargo, el resultado final sería un proceso de desarrollo más complejo que puede incluso limitar la propia habilidad del equipo para llevar a cabo el proyecto. Otra aproximación es centrarse en otras dimensiones, como por ejemplo el factor humano o el producto software. Esta es la filosofía de las **metodologías ágiles**, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo, pero manteniendo una alta calidad. Las metodologías ágiles están revolucionando la manera de producir software, y a la vez generando un amplio debate entre sus seguidores y quienes por escepticismo o convencimiento no las ven como alternativa para las metodologías tradicionales.

Un objetivo de décadas ha sido encontrar procesos y metodologías, que sean sistemáticas, predecibles y repetibles, a fin de mejorar la productividad en el desarrollo y la calidad del producto software.

La evolución de la disciplina de ingeniería del software ha traído consigo propuestas diferentes para mejorar los resultados del proceso de construcción. Las metodologías tradicionales haciendo énfasis en la planificación y las metodologías ágiles haciendo énfasis en la adaptabilidad del proceso.

2.3.8 Clasificación de las Metodologías de Desarrollo

Desarrollar un buen software depende de un gran número de actividades y etapas, donde el impacto de elegir la metodología para un equipo en un determinado proyecto es trascendental para el éxito del producto.

Según la filosofía de desarrollo se pueden clasificar las metodologías en dos grupos [27]. Las metodologías tradicionales, que se basan en una fuerte planificación durante todo el desarrollo, y las metodologías ágiles, en las que el desarrollo de software es incremental, cooperativo, sencillo y adaptado.

a. Metodologías Tradicionales

Las metodologías tradicionales son denominadas, a veces, de forma peyorativa, como metodologías pesadas. Centran su atención en llevar una documentación exhaustiva de todo el proyecto y en cumplir con un plan de proyecto, definido todo esto, en la fase inicial del desarrollo del proyecto.

Otra de las características importantes dentro de este enfoque, son los altos costes al implementar un cambio y la falta de flexibilidad en proyectos donde el entorno es volátil.

Las metodologías tradicionales (formales) se focalizan en la documentación, planificación y procesos (plantillas, técnicas de administración, revisiones, etc.)

- **Rational Unified Process (RUP)**

El proceso unificado Rational (RUP) [43], es un marco de trabajo de proceso de desarrollo de software iterativo creado por Rational Software Corporation, una división de IBM desde 2003. RUP no es un proceso preceptivo concreto individual, sino un marco de trabajo de proceso adaptable, con la idea de ser adaptado por las organizaciones de desarrollo y los equipos de proyecto de software que seleccionarán los elementos del proceso que sean apropiados para sus necesidades.

RUP fue originalmente desarrollado por Rational Software, y ahora disponible desde IBM. El producto incluye una base de conocimiento con artefactos de ejemplo y descripciones detalladas para muchos tipos diferentes de actividades.

RUP resultó de la combinación de varias metodologías y se vio influenciado por métodos previos como el modelo en espiral. Las consideraciones clave fueron el fallo de proyectos usando métodos monolíticos del estilo del modelo en cascada y también la llegada del desarrollo orientado a objetos y las tecnologías GUI, un deseo de elevar el modelado de sistemas a la práctica del desarrollo y de resaltar los principios de calidad que aplicaban a las manufacturas en general al software.

Los creadores y desarrolladores del proceso se centraron en el diagnóstico de las características de diferentes proyectos de software fallidos. De esta forma intentaron reconocer las causas raíz de tales fallos. También se fijaron en los procesos de ingeniería del software existentes y sus soluciones para estos síntomas. El fallo de los proyectos es causado por una combinación de varios síntomas, aunque cada proyecto

falla de una forma única. La salida de su estudio fue un sistema de mejores prácticas del software al que llamaron RUP.

Módulos de RUP (Building blocks)

RUP se basa en un conjunto de módulos o elementos de contenido [43], que describen qué se va a producir, las habilidades necesarias requeridas y la explicación paso a paso describiendo cómo se consiguen los objetivos de desarrollo. Los módulos principales, o elementos de contenido, son:

- Roles (quién): un rol define un conjunto de habilidades, competencias y responsabilidades relacionadas.
- Productos de trabajo (qué): un producto de trabajo representa algo que resulta de una tarea, incluyendo todos los documentos y modelos producidos mientras que se trabaja en el proceso.
- Tareas (cómo): una tarea describe una unidad de trabajo asignada a un rol que proporciona un resultado significante.

Fases de Ciclo de Vida RUP

RUP determina que el ciclo de vida del proyecto consiste en cuatro fases [44]. Estas fases permiten que el proceso sea presentado a alto nivel de una forma similar a como sería presentado un proyecto basado en un estilo en cascada, aunque en esencia la clave del proceso recae en las iteraciones de desarrollo dentro de todas las fases. También, cada fase tiene un objetivo clave y un hito al final que denota que el objetivo se ha logrado.

Las cuatro fases en las que divide el ciclo de vida del proyecto son:

- Fase de iniciación: se define el alcance del proyecto.
- Fase de elaboración: se analizan las necesidades del negocio en mayor detalle y se define sus principios arquitectónicos.
- Fase de construcción: se crea el diseño de la aplicación y el código fuente.
- Fase de transición: se entrega el sistema a los usuarios.

RUP proporciona un prototipo al final de cada iteración. Dentro de cada iteración, las tareas se categorizan en nueve disciplinas:

- Seis disciplinas de ingeniería
  - Modelaje de negocio
  - Requisitos

- Análisis y diseño
- Implementación
- Pruebas
- Despliegue

➢ Tres disciplinas de soporte

- Gestión de la configuración y del cambio
- Gestión de proyectos
- Entorno

- **Rapid Application Development (RAD)**

La metodología de desarrollo rápido de aplicaciones (RAD) [46], se desarrolló para responder a la necesidad de entregar sistemas muy rápido. El enfoque de RAD no es apropiado para todos los proyectos. El alcance, el tamaño y las circunstancias, todo ello determina el éxito de un enfoque RAD.

El método RAD tiene una lista de tareas y una estructura de desglose de trabajo diseñada para la rapidez. El método comprende el desarrollo iterativo, 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 rapidez de ejecución.

A continuación, se muestra un flujo de proceso posible para el desarrollo rápido de aplicaciones:

Figura 7. Flujo de Proceso RAD

Fuente: (INTECO, pág 47.)

RAD requiere el uso interactivo de técnicas estructuradas y prototipos para definir los requisitos de usuario y diseñar el sistema final. Usando técnicas estructuradas, el desarrollador primero construye modelos de datos y modelos de procesos de negocio preliminares de los requisitos. Los prototipos ayudan entonces al analista y los usuarios a verificar tales requisitos y a refinar formalmente los modelos de datos y procesos. El ciclo de modelos resulta a la larga en una combinación de requisitos de negocio y una declaración de diseño técnico para ser usado en la construcción de nuevos sistemas.

Los enfoques RAD pueden implicar compromisos en funcionalidad y rendimiento a cambio de permitir el desarrollo más rápido y facilitando el mantenimiento de la aplicación.

Ventajas

Las principales ventajas que puede aportar este tipo de desarrollo son las siguientes:

- Velocidad de desarrollo
- Calidad: según lo definido por el RAD, es el grado al cual un uso entregado resuelve las necesidades de usuarios así como el grado al cual un sistema entregado tiene costes de mantenimiento bajos. El RAD aumenta la calidad con la implicación del usuario en las etapas del análisis y del diseño.
- Visibilidad temprana debido al uso de técnicas de prototipado.
- Mayor flexibilidad que otros modelos.
- Ciclos de desarrollo más cortos.

Desventajas

Entre los principales inconvenientes que se pueden encontrar en el uso del desarrollo rápido de aplicaciones se pueden encontrar:

- Características reducidas.
- Escalabilidad reducida.
- Más difícil de evaluar el progreso porque no hay hitos clásicos.

b. Metodologías Ágiles

Este enfoque nace como respuesta a los problemas que puedan ocasionar las metodologías tradicionales y se basa en dos aspectos fundamentales, retrasar las

decisiones y la planificación adaptativa. Basan su fundamento en la adaptabilidad de los procesos de desarrollo.

Estas metodologías ponen de relevancia que la capacidad de respuesta a un cambio es más importante que el seguimiento estricto de un plan.

Según Sommerville, Ian [28], expresa que, cada método tiene características propias y hace hincapié en algunos aspectos más específicos. A continuación se resumen algunos métodos ágiles que están siendo utilizados con éxito en proyectos reales:

- **Adaptive Software Development (ASD)**:

Su impulsor fue Jim Highsmith [29]. Sus principales características son: iterativo, orientado a los componentes software más que a las tareas y tolerante a los cambios. El ciclo de vida que propone tiene tres fases esenciales: especulación, colaboración y aprendizaje.

- Especulación: En ésta se inicia el proyecto y se planifican las características del software.
- Colaboración: Se desarrollan las características del software.
- Aprendizaje: Se revisa su calidad, y se entrega al cliente.

La revisión de los componentes sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo.

- **Crystal_Clear**

Desarrollado por Alistair Cockburn [29]. Comprende un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo Crystal Clear (3 a 8 miembros) y Crystal Orange (25 a 50 miembros).

- **Feature Driven Development (FDD)**

Sus impulsores son Jeff De Luca y Peter Coad [29]. Define un proceso iterativo que consta de 5 pasos.

- Desarrollo de un modelo global: Cuando comienza el desarrollo, los expertos del dominio están al tanto de la visión, el contexto y los requerimientos del sistema a construir.
- Construcción de una lista de funcionalidades: Se elabora una lista de funcionalidades que resuma la funcionalidad general del sistema. La lista es elaborada por los desarrolladores y es evaluada por el cliente.
- Planeación por funcionalidad: En este punto se procede a ordenar los conjuntos de funcionalidades conforme a su prioridad y dependencia, y se asigna a los programadores jefes.
- Diseño por funcionalidad: Se selecciona un conjunto de funcionalidades de la lista y se procede a diseñar.
- Construcción por funcionalidad: Seguido del diseño, se construye la funcionalidad mediante un proceso iterativo. Una iteración puede tomar de unos pocos días a un máximo de dos semanas. El proceso iterativo incluye inspección de diseño, codificación, pruebas unitarias, integración e inspección de código.

- **Lean Software Development (LSD)**

Definida por Bob Charettes [29]. En esta metodología, los cambios se consideran riesgos, pero si se manejan adecuadamente se pueden convertir en oportunidades que mejoren la productividad del cliente. Su principal característica es introducir un mecanismo para implementar dichos cambios.

- **Programación Extrema (XP)**

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

- **Método de desarrollo de sistemas dinámicos (DSDM)**

Define el marco para desarrollar un proceso de producción de software. Nace en 1994 con el objetivo de crear una metodología RAD unificada. Sus principales características son: es un proceso iterativo e incremental y el equipo de desarrollo y el usuario trabajan juntos. Propone cinco fases:

- Estudio viabilidad: Se definirá si la metodología se ajusta al proyecto a desarrollar.
- Estudio del negocio: Hace referencia el alcance funcional a automatizar.
- Modelado funcional: Se realizará el diseño funcional que tendrá el proyecto.
- Diseño y construcción, e implementación: Se realizará el diseño técnico y desarrollo del proyecto, las mismas que serán iterativas, además de existir realimentación a todas las fases.

- **Scrum**

Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle [29]. Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos:

- El desarrollo de software se realiza mediante iteraciones sprints: Las cuales constan con una duración de 30 días, tomando en cuenta que el resultado de cada sprint es un incremento ejecutable que se muestra al cliente.
- Las reuniones a lo largo del proyecto: Destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

Los métodos ágiles ofrecen una solución casi a medida para una gran cantidad de proyectos que tienen estas características. Una de las cualidades más destacables en un método ágil es su sencillez, tanto en su aprendizaje como en su aplicación, reduciéndose así los costos de implantación en un equipo de desarrollo.

c. Comparación de Metodologías Tradicionales y Ágiles

Según Arboleda, Hugo F. [46], en las metodologías tradicionales el principal problema es que nunca se logra planificar bien el esfuerzo requerido para seguir la metodología. Pero entonces, si logramos definir métricas que apoyen la estimación de las actividades de desarrollo, muchas prácticas de metodologías tradicionales podrían

ser apropiadas. El no poder predecir siempre los resultados de cada proceso no significa que estemos frente a una disciplina de azar. Lo que significa es que estamos frente a la necesidad de adaptación de los procesos de desarrollo que son llevados por parte de los equipos que desarrollan software.

Tener metodologías diferentes para aplicar de acuerdo con el proyecto que se desarrolle resulta una idea interesante. Estas metodologías pueden involucrar prácticas tanto de metodologías ágiles como de metodologías tradicionales. De esta manera podríamos tener una metodología por cada proyecto, la problemática sería definir cada una de las prácticas, y en el momento preciso definir parámetros para saber cuál usar.

Es importante tener en cuenta que el uso de un método ágil no vale para cualquier proyecto. Sin embargo, una de las principales ventajas de los métodos ágiles es su peso inicialmente ligero y por eso las personas que no estén acostumbradas a seguir procesos encuentran estas metodologías bastante agradables.

En la tabla que se muestra a continuación aparece una comparativa entre estos dos grupos de metodologías.

Tabla 1. Comparativa entre metodologías tradicionales y desarrollo ágil

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

2.4 Antecedentes Contextuales

A partir del 04 de junio de 1954 y durante 59 años consecutivos, la Fuerza Aérea Ecuatoriana, a través de la Escuela de Especialidades y hoy en día con el nombre de Escuela Técnica de la Fuerza Aérea (ETFA) establecida en la ciudad de Latacunga, Provincia de Cotopaxi, ha venido cumpliendo con la noble tarea de formar, capacitar y profesionalizar al personal de Aerotécnicos en las diferentes especialidades de la aviación, pasando por sus aulas un total de 51 Promociones. Paralelamente, el Departamento de Tecnologías de la Información y Comunicaciones (TIC), departamento de la ETFA, ha desempeñado un trabajo arduo y prolongado, las 24 horas del día, a lo largo de la historia de la Escuela; manteniendo las comunicaciones y el manejo de las tecnologías que han aparecido a través del tiempo de manera efectiva, y optimizando al máximo sus recursos, en pro de la formación de Alumnos Aspirantes a Tropa de la Fuerza Aérea.

La tendencia gubernamental es fortalecer la automatización de los procesos, para concordar con las áreas administrativas en una verdadera cultura de procesos que conlleve una optimización de los recursos del Estado, razón por la cual el Departamento TIC se encuentra incursionando en proporcionar guías que normen el proceso de desarrollo de ingeniería software y que contribuya al mejoramiento de los procesos de automatización.

Una de las tareas fundamentales del Departamento TIC es la creación de sistemas software, mediante el análisis, diseño e implementación de soluciones informáticas ajustado a las necesidades institucionales y de cada uno de los departamentos de la ETFA. A la vez, el Departamento se dedica a una serie de actividades tales como Operaciones TIC’s donde se realiza actividades acerca del software (instalación, mantenimiento y administración de servidores, servicios, sistemas, desarrollo de software), redes, manejo de mensajes e información militar; Mantenimiento TIC’s donde se realiza el mantenimiento preventivo y correctivo de equipos e infraestructura de comunicaciones, telefonía, redes e informática de las áreas administrativas, laboratorios y campus de la ETFA; se maneja de manera minuciosa la Seguridad de la Información donde se aplica la implementación, difusión y control de normas de la seguridad de la información y comunicaciones, estudio de las vulnerabilidades y backups; y finalmente el Soporte Técnico donde se actualiza y distribuye guías telefónicas y asesoría técnica en telefonía, comunicaciones e informática.

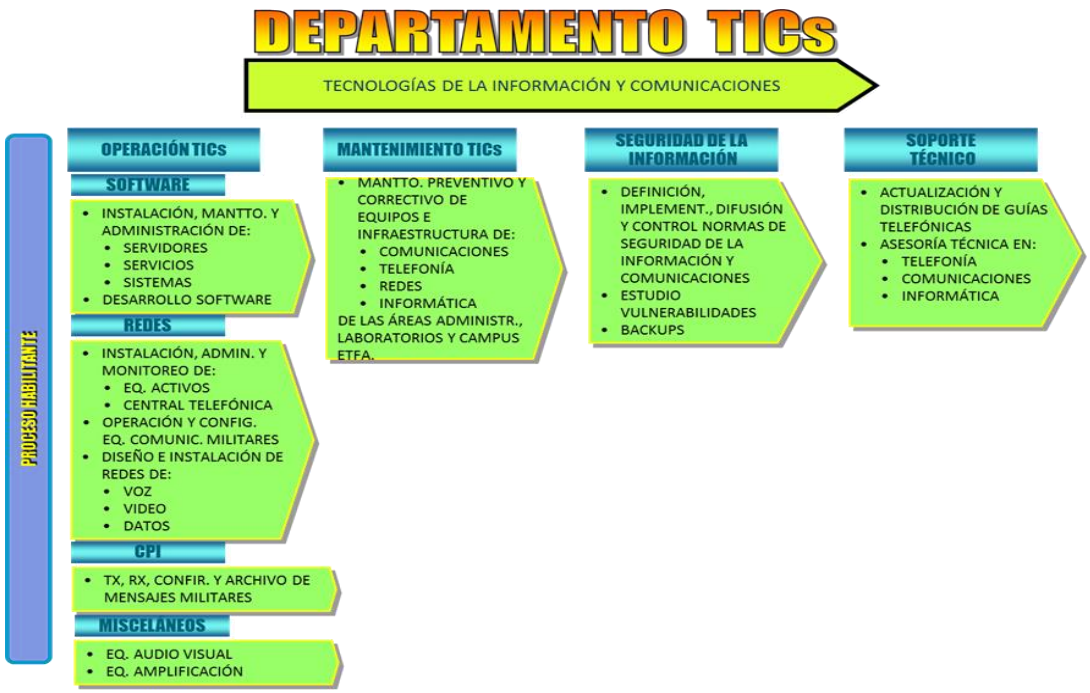


Figura 8. Procesos Habilitantes del Departamento TIC

Fuente: (Archivo Departamento TIC ETFA)

Todas estas actividades que se realiza dentro del Departamento TIC van orientados hacia la consecución de los objetivos instituciones y apegados al cumplimiento de la misión que se le ha encomendado a la Escuela Técnica de la Fuerza Aérea.

Para el desarrollo de productos se enfoca principalmente en la producción de software a medida, ofreciendo un servicio netamente público al 100%, por ser una entidad estatal, atendiendo los requerimientos para todos los departamentos de la ETFA. Opera con aproximadamente el 25% en el Departamento Académico, el 20% en el Departamento Evaluación, el 20% en el Departamento Instrucción Militar, el 10% para el Departamento Personal, el 5% Compras Públicas, el 5% para Infraestructura, el 5% en el Sistema Integrado de Seguridad y Comunicación Social con software personal y, el 10% en el Departamento Finanzas con software contable; dando prioridad, atención y entregando un producto software de calidad para las actividades en pro de la formación de los Aspirantes a Tropa.

Para la justificación del problema se ha elaborado un instrumento de investigación (Encuesta) el mismo que se ha aplicado a los 5 miembros del equipo de trabajo del Departamento TIC de la ETFA, de los cuales se tiene los siguientes resultados, tomando en cuenta preguntas con el fin de ser necesarias para la investigación:

1. ¿Se han presentado problemas en el proceso de desarrollado software en el Departamento TIC de la ETFA anteriormente?

Si	4
No	1
Ocasionalmente	0

Resultado: Se obtuvo como resultado un 80% de respuestas que confirman la presencia de problemas en el proceso de desarrollo software y un 20% de respuestas negativas. Por lo que se confirma la existencia de problemas.

2. ¿Se han presentado retrasos en la entrega del sistema software?

Si	5
No	0
Ocasionalmente	0

Resultado: Se obtuvo como resultado un 100% de respuestas que confirman la presencia de retrasos en la entrega del sistema software.

3. ¿Ha existido algún tipo de inconformidad de los usuarios con el producto software entregado por el Departamento TIC?

Si	1
No	2
Ocasionalmente	2

Resultado: Se obtuvo como resultado un 20% de respuestas que confirman la inconformidad de los usuarios del producto entregado, un 40% de respuestas negativas y un restante de 40% que ocasionalmente han presenciado algún tipo de inconformidad. Sumando las respuestas ocasionales a las afirmativas podemos concluir que se suma a la problemática la inconformidad de los usuarios con el software entregado.

4. ¿Se ha dado sobrecarga de trabajo en el equipo desarrollador de software?

Si	3
No	2
Ocasionalmente	0

Resultado: Se obtuvo como resultado un 60% de respuestas que aseguran la sobrecarga de trabajo a lo que son sometidos y el 40% de respuestas mencionan que no existe sobrecarga. Por lo que se confirma la sobrecarga de trabajo del equipo.

5. ¿El Departamento TIC cuenta con el personal y recursos necesarios para las actividades de desarrollo del sistema software?

Si	1
No	4
Ocasionalmente	0

Resultado: Se obtuvo como resultado un 80% de respuestas que aseguran no disponer con el personal y recursos para el desarrollo, y un 20% de respuestas afirmativas que mencionan disponerlos. Por lo que se confirma la problemática para el normal desenvolvimiento de las actividades de desarrollo del sistema software.

6. ¿Se ha establecido o estructurado un Marco de Trabajo basado en Metodologías de Desarrollo Software para el proceso Operación TIC´s de la actividad Desarrollo de Software?

Si	1
No	4
Ocasionalmente	0

Resultado: Se obtuvo como resultado un 80% de respuestas que aseguran no disponer de un Marco de Trabajo, frente a un 20% de respuestas que afirman disponerlo. Mediante el resultado de respuestas negativas se ratifica la inexistencia de algún Marco de Trabajo basado en Metodologías de Desarrollo Software.

7. ¿Existen estándares, normas o procedimientos reglamentarios para desarrollo del sistema software?

Si	1
No	3
Ocasionalmente	1

Resultado: Se obtuvo como resultado un 60% de respuestas que niegan la existencia estándares, normas o procedimientos reglamentarios, un 20% de respuestas afirmativas y un restante de 20% que han presenciado ocasionalmente la existencia. Sumando las respuestas ocasionales a las negativas podemos concluir que no existen estándares, normas o procedimientos reglamentarios para desarrollo del sistema software.

8. ¿Cree conveniente el uso de métodos y/o metodologías, que mejor se adapten a los requerimientos o necesidades institucionales, para el desarrollo de sistemas software?

Si	5
No	0
Ocasionalmente	0

Resultado: Se obtuvo como resultado un 100 % de respuestas que confirman la necesidad de establecer el uso de métodos y/o metodologías, que mejor se adapten a los requerimientos o necesidades institucionales, para el desarrollo de sistemas software.

El instrumento (Encuesta) aplicado al equipo de trabajo del Departamento TIC de la ETFA, puede ser apreciada en el ANEXO A.

Según los resultados arrojados por el instrumento, aplicado al equipo de trabajo del Departamento TIC, el 80% del personal que labora afirma que existen dificultades que han ido ocurriendo durante el proceso de desarrollo software, demostrando así la existencia de dificultad que se produce en dicho proceso, entre las que destaca: Retrasos en las entregas de software, insatisfacción del cliente, sobrecarga de trabajo,

falta de personal y recursos necesarios, carencia de un marco de trabajo, estándares, métodos y/o metodologías a seguir, entre otros.

Los resultados también arrojaron que existe la necesidad de tener un Marco de Trabajo basado en Metodologías de Desarrollo Software que guíe eficientemente el proceso de ingeniería del software, hecho demostrado con el 100% del personal de acuerdo con la creación del Marco de Trabajo a través del uso de métodos y/o metodologías que mejor se adapten a la realidad institucional, apoyados con estándares, normas y/o procedimientos reglamentarios preestablecidos; justificando de ésta manera la necesidad de implementar una Metodología de Desarrollo Software propuesta, con el fin de optimizar el análisis, diseño e implementación del sistema software, y así mejorar la calidad del producto que el Departamento TIC genera para la ETFA.

2.5 Conclusión del Capítulo

A lo largo de la evolución cronológica que se pudo apreciar en éste capítulo, se puede determinar que no existe una metodología universal para hacer frente con éxito a cualquier proyecto de desarrollo de software. Toda metodología debe ser adaptada al contexto del proyecto (recursos técnicos y humano, tiempo de desarrollo, tipo de sistema, etc.). Históricamente, las metodologías tradicionales han intentado abordar la mayor cantidad de situaciones de contexto del proyecto, exigiendo un esfuerzo considerable para ser adaptadas, sobre todo en proyectos pequeños y con requisitos muy cambiantes. Las metodologías ágiles ofrecen una solución casi a medida para una gran cantidad de proyectos que tienen estas características. Una de las cualidades más destacables en una metodología ágil es su sencillez, tanto en su aprendizaje como en su aplicación, reduciéndose así los costes de implantación en un equipo de desarrollo. Esto ha llevado hacia un interés creciente en las metodologías ágiles. Sin embargo, hay que tener presente una serie de inconvenientes y restricciones para su aplicación, tales como: están dirigidas a equipos pequeños o medianos, el entorno físico debe ser un ambiente que permita la comunicación y colaboración entre todos los miembros del equipo durante todo el tiempo, cualquier resistencia del cliente o del equipo de desarrollo hacia las prácticas y principios puede llevar al proceso al fracaso, el uso de tecnologías que no tengan un ciclo rápido de realimentación o que no soporten fácilmente el cambio.

CAPÍTULO 3
DETERMINACIÓN DE LA METODOLOGÍA ADAPTABLE AL DEPARTAMENTO TIC

3.1 Introducción del Capítulo

En el desarrollo del presente capítulo se hará el estudio de comparación de diversas metodologías ágiles, las mismas que luego de la evaluación respectiva determinarán las características que satisfagan las necesidades intrínsecas del Departamento TIC de la ETFA para el proceso de desarrollo del sistema software, y que servirá de base fundamental de posteriores desarrollos de sistemas. Es necesario mencionar que el Departamento desarrolla software según requerimientos y exigencias de la ETFA, pero la carencia de una esquematización basado en normas, estándares, métodos o metodologías, ha mermado una brecha objetiva entre lo que el Departamento desarrollo y el usuario del sistema, ocasionando la inconformidad del mismo. Por tal razón, la implementación de una metodología o un marco de trabajo basado en metodologías, resulta imprescindible para lograr un sistema software de calidad y por ende el mejoramiento de la gestión de desarrollo del software.

A principios de la década de los 90, surgió un enfoque que fue bastante revolucionario para su momento ya que iba en contra de la creencia de que mediante procesos altamente definidos se iba a lograr obtener software en tiempo, costo y con la requerida calidad. El enfoque fue planteado por primera vez por japoneses y se dio a conocer en la comunidad de ingeniería de software.

"En febrero de 2001, tras una reunión celebrada en los Estados Unidos, nace el término "ágil" el cual se aplica al desarrollo de software. Su objetivo fue esforzar los valores y principios 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.

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 fue el Manifiesto Ágil, un documento que resume la filosofía "ágil"[1].

A través del manifiesto surgen las Metodologías Ágiles, las cuales a continuación mencionaremos las características que vinculan con la realidad del Departamento TIC, acoplándose ágilmente al mencionar situaciones puntuales como la optimización de recursos, tiempos, costos y básicamente, la orientación a grupos multidisciplinarios.

3.2 Selección de las Metodologías a ser investigadas

3.2.1 El Manifiesto Ágil

Como ya se había argumentado en la introducción, a principio del año 2001, luego de celebrarle una reunión en Utah-EEUU, nace el término ágil, aplicado al desarrollo de software.

Participaron en esta reunión un seleccionado grupo de 17 expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías del software. Con un objetivo específico de esbozar los valores y principios que posibilitaran a los equipos desarrollar software rápidamente y respondiendo a su vez, a los cambios que puedan surgir a lo largo del proyecto que se estuviera desarrollando.

En esta reunión se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, debido a que estos se caracterizaban por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades que se desarrollaban.

Luego de desarrollarse esta reunión se creó "The Agile Alliance"[2], una organización fundada sin ánimos de lucro, que se dedicaba a promover conceptos relacionados con el desarrollo ágil de software y también a impulsar dichas organizaciones para que adopten dichos conceptos. Donde se utilizó como punto de partida el Manifiesto Ágil, documento que resume la filosofía ágil.

"Según este Manifiesto, se valora:

Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas: Considerando a la gente como el principal factor de éxito de un proyecto software. Como primicia se destaca que es más importante construir un buen equipo

[1] Las Metodologías de desarrollo ágil como una oportunidad para la ingeniería del software educativo, (2008).

[2] www.agilealliance.com

que construir el entorno. Muchas veces se comete el error de hacer lo contrario; se construye primero el entorno y se espera que el equipo se adapte automáticamente. Evidentemente, es satisfactorio crear el equipo primero y que luego éste configure su propio entorno de desarrollo en base a sus necesidades.

Desarrollar software que funciona más que conseguir una buena documentación: Se impone como regla general a seguir, no producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante. Acentuar que estos documentos deben ser cortos y centrarse en lo fundamental.

La colaboración con el cliente más que la negociación de un contrato: Se propone que debe existir una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto, y asegure su éxito.

Responder a los cambios más que seguir estrictamente un plan: La habilidad de responder a los cambios que puedan surgir en el transcurso del proyecto; entre ellos: cambios en los requisitos, en la tecnología, en el equipo, etc., determina también el éxito o fracaso de este. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta"[3].

Estos valores mencionados, fueron los que inspiraron los doce principios del manifiesto. A demás, son características que trazan la diferencia entre un proceso ágil y uno tradicional. Se puede argumentar que los dos primeros principios son generales y resumen gran parte del espíritu ágil, mientras que el resto tienen que ver con el proceso a seguir y con el equipo de desarrollo, en cuanto metas a seguir y organización del mismo.

"Son 12 los principios a los que nos referimos y se enumeran a continuación:

I. La principal prioridad es satisfacer al cliente a través de tempranas y continuas entregas de software que le aporten un valor.

II. Dar la bienvenida a los cambios o recibirlos con satisfacción. De esta manera se capturan los cambios para que el cliente tenga una ventaja competitiva.

III. Mantener la entrega frecuente de software que funcione desde un par de semanas hasta un par de meses, con el menor intervalo de tiempo posible entre las entregas.

[3]http://noqualityinside.com.ar/nqi/nqifiles/XP_Agil.pdf

IV. La gente del negocio o comerciantes y los desarrolladores o clientes deben trabajar juntos a lo largo del proyecto.

V. Construir el proyecto en torno a individuos motivados. Darles a estos, el entorno y el apoyo que necesitan y confiar en ellos en todos los aspectos para conseguir finalizar el trabajo o en este caso el proyecto.

VI. El diálogo cara a cara es evidentemente el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.

VII. El software que funciona es acertadamente la medida principal de progreso.

VIII. Los procesos ágiles promueven y evidencian un desarrollo sostenible. Los promotores, desarrolladores y usuarios o clientes deberían ser capaces de mantener una paz constante.

IX. La atención continua a la calidad técnica y al buen diseño mejora la agilidad en gran medida.

X. La simplicidad es esencial.

XI. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.

XII. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento"[4].

3.2.2 Estudio de las Metodologías Ágiles más relevantes

A fin de garantizar el estudio selectivo y que sea válido y justificable, se dispondrá de la información necesaria de las metodologías con las que se trabajará y que fuesen lo suficientemente relevantes.

El objetivo de este apartado es seleccionar un total de cinco metodologías ágiles, que posteriormente serán caracterizadas y sobre las cuales realizaremos seguidamente una comparativa. Las metodologías que hemos considerado a la hora de realizar la selección han sido:

- Adaptative Software Development
- Agile Modeling
- Agile Model Driven Development
- Agile Project Management

[4]http://noqualityinside.com.ar/nqi/nqifiles/XP_Agil.pdf

- Agile Unified Process
- Crystal Methods
- Dynamic Systems development methods
- Feature driven development
- Internet Speed Development
- Lean development
- Pragmatic programming
- Scrum
- Test Driven Development
- XBreed
- Extreme Programming
- WinWinSpiral.

Todas estas metodologías están documentadas o referenciadas como metodologías ágiles en las siguientes referencias [56][57][58][59][60][61].

No son todas las metodologías ágiles que existen, pero sí son una gran representación de ellas. A continuación mostramos algunas metodologías ágiles que no hemos incluido en la preselección:

- Evolutionary Project Management.
- Story cards driven development.
- Agile Unified Process
- Open Unified Process

Tal y como hemos mencionado anteriormente, nos interesa trabajar con metodologías lo suficientemente documentadas, que nos faciliten la obtención de información, pero también es interesante trabajar con metodologías que dispongan de algún tipo de certificación y training. Según estas condiciones hemos determinado seis clasificaciones donde escogeremos a las seis metodologías que se encuentran mejor posicionadas; estas son las clasificaciones:

i. La metodología con mayor presencia en Internet.
ii. La metodología mejor documentada.
iii. Metodologías certificadas y con training.
iv. Metodologías con comunidades.
v. Metodología más utilizada por empresas. Presencia empresarial.

vi. Metodología más utilizada en proyectos software.

En la Tabla 2 se muestra algunas consideraciones a la hora de realizar estas clasificaciones y en la Tabla 3, los resultados de los datos considerados para elaborar las listas.

Tabla 2. Criterios para valorar metodologías ágiles

Libros	Safari Books, Google Books, Amazon, Abacus, Díaz de Santos, CCUC
Certificación y Training	Se han considerado metodologías certificadas aquellas que emiten un certificado que aseguran el cumplimiento y seguimiento de la metodología, así como sus técnicas y prácticas. Indicamos que una metodología dispone de training, si hemos encontrado alguna institución, organización o compañía que ofrezca formación de la metodología. *XP está especialmente bien documentada con muchos recursos online disponibles, comunidades libres y grupos de noticias.
Comunidades	Contemplamos tanto si ha formado una comunidad relevante o si está asociada a la Agile Alliance, soportándola y cumpliendo sus principios. No se incluyen grupos de noticias o de correo, aunque sería una propuesta interesante a tener en cuenta.
Proyectos Realizados	La mayoría de metodologías se han aplicado en empresas privadas y por lo tanto no existe mucha documentación al respecto, no se ha realizado una búsqueda exhaustiva, ya que no es el propósito de este proyecto.

Fuente: (José Carvajal, Metodologías Ágiles) [55]

Tabla 3. Resultados de los criterios para valorar metodologías ágiles

Metodologías	Nº Papers	Google	Yahoo	Live	Libros en español	Libros en otros Idiomas	Certificación/ Training	Comunidades (Alliances)	Presencia empresarial	Proyectos realizados
Adaptative Software Development (ASD)	2	15300	46100	23100	0	1	No	Agile alliance	-	-
Agile Modeling (AM)	6	57200	203000	538000	0	1	Training	Agile alliance	-	-
Agile Model Driven Development (AMDD)	2	10200	28400	83000	0	1	Training	Agile alliance	-	-
Agile Project Management (APM)	8	170000	766000	311000	0	1	Training	Declaration of Interdependence for modern management	-	-
Agile Unified Process (AUP)	0	11000	19700	8940	0	0	No	-	-	-
Crystal Methods	0	344000	2930000	724000	0	1	Training	Agile alliance	-	Proyecto Winifred
Dynamic Systems Development Method (DSDM)	70	16900	41200	24200	0	6	DSDM Certified Training	DSDM Consortium	-	-
Feature Driven Development (FDD)	3	31200	177000	68000	0	1	FDD Certified Training	Agile alliance	-	-
Internet Speed Development (ISD)	0	74	326	127	0	0	No	Agile alliance	-	-
Lean Development	2	16300	6890	16100	0	0	Training	Agile alliance	-	-
Pragmatic Programming	0	346	1340	648	0	0	No	Agile alliance	-	-
Scrum	43	3420000	5120000	1970000	2	4	Scrum Certified Trainning	Agile alliance	Yahoo, Google, etc ...	Desarrollos internos principalmente
Test Driven Development	55	492000	2800000	1040000	0	9	No	Scrum Alliance Agile Alliance	-	-
XBreed (Agile Enterprise)	0	601	1450	624	0	0	No	Agile Alliance	-	-
Extreme Programming (XP)	+100	1180000	4470000	1470000	2	+20	Training*	-	Chrysler, Sabre Airlines, CSEE Transport, etc...	Control automatizado de tareas *
Win Win Spiral	3	1700	1640	447	0	0	No	Agile Alliance	-	-

En la Tabla 3 que mostramos en la página anterior podemos ver los resultados de los criterios escogidos. Estos son los resultados extraídos del estudio previo, según los criterios establecidos anteriormente:

i. La metodología con mayor presencia en Internet

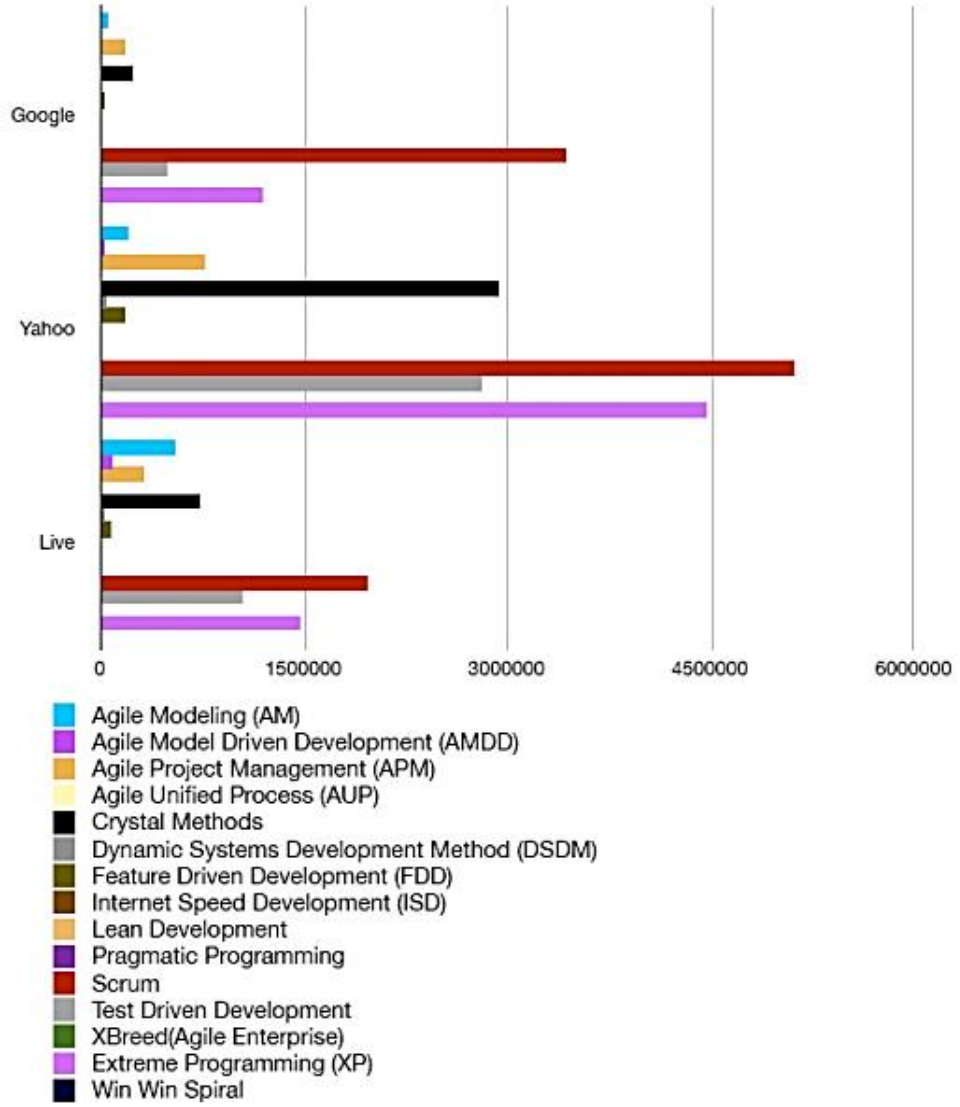


Figura 9. Mayor Presencia en Internet de las Metodologías Ágiles

Fuente: (José Carvajal, Metodologías Ágiles) [55]

Según el número de resultados obtenidos en las búsquedas por Yahoo, Google y Microsoft Live, las cinco metodologías con mayor presencia en la red y en este orden son (ver Figura 9):

1) Scrum
2) Extreme Programming (XP)
3) Feature Driven Development (FDD)
4) Crystal Methods
5) Adaptative Software Development (ASD)

ii. La metodología mejor documentada.

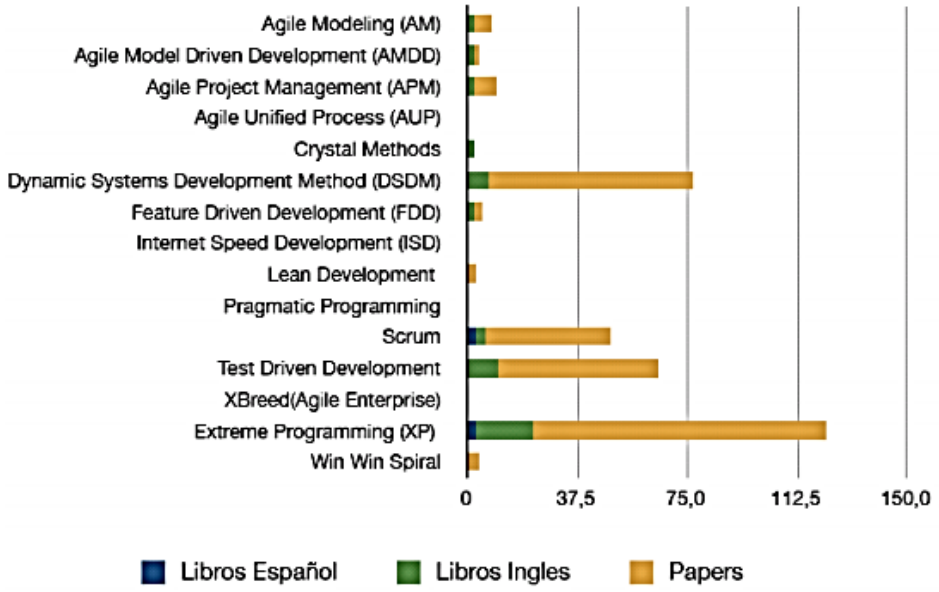


Figura 10. Mejor documentación de las Metodologías Ágiles

Fuente: (José Carvajal, Metodologías Ágiles) [55]

Las cinco metodologías mejor documentadas son (ver figura 3.2):

1) Extreme Programming (XP)
2) Feature Driven Development (FDD)
3) Dynamic System Development Method (DSDM)
4) Scrum
5) Adaptative Software Development (ASD)

iii. Metodologías certificadas y con training.

Metodologías con certificación:

- Dynamic System Development Method tieneun DSDM certified.
- Feature Driven Development tiene FDD Certified.
- Scrum dispone de ScrumCertified.

Metodologías con training:

- Agile Modeling.
- Agile Model Driven Development.
- Adaptative Software Development.
- Crystal methods.
- Dynamic System Development Method.

- Feature Driven Development.
- Lean Development.
- Scrum.
- Extreme Programing.

Con certificación y training:

- Todas las metodologías certificadas ofrecen training

iv. Metodologías con comunidades.

La mayoría pertenecen a la Agile Alliance, pero algunas han montado auténticas comunidades y alianzas a su alrededor.

Metodologías asociadas a la Agile Alliance:

- Agile Modeling.
- Agile Model Driven Development.
- Crystal methods.
- Dynamic System Development Method.
- Feature Driven Development.
- Internet Speed Development.
- Lean Development.
- Pragmatic Programming.
- Scrum.
- Test Driven Development.
- Extreme Programing.
- WinWinSpiral.

Metodologías con comunidades o alianzas diferentes:

- Adaptative Software Development, con Declaration of Interdependence for modern management.
- Dynamic System Development Method, con DSDM Consortium.
- Scrum, con Scrumalliance.

v. Metodología más utilizada por empresas. Presencia empresarial.

Como se pudo observar es realmente complicado encontrar ejemplos de proyectos realizados en una empresa privada y con una metodología en concreto. Por lo que los

resultados obtenidos en este apartado no se tienen en cuenta en la selección inicial de las metodologías (ver Tabla 3).

vi. Metodología más utilizada en proyectos software.

Exactamente igual que el punto anterior.

3.2.3 Selección de las metodologías para el estudio.

El objetivo inicial era seleccionar un máximo de cinco metodologías, pero en vista de los resultados estas han sido las metodologías seleccionadas:

1) Adaptative Software Development.
2) Crystal Methods.
3) Dynamic Sytems Development Method.
4) Scrum.
5) Feature Driven Development.
6) Extreme Programming.

La elección de estas metodologías para la comparativa se ha realizado a partir de los datos presentados en los apartados anteriores. Se han incluido las 5 metodologías mejor documentadas y las 5 con mayor presencia en Internet. Nos encontramos con que son 6 las metodologías que surgen de esta unión es por esta razón que se ha incluido una sexta metodología dentro del estudio.

3.3 Determinación de las características de cada una de las metodologías seleccionadas.

A continuación se resumen los seis métodos ágiles, seleccionados anteriormente, con sus definiciones, funciones y características, que además están siendo utilizados con éxito en proyectos reales:

Adaptive Software Development (ASD): Su impulsor Jim Highsmith. Tiene como principales características ser: iterativo, orientado a los componentes software más que a las tareas y tolerante a los cambios. El ciclo de vida que propone tiene tres fases esenciales: especulación, colaboración y aprendizaje.

- Especulación: En ésta se inicia el proyecto y se planifican las características del software.
- Colaboración: Se desarrollan las características del software.
- Aprendizaje: Se revisa su calidad, y se entrega al cliente. La revisión de los componentes sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo.

Crystal Methods: Desarrollado por Alistair Cockburn. Comprende un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas.

Crystal no especifica ningún ciclo de vida concreto [50], ni ningún modelo de procesos, en vez de eso lo que hace es determinar una guía de las políticas estándar, productos de trabajo, asuntos locales, herramientas, estándares y roles:

- Planificación por etapas: Básicamente consiste en la planificación del siguiente incremento del sistema. La planificación debe finalizar con una versión ejecutable cada tres o cuatro meses como máximo. El equipo selecciona los requisitos que serán implementados en el incremento y planifican lo que creen que serán capaces de hacer.
- Revisiones y resúmenes: Cada incremento consta de diferentes iteraciones y cada iteración incluye las actividades como construcción, demostración y resumen de los objetivos del incremento.
- Monitorización: Los progresos del proyecto son monitorizados a partir delas diferentes entregas del equipo durante el proceso de desarrollo. El progreso se mide con los hitos clave y la estabilidad de las fases (muy inestable, fluctúa y lo suficiente estable para revisar).
- Paralelismo y flujo: Cuando el monitor de estabilidad nos indica un estado lo suficientemente estable para su revisión, entonces es el momento para pasar a la siguiente tarea. Con tal de poder llevar esto a cabo, los equipos de seguimiento y arquitectura deben revisar sus planes de trabajo, su estabilidad y sincronización.

Feature Driven Development (FDD): Sus impulsores son Jeff De Luca y Peter Coad. Este define un proceso iterativo que consta de 5 pasos específicos.

- Desarrollo de un modelo global: Cuando comienza el desarrollo, los expertos del dominio están al tanto de la visión, el contexto y los requerimientos del sistema a construir.
- Construcción de una lista de funcionalidades: Se elabora una lista de funcionalidades que resuma la funcionalidad general del sistema. La lista es elaborada por los desarrolladores y es evaluada por el cliente.
- Planeación por funcionalidad: En este punto se procede a ordenar los conjuntos de funcionalidades conforme a su prioridad y dependencia, y se asigna a los programadores jefes.
- Diseño por funcionalidad: Es seleccionado un conjunto de funcionalidades de la lista y se procede a diseñar.
- Construcción por funcionalidad: Seguido del diseño, se construye la funcionalidad mediante un proceso iterativo. Una iteración puede tomar de unos pocos días a un máximo de dos semanas. El proceso iterativo incluye inspección de diseño, codificación, pruebas unitarias, integración e inspección de código.

Programación Extrema (XP): Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. A continuación se presenta las fases del ciclo de vida [51]:

- Exploración: En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto.
- Planificación de la Entrega (*Release*): En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los

programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente.

- Iteraciones: Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas. Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la Iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior.
- Producción: La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase.
- Mantenimiento: Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente.
- Muerte del Proyecto: Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura.

Método de desarrollo de sistemas dinámicos (DSDM): Define el marco para desarrollar un proceso de producción de software. Nace en 1994 con el objetivo de crear una metodología RAD unificada. Es un proceso iterativo e incremental y el equipo de desarrollo y el usuario trabajan juntos. Propone cuatro fases:

- Estudio viabilidad: Se definirá si la metodología se ajusta al proyecto a desarrollar.
- Estudio del negocio: Hace referencia el alcance funcional a automatizar.
- Modelado funcional: Se realizará el diseño funcional que tendrá el proyecto.
- Diseño y construcción, e implementación: Se realizará el diseño técnico y desarrollo del proyecto, las mismas que serán iterativas, además de existir realimentación a todas las fases.

Scrum: Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. A continuación se presenta las fases del ciclo de vida [52]:

- Pre-Juego: Planeamiento. El propósito es establecer la visión, definir expectativas y asegurarse la financiación. Las actividades son la escritura de la visión, el presupuesto, el registro de acumulación o retraso (backlog) del producto inicial y los ítems estimados, así como la arquitectura de alto nivel, el diseño exploratorio y los prototipos. El registro de acumulación es de alto nivel de abstracción.
- Pre-Juego: Montaje (Staging). El propósito es identificar más requerimientos y priorizar las tareas para la primera iteración. Las actividades son planificación, diseño exploratorio y prototipos.
- Juego o Desarrollo. El propósito es implementar un sistema listo para entrega en una serie de iteraciones de treinta días llamadas "corridas" (sprints). Las actividades son un encuentro de planeamiento de corridas en cada iteración, la definición del registro de acumulación de corridas y los estimados, y encuentros diarios de Scrum.
- Pos-Juego: Liberación. El propósito es el despliegue operacional. Las actividades, documentación, entrenamiento, mercadeo y venta.

Los métodos ágiles ofrecen una solución casi a medida para una gran cantidad de proyectos que tienen estas características. Una de las cualidades más destacables en un método ágil es su sencillez, tanto en su aprendizaje como en su aplicación, reduciéndose así los costos de implantación en un equipo de desarrollo.

3.4 Estudio comparativo de las metodologías seleccionadas a fin de determinar la metodología que satisface las necesidades del Departamento TIC de la ETFA

Para la selección e implantación de una metodología ágil existe una importante labor de documentación previa. Considerando el estudio realizado en el sub apartado 3.2.2, se seleccionó seis metodologías ágiles y, a partir de aquello, lo que se pretende es escoger la metodología más adaptable a las necesidades del Departamento TIC, de tal manera que sirva de base fundamental para su aplicación en el día a día.

En este caso, se ha dado la vuelta al proceso, de manera que conociendo un marco de trabajo determinado, lleguemos a una metodología de trabajo apropiada.

Para la determinación de la metodología apropiada, este apartado se ha dividido en cinco partes. La primera parte consta de un formulario para conocer y justificar la orientación del Departamento TIC: ágil o tradicional; la segunda tiene un formulario que permite identificar si el Departamento cumple los principios ágiles; la tercera parte consta de un formulario que estará en base a un Framework que determinará la metodología que mejor se adapta al marco de trabajo del Departamento; la cuarta parte que describe el estado actual de la metodología; y la quinta parte referente a la determinación final de la metodología ágil adaptable al Departamento TIC [54].

3.4.1 Primera Parte: Orientación del Departamento TIC de la ETFA

En esta fase se extraerá el enfoque del Departamento, bien un enfoque tradicional o un enfoque ágil. Si el Departamento está en un porcentaje alto de implantación de las directrices de las metodologías ágiles, se podría pasar a la segunda parte del formulario, mientras que si en el primer formulario se detecta que la cultura de trabajo es más cercana a una metodología tradicional, sería necesario conocer y adquirir las prácticas de una metodología ágil.

Primer Formulario: Orientación tradicional vs Orientación ágil

Para obtener este dato de forma objetiva, se analizará cada valor ágil y su relación con la organización.

Se han desglosado los valores del manifiesto ágil y se han dividido entre orientación ágil vs orientación tradicional, estos valores serán evaluados por la organización según una escala de importancia.

Valores de importancia:

0: Ninguna.

1: Baja importancia.

2: Media importancia.

3: Alta importancia.

Tabla 4. Orientación tradicional vs orientación ágil

ORIENTACIÓN ÁGIL		ORIENTACIÓN TRADICIONAL	
VALOR	**IMPORTANCIA**	**VALOR**	**IMPORTANCIA**
Individuo y las interacciones del equipo	x1	El proceso y las herramientas	y1
Desarrollar software que funciona	x2	Conseguir una buena documentación	y2
Colaboración con el Cliente	x3	Negociación contractual	y3
Respuesta al cambio	x4	Seguimiento de un plan	y4

Siendo x1, x2, x3 y x4 los valores asignados a cada valor con enfoque ágil, e y1, y2, y3 e y4 los valores asignados a cada valor con enfoque tradicional.

La escala mostrada anteriormente califica la importancia del atributo "valor del manifiesto", la misma que será evaluada por el equipo de desarrollo de software del Departamento TIC de acuerdo a la realidad en la que se desenvuelve a fin de verificar la tendencia hacia el manifiesto u orientación ágil o hacia una orientación tradicional; teniendo en consideración esto, se define:

- ✓ La importancia 0 (cero) calificará al valor del manifiesto sin significancia alguna o inexistencia para el equipo de desarrollo del Departamento;
- ✓ La importancia 1 (uno) calificará al valor del manifiesto con poca presencia o baja relevancia del atributo en el desenvolvimiento del equipo;
- ✓ La importancia 2 (dos) calificará al valor del manifiesto con una injerencia mediana del atributo o de normal aplicación en el desenvolvimiento del equipo, sin que comprometa una significancia baja o alta;
- ✓ Y la importancia 3 (tres) calificará al valor del manifiesto con extrema o alta importancia por ser de imprescindible naturaleza o constante aplicabilidad que resalta en el desenvolvimiento del equipo del Departamento.

Tabla 5. Resultado orientación tradicional vs orientación ágil del Departamento TIC de la ETFA

ORIENTACIÓN ÁGIL		ORIENTACIÓN TRADICIONAL	
VALOR	**IMPORTANCIA**	**VALOR**	**IMPORTANCIA**
Individuo y las interacciones del equipo	3	El proceso y las herramientas	2
Desarrollar software que funciona	3	Conseguir una buena documentación	2
Colaboración con el Cliente	2	Negociación contractual	2
Respuesta al cambio	3	Seguimiento de un plan	2
MEDIA	**2,75**		**2**

En este caso, de acuerdo a los resultados arrojados por la tabla 3.4 se demuestra que se sobrevalora lo indicado por los valores del manifiesto ágil, con una orientación ágil de media 2,75 mayor a una orientación tradicional de media 2.

3.4.2 Segunda Parte: Segundo Formulario de Cumplimiento principios ágiles

Se ha extraído a un formulario cada principio ágil y extraerá su relación con el Departamento TIC de acuerdo a la escala de prioridad.

Valores en prioridad:

0: Ninguna.
1: Baja prioridad.
2: Media prioridad.
3: Alta prioridad.

Tabla 6. Cumplimiento principios ágiles

	Principios del Manifiesto Ágil	**Prioridad**
1	La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.	
2	Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.	
3	Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.	
4	La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.	
5	Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo	
6	El diálogo cara a cara es el método más efectivo para comunicar información dentro de un equipo de desarrollo.	
7	El software que funciona es la medida principal de progreso.	
8	Los procesos ágiles promueven un desarrollo sostenible. Los promotores, los desarrolladores y usuarios deberían ser capaces de mantener una paz constante	
9	La atención continua a la calidad técnica y al buen diseño mejora la agilidad.	
10	La simplicidad es esencial.	
11	Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.	
12	En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.	
	TOTAL	∑ **prioridades asignadas**

Siendo los principios ágiles 12 y la prioridad más alta tiene un valor de 3, entonces el valor más alto en cuanto al cumplimiento de estos principios de 36.

Según el resultado obtenido Σ prioridades asignadas, se deduce que las metas según el enfoque del equipo de desarrollo del Departamento TIC se orientan en un **Σ prioridades asignadas * 100/36 %** al cumplimiento de los principios ágiles.

De igual manera, la escala mostrada anteriormente califica la importancia del atributo "principio del manifiesto ágil", la misma que será evaluada por el equipo de desarrollo de software del Departamento TIC de acuerdo a la prioridad en la que se desenvuelve a fin de verificar la tendencia hacia el manifiesto; teniendo en consideración esto, se define:

- ✓ La prioridad 0 (cero) calificará al manifiesto sin significancia alguna o inexistencia para el equipo de desarrollo del Departamento;

- ✓ La prioridad 1 (uno) calificará al manifiesto con poca o baja significancia en el desenvolvimiento del equipo;
- ✓ La prioridad 2 (dos) calificará el manifiesto con una injerencia mediana del atributo o de normal aplicación y significancia en el desenvolvimiento del equipo, sin que comprometa una significancia baja o alta;
- ✓ Y la prioridad 3 (tres) calificará el manifiesto con extrema o alta importancia por ser de imprescindible naturaleza o constante aplicabilidad que resalta en el desenvolvimiento del equipo del Departamento.

Departamento TIC

Indicadores de los principios ágiles en una organización según un desarrollador de la organización:

Tabla 7. Cumplimiento principios ágiles del Departamento TIC de la ETFA

	Principios del Manifiesto Ágil	**Prioridad**
1	La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.	2
2	Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.	3
3	Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.	2
4	La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.	2
5	Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo	3
6	El diálogo cara a cara es el método más efectivo para comunicar información dentro de un equipo de desarrollo.	3
7	El software que funciona es la medida principal de progreso.	3
8	Los procesos ágiles promueven un desarrollo sostenible. Los promotores, los desarrolladores y usuarios deberían ser capaces de mantener una paz constante	3
9	La atención continua a la calidad técnica y al buen diseño mejora la agilidad.	3
10	La simplicidad es esencial.	3
11	Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.	2
12	En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.	2
	TOTAL	**31**

Siendo los principios ágiles 12 y la prioridad más alta tiene un valor de 3, entonces el valor más alto en cuanto al cumplimiento de estos principios de 36.

Según el resultado obtenido de 31, se deduce que las metas según el enfoque del equipo de desarrollo del Departamento TIC se orientan se orientan en un **86,11%** al cumplimiento íntegro o total de los principios ágiles.

En el Departamento se han obtenido datos objetivos que indican que la organización tiene un enfoque ágil, por tanto, el siguiente paso sería conocer qué metodología ágil, de entre las que se detallaron anteriormente, se adapta mejor a la organización.

3.4.3 Tercera Parte: Framework para elección de la metodología ágil

En este apartado se evaluará la forma de trabajo del Departamento TIC basándose en los cuatro puntos de vista de Iacovelli[5]. Para ello, se ha elaborado un nuevo formulario agrupando estos cuatro puntos: Uso, capacidad de agilidad, aplicación, procesos y productos [53]. Cada uno de ellos con sus respectos atributos, cuyos valores serán asignados por el Departamento en evaluación.

El objetivo del estudio realizado por Iacovelli: "Framework para la clasificación de metodologías ágiles", es clasificar los métodos a través de cuatro puntos de vista, cada uno representando un aspecto de las metodologías. Cada punto de vista se caracteriza por un conjunto de atributos.

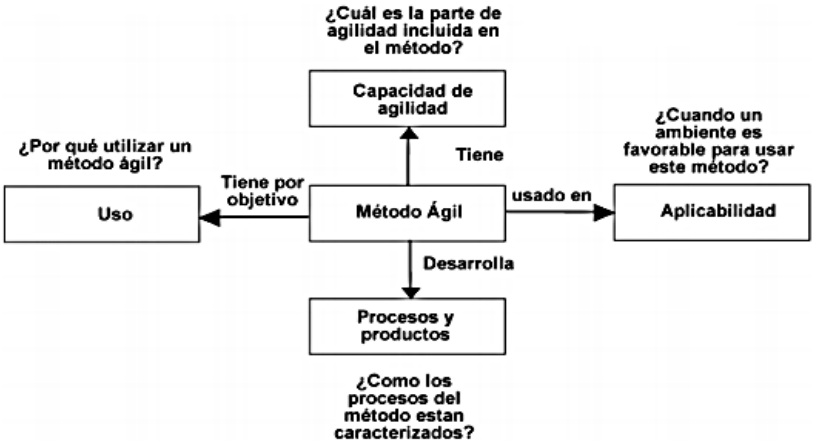


Figura 11. Cuatro vistas de las metodologías ágiles de Iacovelli

Fuente: (Iacovelli, Introducción Ágil a los Métodos 2007)

[5]Adrian Iacovelli, autor de "Framework for Agile Methods Classification".

a. Uso

Refleja por qué utilizar metodologías ágiles. Los atributos de esta vista tratan de evaluar todos los beneficios de que el equipo de desarrollo y el cliente obtienen utilizando este tipo de metodologías: incremento de la productividad, calidad y satisfacción.

Las metodologías ágiles integran los cambios en el proceso de desarrollo, aportan reglas y directrices para trabajar en proyectos con requisitos cambiantes manteniendo fechas de entrega. Las metodologías ágiles aportan flexibilidad.

Los atributos de este punto de vista son:

- Adaptarse a los entornos turbulentos.
- Satisfacción del usuario final.
- Favorable al offshoring (outsourcing internacional).
- Aumento de la productividad.
- El respeto de un nivel de calidad.
- El respeto de las fechas de entrega.
- Cumplimiento de los requisitos.

b. Capacidad de Agilidad

Representa cuál es la parte ágil de la metodología. Los atributos de esta vista representan todos los aspectos del concepto de agilidad y su evaluación refleja que aspectos están incluidos en una metodología.

Una metodología de desarrollo de software está compuesta por un ciclo de vida. En Ingeniería del Software existen diferentes ciclos de vida, como por ejemplo modelo en V, modelo en espiral, etc. De acuerdo a la cronología de aparición de las metodologías ágiles, la mayoría de las metodologías derivan directamente del modelo en espiral. Esto se explica ya que las dos principales características de modelo en espiral son un ciclo de vida iterativo e incremental. Por tanto, los cambios de requisitos se pueden ir integrando en cada iteración, de manera que el plan de trabajo no tiene que ser modificado, irá cambiando a lo largo de las iteraciones. Otro punto interesante es la duración de las iteraciones, con iteraciones cortas aumenta el número de reuniones con el cliente para definir y detallar sus necesidades de forma incremental.

En cuanto a la interacción de las personas, las metodologías ágiles tienden a romper las relaciones contractuales entre los clientes y los equipos de desarrollo. Esta relación se expresa en este punto de vista por el atributo de colaboración. Un equipo ágil es un tipo de organización holográfica en la que cada miembro tiene el conocimiento del sistema en su conjunto, así que, si un miembro deja el equipo, no se ha perdido conocimiento.

El principal concepto de la agilidad son los procesos ligeros. Generalmente, las metodologías ágiles incluyen menos documentación. Las pruebas son una práctica muy importante, así como la refactorización.

Los atributos de este punto de vista son:

- Indicadores de cambio.
- Colaboración.
- Los requisitos funcionales pueden cambiar.
- Los recursos humanos pueden cambiar.
- Integración de los cambios.
- Nivel de intercambio de conocimientos (baja, alta).
- De peso ligero.
- Requisito no funcional puede cambiar.
- Centrado en las personas.
- Reactividad (al comienzo del proyecto, cada etapa, cada iteración).
- Refactoring político.
- Iteraciones cortas.
- Pruebas de política.
- Plan de trabajo se puede cambiar.

c. Aplicabilidad

El objetivo de esta vista es mostrar el impacto de los aspectos ambientales en el método. Representa cuando el entorno es favorable para la aplicación de metodologías ágiles. Este aspecto se describe por atributos, cada uno correspondiente a una característica del entorno.

Los atributos de este punto de vista son:

- Grado de interacción entre los miembros del equipo (baja, alta).
- El grado de interacción con el cliente (baja, alta).

- Grado de interacción con los usuarios finales (baja, alta).
- Grado de integración de la novedad (baja, alta).
- La complejidad del proyecto (baja, alta).
- Los riesgos del proyecto (baja, alta).
- Tamaño del proyecto (pequeño, grande).
- La organización del equipo (auto-organización, la organización jerárquica).
- El tamaño del equipo (pequeño, grande).

d. Procesos y Productos

La vista de los procesos y productos representa cómo se caracteriza la metodología. Los atributos caracterizarán a los procesos ágiles por dos dimensiones y listarán los productos de las actividades del proceso.

El proceso se compone de dos dimensiones. La primera dimensión son las actividades de desarrollo de software cubiertas por las metodologías ágiles. La segunda representa el nivel de abstracción de sus directrices y reglas. Estas dos dimensiones se evalúan con atributos de esta vista.

Los atributos de los procesos y los productos son:

- Nivel de abstracción de las normas y directrices:
    - Gestión de proyectos
    - Descripción de procesos
    - Normas y orientaciones concretas sobre las actividades y productos
- Las actividades cubiertas por el método ágil:
    - Puesta en marcha del proyecto
    - Definición de requisitos
    - Modelado
    - Código
    - Pruebas unitarias
    - Pruebas de integración
    - Prueba del sistema
    - Prueba de aceptación
    - Control de calidad
    - Sistema de uso
- Productos de las actividades del método:

- Modelos de diseño
- Comentario del código fuente
- Ejecutable
- Pruebas unitarias
- Pruebas de integración
- Pruebas de sistema
- Pruebas de aceptación
- Informes de calidad
- Documentación de usuario

Tercer Formulario para Elección de una metodología ágil mediante Framework

a. Uso

Verdadero (V) o Falso (F) en cada una de las premisas.

Tabla 7. Valoración del Uso

	Premisa	**Respuesta**
1	Respeto de las fechas de entrega	
2	Cumplimiento de los requisitos	
3	Respeto al nivel de calidad	
4	Satisfacción del usuario final	
5	Entornos turbulentos	
6	Favorable al Off shoring	
7	Aumento de la productividad	

b. Capacidad de Agilidad

Verdadero (V) o Falso (F) en cada una de las premisas.

Tabla 8. Valoración de la Capacidad de agilidad

	Premisa	**Respuesta**
1	Iteraciones cortas	
2	Colaboración	
3	Centrado en las personas	
4	Refactoring político	
5	Prueba político	
6	Integración de los cambios	
7	De peso ligero	
8	Los requisitos funcionales pueden cambiar	

9	Los requisitos no funcionales pueden cambiar	
10	El plan de trabajo puede cambiar	
11	Los recursos humanos pueden cambiar	
12	Cambiar los indicadores	
13	Reactividad (AL COMIENZO DEL PROYECTO, CADA ETAPA, CADA ITERACIÓN)	
14	Intercambio de conocimientos (BAJO, ALTO)	

c. Aplicación

Verdadero (V) o Falso (F) en cada una de las premisas.

Tabla 9. Valoración de la Aplicación

	Premisa	**Respuesta**
1	Tamaño del proyecto (PEQUEÑO, GRANDE)	
2	La complejidad del proyecto (BAJA, ALTA)	
3	Los riesgos del proyecto (BAJO, ALTO)	
4	El tamaño del equipo (PEQUEÑO, GRANDE)	
5	El grado de interacción con el cliente (BAJA, ALTA)	
6	Grado de interacción con los usuarios finales (BAJA, ALTA)	
7	Grado de interacción entre los miembros del equipo (BAJA, ALTA)	
8	Grado de integración de la novedad (BAJA, ALTA)	
9	La organización del equipo (AUTO-ORGANIZACIÓN,ORGANIZACIÓNJERÁRQUICA)	

d. Procesos y Productos

Verdadero (V) o Falso (F) en cada una de las premisas.

Nivel de abstracción de las normas y directrices:

Tabla 10. Valoración normas y directrices ágiles

	Premisa	**Respuesta**
1	Gestión de proyectos	
2	Descripción de procesos	
3	Normas y orientaciones concretas sobre las actividades y productos	

Las actividades cubiertas por el método ágil:

Tabla 11. Valoración actividades cubiertas por el método ágil

	Premisa	**Respuesta**
1	Puesta en marcha del proyecto	
2	Definición de requisitos	
3	Modelado	
4	Código	

5	Pruebas unitarias	
6	Pruebas de integración	
7	Prueba del sistema	
8	Prueba de aceptación	
9	Control de calidad	
10	Sistema de uso	

Productos de las actividades del método:

Tabla 12. Valoración de los productos de las actividades de la metodología ágil

	Premisa	**Respuesta**
1	Modelos de diseño	
2	Comentario del código fuente	
3	Ejecutable	
4	Pruebas unitarias	
5	Pruebas de integración	
6	Pruebas de sistema	
7	Pruebas de aceptación	
8	Informes de calidad	
9	Documentación de usuario	

Los datos extraídos de este formulario se compararán con la clasificación de las diferentes metodologías que se muestra a continuación.

Tabla 13. Clasificación de las Metodologías Ágiles por Iacovelli

			METODOLOGÍAS ÁGILES					
			ASD	**CRISTAL**	**DSDM**	**XP**	**FDD**	**SCRUM**
USO	¿Por qué utilizar un método ágil?	Respeto de las fechas de entrega	VERDADERO	VERDADERO	VERDADERO	FALSO	VERDADERO	VERDADERO
		Cumplimiento de los requisitos	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Respeto al nivel de calidad	VERDADERO	FALSO	FALSO	FALSO	FALSO	FALSO
		Satisfacción del usuariofinal	FALSO	FALSO	VERDADERO	FALSO	FALSO	FALSO
		Entornos conflictivos	FALSO	FALSO	VERDADERO	VERDADERO	FALSO	VERDADERO
		Favorable al Off shoring	VERDADERO	VERDADERO	VERDADERO	FALSO	FALSO	VERDADERO
		Aumento de la productividad	FALSO	FALSO	FALSO	VERDADERO	FALSO	FALSO
CAPACIDAD DE AGILIDAD	¿Cuál es la parte de agilidad incluida en el método?	Iteraciones cortas	FALSO	FALSO	FALSO	VERDADERO	VERDADERO	VERDADERO
		Colaboración	VERDADERO	VERDADERO	VERDADERO	VERDADERO	FALSO	FALSO
		Centrado en las personas	VERDADERO	VERDADERO	VERDADERO	VERDADERO	FALSO	FALSO
		Refactoring político	FALSO	FALSO	VERDADERO	VERDADERO	FALSO	FALSO
		Prueba político	VERDADERO	FALSO	VERDADERO	VERDADERO	FALSO	VERDADERO
		Integración de los cambios	VERDADERO	FALSO	VERDADERO	VERDADERO	FALSO	VERDADERO
		De peso ligero	FALSO	FALSO	FALSO	VERDADERO	VERDADERO	VERDADERO
		Los requisitos funcionales pueden cambiar	VERDADERO	FALSO	VERDADERO	VERDADERO	FALSO	VERDADERO
		Los requisitos no Funcionales pueden cambiar	VERDADERO	FALSO	FALSO	FALSO	FALSO	FALSO
		El plan de trabajo puede cambiar	FALSO	FALSO	FALSO	VERDADERO	FALSO	FALSO

		Los recursos humanos pueden cambiar	VERDADERO	VERDADERO	FALSO	VERDADERO	FALSO	FALSO
		Cambiar los indicadores	FALSO	FALSO	FALSO	VERDADERO	FALSO	FALSO
		Reactividad (AL COMIENZO DEL PROYECTO, CADA ETAPA, CADA ITERACIÓN)	CADA ETAPA	CADA ETAPA	CADA ITERACIÓN	CADA ITERACIÓN	AL COMIENZO DEL PROYECTO	CADA ETAPA
		Intercambio de conocimientos (BAJO, ALTO)	ALTO	ALTO	BAJO	ALTO	BAJO	BAJO
APLICABILIDAD	¿Cuándo un ambiente es favorable para usar este método?	Tamaño del proyecto (PEQUEÑO, GRANDE)	GRANDE	GRANDE	GRANDE	PEQUEÑO	GRANDE	GRANDE
		La complejidad del proyecto (BAJA, ALTA)	ALTA	ALTA	ALTA	BAJA	ALTA	ALTA
		Los riesgos del proyecto (BAJO, ALTO)	ALTO	ALTO	ALTO	BAJO	ALTO	ALTO
		El tamaño del equipo (PEQUEÑO, GRANDE)	GRANDE	PEQUEÑO	GRANDE	PEQUEÑO	GRANDE	GRANDE
		El grado de interacción con el cliente (BAJA, ALTA)	ALTA	BAJA	ALTA	ALTA	BAJA	BAJA
		Grado de interacción con los usuarios finales (BAJA, ALTA)	BAJA	BAJA	ALTA	BAJA	BAJA	BAJA
		Grado de interacción entre los miembros del equipo (BAJA, ALTA)	BAJA	ALTA	ALTA	ALTA	BAJA	BAJA
		Grado de integración de la novedad (BAJA, ALTA)	ALTA	BAJA	ALTA	ALTA	BAJA	BAJA
		La organización del equipo(AUTO-ORGANIZACIÓN, ORGANIZACIÓN JERÁRQUICA)	ORGANIZACIÓN JERÁRQUICA	AUTO-ORGANIACIÓN	ORGANIZACIÓN JERÁRQUICA	AUTO-ORGANIACIÓN	ORGANIZACIÓN JERÁRQUICA	AUTO-ORGANIACIÓN
PROCESOS Y PRODUCTOS	¿Cómo están Caracterizados los procesos del método?	**Nivel de abstracción de las normas y directrices**						
		Gestión de proyectos	VERDADERO	VERDADERO	VERDADERO	FALSO	VERDADERO	VERDADERO
		Descripción de procesos	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO	FALSO
		Normas y orientaciones concretas sobre las actividades y productos	FALSO	FALSO	FALSO	VERDADERO	VERDADERO	FALSO
		Las actividades cubiertas por el método ágil						
		Puesta en marcha del proyecto	FALSO	FALSO	VERDADERO	FALSO	FALSO	FALSO
		Definición de requisitos	VERDADERO	FALSO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Modelado	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Código	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Pruebas unitarias	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Pruebas de integración	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Prueba del sistema	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Prueba de aceptación	VERDADERO	FALSO	FALSO	FALSO	FALSO	FALSO
		Control de calidad	VERDADERO	FALSO	FALSO	FALSO	VERDADERO	FALSO
		Sistema de uso	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO
		Productos de las actividades del método ágil						
		Modelos de diseño	FALSO	FALSO	VERDADERO	FALSO	VERDADERO	VERDADERO
		Comentario del código fuente	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Ejecutable	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Pruebas unitarias	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Pruebas de integración	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Pruebas de sistema	VERDADERO	VERDADERO	FALSO	VERDADERO	VERDADERO	VERDADERO
		Pruebas de aceptación	VERDADERO	FALSO	VERDADERO	FALSO	FALSO	FALSO
		Informes de calidad	VERDADERO	FALSO	FALSO	FALSO	FALSO	FALSO
		Documentación de usuario	FALSO	FALSO	VERDADERO	FALSO	FALSO	FALSO

Comparando los resultados obtenidos del formulario y la tabla de clasificación anterior, se identificará la metodología que mejor se adapta a la forma de trabajo del Departamento TIC de la ETFA. La metodología adecuada será la que mayor número de coincidencias tenga con el cuestionario anterior.

Departamento TIC

a. Uso

Verdadero (V) o Falso (F) en cada una de las premisas.

Tabla 14. Valoración del Uso del Departamento TIC

	Premisa	**Respuesta**
1	Respeto de las fechas de entrega	V
2	Cumplimiento de los requisitos	V
3	Respeto al nivel de calidad	V
4	Satisfacción del usuario final	V
5	Entornos turbulentos	V
6	Favorable al Off shoring	F
7	Aumento de la productividad	V

b. Capacidad de Agilidad

Verdadero (V) o Falso (F) en cada una de las premisas.

Tabla 15. Valoración de la Capacidad de agilidad

	Premisa	**Respuesta**
1	Iteraciones cortas	V
2	Colaboración	V
3	Centrado en las personas	V
4	Refactoring político	F
5	Prueba político	F
6	Integración de los cambios	V
7	De peso ligero	V
8	Los requisitos funcionales pueden cambiar	V
9	Los requisitos no funcionales pueden cambiar	V
10	El plan de trabajo puede cambiar	V
11	Los recursos humanos pueden cambiar	V
12	Cambiar los indicadores	V
13	Reactividad (AL COMIENZO DEL PROYECTO, CADA ETAPA, CADA ITERACIÓN)	ITERACIÓN
14	Intercambio de conocimientos (BAJO, ALTO)	BAJO

c. Aplicación

Verdadero (V) o Falso (F) en cada una de las premisas.

Tabla 16. Valoración de la Aplicación del Departamento TIC

	Premisa	**Respuesta**
1	Tamaño del proyecto (PEQUEÑO, GRANDE)	PEQUEÑO
2	La complejidad del proyecto (BAJA, ALTA)	BAJA
3	Los riesgos del proyecto (BAJO, ALTO)	BAJO
4	El tamaño del equipo (PEQUEÑO, GRANDE)	PEQUEÑO
5	El grado de interacción con el cliente (BAJA, ALTA)	ALTA
6	Grado de interacción con los usuarios finales (BAJA, ALTA)	ALTA
7	Grado de interacción entre los miembros del equipo (BAJA, ALTA)	ALTA
8	Grado de integración de la novedad (BAJA, ALTA)	ALTA
9	La organización del equipo (AUTO-ORGANIZACIÓN, ORGANIZACIÓN JERÁRQUICA)	JERÁRQUICA

d. Procesos y Productos

Verdadero (V) o Falso (F) en cada una de las premisas.

Nivel de abstracción de las normas y directrices:

Tabla 17. Valoración normas y directrices ágiles del Departamento TIC

	Premisa	**Respuesta**
1	Gestión de proyectos	V
2	Descripción de procesos	V
3	Normas y orientaciones concretas sobre las actividades y productos	F

Las actividades cubiertas por el método ágil:

Tabla 18. Valoración actividades cubiertas por el método ágil del Departamento TIC

	Premisa	**Respuesta**
1	Puesta en marcha del proyecto	V
2	Definición de requisitos	V
3	Modelado	V
4	Código	V
5	Pruebas unitarias	V
6	Pruebas de integración	V

7	Prueba del sistema	V
8	Prueba de aceptación	V
9	Control de calidad	V
10	Sistema de uso	V

Productos de las actividades del método:

Tabla 19. Valoración de los productos de las actividades de la metodología ágil del Departamento TIC

	Premisa	**Respuesta**
1	Modelos de diseño	V
2	Comentario del código fuente	V
3	Ejecutable	V
4	Pruebas unitarias	V
5	Pruebas de integración	V
6	Pruebas de sistema	V
7	Pruebas de aceptación	V
8	Informes de calidad	V
9	Documentación de usuario	V

En los casos en los que la respuesta del Departamento TIC coincida con el valor asociado a la metodología se sumará 1, en caso contrario 0.

Tabla 20. Metodología ágil adecuada para el Departamento TIC

			METODOLOGÍAS ÁGILES					
			ASD	**CRISTAL**	**DSDM**	**XP**	**FDD**	**SCRUM**
USO	¿Por qué utilizar un método ágil?	Respeto de las fechas de entrega	1	1	1	0	1	1
		Cumplimiento de los requisitos	1	1	1	1	1	1
		Respeto al nivel de calidad	1	0	0	0	0	0
		Satisfacción del usuario final	0	0	1	0	0	0
		Entornos turbulentos	0	0	1	1	0	1
		Favorable al Off shoring	0	0	0	1	1	0
		Aumento de la productividad	0	0	0	1	0	0
CAPACIDAD DE AGILIDAD	¿Cuál es la parte de agilidad incluida en el método?	Iteraciones cortas	0	0	0	1	1	1
		Colaboración	1	1	1	1	0	0
		Centrado en las personas	1	1	1	1	0	0
		Refactoring político	1	1	0	0	1	1
		Prueba político	0	1	0	0	1	0
		Integración de los cambios	1	0	1	1	0	1
		De peso ligero	0	0	0	1	1	1
		Los requisitos funcionales pueden cambiar	1	0	1	1	0	1
		Los requisitos no Funcionales pueden cambiar	1	0	0	0	0	0
		El plan de trabajo puede cambiar	0	0	0	1	0	0
		Los recursos humanos pueden cambiar	1	1	0	1	0	0

		Cambiar los indicadores	0	0	0	1	0	0
		Reactividad (AL COMIENZO DEL PROYECTO, CADA ETAPA, CADA ITERACIÓN)	0	0	1	1	0	0
		Intercambio de conocimientos (BAJO, ALTO)	0	0	1	0	1	1
APLICABILIDAD	¿Cuándo un ambiente es favorable parausar este método?	Tamaño del proyecto (PEQUEÑO, GRANDE)	0	0	0	1	0	0
		La complejidad del proyecto (BAJA, ALTA)	0	0	0	1	0	0
		Los riesgos del proyecto (BAJO, ALTO)	0	0	0	1	0	0
		El tamaño del equipo (PEQUEÑO, GRANDE)	0	1	0	1	0	0
		El grado de interacción con el cliente (BAJA, ALTA)	1	0	1	1	0	0
		Grado de interacción con los usuarios finales (BAJA, ALTA)	0	0	1	0	0	0
		Grado de interacción entre los miembros del equipo (BAJA, ALTA)	0	1	1	1	0	0
		Grado de integración de la novedad (BAJA, ALTA)	1	0	1	1	0	0
		La organización del equipo(AUTO-ORGANIZACIÓN, ORGANIZACIÓN JERÁRQUICA)	1	0	1	0	1	0
PROCESOS Y PRODUCTOS	¿Cómo están Caracterizados los procesos del método?	**Nivel de abstracción de las normas y directrices**						
		Gestión de proyectos	1	1	1	0	1	1
		Descripción de procesos	1	1	1	1	1	0
		Normas y orientaciones concretas sobre las actividades y productos	0	0	0	0	0	1
		Las actividades cubiertas por el método ágil1						
		Puesta en marcha del proyecto	0	0	1	0	0	0
		Definición de requisitos	1	0	1	1	1	1
		Modelado	1	1	1	1	1	1
		Código	1	1	1	1	1	1
		Pruebas unitarias	1	1	1	1	1	1
		Pruebas de integración	1	1	1	1	1	1
		Prueba del sistema	1	1	1	1	1	1
		Prueba de aceptación	1	0	0	0	0	0
		Control de calidad	1	0	0	0	1	0
		Sistema de uso	0	0	0	0	0	0
		Productos de las actividades del método ágil						
		Modelos de diseño	0	0	1	0	1	1
		Comentario del código fuente	1	1	1	1	1	1
		Ejecutable	1	1	1	1	1	1
		Pruebas unitarias	1	1	1	1	1	1
		Pruebas de integración	1	1	1	1	1	1
		Pruebas de sistema	1	1	0	1	1	1
		Pruebas de aceptación	1	0	1	0	0	0
		Informes de calidad	1	0	0	0	0	0
		Documentación de Usuario	0	0	1	0	0	0
		TOTAL	**30**	**21**	**31**	**33**	**24**	**23**

En los resultados se observa que XP y DSDM son los que han obtenido más puntuación. Podrían aplicar una u otra, incluso podemos no limitarnos al uso exclusivo de una sola, las metodologías (XP, DSDM, ASP, Scrum) se pueden combinar, por ejemplo muchos equipos que utilizan las prácticas de Scrum permiten hacer reuniones diarias, lo que otorga un valor agregado al software al poder interactuar conjuntamente con el equipo de trabajo, el cliente y usuarios; lo que nos deja como lección que se puede usar lo que funcione en cada equipo.

3.4.4 Cuarta Parte: Estado actual de las metodologías seleccionadas

Los criterios adoptados para la selección de cada una de los estados de las metodologías están fundamentados en la caracterización que se ha venido realizando sobre las metodologías a medida que se desarrolla el capítulo, de tal manera que una metodología está en estado de:

- Recién nacida. Es aquella metodología que tiene un año o menos y de la cual no tenemos evidencias empíricas, ni estudios.
- En construcción. Aquellas metodologías con más de una año de existencia, pero que no disponen de experiencias documentadas ni/o estudios.
- Activa. Son aquellas metodologías que llevan varios en el desarrollo del software y de las cuales podemos encontrar experiencias y estudios que corroboren su ratio efectividad.
- Olvidada. Aquellas metodologías que llevan el suficiente tiempo sin ser utilizadas y de las cuales no se encuentran experiencias actuales, ni estudios.

Tabla 21. Criterios de estados para las metodologías ágiles

Metodología	Explicación	Estado Actual
ASD	Existe desde el año 2004, y hasta la actualidad no ha mostrado una suficiente consolidación en la actividad académica o empresarial como para considerarse activa o ya establecida.	En construcción
Crystal Methods	Se encuentra activa con respecto a Orange y Clear, en construcción en cuanto a familia de metodologías incompletas	En construcción/ Activa
DSDM	Se define como una de las metodologías con mayor adopción en el Reino Unido y disponiendo de un gran número de experiencias se considera como una metodología activa.	Activa

SCRUM	Es una de las metodologías más antiguas y que más de moda esta últimamente, en la actualidad se puede encontrar muchos estudios y experiencia con Scrum.	Activa
FDD	La técnica/ metodología más extendida. Se puede encontrar en un gran número de proyectos, muchas veces de la mano XP.	Activa
XP	Se ha convertido desde 1999, en la punta del iceberg de las metodologías ágiles, es la número uno en internet, en adopciones y experiencias y a su vez la primera en estudios.	Activa

Después de establecer la comparación, se deduce que se puede escoger el grado de madurez que comienzan a tener las metodologías ágiles, o como mínimo las que se han escogido, comienzan a dar señales de que son una realidad palpable, esto indica, que un gran número de empresas están dando el salto y dejando de experimentar con ellas para comenzar a utilizarlas con seriedad.

Haciendo la relación de la Tabla 20, donde se arrojó la metodología XP con la mayor puntuación, con la Tabla 21 podemos apreciar que XP ha alcanzado su grado de madurez significativo constando en estado Activo, encontrándose hoy en día como una de las metodologías mejor documentada, con diferentes adopciones en el estudio, mayores referencias y ampliamente utilizada con éxito en proyectos reales.

3.4.5 Determinación de la Metodología Ágil adaptable al Departamento TIC

A lo largo del presente capítulo, se ha realizado un estudio selectivo fuertemente válido y justificable, del que se dispuso de la información necesaria en base a las metodologías que se consideró más importantes.

Se partió desde la identificación de las Metodologías ágiles, evaluando los criterios válidos en base a información real existente, las que determinaron la selección de seis metodologías.

Luego de la selección se determinaron las características esenciales dando a conocer el proceso o ciclo de vida de las mismas.

Finalmente, se realizó un estudio comparativo de las metodologías seleccionadas en función de las necesidades y el entorno de trabajo del equipo del Departamento TIC, donde se elaboró un cuestionario que consta de dos partes , una inicial para conocer la orientación de la Departamento TIC: ágil o tradicional (necesaria para fundamentar a fondo el estudio), y una segunda parte que permitiría conocer la

metodología ágil que mejor se acopla al Departamento, con la ayuda del Framework propuesto por Adrian Iacovelli.

Además, apoyados con la comparativa del estado actual de las metodologías y en base a todos los resultados arrojados del estudio comparativo, podemos identificar que la metodología que más se adapta es la **Extreme Programming (XP)**; recalcando que no es la única que podría ajustarse y que existen otras como la DSDM, ASP, Scrum (las cuales tuvieron ponderaciones altas) que incluso las podemos combinar.

3.5 Conclusión del Capítulo

En este capítulo, se han desarrollado todas las metodologías ágiles para un mayor conocimiento de las características de las mismas, se pudo establecer una comparación entre las mismas, se logró determinar cuál sería la metodología para el desarrollo del software más adecuada para ser implementada en el Departamento TIC de la ETFA y de esta se disertó sus características fundamentales. Esperando una buena utilización de las fuentes y técnicas de recopilación de información y análisis de datos para el buen desempeño de esta metodología ágil como lo es Extreme Programming (XP); ésta metodología, conjuntamente con otros principios, procesos y artefactos valederos de otras metodologías ágiles (DSDM, ASP, Scrum) que obtuvieron también mayor ponderación, serán la base fundamental que establecerá el marco de trabajo para posteriores desarrollos de sistemas software, y que será demostrable y verificable a través del "Sistema Inteligente para el Control Disciplinario de los Aspirantes a Tropa de la ETFA".

CAPÍTULO 4
DESARROLLO DE LA METODOLOGÍA ADAPTABLE Y APLICACIÓN DEL CASO DE ESTUDIO PRÁCTICO

4.1 Introducción del Capítulo

En el desarrollo del presente capítulo, inicialmente se hará una síntesis de los ciclos de vida de las metodologías ágiles. Seguidamente se establecerá el ciclo de vida de la metodología ágil seleccionada en el capítulo anterior, considerando el entorno y las necesidades en el que se desenvuelve el Departamento TIC de la ETFA, la misma que servirá de aplicación en el caso de estudio práctico propuesto a fin de justificar la funcionalidad del mismo para el proceso de desarrollo del sistema software, y a su vez sirva de base fundamental de posteriores desarrollos.

El Departamento TIC desarrolla software según requerimientos y exigencias de la ETFA. Seguidamente se estructurará la implementación de un marco de trabajo basado en los procesos de una metodología ágil y que será aplicado a un caso de estudio práctico propuesto.

4.2 Fases del Ciclo de Vida de las Metodologías Ágiles

El ciclo de vida permite que los errores se detecten lo antes posible y por lo tanto, permite a los desarrolladores concentrarse en la calidad del software, en los plazos de implementación y en los costos asociados. Para facilitar una metodología común entre el cliente y el desarrollador de software, los modelos de ciclo de vida se han actualizado para reflejar las etapas de desarrollo involucradas y la documentación requerida, de manera que cada etapa se valide antes de continuar con la siguiente etapa.

De manera general, los métodos de desarrollo ágil se desarrollan en un similar ciclo de vida, unas priorizando o haciendo retrospecciones más profundas que otras, aunque la mayoría apuntan a minimizar riesgos desarrollando software en lapsos cortos. Comúnmente, el desarrollo ágil involucra al término “iteración” dentro de sus fases como el software desarrollado en una unidad de tiempo, la cual debe durar de una a cuatro semanas. Cada iteración del ciclo de vida incluye: planificación, análisis de requisitos, diseño, codificación, revisión y documentación. Una iteración no debe agregar demasiada funcionalidad para justificar el lanzamiento del producto al

mercado, sino que la meta es tener un "demo" (sin errores) al final de cada iteración. Al final de cada iteración el equipo vuelve a evaluar las prioridades del proyecto. Dentro de éste esquema genérico se desarrolla un ciclo de vida de una metodología ágil, la cual será apreciada en los apartados siguientes.

4.3 Ciclo de Vida de la Metodología Adaptable "XP TIC´S"

En el estudio realizado en el capítulo anterior, se determinó que la metodología que más se ajusta al entorno de trabajo y necesidades del Departamento TIC fue la Metodología de Desarrollo Ágil Extreme Programming "XP", resaltando que no es la única que podría ajustarse, ya que del estudio realizado también resultó otras como la DSDM, ASP o Scrum, y que incluso las podemos combinar utilizando las prácticas que mejor funcionen en el entorno del equipo desarrollador.

Para el desarrollo del capítulo nos referenciaremos principalmente a la metodología XP, sin olvidar artefactos y prácticas que podrían tomarse de otras. El ciclo de vida de un proyecto XP incluye, al igual que las otras metodologías, entender lo que el cliente necesita, estimar el esfuerzo, crear la solución y entregar el producto final al cliente. Sin embargo, XP propone un ciclo de vida dinámico, donde se admite expresamente que, en muchos casos, los clientes no son capaces de especificar sus requerimientos al comienzo de un proyecto. Por esto, se trata de realizar ciclos de desarrollo cortos (llamados iteraciones), con entregables funcionales al finalizar cada ciclo. Las iteraciones son relativamente cortas ya que se piensa que entre más rápido se le entreguen desarrollos al cliente, más retroalimentación se va a obtener y esto va a representar una mejor calidad del producto a largo plazo. Existe una fase de análisis inicial orientada a programar las iteraciones de desarrollo y cada iteración incluye diseño, codificación y pruebas, fases superpuestas de tal manera que no se separen en el tiempo [62].

Actualmente, el Departamento TIC de la ETFA busca desarrollar software de calidad en un entorno dinámico, mediante la aplicación de herramientas, normas, estándares, marcos de trabajo y metodologías disponibles que proporcionen un soporte tecnológico, conceptual y humano, propendiendo al mejoramiento continuo del desarrollo software en sus diferentes etapas, reduciendo costos y tiempos de entrega.

En vista de esto, el Departamento TIC se ha propuesto objetivos precisos. Uno de ellos es la satisfacción del usuario, la cual puede ser alcanzada mediante la aplicación

de la metodología ágil XP, pues ésta trata de dar al cliente el software que él necesita y cuando lo necesita. Por tanto, el entorno del Departamento TIC debe responder muy rápido a las necesidades del usuario, incluso cuando los cambios sean al final del ciclo de la programación. El segundo objetivo principal es potenciar al máximo el trabajo en grupo, que de igual manera con la aplicación de XP, tanto los jefes de proyecto, los clientes y desarrolladores, son parte del equipo y están involucrados en el desarrollo del software.

La siguiente figura esquematiza el ciclo de vida de la Metodología Seleccionada "XP TIC´S" adaptado a las necesidades del entorno que se desarrollará en el Departamento TIC.

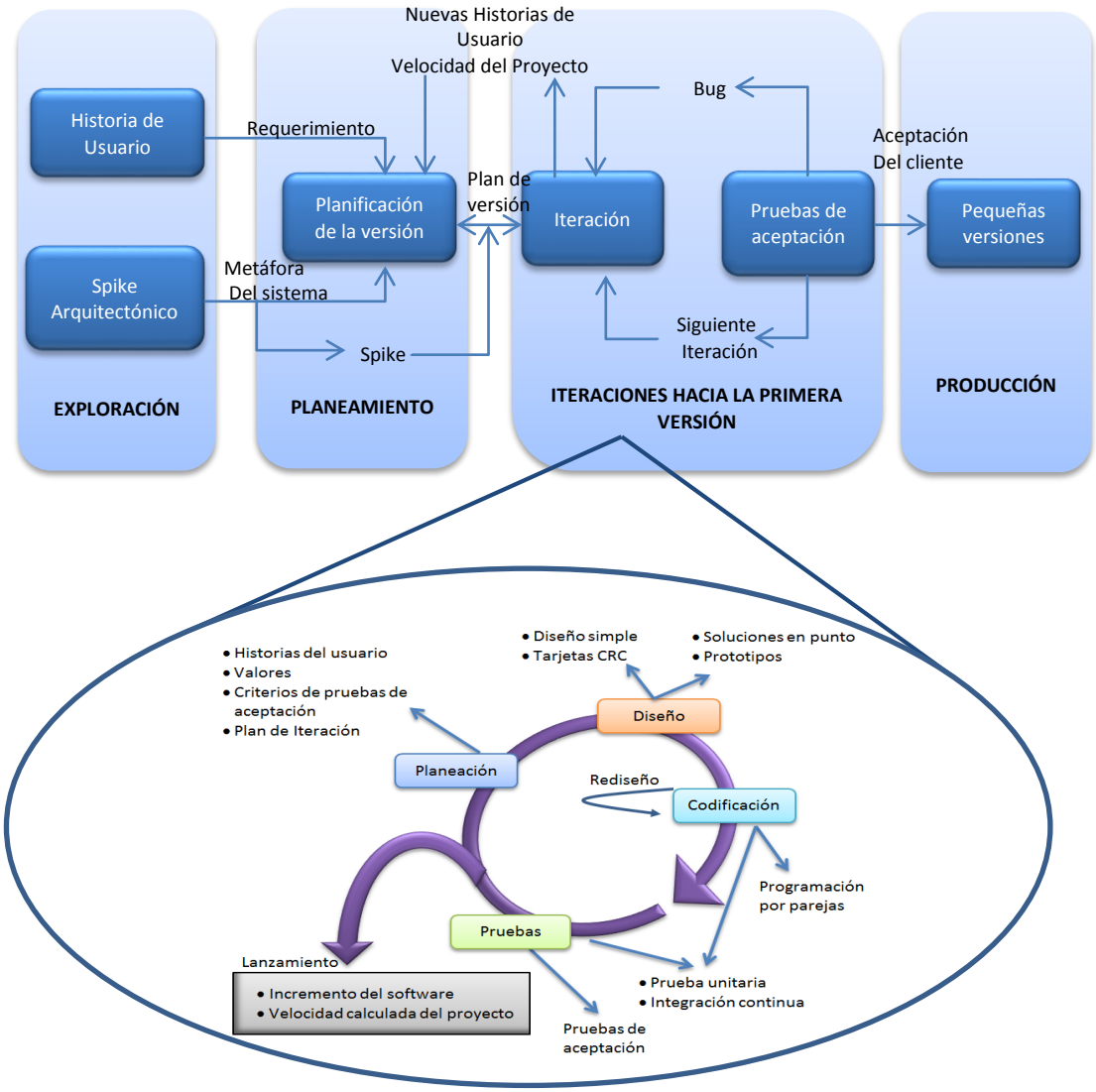


Figura 12. Ciclo de vida Metodología "XP TIC´S" basado en Metodologías Ágiles

Fuente: (Los Autores)

Si bien el ciclo de vida de un proyecto con desarrollo ágil es muy dinámico, se puede determinar sus fases guiándonos con las prácticas XP [63] y ciertas metodologías ágiles:

- Fase de Exploración
- Fase de Planeación del Proyecto
- Fase de Iteraciones
- Fase de Producción
- Fase de Mantenimiento
- Fase de Muerte del Proyecto

4.4 Implementación de la Metodología Adaptable "XP TIC´s"

Seguidamente se expone el formato general para la generación de proyectos, aplicando la Metodología Ágil de Desarrollo de Sistemas Software "XP TIC´s" para el Departamento TIC de la ETFA.

De igual manera, en el ANEXO F se adjunta la aplicación de mencionada Metodología al caso de estudio práctico "Sistema Inteligente de Control Disciplinario para Aspirantes a Tropa de la Escuela Técnica de la Fuerza Aérea", cuyo propósito es verificar la validez y rendimiento de la misma.

Desarrollo del Sistema [Nombre cliente]	Versión: [n.n]
Del Departamento [Nombre Departamento]	Fecha: [dd/mm/aaaa]
Descripción de la metodología de trabajo	

Proyecto

[Nombre del sistema o proyecto]

Metodología de Desarrollo Ágil de Sistemas Software

"XP TIC´s" del Departamento TIC de la ETFA

Versión: [n.n]

HISTORIAL DE REVISIONES

Fecha	Versión	Descripción	Autor
[dd/mm/aaaa]	[n.n]	[Primera versión con los apartados y contenidos básicos]	[Nombre autor]
…			

DESCRIPCIÓN DE LA METODOLOGÍA DE TRABAJO

I. INTRODUCCIÓN

Este documento describe la implementación de la metodología de trabajo "XP TIC´S" y ciertas metodologías ágiles en el Departamento TIC de la ETFA para la gestión del desarrollo del proyecto [Nombre del sistema].

Incluye junto con la descripción de este ciclo de vida iterativo e incremental para el proyecto, los artefactos o documentos con los que se gestionan las tareas de adquisición y suministro: requisitos, monitorización y seguimiento del avance, así como las responsabilidades y compromisos de los participantes en el proyecto.

I.1 PROPÓSITO DE ESTE DOCUMENTO

Facilitar la información de referencia necesaria a las personas implicadas en el desarrollo del sistema [Nombre del sistema].

I.2 ALCANCE

Personas y procedimientos implicados en el desarrollo del sistema [Nombre del sistema].

II. DESCRIPCIÓN GENERAL DE LA METODOLOGÍA

II.1 FUNDAMENTACIÓN

Las principales razones del uso de un ciclo de desarrollo iterativo e incremental de tipo ágil, el cual se basa nuestra metodología, para la ejecución de este proyecto son:

- Sistema modular. Las características del sistema [Nombre del sistema] permiten desarrollar una base funcional mínima y sobre ella ir incrementando las funcionalidades o modificando el comportamiento o apariencia de las ya implementadas.
- Entregas frecuentes y continuas al cliente de los módulos terminados, de forma que puede disponer de una funcionalidad básica en un tiempo mínimo y a partir de ahí un incremento y mejora continua del sistema.
- Previsible inestabilidad de requisitos.
  - Es posible que el sistema incorpore más funcionalidades de las inicialmente identificadas.
  - Es posible que durante la ejecución del proyecto se altere el orden en el que se desean recibir los módulos o historias de usuario terminadas.
  - Para el cliente resulta difícil precisar cuál será la dimensión completa del sistema, y su crecimiento puede continuarse en el tiempo, suspenderse o detenerse.
- [Otras posibles razones…]; definidas según las características y condiciones del usuario.

II.2 VALORES DE TRABAJO

Los valores que deben ser practicados por todos los miembros involucrados en el desarrollo y que hacen posible que la metodología XP TIC´S tenga éxito son:

- Simplicidad
- Comunicación
- Retroalimentación (Feedback)
- Coraje o valentía
- Respeto.

III. PERSONAS Y ROLES DEL PROYECTO

Tabla 22. Persona y roles del proyecto

Persona	**Contacto**	**Rol**
[Nombre]	[e-mail / tel.]	[Programador...]
[Nombre]	[e-mail / tel.]	[Cliente…]

[Nombre]	[e-mail / tel.]	[Tester…]
[Nombre]	[e-mail / tel.]	[Tracker…]
[Nombre]	[e-mail / tel.]	[Entrenador/Coach…]
[Nombre]	[e-mail / tel.]	[Consultor…]
[Nombre]	[e-mail / tel.]	[Gestor/Big_Boss…]
[…]		

IV. RESPONSABILIDADES

Programador

- Escribe las pruebas unitarias.
- Produce el código del sistema.
- ***[Otras posibles implementadas en esta Institución]***

Cliente

- Escribe las historias de usuario y las pruebas funcionales para validar su implementación.
- Asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar el mayor valor de negocio.
- ***[Otras posibles implementadas en esta Institución]***

Tester

- Ayuda al cliente a escribir las pruebas funcionales.
- Ejecuta pruebas regularmente y difunde los resultados en el equipo
- Es responsable de las herramientas de soporte para pruebas.
- ***[Otras posibles implementadas en esta Institución]***

Tracker

- Es el encargado de seguimiento.
- Proporciona realimentación al equipo.
- Debe verificar el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, comunicando los resultados para mejorar futuras estimaciones.

- ***[Otras posibles implementadas en esta Institución]***

Entrenador (coach)

- Responsable del proceso global.
- Guía a los miembros del equipo para seguir el proceso correctamente.
- ***[Otras posibles implementadas en esta Institución]***

Consultor

- Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto.
- Ayuda al equipo a resolver un problema específico.
- ***[Otras posibles implementadas en esta Institución]***

Gestor-Jefe del Proyecto (Big boss)

- Es el dueño del equipo y el vínculo entre clientes y programadores.
- Su labor esencial es la coordinación.
- ***[Otras posibles implementadas en esta Institución]***

V. FASES DE LA METODOLOGÍA ADAPTABLE "XP TIC´S".

1ª FASE: EXPLORACIÓN

1.1 Recolección de Requerimientos Generales

En esta fase, los clientes o demás Departamentos de la ETFA plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto.

Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

Seguidamente se sugiere la Tabla 23 para la recolección de requerimientos.

Tabla 23. Formato de Recolección de Requerimientos Generales

Nombre	Departamento	Requerimiento/Historia Usuario	Prioridad estimada
Persona1	Dpto. “A” ETFA	H1.	Alta/Media/Baja
Persona2	Dpto. “B” ETFA	H2.	Alta/Media/Baja
….			

- Nombre: Hace referencia de la persona que genera el requerimiento.
- Departamento: Hace referencia al Departamento de la ETFA donde surge el requerimiento.
- Requerimiento: Se describirá en términos generales la necesidad del usuario/cliente, esto a su vez se convertirá en las historias de usuario.
- Prioridad estimada: Se establecerá una ponderación estimada del requerimiento por el usuario/cliente del Departamento con la prioridad Alta/Media/Baja, según la importancia dentro del contexto del proyecto.

1.1.1 Supuestos Generales

Los supuestos apuntarán a la posibilidad de alcanzar algo propuesto, es decir, se considerará un dato asumido como cierto a efectos de la planificación del proyecto; como por ejemplo alcanzar la automatización, el mejoramiento, el funcionamiento, rendimiento o productividad de un proyecto y/o Institución.

1.1.2 Principales procesos a generarse

En la [versión 1.0] se tomará en cuenta las [cantidad] (número) primeras historias de usuario ordenadas por prioridad para detallar el desarrollo del software y su avance en el proyecto. Los principales requerimientos para ejecutarse que poseen gran prioridad, se convertirán en los procesos generales del proyecto.

Seguidamente se sugiere la Tabla 24 para la recolección de los principales procesos.

Tabla 24. Recolección de los Principales Procesos

No.	Nombre del Proceso	Descripción del Proceso	Actor	Funciones del Actor
1	Proceso/Gestión A	Procesos lógicos, Interrelaciones, funcionalidades, etc.	Actor A	Función A
…				

- Nombre del Proceso: Corresponde al nombre que se le atribuye al proceso o gestión a realizar.
- Descripción del Proceso: Describirá todo lo referente a la gestión del proceso, indicando las funcionalidades, interrelaciones con las historias de usuario, procesos lógicos, aritméticos, etc.
- Actor: Hace referencia a los diferentes niveles de usuarios del sistema (super-administrador, administrador, cliente, el propio usuario o entre otros definidos por la Institución) que intervienen en cada uno de los procesos señalados.
- Funciones del Actor: Describirá las funciones específicas que tendrá el actor dentro del sistema.

1.1.3 Estimación General del Proyecto

De acuerdo a la experiencia del equipo programador del Departamento TIC, las estimaciones de tiempo asociado a la implementación de las historias, inicialmente se establecerán de acuerdo a la duración por día, procurando que cada historia no exceda más allá de los ocho días.

El equipo de trabajo se reunirá y en función de las historias de usuario se estimará cada una de ellas, obteniendo de esta manera un tiempo estimado total que será fundamental para programar inicialmente el proyecto.

Seguidamente se sugiere la Tabla 25 para la estimación general del proyecto.

Tabla 25. Estimación general del proyecto

HISTORIA	TIEMPO ESTIMADO
Historia 1	1 a 8 horas
Historia 2	1 a 8 horas
…	
Tiempo Estimado Total	Σ horas

1.1.4 Spike Arquitectónico

En caso de que la historia de usuario presente dificultades de ser estimado, ya sea por alguna complejidad técnica o de arquitectura del proyecto o sistema, se desarrollará un prototipo que facilite estimar el tiempo que demora el desarrollo de la historia.

El Spike Arquitectónico comprende en realizar una pequeña aplicación completamente desconectada del proyecto, la cual se realizará por parejas e intentará explorar el problema, proponiendo una solución potencial. Esta puede ser rústica y simple, siempre que brinde la información suficiente para enfrentar el problema encontrado.

El conocimiento aprendido de la elaboración del Spike, deberá ser compartido con el resto del grupo del Departamento, para generar una base de experiencia en el desarrollo de la metodología.

1.2 Plan General de Desarrollo

En este apartado se describirá la planificación que seguirá el proyecto a lo largo de su desarrollo, estableciendo las historias en sus diferentes fases de ciclo de vida y estimando tiempos en referencia al sub-apartado 1.1.3; así como lo indica el ejemplo del siguiente gráfico:

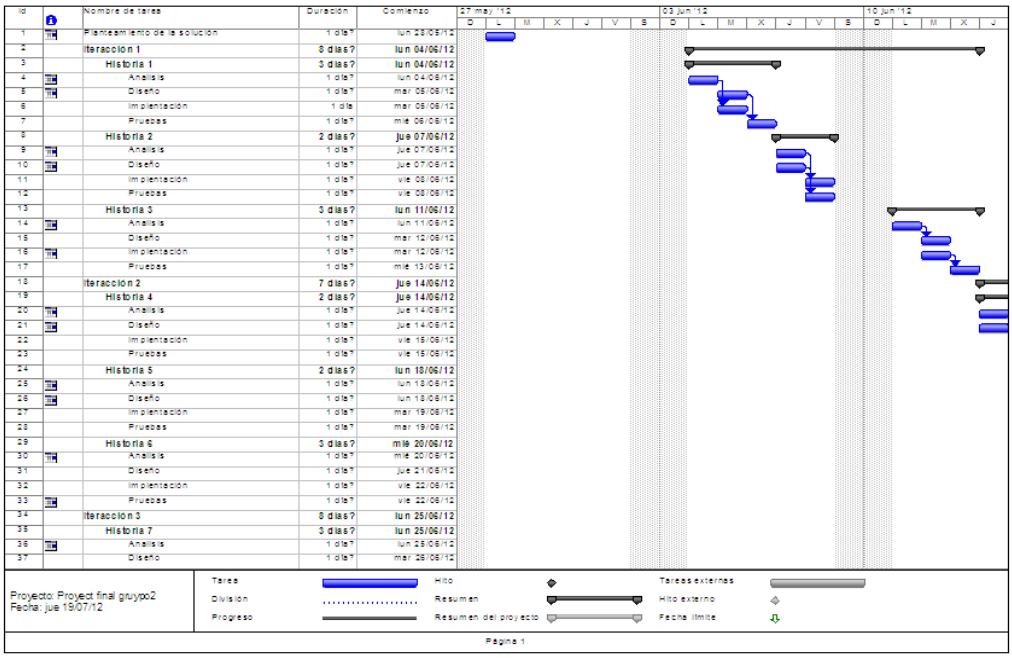


Figura 13. Ejemplo Plan General de un Proyecto en MS Project

Fuente: (Los Autores)

2ª FASE: PLANIFICACIÓN DEL PROYECTO

En esta fase se comenzará a interactuar con el cliente o demás Departamentos de la ETFA y el resto del grupo de desarrollo para descubrir los requerimientos del sistema. Se identificará la prioridad de cada historia de usuario, y correspondientemente, los programadores realizarán una estimación del esfuerzo necesario de cada una de ellas. Se tomarán acuerdos sobre el contenido de la primera entrega y se determinará un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase durará unos pocos días.

Las estimaciones de esfuerzo asociado a la implementación de las historias la establecerán los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos.

2.1 Historias de los usuarios

Las historias de usuario son la técnica para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales (Ver Tabla 26).

El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento las historias de usuario pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas.

Cada historia de usuario debe ser lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.

Las historias de usuario deberán:

- Ser escritas por el usuario.
- Usar terminología del cliente.
- Ser de bajo nivel de detalle.
- Servir de base para estimar los tiempos de implementación.
- No ser menos de 20, ni más de 80.

Seguidamente se sugiere la Tabla 26 para la recolección de las historias de usuario.

Tabla 26. Tarjeta para historias de usuario

Historia de Usuario	
Número:	**Usuario:**
Nombre historia:	
Prioridad en negocio: (Alta / Media / Baja)	**Riesgo en desarrollo:** (Alta / Media / Baja)
Puntos estimados:	**Iteración asignada:**
Programador responsable:	
Descripción:	
Observaciones:	

2.2 Spike solutions

De similar manera como el sub-apartado 1.1.4, se enunció que en muchas de las ocasiones el equipo de desarrollo del Departamento TIC se enfrentará a historias de usuario (requerimientos) de los clientes, los cuales generarán problemas. Es donde nacerá la necesidad de realizar Spike Solutions o Soluciones Rápidas, en caso de surgir dudas referentes a cómo solucionar un problema técnico o de arquitectura del software, se recurrirá a soluciones provisorias que se focalizarán en el tema en cuestión. Se tomará en cuenta que estas soluciones provisorias no se conserven como solución definitiva, sino que generalmente se descarten luego de haber cumplido su propósito. Su objetivo es:

- Reducir el riesgo de sufrir un problema técnico o de diseño y/o,
- Aumentar la fiabilidad de una estimación de la historia de usuario.

El Spike Solution es un bosquejo de una pequeña aplicación completamente desconectada del proyecto, la cual se realizará por parejas e intentará explorar el problema, proponiendo una solución potencial. Esta puede ser burda y simple, siempre que brinde la información suficiente para enfrentar el problema encontrado. El conocimiento aprendido con la elaboración de éstos, deberá ser compartido con el resto

del grupo del Departamento, ya que así se generará una base de conocimiento que durante todo el desarrollo de la metodología.

Se recomienda el uso de la librería JUnit[6] de Java para la simulación de estos bosquejos, entre otras herramientas.

2.3 Prioridad

La prioridad de las historias se las establece de acuerdo al grado de importancia que tiene para el desenvolvimiento de la funcionalidad del sistema. El cliente demarca mucha importancia en el establecimiento de las mismas, en vista de que de él surgen las necesidades primordiales para la generación del sistema. En tal sentido, el cliente establecerá la prioridad con la interacción del equipo de trabajo.

En este aparatado conoceremos el orden de ejecución de las tareas a realizar, priorizando en orden desde la mayor importancia hasta la menor importancia. Seguidamente se sugiere la Tabla 27 para establecer el orden según prioridad de las historias.

Tabla 27. Orden y prioridad según importancia de las historias de usuario

Actividad N°	Descripción	Prioridad
1	[Nombre de la Historia 1]	[Alta/Media/Baja]
2	[Nombre de la Historia 2]	[Alta/Media/Baja]
3	[Nombre de la Historia 3]	[Alta/Media/Baja]
…	[Nombre de la Historia n]	[Alta/Media/Baja]

- Actividad N°: Corresponde al orden secuencial según la importancia y prioridad de la historia.
- Descripción: Se nombra la historia de usuario correspondiente.
- Prioridad: Se establecerá la prioridad según la importancia dentro del contexto del proyecto o sistema generado determinado por el cliente con la interacción del equipo de trabajo del Departamento, de acuerdo con la prioridad Alta/Media/Baja.

[6] Conjunto de bibliotecas que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java.

3ª FASE: ITERACIONES

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción.

Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la Iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior.

Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores.

3.1 División en iteraciones

De acuerdo a la metodología implementada, la creación del proyecto o sistema software se dividirá en etapas para facilitar su realización. Generalmente, los proyectos constan de más de tres etapas, las cuales toman el nombre de iteraciones. La duración ideal de una iteración es de una a tres semanas.

Para cada iteración se define un módulo o conjunto de historias que van a implementarse. Al final de la iteración se obtiene como resultado la entrega del módulo correspondiente, el cual debe haber superado las pruebas de aceptación que establece el cliente para verificar el cumplimiento de los requisitos. Las tareas que no se realicen en una iteración son tomadas en cuenta para la próxima iteración, junto al cliente, si se deben realizar o si deben ser removidas de la planeación del sistema software. En resumen:

- El proyecto se divide en varias iteraciones.
- La duración de una iteración varía entre una y tres semanas.

Seguidamente se sugiere la Tabla 28 para establecer el plan de iteraciones del proyecto.

Tabla 28. Plan de iteraciones (Release Planning)

Primera Iteración	H1, H2, H3
Segunda Iteración	H4, H5, H6
Tercera Iteración	H7, H8, …
….	Hn……

3.2 Velocidad del proyecto

Es la medida de la capacidad que tiene el equipo de desarrollo para evacuar las historias de usuario en una determinada iteración. Esta medida se calcula totalizando el número de historias de usuario realizadas en una iteración. Para la iteración siguiente se podrá (teóricamente) implementar el mismo número de historias de usuario que en la iteración anterior.

Cabe recalcar que la velocidad del proyecto ayuda a determinar la cantidad de historias que se puede implementar en las siguientes iteraciones, aunque no de manera exacta.

3.2.1 40 horas por semana

Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo. Los proyectos que requieren trabajo extra para intentar cumplir con los plazos suelen al final ser entregados con retraso. En lugar de esto se puede realizar el juego de la planificación para cambiar el ámbito del proyecto o la fecha de entrega.

A continuación se muestra la Tabla 29 como una guía para el desarrollo de las historias de usuario en el cual se establece un trabajo de 8 horas diarias de los desarrolladores:

Tabla 29. Guía de Calendario semanal

Horario de Trabajo Semanal				
Lunes	**Martes**	**Miércoles**	**Jueves**	**Viérnes**
8:00 am 12:00pm	8:00 am 12:00pm	8:00 am 12:00pm	8:00 am 12:00pm	8:00 am 12:00pm
Receso	Receso	receso	Receso	Receso
1:00pm5:00pm	1:00pm5:00pm	1:00pm5:00pm	1:00pm5:00pm	1:00pm5:00pm

Este horario se mantendrá constante durante todo el desarrollo del proyecto.

De igual manera, en la Tabla 30 se muestra una guía sobre la distribución de horas, semanas, horas semanales y velocidad del proyecto de acuerdo al número de iteraciones establecidas:

Tabla 30. Velocidad del proyecto

	ITERACIÓN 1	ITERACIÓN 2	ITERACIÓN 3
Horas	90	60	56
Semanas	3	2	2
horas semanales	30	30	28
historia de usuario (velocidad del proyecto)	4	3	3

Si bien la medida de la velocidad del proyecto de la Tabla 4.8 es tenida para el análisis de tiempos, resultará de mayor utilidad al estimar el número de horas que tomaría implementar cada historia de usuario y planificar las entregas acorde a esta medida. De esta forma, al tener la disponibilidad de cada desarrollador en horas por semana, se podrá estimar con mucha precisión cuántas horas de usuario podrán ser asignadas por iteración, y de igual manera con mucha precisión el plan de entregas.

3.3 Entregas pequeñas

La duración de una iteración varía entre una y tres semanas, al final de la cual habrá una entrega de los avances del producto, los cuales deberán ser completamente funcionales. Estas entregas deben caracterizarse por frecuentes, basándose en los siguientes aspectos:

- Reunión al inicio del proyecto.
- Cuales historias de usuario serán implementadas en cada entrega.
- Grado de relevancia para cada historia de usuario.
- Se aproxima el tiempo para la realización de cada iteración.

3.4 Reuniones diarias

El formato siguiente se utilizará para el control del desarrollo en las revisiones diarias por cada iteración realizada.

Tabla 31. Reuniones Diarias por cada iteración

[Nombre del Proyecto o Sistema]
Control Diario

N° ……
Iteracción

FECHA DEFINICIÓN DE LA HISTORIA			PROCESO	DESCRIPCIÓN TAREA	TIPO DE TAREA	RESPONSABLE	FECHA SEGUIMIENTO			ESTADO	OBSERVACIONES
DIA	MES	AÑO					DIA	MES	AÑO		
										CUMPLIDA/ PENDIENTE	
										CUMPLIDA/ PENDIENTE	
										CUMPLIDA/ PENDIENTE	

- Fecha de Definición de la historia: Corresponde al día, mes y año el cual se definió la historia según la planificación determinada.
- Proceso: Corresponde a la etapa de análisis, diseño, implementación y/o pruebas que se realizan para el desarrollo de la historia.
- Descripción de la tarea: Corresponde a la descripción de la actividad propia para el desarrollo de la historia, como la funcionalidad, gestión o tarea a realizar. Ej: altas, bajas, cambios, niveles de acceso, etc.
- Responsable: Se ubicará el nombre de la persona o personas quienes las ejecutan.
- Fecha de seguimiento: Corresponde al día, mes y año, al cual se ha realizado las reuniones para efectos de control y cumplimiento.
- Estado: Corresponde al estado o situación que se encuentra el desarrollo de la historia o tarea. Se colocará “Pendiente” cuando la tarea o historia no esté cumplida en su totalidad. Se colocará “Cumplida” cuando la tarea o historia esté 100% efectuada.

3.5 Diario de actividades

Es una herramienta muy útil pues nos permite anotar las diferentes actividades que se realizan durante todo el proceso de desarrollo, desde la recolección de información del cliente hasta la entrega final del proyecto. De mucha importancia pues permite alimentar de la experiencia que se comparte con el cliente, el equipo desarrollador, se

apunta los cambios continuos que sufren todos sus elementos y apuntan a la mejora continua. Se debe tomar en cuenta que se lo debe llevar el registro cada vez que se realice cualquier actividad y anotando cualquier observación que se suceda.

Seguidamente se sugiere la Tabla 32 para establecer el diario de actividades del equipo de trabajo del Departamento.

Tabla 32. Diario de Actividades

Nombre: [Nombre de quien la realiza...]

Equipo: [Nombre del equipo...]

Rol desempeñado:[Manager/Desarrollador/Tester...]

Fecha (dd/mes)	*Actividad Realizada*	*Tiempo Dedicado (en horas)*	*Observaciones*
18/02	Primera reunión con el cliente.	2	No sabemos lo que hacemos y es frustrante.
19/02	Reunión con el Trainer	0.5	Productiva
…			

- Fecha: Corresponde al día y mes que se efectúa la actividad.
- Actividad realizada: Se redactará corta y puntualmente la actividad que se realizó. Por ejemplo reuniones, revisiones, citas, sesiones, presentación de iteraciones, presentación de prototipos, etc.
- Tiempo dedicado: Se indicará el tiempo empleado para efecto de la actividad en horas.
- Observaciones: Las observaciones se refieren a cualquier comentario o incidencia asociado a la actividad asociada. Por ejemplo, inconvenientes o dificultades para desarrollar la actividad, percepción final de éxito o fracaso en el objetivo de la actividad, grado de motivación/desinterés, etc.

3.6 Diseño

3.6.1 Metáfora.

Una metáfora es una historia que todo el mundo puede contar acerca de cómo funciona el sistema. Se debe buscar unas frases o nombres que definan cómo funcionan

las distintas partes del programa, de forma que sólo con los nombres se pueda hacer una idea de qué es lo que hace cada parte del programa. Esto ayuda a que todos los programadores y el cliente, sepan de qué se está hablando y que no haya mal entendidos. Algunas veces podremos encontrar metáforas sencillas como por ejemplo: Programa de gestión de compras o ventas, gestión de cartera y almacén, etc. Las metáforas ayudan a cualquier persona a entender el objeto del programa.

3.6.2 Diseño sencillo.

Este diseño significa hacer siempre lo mínimo imprescindible de la forma más sencilla posible, manteniendo siempre sencillo el código.

El diseño adecuado para el software es aquel que:

- Funciona con todas las pruebas.
- No tiene lógica duplicada.
- Manifiesta cada intención importante para los programadores
- Tiene el menor número de clases y métodos.

En este apartado se diseñará mediante diferentes herramientas para la modelación, dependiendo de la arquitectura y/o lenguaje de programación que el equipo programador mantenga. Entre ellos se destacan los más comunes:

- **Modelo Funcional:** Es un instrumento que sirve a su propósito en forma adecuada y que deja satisfecho al utilizador. Un buen modelo funcional toma en cuenta todos los factores esenciales e ignora por completo los detalles superfinos. Por eso, es de suma importancia disponer de un propósito muy claro y preciso antes de comenzar a elaborar el modelo. Este Modelo se basará mediante la notación clásica de Diagramas de Flujo de Datos (DFD), señalando sus elementos fundamentales como:
  - Procesos
  - Flujos de datos (y en ocasiones, de control)
  - Entidades externas (Actores)
  - Almacenaje de datos

- **Modelo Entidad-Relación:** Esta herramienta permite representar las entidades relevantes de un sistema de información así como sus interrelaciones y

propiedades. Se ubicará el esquema gráfico del modelo y además los siguientes recursos:

- **Entidades:** Del modelo se identificará y ubicará todas las entidades con su nombre, función y sus atributos. Seguidamente se sugiere la Tabla 33 para ingreso de la entidad.

Tabla 33. Tabla Entidad

Entidad: [Nombre de la entidad]	**Función:** [Descripción corta de la funcionalidad de la entidad]
Atributo	**Descripción**
[Nombre del atributo1]	[Descripción corta del atributo]
[Nombre del atributo2]	[Descripción corta del atributo]
…	

- **Relaciones:** Hasta ahora solo se han mencionado las entidades y sus atributos, a continuación se mostrará la relación que hay entre ellos y su conectividad, expresando la conectividad de la siguiente manera:
  - Uno a Muchos = 1:M
  - Muchos a 1 = M:1
  - Muchos a Muchos = M:M
  - Etc.

  Seguidamente se sugiere la Tabla 34 para ingreso de las relaciones.

Tabla 34. Tabla Relaciones de un Sistema

Entidad	**Relación**	**Conectividad**	**Entidad**
[Entidad1]	[Descripción relación. Ej: Tiene, contiene, hecha por, etc]	[M:1/1:M/ M:M/etc]	[Entidad2]
[Entidad2]	[Descripción relación. Ej: Tiene, contiene, hecha por, etc]	[M:1/1:M/ M:M/etc]	[Entidad3]
…			

❖ **Modelo Relacional:** El Modelo Relacional se obtiene del Modelo Entidad Relación. Se presentaran las tablas que se obtuviesen solamente del nombre de la entidad y sus atributos**:**

Tabla 35. Ejemplo de una entidad y sus atributos

[ENTIDAD]	**TIPO**
[Atributo1]	[Tipo de Atributo]
[Atributo2]	[Tipo de Atributo]
…	

- Entidad: Hace referencia al nombre de la entidad. Ej: Empleado.
- Atributo: Hace referencia al atributo de la entidad. Ej: Cédula, nombre, etc.
- Tipo de atributo: Hace referencia al tipo del atributo. Ej: alfabético, numérico, alfanumérico, moneda, etc. (Char, varchar, float, double, etc.).

❖ **Modelado UML**: El Lenguaje de Modelado Unificado (UML) es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema, incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados. La programación orientada a objetos viene siendo un complemento perfecto de UML, pero no por eso se toma UML sólo para lenguajes orientados a objetos. Se utilizará los siguientes diagramas de acuerdo a la necesidad [64].

- Diagramas de Casos de Uso: Para modelar los procesos de negocio.
- Diagramas de Secuencia: Para modelar el paso de mensajes entre objetos.
- Diagramas de Colaboración: Para modelar interacciones entre objetos.
- Diagramas de Estado: Para modelar el comportamiento de los objetos en el sistema.
- Diagramas de Actividad: Para modelar el comportamiento de los Casos de Uso, objetos u operaciones.
- Diagramas de Clases: Para modelar la estructura estática de las clases en el sistema.
- Diagramas de Objetos: Para modelar la estructura estática de los objetos en el sistema.
- Diagramas de Componentes: Para modelar componentes.
- Diagramas de Implementación: Para modelar la distribución del sistema.

3.6.3 Glosario de Términos

Se ubicará un listado de términos que, dentro del proceso, el equipo desarrollar idee o conceptualice, siendo necesario su exposición para que cada miembro del equipo o el cliente logre su identificación. Seguidamente se sugiere la Tabla 36 para el ingreso del glosario de términos.

Tabla 36. Ejemplo de un Glosario de Términos en un sistema

Término	Significado
TIC	Tecnologías de Información y Comunicación
Sistema COND_ATROS	Sistema Control de **COND**ucta de **A**spirantes a **TRO**pa de la ETFA
HU	Historia de Usuario
DFD	Diagrama de Flujo de Datos
Nómina	Listado en general de artículos

3.6.4 Tarjetas CRC (Clase - Responsabilidad – Colaborador)

Estas tarjetas se dividen en tres secciones que contienen la información del nombre de la clase, sus responsabilidades y sus colaboradores. En la siguiente tabla se muestra cómo se distribuye esta información.

Tabla 37. Tarjetas CRC

Nombre de la Clase	
Responsabilidades	Colaboradores

Una clase es cualquier persona, cosa, evento, concepto, pantalla o reporte. Las responsabilidades de una clase son las cosas que conoce y las que realiza, sus atributos y métodos. Los colaboradores de una clase son las demás clases con las que trabaja en conjunto para llevar a cabo sus responsabilidades.

En la práctica conviene tener pequeñas tarjetas de cartón, que se llenarán y que son mostradas al cliente, de manera que se pueda llegar a un acuerdo sobre la validez de las abstracciones propuestas.

Los pasos a seguir para llenar las tarjetas son los siguientes:

- Encontrar clases

- Encontrar responsabilidades
- Definir colaboradores
- Disponer las tarjetas

Para encontrar las clases debemos pensar qué cosas interactúan con el sistema (por ejemplo un usuario), y qué cosas son parte del sistema, así como las pantallas útiles a la aplicación (un despliegue de datos, una entrada de parámetros y una pantalla general, entre otros). Una vez que las clases principales han sido encontradas se procede a buscar los atributos y las responsabilidades, para esto se puede formular la pregunta ¿Qué sabe la clase? y ¿Qué hace la clase? Finalmente se buscan los colaboradores dentro de la lista de clases que se tenga.

3.6.5 Diseño de Estilos de las Interfaces

A continuación se presentan los estilos predefinidos de las interfaces gráficas de usuario diseñadas como aplicación genérica para cualquier sistema software que se realice. Para su presentación se ha separado por las distintas partes funcionales de la aplicación.

a) Interfaz de Ingreso al Sistema

En el momento de ingresar al sistema software, se encontrará con la siguiente interfaz:

Figura 14. Estilos para Interfaz de Ingreso al Sistema

Fuente: (Los Autores)

Se definirá el siguiente diseño para la interfaz de ingreso de acuerdo a la Figura 14:

- Color fondo interfaz ingreso: #0866C6.
- Texto encabezado: fuente Verdana, tamaño 34 puntos, color #000000.
- Cajas de Texto: fuente texto Verdana, tamaño 14 puntos, color texto #999DC3, fondo cajas #FFFFFF, dimensión cajas 6 cm x 1cm.
- Botones de comando: fuente texto Verdana, tamaño 15 puntos, color texto #FFFFFF, color fondo #1E82E, dimensión botones 6 cm x 1cm.
- Check Box: fuente texto Verdana, tamaño 10 puntos, color texto #FFFFFF.
- Logotipo: Imagen logotipo ETFA genérico para toda aplicación. Tamaño 10cm x 8cm.

b) Interfaz de Inicio

En el momento de ingresar al sistema software, se encontrará con la siguiente interfaz:

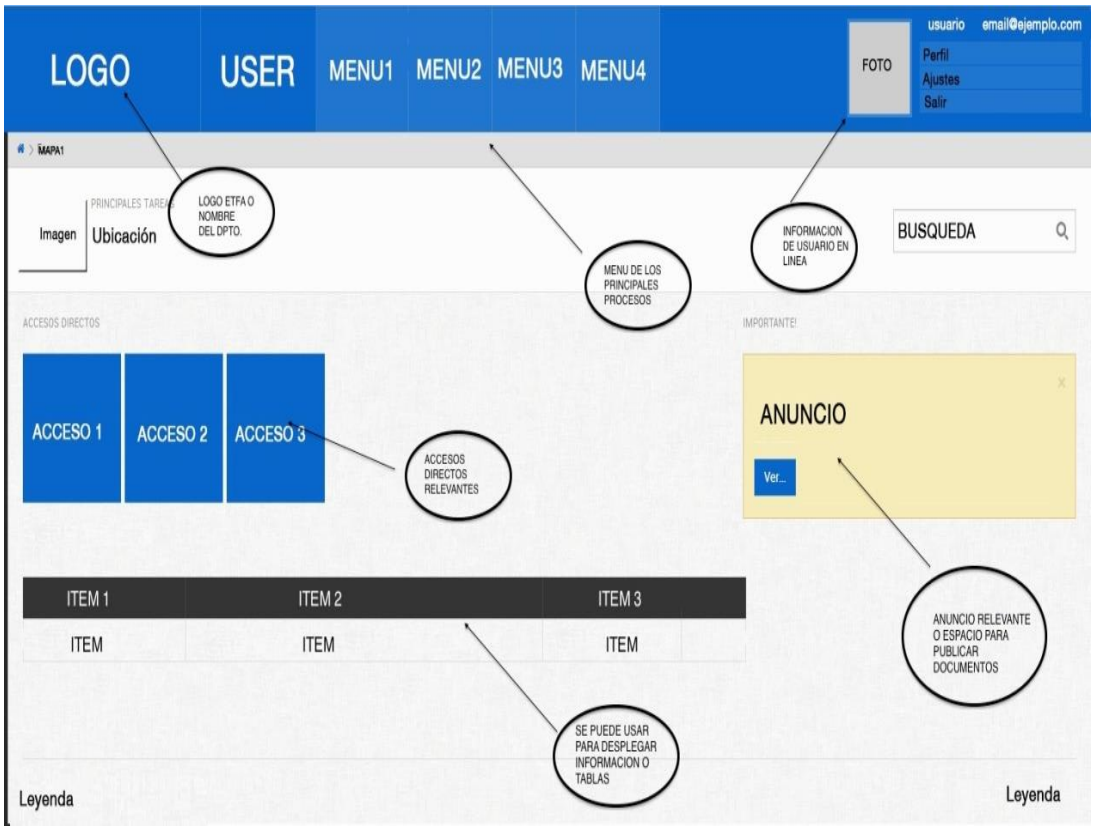


Figura 15. Estilos para Interfaz de Inicio

Fuente: (Los Autores)

Se definirá el siguiente diseño para la interfaz de inicio de acuerdo a la Figura 15:

Tabla 38. Estructura de diseño y características de Interfaz de Inicio

ESTRUCTURA DE DISEÑO	CARACTERÍSTICAS	SUB-ESTRUCTURA	ELEMENTOS Y CARACTERÍSTICAS
Encabezado	Color fondo: #0866C6 Texto: fuente Verdana, tamaño 32 puntos, color #000000	Área de navegación	• Menús • Imagen usuario • Información de usuario
Menú de posicionamiento	Color fondo: #000000 Texto: fuente Verdana, tamaño 14 puntos, color #000000	Caja de búsqueda	Color fondo interfaz ingreso: #0866C6 Cajas de Texto: fuente texto Verdana, tamaño 12 puntos, color texto #0866C7, fondo cajas #FFFFFF, dimensión cajas 4 cm x 0,6 cm. Botones de comando: fuente texto Verdana, tamaño 12 puntos, color texto #FFFFFF, color fondo #1E82E, dimensión botones 4 cm x0,6cm
		Imagen	Check Box: fuente texto Verdana, tamaño 10 puntos, color texto #FFFFFF. Logotipo: Imagen logotipo ETFA genérico para toda aplicación. Tamaño 10cm x 8cm.
Cuerpo	Color fondo: #000000 Texto: fuente Verdana, tamaño 14 a 20 puntos, color #00012E	Elementos auxiliares	Color fondo interfaz: #0866C6 Texto encabezado: fuente Verdana, tamaño 34 puntos, color #000000 Cajas de Texto: fuente texto Verdana, tamaño 14 puntos, color texto #999DC3, fondo cajas #FFFFFF, dimensión cajas 6 cm x 1cm. Botones de comando: fuente texto Verdana, tamaño 15 puntos, color texto #FFFFFF, color fondo #1E82E, dimensión botones 6 cm x 1cm Check Box: fuente texto Verdana, tamaño 10 puntos, color texto #FFFFFF.

c) Interfaz para Gestión de Procesos

En el momento de ingresar al sistema software, se encontrará con la siguiente interfaz:

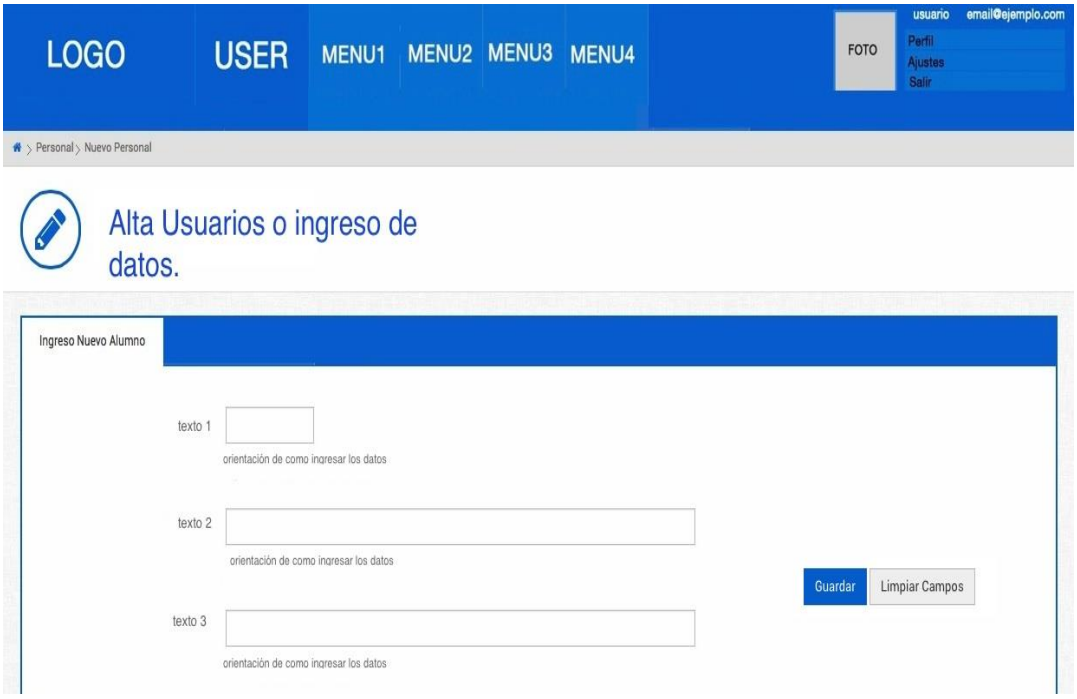


Figura 16. Estilos para Interfaz de Inicio

Fuente: (Los Autores)

Se definirá el siguiente diseño para la interfaz de gestión de acuerdo a la Figura 16:

Tabla 39. Estructura de diseño y características de Interfaz de Inicio

ESTRUCTURA DE DISEÑO	CARACTERÍSTICAS	SUB-ESTRUCTURA	ELEMENTOS Y CARACTERÍSTICAS
Encabezado	Color fondo: #0866C6 Texto: fuente Verdana, tamaño 32 puntos, color #000000	Área de navegación	• Menús • Imagen usuario • Información de usuario
Cuerpo	Color fondo: #000000 Texto: fuente Verdana, tamaño 14 a 20 puntos, color #00012E	Elementos auxiliares	Color fondo interfaz: #0866C6 Texto encabezado: fuente Verdana, tamaño 34 puntos, color #000000 Cajas de Texto: fuente texto Verdana, tamaño 14 puntos, color texto #999DC3, fondo cajas #FFFFFF, dimensión cajas 6 cm x 1cm. Botones de comando: fuente texto Verdana, tamaño 15 puntos, color texto #FFFFFF, color fondo #1E82E, dimensión botones 6 cm x 1cm Check Box: fuente texto Verdana, tamaño 10 puntos, color texto #FFFFFF.

3.7 Codificación

3.7.1 Requerimientos Tecnológicos

En este sub apartado se detallarán las herramientas necesarias para el desarrollo en la implementación del sistema software. La descripción de estos requerimientos dependerá del grado de conocimiento que tenga el equipo desarrollador sobre ciertas herramientas, pues siempre se encontrará diversidad en los mismos; es decir, un integrante tendrá conocimientos sobre programación en objetos, otro sabrá programación web, etc. A pesar de los antagonismos entre los integrantes, lo importante es acordar las herramientas que sea de dominio general para el equipo desarrollador.

Tomando en cuenta la experiencia del Departamento TIC ETFA, se sugiere las herramientas a continuación señaladas.

Software requerido para la ejecución del sistema software:

- Servidor de aplicaciones web
  - Apache,
  - Tomcat.
- Sistema gestor de base de datos relacional
  - MySQL
- Navegador web
  - Mozilla,
  - Internet Explorer,
  - Google Chrome.
- Tecnologías
  - PHP,
  - JSP,
  - HTML
  - jQuery, etc.

3.7.2 Recodificación

En este sub apartado se detallará todas las incidencias y/o comportamientos que se han generado dentro del equipo programador, teniendo presente las siguientes consideraciones:

- Cuando implementamos nuevas características en nuestros programas nos planteamos la manera de hacerlo lo más simple posible, después de implementar esta característica, nos preguntamos cómo hacer el programa más

simple sin perder funcionalidad, este proceso se le denomina recodificar o refactorizar (refactoring).

- La recodificación puede llevar a hacer más trabajo del necesario, pero a la vez se estará preparando al sistema para que en un futuro acepte nuevos cambios y pueda albergar nuevas características.
- No se debe de recodificar ante especulaciones, sino solo cuándo el sistema lo pida.

3.7.3 Programación por parejas

Para la programación en parejas como su nombre lo indica los dos integrantes del equipo programador realizaran las tareas asignadas (generalmente proceso de análisis, diseño, implementación y pruebas).

En este sub apartado se detallará todas las incidencias y comportamientos que se han generado dentro del equipo programador, tomando en cuenta que:

- La producción de código procurará ser hecha en parejas frente a un único computador.
- Al trabajar en parejas se tiene un diseño de mejor calidad y un código más organizado, sin impactar el tiempo para cumplir lo prometido. Cada miembro de la pareja juega su papel: uno codifica en el ordenador y piensa la mejor manera de hacerlo, el otro piensa más estratégicamente, ¿Va a funcionar?, ¿Puede haber pruebas donde no funcione?, ¿Hay forma de simplificar el sistema global para que el problema desaparezca?
- Al trabajar en parejas se solucionan los problemas más fácilmente. Se posibilita la transferencia de conocimientos de programación entre los miembros del equipo, varias personas entienden las diferentes partes del sistema, los programadores conversan mejorando así el flujo de información y la dinámica del equipo, y finalmente, los programadores disfrutan más su trabajo. Dichos beneficios se consiguen después de varios meses de practicar la programación en parejas.

Seguidamente se sugiere la Tabla 40 para el ingreso de la información de la programación en parejas, para cada una de las actividades que se desarrollarán, principalmente relacionado a la creación de las historias y/o iteraciones.

Tabla 40. Tabla para programación en parejas

Proceso	Actividad	Tareas	Responsable	Tiempo
Análisis	[Descripción de actividad]	[Historia No...]	[Nombres]	[Horario..]
Diseño	[Descripción de actividad]			[Horario..]
Implementación	[Descripción de actividad]			[Horario..]
Pruebas	[Descripción de actividad]			[Horario..]

3.7.4 Propiedad colectiva

Esta es una práctica que el equipo desarrollador del Departamento TIC debe inculcar el valor significativo de la propiedad colectiva sobre el código, principalmente entre sus programadores. Cualquiera que crea que puede aportar valor al código en cualquier parcela puede hacerlo, ningún miembro del equipo es propietario del código. Si alguien quiere hacer cambios en el código puede hacerlo. Si hacemos el código propietario, y necesitamos de su autor para que lo cambie entonces estaremos alejándonos cada vez más de la comprensión del problema, si necesitamos un cambio sobre una parte del código lo hacemos. Esto preparará al equipo para la sustitución no traumática de cada miembro.

3.7.5 Integración continua

Es una práctica adicional. El equipo desarrollador del Departamento TIC deberá integrar el código como mínimo una vez al día, y realizar las pruebas sobre la totalidad del sistema. Una pareja de programadores se encargará de integrar todo el código en una máquina y realizar todas las pruebas hasta que estas funcionen al 100%.

3.7.6 Cliente In-situ (Cliente siempre presente)

Esta es otra práctica esencial para el desenvolvimiento del proyecto. Un cliente real (jefe, supervisor o encargado de algún Departamento de la ETFA) debe sentarse con el equipo de programadores, estar disponible para responder a sus preguntas, resolver discusiones y fijar las prioridades. Lo difícil es que el cliente ceda una persona que conozca el negocio para que se integre en el equipo, normalmente estos elementos son muy valiosos, pero se debe hacerles ver que será mejor para su departamento o

Institución tener un software pronto en funcionamiento, y esto no implica que el cliente no pueda realizar cualquier otro trabajo.

3.7.7 Estándares de código

El equipo de programadores define en forma consensuada los estándares de programación que utilizará. De esta manera se logra obtener un código uniforme, como si el sistema fuera programado por una sola persona. Esta práctica fomenta la comunicación y facilita la implementación de las prácticas de refactoring[7], programación en pareja y propiedad colectiva (ver ANEXO B).

3.7.8 Desarrollo de la Codificación

En este sub apartado, se sugiere enunciar la herramienta utilizada con la codificación respectiva:

- Primero, enunciando la generación de código de la base de datos, creación de tablas a través de sentencias SQL, el código de las conexiones entre las mismas, etc.
- Y segundo, con la herramienta y el lenguaje de programación acordado por el equipo, mostrar todo o parte del código, guiándose en relación a los procesos (gestión) del proyecto o a su vez, en relación a la mejor presentación de la codificación respectiva que determine el equipo de trabajo del Departamento TIC.

3.8 Pruebas

Una vez finalizada la codificación, el equipo programador del Departamento TIC procederá a la creación de las pruebas para conocer la efectiva funcionalidad del sistema desarrollado, según los requerimientos iniciales y el cumplimiento exacto de estos en el proceso del sistema software. Cabe señalar que el diseño de pruebas se realizará para todas las partes del sistema como una práctica para garantizar el buen funcionamiento.

[7] Técnica de la ingeniería de software para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.

Para esto se realizará el siguiente plan de pruebas basado en las pruebas más comunes de la metodología XP:

- Pruebas unitarias
- Pruebas de aceptación

3.8.1 Pruebas Unitarias

Estas pruebas se realizan para controlar el funcionamiento de un módulo (unidad de código) como puede ser subprogramas o métodos. Las pruebas unitarias aseguran que un determinado módulo cumpla con un comportamiento esperado en forma aislada antes de ser integrado al sistema.

Generalmente son realizadas por los mismos programadores, puesto que conocen con mayor detalle el código. Los programadores realizan estas pruebas cuando: la interfaz de un método no es clara, la implementación es complicada, para testear entradas y condiciones inusuales, luego de modificar algo. Éstas deben contemplar cada módulo del sistema que pueda generar fallas. Para poder integrar el código realizado al ya existente, el mismo debe aprobar satisfactoriamente todos los casos de prueba definidos. En XP los programadores deben escribir las pruebas unitarias para cada módulo antes de escribir el código. No es necesario escribir casos de prueba para todos los módulos, sólo para aquellos en que exista la posibilidad de que puedan fallar. Luego de escribir el código, los programadores ejecutan las pruebas, las cuales deben resultar 100% efectivas para que el código pueda integrarse al sistema. En caso contrario hay que solucionar los errores y ejecutar nuevamente los casos de prueba hasta lograr que ninguno de ellos.

Las pruebas son automatizadas utilizando herramientas como XUnit[8], de forma tal de poder soportar un testing[9] continuo y mantener organizados los casos de pruebas. La ausencia de las pruebas unitarias lleva a tener que invertir una gran cantidad de horas en sesiones de debugging[10] al momento de integrar el código con el sistema existente.

Las pruebas unitarias brindan una inmediata retroalimentación a la realización del proyecto, permiten al programador saber si una determinada funcionalidad se puede agregar al sistema existente sin alterar el funcionamiento actual del mismo. También

[8] Frameworks de pruebas conducidos por código.
[9] Pruebas de software.
[10] Depuración, corrección de errores o bugs.

permiten la aplicación de otras prácticas como refactoring y diseño simple al estar respaldados por efectivos casos de prueba.

Para el efecto de las mismas, se propone un esquema de pruebas unitarias de acuerdo al ANEXO C, apartado a).

a) Herramientas para automatizar las pruebas unitarias

En los últimos años se han creado una serie de frameworks[11] que permiten automatizar las pruebas unitarias, permitiendo definir estas y ejecutarlas en reiteradas ocasiones. Estos frameworks son denominados xUnit.

Estos frameworks se basan en los conceptos de Test Case y Test Suite, el primer es la prueba unitaria del componente a testear, la segunda nos permite agrupar varias pruebas unitarias para poder ejecutarlas en forma conjuntas.

La finalidad de los Test Case es probar funcionalidad de un determinado modulo. En lenguajes orientados a objetos seria probar los métodos de una clase. Los Test Suite permiten agrupar todas las pruebas de un componente, permitiendo ejecutar de forma conjunta todas las pruebas definidas para un componente.

3.8.2 Pruebas de aceptación

Las pruebas de aceptación, también llamadas pruebas funcionales, son pruebas de caja negra definidas por el cliente para cada historia de usuario, y tienen como objetivo asegurar que las funcionalidades del sistema cumplen con lo que se espera de ellas. En efecto, las pruebas de aceptación corresponden a una especie de documento de requerimientos según XP, ya que marcan el camino a seguir en cada iteración, indicándole al equipo de desarrollo hacia donde tiene que ir y en qué puntos o funcionalidades debe poner el mayor esfuerzo y atención.

Las pruebas de aceptación tienen una importancia crítica para el éxito de una iteración. Por lo tanto el tester debe tenerlas prontas lo antes posible a partir del comienzo de la iteración, y lograr que el cliente las apruebe para poder presentárselas cuanto antes al equipo de desarrollo.

Para el efecto de las mismas, se propone un esquema de pruebas de aceptación de acuerdo al ANEXO C, apartado b).

[11] Marco de trabajo, conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar

a) El rol del cliente

Según XP, las pruebas de aceptación son en última instancia responsabilidad del cliente ya que deben reflejar los requerimientos y las funcionalidades que se quieren obtener en cada iteración. Es más, en algunos proyectos realizados, el cliente llega a escribir las pruebas de aceptación.

Sin embargo la mayoría de los clientes, especialmente cuando son externos al proyecto, son personas muy ocupadas con trabajos de tiempo completo, por lo tanto no disponen de demasiado tiempo para dedicarle al proyecto. Además generalmente no tienen ni la experiencia ni los conocimientos técnicos necesarios para escribir correctamente las pruebas de aceptación. Por esta razón el tester debe reunirse con el cliente para interpretar sus ideas y sus necesidades y luego poder escribir los casos de prueba correspondientes. También es importante que el tester guíe al cliente en la definición de los criterios de calidad, realizando diferentes cuestionarios e identificando lo que es crucial para él.

b) Criterios de aprobación

En XP las pruebas de aceptación cumplen con el objetivo de indicarnos cuando las funcionalidades de una iteración han sido completadas exitosamente. A diferencia de las pruebas unitarias, el criterio de aprobación de las pruebas de aceptación no tiene que ser necesariamente de 100% de efectividad. Todos sabemos que es imposible esperar un código totalmente libre de errores, por lo tanto es necesario definir un criterio de aprobación para saber cuándo el software está listo para ser liberado.

El cliente podría demandar que el 100% de los casos de prueba sean exitosos para pasar de iteración. Sin embargo como en XP las iteraciones tienen que terminar en fecha y los tiempos límites de entrega no son negociables, esto provocaría un tiempo de estimación mayor para cada historia de usuario. Por lo tanto, generalmente se sugiere que el cliente incluya pruebas de aceptación no críticas. Si estas pruebas se superan exitosamente generan una cuota extra de confianza en el cliente, pero si las mismas fallan en el fin de una iteración el cliente puede decidir repetir dichas pruebas al final de la siguiente iteración aumentando el grado de criticidad de las mismas.

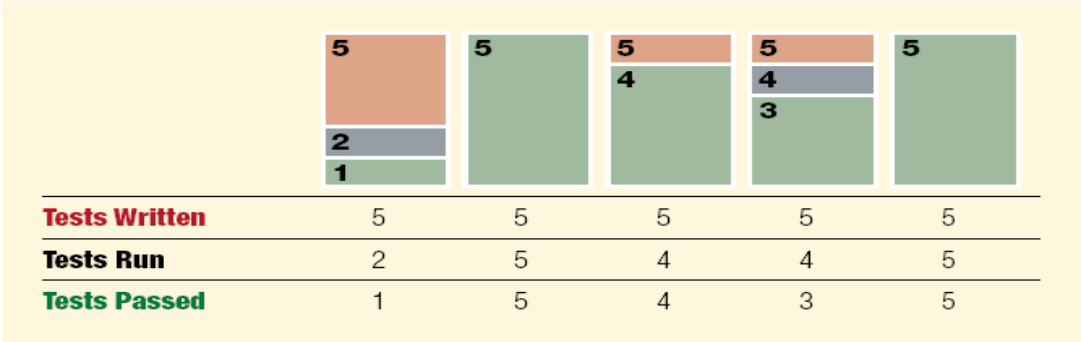
c) Definición de los casos de prueba

Para escribir los casos de prueba se deben tener en cuenta ciertas consideraciones importantes. En primer lugar cada caso debe servir para obtener un feedback[12] rápido y concreto de cómo se está desarrollando la iteración y el proyecto. Se tienen que tratar de evitar los casos de pruebas extensos que incluyan un gran número de pasos. Los casos escritos deben ser concisos y hay que documentar por separado los pasos del caso y los datos de prueba en sí mismos. Es importante señalar también que todos los casos de prueba cumplidos con éxito en iteraciones anteriores se deben seguir cumpliendo en todas las iteraciones, y si se produce un error aunque sea en un único paso del caso de prueba se considera que todo el caso falló.

d) Presentación de los resultados de las pruebas de aceptación

Otra importante diferencia entre las pruebas de aceptación y las pruebas unitarias es que para las de aceptación la presentación de los resultados es importante, en cambio para las unitarias no tiene mucho sentido ya que siempre se requiere un 100% de efectividad. Beck (2000) recomienda la exhibición de los resultados que se obtienen al ejecutar las pruebas de aceptación, generando reportes y gráficas que desplieguen los porcentajes de efectividad obtenidos. Estos índices permiten evaluar si el equipo de desarrollo está realizando un buen trabajo o no. Es importante mantener esta información estadística actualizada y visible para todos los integrantes del proyecto.

Crispin (2001) propone el siguiente cuadro de resumen donde presenta tanto gráfica como numéricamente los casos de pruebas escritos, ejecutados y exitosos.



Tests Written	5	5	5	5	5
Tests Run	2	5	4	4	5
Tests Passed	1	5	4	3	5

Figura 17. Cuadro de resumen de los casos y resultados de las pruebas de aceptación
Fuente: (Crispin, 2001) [65]

[12] Retroalimentación en los procesos

Otro ejemplo presentado en Jeffries 1999 intenta mostrar la cantidad de casos de pruebas de aceptación ejecutados en cada iteración, diferenciándolos entre los casos exitosos, los casos que fallaron y los que no fueron validados por el cliente.

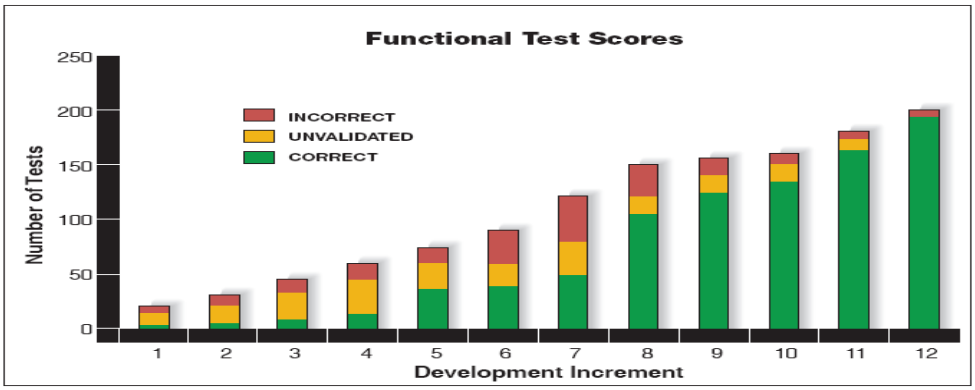


Figura 18. Resultados de las pruebas de aceptación al final de cada iteración

Fuente: (Jeffriess, Ron 1999) [66]

En la Figura 17 se observa que hay una clara tendencia que indica que en cada iteración se incrementen los casos escritos y ejecutados con respecto a la anterior. Además para que el proyecto culmine de forma exitosa es necesario que en las iteraciones finales los casos con error y los no validados por el cliente disminuyan, llegando a la última iteración con el 100% de los casos validados.

4ª FASE: PRODUCCIÓN

Esta fase corresponde la realización de las pruebas necesarias previo al traslado del sistema al entorno del cliente. Si bien al final de cada iteración se entregan módulos funcionales y sin errores, puede ser deseable por parte del cliente no poner el sistema en producción hasta tanto no se tenga la funcionalidad completa.

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase.

Es posible que se rebaje el tiempo que toma cada iteración, de tres a una semana. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación (por ejemplo, durante la fase de mantenimiento).

En resumen, el equipo de trabajo del Departamento tomará en cuenta:

- Realizar revisiones de rendimiento del entregable.

- Analizar la inclusión de nuevas características a la versión actual debido a cambios en las historias de usuario.
- Documentar ideas y sugerencias propuestas para ser implementadas durante la fase de mantenimiento.

5ª FASE: MANTENIMIENTO

Esta fase se concentra en la realización de tareas de soporte al cliente. Se efectúa luego de que el primer entregable se encuentra en producción; es decir, mientras la primera versión se encuentra en producción, el proyecto debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción.

En resumen, el equipo de trabajo del Departamento tomará en cuenta que:

- Los programadores se encargan de mantener el sistema puesto en producción en funcionamiento.
- El registro de velocidad del proyecto normalmente marcará una baja en la velocidad debido a estas tareas de soporte.
- Adicional, la fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

6ª FASE: MUERTE DEL PROYECTO

Esta fase corresponde a la última dentro del ciclo de vida del proyecto, en la cual se busca esencialmente satisfacer las necesidades del cliente en aspectos de rendimiento y confiabilidad del sistema.

La muerte del proyecto ocurrirá cuando:

- El cliente no tiene más historias o dentro del proyecto en desarrollo no existen más historias para ser incluidas en el sistema.
- El sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo.

Para lo cual el equipo de trabajo del Departamento:

- Generará la documentación final del sistema
- No realizan más cambios en la arquitectura.

4.5 Conclusiones del Capítulo

La implementación de una metodología ágil al Departamento TIC resulta imprescindible, pues la falta de un marco de trabajo, técnicas, métodos o metodologías a seguir incurría en tiempos tardíos de entrega del software y en consecuencia la insatisfacción de los diferentes usuarios en la ETFA. La implementación de la misma, es el resultado de un estudio previo, adaptando artefactos y principios de las metodologías ágiles más importantes como DSDM, SCRUM y principalmente XP, a las necesidades específicas del Departamento.

Es notable que la planificación previa y sistemática de la metodología implementada, organiza y orienta al equipo desarrollador, y más que nada, los reajustes que se pueden realizar en etapas tempranas, hacen que la metodología sea ideal y adaptable a los cambios. Además, la participación activa del cliente/usuario permite optimizar los recursos que se utiliza para el desarrollo del sistema software.

La experiencia alcanzada, mediante la aplicación de la metodología ágil dadas las características del problema del caso de estudio práctico, resultó satisfactoria, arrojando resultados positivos principalmente en términos de satisfacción del cliente (usuarios del sistema del o los Departamentos de la ETFA); cumpliendo con los plazos establecidos y creando un buen ambiente de trabajo del equipo. La aplicación de mencionado caso de estudio, permitió concluir que la metodología se adapta a las necesidades del cliente, a las características del problema, al entorno de trabajo y a las características de los desarrolladores. Inicialmente, se pensaba que al estar en una Institución jerárquica por ser militar, podría existir cierta barrera en tanto a la comunicación dentro del equipo y con el cliente; pero en realidad la metodología más que una barrera, se constituyó en una poderosa herramienta de trabajo, al disponer de una serie de principios y prácticas, que en cualquier Institución podría adaptarse sin grandes esfuerzos.

CAPÍTULO 5
VALIDACIÓN DE LA METODOLOGÍA

En el presente capítulo se desarrollará el análisis de los resultados obtenidos al validar la implementación de la metodología ágil en el Departamento TIC de la ETFA. Para la valoración de los resultados se ha elaborado un instrumento de recopilación de información (encuesta), la misma que pretende evaluar los objetivos planteados inicialmente, que implica la optimización del proceso de desarrollo del sistema software.

Para el análisis y obtención de los resultados se trabajará con tres grupos de personas, quienes integran el equipo de trabajo del Departamento TIC de la ETFA. En tal sentido, la investigación se orientará de la siguiente manera [67]:

- Población (Departamento TIC de la ETFA)
- Muestra (Equipo de trabajo)
    - Encuesta orientada al primer grupo de personas (Jefe del proyecto, Consultor), previa aplicación de la Metodología "XP TIC´s".
    - Encuesta orientada al segundo grupo de personas (Programador, Tester), según la experiencia obtenida con la aplicación de la Metodología "XP TIC´s".
    - Encuesta orientada al tercer grupo de personas (Jefe del Proyecto, Tracker, Entrenador), de los gestores del proyecto.

5.1 Procesamiento de datos y corroboración de los resultados

A continuación se presenta la estructura del instrumento elaborado para la recolección de datos:

- Valoración de resultado entre 1 y 5 para cada pregunta.

    1.- Ocasionalmente (en escasas ocasiones)
    2.-Ordinariamente (en ocasiones puntuales)
    3.- Frecuentemente (en la mayor parte)
    4.- Muy Frecuentemente (en casi todas las ocasiones)
    5.- Siempre

- Se marcará con una equis (X) el valor seleccionado.

PREGUNTA	Valor				
	1	2	3	4	5
Pregunta No…					

- Se sumará las equis (X) por cada columna respectivamente.
- El resultado se multiplicará por el valor que se indica para la columna, obteniendo de esta forma el total del puntaje de la columna.
- Se sumará los totales de cada columna y se dividirá para el valor que resulta de multiplicar el número de preguntas cerradas de la encuesta (tres tipos de encuesta) por el valor más alto de la misma.
- Los instrumentos constan de preguntas cerradas y abiertas las cuales permitirán conocer el criterio de los encuestados en base a la metodología ágil establecida, originando una idea general de su aplicación en el ámbito de trabajo en el Departamento TIC.

Para la interpretación, se ha dividido a los resultados en cuatro grupos de puntajes, que abarcan el 100% de los datos. Estos grupos se han distribuido en un intervalo de porcentajes, según una estimación objetiva a fin de analizar los resultados que se obtendrán de la aplicación de los instrumentos de medición (encuestas) y se identificarán al grupo que pertenezcan y de esta manera se verificará el nivel de aceptación de la Metodología propuesta dentro del equipo de trabajo del el Departamento TIC.

A continuación se enuncia la distribución, tal como se muestra a continuación:

De 0% a 39,99%: La metodología propuesta no se cumple o en su defecto se cumple en aspectos parciales o tiene una confianza muy baja con las actividades realizadas, y deben tomarse acciones correctivas con extrema urgencia y de manera general para implantar una metodología óptima.

De 40% a 59,99%: La metodología propuesta se cumple, pero con falencias en cuanto al proceso de desarrollo del software o a la continuidad y sistemática de su cumplimiento, o tiene una confianza baja con las actividades realizadas. Se tomará acciones correctivas con urgencia para implantar una metodología óptima.

De 60% a 85,99%: La metodología propuesta se cumple, pero con leves inconvenientes. Las falencias se solucionarán a corto plazo, para que la metodología no deje de ser óptima. La tendencia a establecer una metodología al proceso de

desarrollo software es muy positiva. Es necesario analizar las preguntas sobresalientes, como también dar el énfasis respectivo a las preguntas con más baja puntuación.

De 86% a 100%: La metodología propuesta optimizará el proceso de desarrollo del sistema software, como un marco de trabajo estándar para la producción de software de calidad en la ETFA.

La aplicación de tales instrumentos constituye un método empírico, cuyos resultados arrojados se recolectarán en tablas y por cada caso se hace la valoración respectiva a fin de documentar, verificar y validar las evidencias que demuestren la optimización del proceso de desarrollo del sistema software.

Los instrumentos arrojarán resultados cuantitativos, los cuales permitirán analizar parámetros sobre la implementación de la metodología de desarrollo ágil y los aspectos generales propios de la optimización del análisis, diseño e implementación del sistema software; con lo que se conocerá la óptica de los encuestados frente a la metodología ágil propuesta.

5.1.1 Procesamiento, análisis y resultados de la aplicación del instrumento previa aplicación de la Metodología "XP TIC´s" (Jefe del proyecto, Consultor).

A continuación se presenta el análisis e interpretación de la información recopilada del instrumento previa la aplicación de la Metodología, tal como se muestra en el ANEXO D (Encuesta No.1). Para el efecto se contó con la colaboración del Jefe del Proyecto y Consultor del equipo de trabajo.

Para el cálculo de estos valores se dividió la suma total de puntos obtenidos (ST) para el valor que resulta de multiplicar el número total de preguntas cerradas de la encuesta por el valor más alto de la misma, y multiplicar por 100%. Se debe tomar en cuenta que el instrumento se aplica a dos personas, por tanto el número total de preguntas es el doble. Estos cálculos se observan en la Tabla 41, y finalmente se muestran los resultados mediante una incorporación de los mismos en la Figura 19.

Tabla 41. Resultados de aceptación previa aplicación de la Metodología "XP TIC´s"

Valor	1	2	3	4	5
Total de (X)	**0**	**0**	**6**	**6**	**14**
Multiplicación	*1	*2	*3	*4	*5
Resultado parcial	**0**	**0**	**18**	**24**	**70**
Suma total de puntos obtenidos (ST)					**112**
RESULTADO [(ST/(26*5))*100%]	**86,15 % de aceptación**				

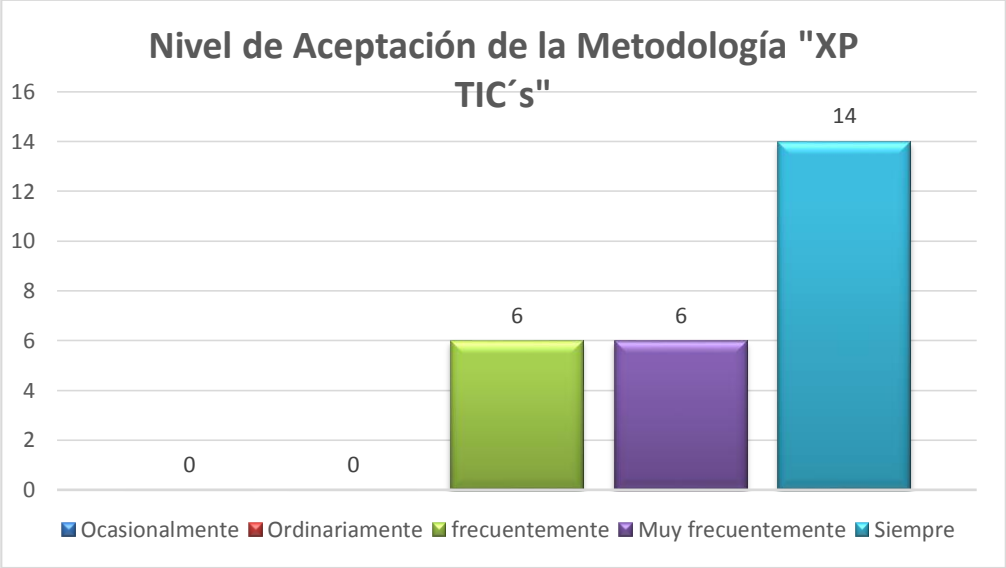


Figura 19. Nivel de Aceptación previa aplicación de la Metodología Ágil "XP TIC´s" en el Departamento TIC de la ETFA

Fuente: (Los Autores)

La valoración proporcionada por el Jefe del Proyecto y Consultor, previa aplicación de la Metodología Ágil "XP TIC´s", alcanzó un 86,15% de aceptación. Por tal motivo, se concluye en este sub apartado, que la Metodología propuesta es necesaria, tendiente a disminuir tiempos de entrega y costos excesivos, a establecer un marco de trabajo lógico y ordenado con herramientas válidas para un buen desempeño del equipo de trabajo; la misma que se aplicaría siempre en el proceso de desarrollo del sistema software en el Departamento TIC de la ETFA.

5.1.2 Procesamiento, análisis y resultados de la aplicación del instrumento según la experiencia obtenida con la aplicación de la Metodología "XP TIC´s" (Programador, Tester).

A continuación se presenta el análisis e interpretación de la información recopilada del instrumento previa la aplicación de la Metodología, tal como se muestra en el ANEXO D (Encuesta No.2). Para el efecto se contó con la colaboración del Programador y Tester del equipo de trabajo.

Para el cálculo de estos valores se aplica la misma temática de evaluación del sub apartado 5.1.1 al contar con dos personas encuestadas. Estos cálculos se observan en la Tabla 42 de la experiencia obtenida con la Metodología "XP TIC´s", y finalmente se muestran los resultados mediante una incorporación de los mismos en la Figura 20.

Tabla 42. Resultados de aceptación según la experiencia obtenida

Valor	1	2	3	4	5
Total de (X)	**0**	**0**	**0**	**12**	**18**
Multiplicación	*1	*2	*3	*4	*5
Resultado parcial	**0**	**0**	**0**	**48**	**90**
Suma total de puntos obtenidos (ST)					**138**
RESULTADO [(ST/(30*5))*100%]	**92,00 % de aceptación**				

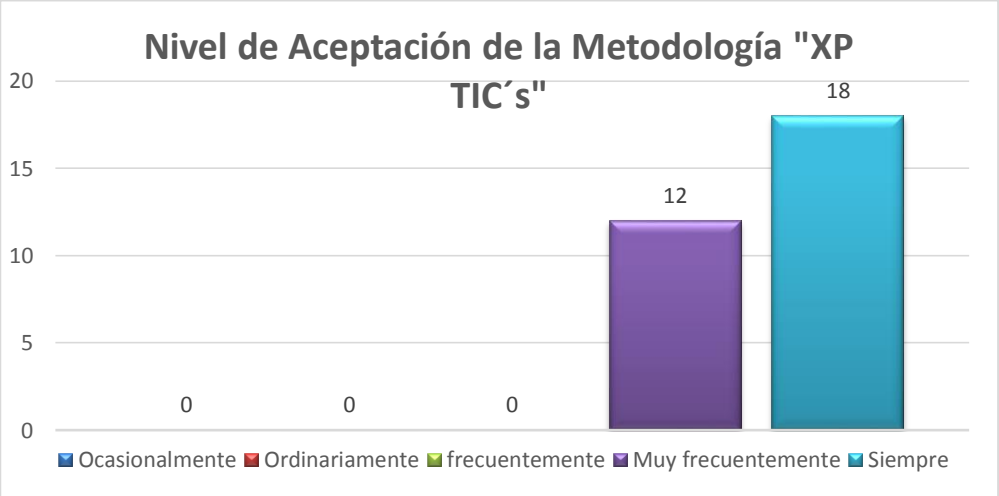


Figura 20. Nivel de Aceptación según la experiencia obtenida de la aplicación de la Metodología Ágil "XP TIC´s" en el Departamento TIC de la ETFA

Fuente: (Los Autores)

Los resultados obtenidos por el Programador y Tester (desarrolladores), según la experiencia obtenida mediante la aplicación de la Metodología Ágil "XP TIC´s", alcanzaron un 92 % de aceptación. Por tal motivo, se concluye en este sub apartado, que la Metodología propuesta es necesaria, tendiente a disminuir tiempos de entrega y costos excesivos, a establecer un marco de trabajo lógico y ordenado con herramientas válidas para un buen desempeño del equipo de trabajo; la misma que aplicaría siempre el equipo desarrollador en el proceso de desarrollo del sistema software en el Departamento TIC de la ETFA.

5.1.3 Procesamiento, análisis y resultados de la aplicación del instrumento a los gestores del proyecto (Jefe del Proyecto, Tracker, Entrenador).

Seguidamente se presenta el análisis e interpretación de la información recopilada del instrumento previa la aplicación de la Metodología, tal como se muestra en el ANEXO D (Encuesta No.3). Para el efecto se contó con la colaboración del Jefe del Proyecto, Tracker y Entrenador del equipo de trabajo.

Para el cálculo de estos valores se aplica la misma temática de evaluación del sub apartado 5.1.1 al contar con tres personas encuestadas, los mismos quienes realizan la gestión y administración pertinente dentro del proceso de desarrollo. Estos cálculos se observan en la Tabla 43, y finalmente se muestran los resultados mediante una incorporación de los mismos en la Figura 21.

Tabla 43. Resultados de aceptación según los gestores del proyecto con la Metodología "XP TIC´s"

Valor	1	2	3	4	5
Total de (X)	**0**	**0**	**0**	**12**	**36**
Multiplicación	*1	*2	*3	*4	*5
Resultado parcial	**0**	**0**	**0**	**48**	**180**
Suma total de puntos obtenidos (ST)					**228**
RESULTADO [(ST/(48*5))*100%]	**95,00 % de aceptación**				

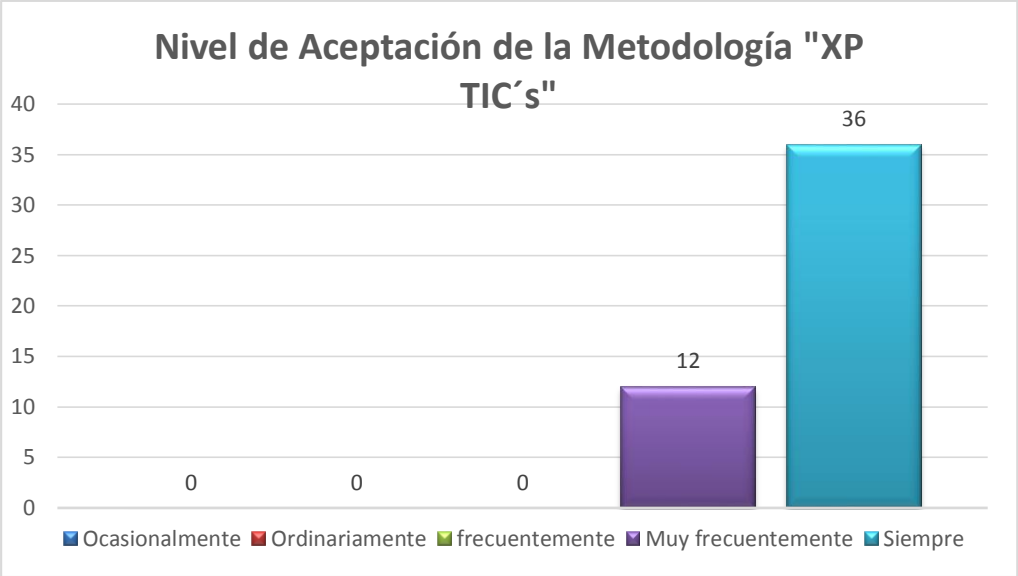


Figura 21. Nivel de Aceptación según los Gestores del Proyecto de la Metodología Ágil "XP TIC´s" en el Departamento TIC de la ETFA
Fuente: (Los Autores)

Los resultados obtenidos por el del Jefe del Proyecto, Tracker y Entrenador (gestores), según la experiencia obtenida mediante la aplicación de la Metodología Ágil "XP TIC´s", alcanzaron un 95 % de aceptación. Por tal motivo, se concluye en este sub apartado, que la Metodología propuesta es necesaria, tendiente a establecer una correcta e importante planificación inicial del proyecto, como también la planificación en cada uno de las fases del proceso de desarrollo software, estimación de costos, esfuerzo y recursos; la misma que aplicaría siempre los gestores del proyecto para una correcta gestión de cualquier proyecto software en el Departamento TIC de la ETFA.

5.2 Prueba de Hipótesis con Chi Cuadrado

5.2.1 Planteamiento de la Hipótesis

Mediante la prueba de hipótesis con Chi Cuadrado se realizará la comparación de dos atributos claramente identificados para determinar una posible relación entre ellos, de tal forma que permita conocer si una variable depende de la otra para su cumplimiento, estableciendo a continuación las variables propósito del estudio.

a) **Hipótesis de Investigación:** Si se implementa una metodología de desarrollo ágil, entonces se optimiza el análisis, diseño e implementación del sistema software, en el Departamento de Tecnologías de Información y Comunicaciones de la Escuela Técnica de la Fuerza Aérea de la ciudad de Latacunga, durante el período abril – diciembre del 2013.

b) **Variable Independiente:** Implementación de una Metodología de desarrollo ágil.

c) **Variable Dependiente:** Optimización del análisis, diseño e implementación del sistema software, en el Departamento de Tecnologías de Información y Comunicaciones de la Escuela Técnica de la Fuerza Aérea de la ciudad de Latacunga, durante el período abril – diciembre del 2013.

- **Hipótesis Nula (Ho):** La implementación de una metodología de desarrollo ágil y la optimización del análisis, diseño e implementación del sistema software, son independientes.
- **Hipótesis Alternativa (Ha):** La implementación de una metodología de desarrollo ágil y la optimización del análisis, diseño e implementación del sistema software, son dependientes.

5.2.2 Cálculos de frecuencias esperadas, correspondientes a cada frecuencia observada

a) Frecuencia Observada

Seguidamente se representan los datos correspondientes a las variables independiente y dependiente de los resultados obtenidos de las encuestas realizadas al equipo de trabajo del Departamento TIC de la ETFA, los mismos que se encuentran en el ANEXO E.

En la Tabla 44 se muestra la valoración de la primera variable, mientras en la Tabla 45 se muestra la valoración de la segunda variable.

Tabla 44. Variable Metodología de desarrollo ágil

Valoración	Previa aplicación	Según la experiencia obtenida	Gestores del Proyecto	Total
1.- Ocasionalmente	0	0	0	0
2.- Ordinariamente	0	0	0	0
3.- Frecuentemente	0	0	0	0
4.- Muy frecuentemente	3	3	5	11
5.- Siempre	5	7	13	25

Tabla 45. Variable optimización del análisis, diseño e implementación del sistema software

Valoración	Previa aplicación	Según la experiencia obtenida	Gestores del Proyecto	Total
1.- Ocasionalmente	0	0	0	0
2.- Ordinariamente	0	0	0	0
3.- Frecuentemente	6	0	0	6
4.- Muy frecuentemente	3	9	7	19
5.- Siempre	9	11	23	43

b) Frecuencia Esperada

En la Tabla 46 se muestra la intersección entre los datos de las Variables Metodología de desarrollo ágil y Optimización del análisis, diseño e implementación del sistema software, para el cálculo pertinente.

Tabla 46. Frecuencia Esperada para Variables Independiente y Dependiente

		Metodología de Desarrollo Ágil					
	Valoración	**1**	**2**	**3**	**4**	**5**	**Totales**
Optimización del análisis, diseño e implementación del sistema software	**1**	0	0	0	11	25	36
	2	0	0	0	11	25	36
	3	6	6	6	17	31	66
	4	19	19	19	30	44	131
	5	43	43	43	54	68	251
	Totales	68	68	68	123	193	520

Seguidamente se muestra la ecuación Ec. 5.1 para el cálculo de los valores para las Variables Metodología de desarrollo ágil frente a la Optimización del análisis, diseño e implementación del sistema software.

$$E_{i,j} = \frac{\Sigma_{i=1}^{m} O_{i,j} * \Sigma_{j=1}^{n} O_{i,j}}{\Sigma_{i=1}^{m} {}_{j=1}^{n} O_{i,j}}$$

Ec. 5.1

Dónde:

m: número de columnas

n: número de filas

i: posición de columnas

j: posición de filas

O: frecuencia observada

E: frecuencia esperada

Mediante la aplicación de la ecuación Ec. 5.1, en la Tabla 47 se representa la Frecuencia Esperada obtenida de las dos variables.

Tabla 47. Frecuencia Esperada para Variables Independiente y Dependiente obtenida de la ecuación Ec. 5.1

		Metodología de Desarrollo Ágil					
	Valoración	**1**	**2**	**3**	**4**	**5**	**Totales**
Optimización del análisis, diseño e implementación del sistema software	**1**	4,7077	4,7077	4,7077	8,5154	13,3615	36,0000
	2	4,7077	4,7077	4,7077	8,5154	13,3615	36,0000
	3	8,6307	8,6307	8,6307	15,6115	24,4961	66,0000
	4	17,1307	17,1307	17,1307	30,9865	48,6211	131,0000
	5	32,8231	32,8231	32,8231	59,3711	93,1596	251,0000
	Totales	68,0000	68,0000	68,0000	123,0000	193,0000	520,0000

5.2.3 Cálculo del valor de Chi Cuadrado

Continuando con el estudio se procede a realizar el cálculo del valor de Chi Cuadrado, el mismo que nos permitirá corroborar la dependencia o no, de las Variables Metodología de desarrollo ágil y Optimización del análisis, diseño e implementación del sistema software.

A continuación se presenta la ecuación Ec. 5.2 para el cálculo del valor de Chi Cuadrado.

$$x^2 = \Sigma_{i=1}^{m} \Sigma_{j=1}^{n} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}}$$

Ec. 5.2

Dónde:

m: número de columnas

n: número de filas

i: posición de columnas

x: valor de Chi Cuadrado

j: posición de filas

O: frecuencia observada

E: frecuencia esperada

En la Tabla 48 se muestra el cálculo de Chi Cuadrado para ambas variables.

Tabla 48. Cálculo de Chi Cuadrado para Variables Independiente y Dependiente obtenida de la ecuación Ec. 5.2

		Metodología de Desarrollo Ágil					
	Valoración	**1**	**2**	**3**	**4**	**5**	**Totales**
Optimización del análisis, diseño e implementación del sistema software	**1**	4,7077	4,7077	4,7077	0,7249	10,1377	24,9857
	2	4,7077	4,7077	4,7077	0,7249	10,1377	24,9857
	3	0,8018	0,8018	0,8018	0,1235	1,7268	4,2559
	4	0,2040	0,2040	0,2040	0,0314	0,4392	1,0825
	5	3,1554	3,1554	3,1554	0,4859	6,7948	16,7469
	Totales	13,5766	13,5766	13,5766	2,0907	29,2363	**72,0568**

Realizando los cálculos correspondientes en relación a la Tabla 48, se puede observar que la sumatoria para hallar el valor de Chi Cuadrado es:

$$\boldsymbol{x^2_{observado} = 72,0568}$$

5.2.4 Cálculo del Valor Crítico de Chi Cuadrado

Para el cálculo del Valor Crítico de Chi Cuadrado, inicialmente se procede a calcular los Grados de Libertad para Chi Crítico tomando el Nivel de Significancia supuesto de 0.05, que indica que hay una probabilidad del 0.95 de que la hipótesis nula sea verdadera.

Nivel de Significancia (α):

$$\boldsymbol{\alpha = 0.05}$$

Grados de Libertad (ν):

$$\nu = (número\ de\ filas - 1) * (número\ de\ olumnas - 1)$$

$$\nu = (5 - 1) * (5 - 1)$$

$$\nu = (4) * (4)$$

$$\boldsymbol{\nu = 16}$$

Con los datos obtenidos anteriormente, en la Tabla 49 se procede a la búsqueda del Valor Crítico de Chi Cuadrado según el Nivel de Significancia (filas) y Grados de Libertad (columnas).

Tabla 49. Distribución Chi Cuadrado Crítico

Grados de Libertad (ν)	**Áreas de Extremos Superior (α)**					
	0.25	**0.10**	**0.05**	**0.025**	**0.01**	**0.005**
1	1.323	2.706	3.841	5.024	6.635	7.879
2	2.773	4.605	5.991	7.378	9.210	10.597
3	4.108	6.251	7.815	9.348	11.345	12.838
4	5.385	7.779	9.488	11.143	13.277	14.860
5	6.662	9.236	11.071	12.883	15.086	16.750
6	7.841	10.645	12.592	14.449	16.812	18.548
7	9.037	12.017	14.067	16.013	18.475	20.278
8	10.129	13.362	15.507	17.535	20.090	21.955
9	11.389	14.684	16.919	19.023	21.666	23.589
10	12.549	15.987	18.307	20.483	23.209	25.188
11	13.701	17.275	19.675	21.920	24.725	26.757
12	14.845	18.549	21.026	23.337	26.217	28.299
13	15.984	19.812	22.362	24.736	27.688	29.819
14	17.117	21.064	23.685	26.119	29.141	31.319
15	18.245	22.307	24.996	27.488	30.578	32.801
16	19.369	23.542	**26.296**	28.845	32.000	34.267
17	20.489	24.769	27.587	30.191	33.409	35.718
18	21.605	25.989	28.869	31.526	34.805	37.156

Por tanto, el Valor Crítico de Chi Cuadrado es:

$$x^2_{crítico} = 26,296$$

5.2.5 Comparación entre el valor observado y el valor crítico.

De acuerdo a la Figura 22 se puede notar que el valor Chi Observado se encuentra en la Zona de Rechazo para la aceptación de la Hipótesis Nula (Ho).

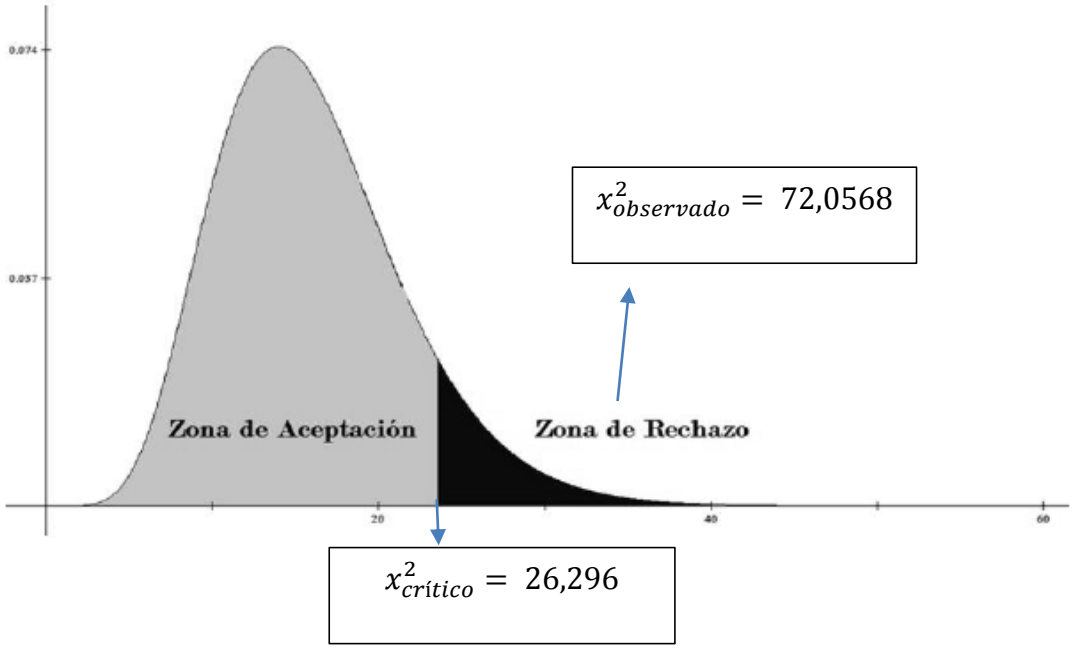


Figura 22. Prueba de Chi Cuadrado Dependiente

Fuente: (Los Autores)

Para corroborar el cumplimiento de la hipótesis planteada, motivo del presente estudio, se procede a aplicar la siguiente regla de decisión entre los valores observado y crítico.

Regla de decisión:

- Se acepta la Hipótesis Nula (Ho), si: $x^2_{observado} < x^2_{crítico}$
- Se acepta la Hipótesis Alternativa (Ha), si: $x^2_{observado} > x^2_{crítico}$

Siendo: $\boldsymbol{x^2_{observado} = 72,0568}$ y $\boldsymbol{x^2_{crítico} = 26,296}$

$$\boldsymbol{72,0568 > 26,296}$$

Entonces: $\boldsymbol{x^2_{observado} > x^2_{crítico}}$

Por lo tanto**: "Se acepta la Hipótesis Alternativa (Ha)"**

5.3 Conclusiones del Capítulo

Mediante los cálculos realizados aplicando la Prueba de Chi Cuadrado y el análisis de los resultados obtenidos, se puede concluir que la optimización del análisis, diseño e implementación del sistema software depende de la implementación correcta de una Metodología de Desarrollo Ágil en el Departamento TIC de la ETFA.

Según los datos arrojados de la aplicación de los instrumentos de medición (encuestas) hacia los miembros del equipo de trabajo del Departamento TIC, se puede evidenciar que la Metodología de Desarrollo Ágil "XP TIC´s", se acopla a las necesidades institucionales para la generación de software de calidad, lo cual corrobora y ratifica la hipótesis planteada al inicio del presente estudio, la misma que hace referencia a la optimización del análisis, diseño e implementación del sistema software mediante la implementación de una Metodología de Desarrollo Ágil en el Departamento TIC de la ETFA.

La aplicación de la Metodología propuesta, encaminará hacia una correcta gestión y a una mejor forma de generar software de calidad, buscando reducir la brecha existente entre tiempos tardíos de entrega, costos elevados y recursos limitados, ocasionado principalmente por la carencia de un marco de trabajo, estándares o metodologías a seguir en el Departamento TIC.

CAPÍTULO 6
CONCLUSIONES Y RECOMENDACIONES

6.1 Conclusiones

- La tendencia actual del Departamento TIC de la ETFA es hacia la optimización de los procesos. Sin embargo, una de las falencias más importantes ha sido, desde las etapas iniciales en el desarrollo y gestión de proyectos software, la carencia de estándares, procedimientos, métodos, técnicas, etc. a seguir, causando graves inconvenientes en las fases del desarrollo del sistema software. En este sentido, se ha decidido dar una solución mediante la implementación de una metodología que se adapte a los procesos de la ETFA y cuyo fin es la optimización del análisis, diseño e implementación del sistema software.
- La evolución de las Metodologías de Desarrollo de la Ingeniería de Software en el Proceso de Ingeniería de Sistemas Software, permite dar a conocer la cronología desde el nacimiento de las primeras técnicas de desarrollo software como "Code and Fix" (Codificar y corregir) hasta el establecimiento de las Metodologías Tradicionales y Ágiles como instrumentos ordenados y sistemáticos para el desarrollo software, los cuales en la actualidad tienen una gran acogida en las pequeñas, medianas y grandes empresas a nivel mundial. Tales Metodologías nos enseñan que no existe una metodología universal para hacer frente con éxito a cualquier proyecto de desarrollo de software; esto irá de la mano de acuerdo al contexto del proyecto y las necesidades particulares de cada empresa o institución.
- La determinación de la metodología adaptable al Departamento TIC de la ETFA, depende de una investigación exhaustiva, partiendo desde un proceso de selección de las metodologías hasta un estudio comparativo de las mismas, que mediante la utilización de parámetros de trabajo, instrumentos de investigación, frameworks, etc., determinó a la Metodología Extreme Programming (XP) como la más adaptable al contexto de trabajo del equipo del Departamento TIC. La investigación permite identificar otras metodologías ágiles (DSDM, ASP, Scrum) que también podrían utilizarse en el

Departamento, asimilando de ellas principios, procesos y artefactos valederos; el uso de los mismos dependerá siempre de los requerimientos de cualquier empresa o institución.

- La implementación de la Metodología de Desarrollo Ágil "XP TIC´s", ha sido descrita en base a principios del desarrollo ágil, fundamentada principalmente en la Metodología Ágil Extreme Programming (XP) y en otras como DSDM y Scrum, las mismas que sustentan todo el proceso de desarrollo del sistema software, permitiendo dotar de un coherente marco de trabajo a seguir por el equipo del Departamento TIC. Esta Metodología no limita al equipo de trabajo a operar exclusivamente bajo todas sus condiciones, más bien sirve al equipo a tomar la mejor decisión para ejecutar cualquier proyecto de desarrollo software utilizando únicamente lo necesario; es decir queda a discreción omitir ciertos procesos según las características del proyecto.
- La presente investigación dio a conocer la problemática actual del desarrollo del software en el Departamento TIC de la ETFA, y a su vez arrojó una propuesta alentadora, aceptada y aprobada por los miembros del equipo de trabajo, la Metodología de Desarrollo Ágil "XP TIC´s", como una respuesta urgente a la necesidad de desarrollar software de calidad orientado a la optimización de sus procesos y que sea adaptable al contexto de trabajo del Departamento. La experiencia obtenida mediante la aplicación de esta Metodología en el caso de estudio práctico, nos permite mencionar que es indispensable seguir una estructura organizada y sistemática, amparada en el uso de principios de las metodologías de desarrollo, la aplicación de normas y estándares para el éxito de cualquier proyecto software.
- Finalmente, se puede concluir que la Metodología de Desarrollo Ágil "XP TIC´s" es adaptable al ámbito y contexto del Departamento TIC, presenta instrumentos y procesos predeterminados que facilitan la recolección de la información y progresivamente organiza las diferentes fases del ciclo de vida del software. A pesar de esto no se garantiza la inexistencia de errores o inconvenientes del producto software final, pero es ineludible admitir que esta es una respuesta inmediata a la carencia de un marco metodológico que orienta la producción del software, cuyo objetivo final está orientado a la satisfacción del cliente en relación al producto entregado.

6.2 Recomendaciones

- Se recomienda el uso de estándares, procedimientos, métodos, técnicas, metodologías, etc., los cuales permitan optimizar todo el proceso de desarrollo del software a fin de alcanzar un producto software de calidad y lograr la satisfacción del cliente dentro de la Institución.
- Se recomienda también el estudio de los estándares IEEE y las normas ISO/IEC, las mismas que garanticen la construcción de los procesos y mejoren la gestión de desarrollo del software.
- Se recomienda, para un estudio de adaptación a una metodología de desarrollo software en cualquier Institución, realizar una investigación profunda tomando en cuenta siempre el contexto de trabajo (costo, tiempo, recursos humanos, materiales y tecnológicos) y sus requerimientos. Además, el estudio nos sugiere ceñirnos no únicamente al pie de una metodología, sino también asimilar prácticas válidas de otras metodologías; esto dependerá siempre de las necesidades del equipo desarrollador de software.
- Se recomienda la aplicación de la Metodología de Desarrollo Ágil "XP TIC´s" en el Departamento TIC de la ETFA, la cual posee una estructura flexible que norma el proceso de desarrollo software y el desempeño del equipo de trabajo, sugiriendo para su ejecución a utilizar los procesos, herramientas o artefactos que sean necesarios o adaptables al equipo.
- De acuerdo a los resultados obtenidos en la presente investigación y a través de su aplicación en el caso de estudio práctico, la Metodología propuesta es aplicable y aceptada en el Departamento TIC, recomendando ejecutar de la mejor manera la estructura que posee, ya que de esto dependerá obtener el producto software esperado.
- Se recomienda que durante la ejecución de la Metodología propuesta, se lleve el control respectivo a través del Diario de Actividades, el mismo que permite registrar las anotaciones necesarias durante todo el proceso de desarrollo, verificando el cumplimiento de las tareas y los entregables esperados de acuerdo a las fechas establecidas en el mismo.

BIBLIOGRAFÍA

[3] Hernán, M. (2004). Diseño de una Metodología Ágil de Desarrollo de Software. Tesis de Grado de Ingeniería en Informática. Buenos Aires: Universidad de Buenos Aires.

[4] Sommerville, I. (2006). Ingeniería del software. 7ma.Ed. Madrid.

[5] Ingeniería del Software: metodologías y ciclos de vida (2009). 1ra. Ed. España: Laboratorio Nacional de Calidad del Software de INTECO.

[7] Rising, L., Janoff, N.S. (2000). The Scrum Software Development Process for Small Teams Retrieved.

[9] Canós J. y Letelier P. (2003). Metodologías ágiles en el desarrollo de software [resumen]. Taller realizado en el marco de las VIII jornadas de Ingeniería del software y bases de datos en la Universidad politécnica de Valencia, España: Alicante.

[10] Kuhn Thomas S. (2006). La Estructura de las revoluciones científicas. 3ra. Ed. Fondo de Cultura Económica.

[11] Alonso, F. y Martínez, L. (2005). Introducción a la ingeniería del software: modelos de desarrollo de programas. 1ra. Ed. España: Delta Publicaciones.

[12] Sommerville, I. (2005). Ingeniería del software. 7ma. Ed. España: Pearson Educación.

[15] Jacobson, I., Booch G. y Rumbaugh J. (2005). The Unified Modeling Language Referente Manual. 2da. Ed. California: Addison Wesley.

[16] Beck K. (1999). Extreme Programming Explained: Embrace Change. California: Addison Wesley.

[17] Shaw, M. y Garlan, D. (1996). Software Architecture, Perspectives on an Emerging Discipline. 4ta. Ed. Prentice-Hall.

[18] Gamma, E., Helm, R., Jonson, R. y Vlissides, J. Design Patterns, Elements of reusable object-oriented software. Ed. Addsion Wesley.

[19] Beedle, M., Devos, M., Sharon, Y., Schwaber, K. y Sutherland, J. (1998). SCRUM: A pattern language for hyperproductive software development". En N. Harrison, B. 2. Foote, and H. Rohnert, eds., Pattern Languages of Program Design, vol. 4, pp. 637-651. Reading, Addison-Wesley.

[20] Jacobson, I., Booch G. y Rumbaugh J. (2005). The Unified Modeling Language Referente Manual. 2da. Ed. California: Addison Wesley.

[21] Kleppe, A., Warmer, J. y Basts, W. (2003). MDA Explained, The Model Driven Architecture: Practice and Promise. Ed. Pearson Education.

[22] Patton, R. (2006). "Software Testing". 2da. Ed. New York: Sams.

[23] Gutiérrez, H. y De la Vara, R. (2008). Análisis y Diseño de Experimentos. 5ta. Ed. México: Mc Graw Hill.

[25] Pressman, R. (2003). «El proceso». Ingeniería del Software, un enfoque Práctico. 5ta. Ed. Mexico: Mc Graw Hill.

[26] Sommerville, I. (2005). Ingeniería del software. 7ma. Ed. pp. 611

[27] Laboratorio Nacional de Calidad del Software de INTECO. (2009). Ingeniería del Software: metodologías y ciclos de vida. 1ra. Ed. España.

[28] Sommerville, I. (2005). Ingeniería del software. 7ma. Ed. pp. 373.

[30] Pressman, R. (2003). «El proceso». Ingeniería del Software, un enfoque Práctico. 5ta. Ed. Mexico: Mc Graw Hill.

[31] Giraldo, L. and Zapata, Y. (2005). Herramientas de desarrollo de ingeniería de software para Linux. Monitoria de Ingesoft.

[32] Análisis y Diseño de Sistemas. 3ra. Ed. Kendall & Kendall.

[33] Fuster, G. G., J. M. F. Torres, et al. (2006). Evaluación comparativa de herramientas CASE para UML desde el punto de vista notacional. Tecnología de Objetos Secciones Técnicas. Dpto de Informática. Madrid: Universidad Carlos III.

[38] Zhao, J. y Thomas D. (2005). Comparación de Herramientas de modelado UML: Enterprise Architect y Rational Rose.

[40] Magic Draw Architecture Made Simple, Instituto Politécnico Nacional. Bogotá: Unidad Politécnica para la Educación Virtual.

[42] Duvall, P. M. (2007). Continuous Integration. Improving Software Quality and Reducing Risk.

[43] Laboratorio Nacional de Calidad del Software de INTECO. (2009). Ingeniería del Software: metodologías y ciclos de vida. 1ra. Ed. España, pp.49.

[44] Laboratorio Nacional de Calidad del Software de INTECO. (2009). Ingeniería del Software: metodologías y ciclos de vida. 1ra. Ed. España, pp.50.

[45] Laboratorio Nacional de Calidad del Software de INTECO. (2009). Ingeniería del Software: metodologías y ciclos de vida. 1ra. Ed. España, pp.46.

[47] Programación en Pascal, Introducción a las Computadoras y a los lenguajes de programación. México: Mc. Graw Hill, pp. 34.

[48] Sommerville, I. (2011). Ingeniería del software. 9na. Ed., pp.34. México.

[49] Sommerville, I. (2011). Ingeniería del software. 9na. Ed., pp.52. México.

[56] Avison, D. E. y Fitzgerald, G. (1995). Information Systems Development: Methodologies, Techniques, and Tools. México: McGraw-Hill.

[57] Mnkandla, E. y Dwolatzky, B. Agile Methodologies Selection Toolbox. International Conference on Software Enfineering Advances (ICSEA 2007). IEEE 2007.

[58] Qumer, A. y Henderson-Sellers, B. (2008). An Evaluation of the degree of agility in six agile methods and itsapplicability for method engineering. Information and Software Technology vol 50.

[60] Abrahamsson, P. (2002). Agile software development methods, review and analysis. VTT Publications ESPOO.

[61] Abrahamsson, P., Siponen, M. y Ronkainen, J. New Directions on Agile Methods: A comparative Analysis. Procedings of the 25th International Conference on Software Engineering (ICSE'03). IEEE 2003.

NETGRAFÍA

[1] Historia de la Ingeniería de Software. http://es.wikipedia.org/wiki/Historia_de_la_ingenier%C3%ADa_del_software: [citado: 18/08/2013]

[2] Carballar, D. Ingeniería de software. www.eduinnova.es/dic09/Ingenieria_Software.pdf : [citado: 18/08/2013]

[6] Metodologías de Desarrollo Software. http://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software : [citado: 18/08/2013].

[8] Prácticas de la Metodología XP. http://es.slideshare.net/CrisCobol/metodologia-xp-6697221 : [citado: 23/08/2013].

[13] Métodos rápidos de la Ingeniería de Software. http://es.kioskea.net/contents/227-metodos-rapidos-rad-xp : [citado: 11/09/2013].

[14] Principios para la formalización de la Ingeniería de Software, http://revistas.udistrital.edu.co/ojs/index.php/reving/article/view/2118/2807 : [citado: 12/09/2013].

[24] Etapas del proceso de la Ingeniería de Software. http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software#Especificaci.C3.B3n : [citado: 11/10/2013].

[29] Metodologías Ágiles en el Desarrollo del Software. http://noqualityinside.com.ar/nqi/nqifiles/XP_Agil.pdf : [citado: 23/11/2013].

[34] System Architect, from http://www.popkin.com/products/sa2001/product.htm. : [citado: 07/12/2013].

[35] Power Designer, from http://www.sybase.com/products/powerdesigner : [citado: 07/12/2013].

[36] Oracle Designer. http://www.oracle.com/tools/designer/quicktour/contents.htm#features. : [citado: 07/12/2013].

[37] Micrsoft Proyect. http://www.microsoft.com/products/info/product.aspx?view=22&pcid=13f97e5e-0a49-4e27-ac77-fe647e54dd26 : [citado: 07/12/2013].

[39] R.S. Pressman & Associates.Products that improve your software engineering. www.casecomplete.com : [citado: 08/12/2013].

[41] Visual Paradigm. http://www.visual-paradigm.com : [citado: 08/12/2013].

[46] Arboleda, H. MSc. Ensayo sobre: Modelos de Ciclo de Vida de Desarrollo de Software en el Contexto dela Industria Colombiana de Software, Pp. 4. http://www.emn.fr/z-info/harbol07/ACIS.pdf : [citado: 12/12/2013].

[50] Cristal Clear. http://www.scribd.com/doc/55555056/Crystal-Clear-Version-Open-Document : [citado: 22/12/2013].

[51] Ciclo de vida XP. http://oness.sourceforge.net/proyecto/html/ch05s02.html : [citado: 22/12/2013].

[52] Ciclo de vida Scrum, pp. 20. http://carlosreynoso.com.ar/archivos/arquitectura/Metodos-Agiles.PDF : [citado: 22/12/2013].

[53] Framework for Agile Methods Classification. http://ceur-ws.org/Vol-341/paper9.pdf : [citado: 24/12/2013].

[54] Método comparativo de Metodologías Ágiles. pp 51-61. http://uvadoc.uva.es/bitstream/10324/1495/1/TFG-B.117.pdf : [citado: 04/01/2014].

[55] Carvajal, J. (2008). Metodologías ágiles: herramientas y modelo de desarrollo para aplicaciones java como metodología empresarial. Tesis Final de Máster. http://upcommons.upc.edu/pfc/bitstream/2099.1/5608/1/50015.pdf : [citado: 04/01/2014].

[59] Fowler, M. The New Methodology. http://martinfowler.com/articles/newMethodology.html : [citado: 18/01/2014].

[62] Solis, M. (2003). Una explicación de la programación extrema (XP). http://www.apolosoftware.com : [citado: 18/02/2014].

[63] Baird, S. (2002). XP: A Project Manager's Primer. http://www.phptr.com/articles/article.asp?p=26060&seqNum=6&rl=1 : [citado: 22/02/2014].

[64] Diagramas UML. http://www.clubdelsuran.com.ar/site/materiales/proyecto/diagramas_del_uml.pdf : [citado: 27/02/2014].

[65] Crispin, L. (2001). Pruebas de aceptación. Extreme Rules of the Road. Disponible en internet: http://www.testing.com/agile/crispin-xp-article.pdf [online] [12/04/2004].

[66] Jeffries, R. (1999). Pruebas de aceptación Extreme Testing. Disponible en internet: http://www.xprogramming.com/publications/SP99 Extreme for Web.pdf [online] [citado 12/04/2004].

[67] Orna, C. (2013). Tesis propuesta de Modelo de Calidad para Proyectos Medianos de Software. http://repositorio.espe.edu.ec/handle/21000/2718/browse?type=author&order=ASC&rpp=20&value=Orna+Jij%C3%B3n%2C+Cristina+Nataly : [citado: 03/03/2014].

ANEXO A

ENCUESTA DE LA PROBLEMÁTICA EN EL DEPARTAMENTO TIC ETFA

La siguiente encuesta está dirigida al equipo de trabajo del Departamento TIC de la ETFA, para evidenciar las falencias dentro del mismo y la factibilidad de aplicar una metodología de desarrollo software.

Deberá contestar marcando con una equis (X) según como corresponda, de la forma más veraz posible.

1. Se han presentado problemas en el proceso de desarrollado software en el Departamento TIC de la ETFA anteriormente?

Si	
No	
Ocasionalmente	

2. Se han presentado retrasos en la entrega del sistema software?

Si	
No	
Ocasionalmente	

3. Ha existido algún tipo de inconformidad de los usuarios con el producto software entregado por el Departamento TIC?

Si	
No	
Ocasionalmente	

4. Se ha dado sobrecarga de trabajo en el equipo desarrollador de software?

Si	
No	
Ocasionalmente	

5. El Departamento TIC cuenta con el personal y recursos necesarios para las actividades de desarrollo del sistema software?

Si	
No	
Ocasionalmente	

6. Se ha establecido o estructurado un Marco de Trabajo basado en Metodologías de Desarrollo Software para el proceso Operación TIC´s de la actividad Desarrollo de Software?

Si	
No	
Ocasionalmente	

7. Existen estándares, normas o procedimientos reglamentarios para desarrollo del sistema software?

Si	
No	
Ocasionalmente	

8. Cree conveniente el uso de métodos y/o metodologías, que mejor se adapten a los requerimientos o necesidades institucionales, para el desarrollo de sistemas software?

Si	
No	
Ocasionalmente	

ANEXO B
ESTÁNDARES DE CÓDIGO

Codificación PHP

Esta sección comprende lo que debe considerarse sobre la norma de codificación de los elementos que se requieren para garantizar un alto nivel técnico de interoperabilidad entre el código PHP.

1. Generalidades

- Los archivos DEBEN utilizar solamente las etiquetas `<?php` y `<?=`.
- Los archivos DEBEN emplear solamente la codificación UTF-8 sin BOM para el código PHP.
- Los archivos DEBERÍAN declarar *cualquier* estructura (clases, funciones, constantes, etc.) o realizar partes de la lógica de negocio (por ejemplo, generar una salida, cambio de configuración ini, etc.) pero NO DEBERÍAN hacer las dos cosas.
- Los espacios de nombres y las clases DEBEN cumplir el estándar PSR-0[13].
- Los nombres de las clases DEBEN declararse en notación `StudlyCaps`[14].
- Las constantes de las clases DEBEN declararse en mayúsculas con guiones bajos como separadores `CONSTANTE_DE_CLASE`.
- Los nombres de los métodos DEBEN declararse en notación `camelCase`[15].

2. Archivos

2.1. Etiquetas PHP

El código PHP DEBE utilizar las etiquetas largas `<?php ?>` o las etiquetas cortas para imprimir salida de información `<?= ?>`; NO DEBE emplear otras variantes.

2.2. Codificación de caracteres

El código PHP DEBE utilizar codificación UTF-8 sin BOM.

---

[13] PSR-0: Requisitos obligatorios que deben cumplirse para la interoperabilidad del autoloader.

[14] StudlyCaps, es una forma de notación de texto que sigue el patrón de palabras en minúscula sin espacios y con la primera letra de cada palabra en mayúscula.

[15] camelCase, es una forma de notación de texto que sigue el patrón de palabras en minúscula sin espacios y con la primera letra de cada palabra en mayúsculas exceptuando la primera palabra.

2.3. Efectos secundarios

Un archivo DEBERÍA declarar estructuras (clases, funciones, constantes, etc.) y no causar efectos secundarios, o DEBERÍA ejecutar partes de la lógica de negocio, pero NO DEBERÍA hacer las dos cosas.

La frase "efectos secundarios" significa: que la ejecución de la lógica de negocio no está directamente relacionado con declarar clases, funciones, constantes, etc., *simplemente la de incluir el archivo*.

"Efectos secundarios" incluyen, pero no se limitan a: generar salidas, uso explícito de `requiere` o `include`, conexiones a servicios externos, modificación de configuraciones iniciales, enviar errores o excepciones, modificar variables globales o estáticas, leer o escribir un archivo, etc.

3. Espacios de nombres y nombres de las Clases

Los espacios de nombres y las clases DEBEN seguir el estándar PSR-0.

Esto significa que cada clase estará en un fichero independiente y está dentro de un espacio de nombres en al menos un nivel: un nombre de proveedor de nivel superior.

Los nombres de las clases DEBEN declararse con notación `StudlyCaps`.

El código escrito para PHP 5.3 o superior DEBE hacer un uso formal de los espacios de nombres.

4. Constantes de Clases, Propiedades y Métodos

El término "clases" hace referencia a todas las clases, interfaces y traits.

4.1. Constantes

Las constantes de las clases DEBEN declararse siempre en mayúsculas y separadas por guiones bajos.

4.2. Propiedades

Esta guía evita intencionadamente cualquier recomendación respecto al uso de las notaciones `$StudlyCaps`, `$camelCase`, o `$guion_bajo` en los nombres de las propiedades.

Cualquiera que sea la convención en nomenclatura, DEBERÍA ser utilizada de forma coherente con un alcance razonable. Este alcance PUEDE ser a nivel de proveedor, a nivel de paquete, a nivel de clase o a nivel de método.

4.3. Métodos

Los nombres de los métodos DEBEN declararse en notación `camelCase()`.

Codificación HTML

HTML, siglas de **HyperText Markup Language** («lenguaje de marcas de hipertexto»), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia para la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, etc.

- `<html>`: define el inicio del documento HTML, le indica al navegador que lo que viene a continuación debe ser interpretado como código HTML. Esto es así *de facto*, ya que en teoría lo que define el tipo de documento es el DOCTYPE, que significa la palabra justo tras DOCTYPE el tag de raíz.
- `<script>`: incrusta un script en una web, o llama a uno mediante `src="url del script"`. Se recomienda incluir el tipo MIME en el atributo `type`, en el caso de JavaScript `text/javascript`.
- `<head>`: define la cabecera del documento HTML; esta cabecera suele contener información sobre el documento que no se muestra directamente al usuario como, por ejemplo, el título de la ventana del navegador. Dentro de la cabecera `<head>` es posible encontrar:
  - `<title>`: define el título de la página. Por lo general, el título aparece en la barra de título encima de la ventana.
  - `<link>`: para vincular el sitio a hojas de estilo o iconos. Por ejemplo:`<link rel="stylesheet" href="/style.css" type="text/css">`.
  - `<style>`: para colocar el estilo interno de la página; ya sea usando CSS u otros lenguajes similares. No es necesario colocarlo si se va a vincular a un archivo externo usando la etiqueta `<link>`.
  - `<meta>`: para metadatos como la autoría o la licencia, incluso para indicar parámetros http (mediante `http-equiv=""`) cuando no se pueden modificar por no estar disponible la configuración o por dificultades con server-side scripting.

- <body>: define el contenido principal o cuerpo del documento. Esta es la parte del documento html que se muestra en el navegador; dentro de esta etiqueta pueden definirse propiedades comunes a toda la página, como color de fondo y márgenes. Dentro del cuerpo <body> es posible encontrar numerosas etiquetas. A continuación se indican algunas a modo de ejemplo:
  - <h1> a <h6>: encabezados o títulos del documento con diferente relevancia.
  - <table>: define una tabla.
    - <tr>: fila de una tabla.
    - <td>: celda de una tabla (debe estar dentro de una fila).
  - <a>: hipervínculo o enlace, dentro o fuera del sitio web. Debe definirse el parámetro de pasada por medio del atributo *href*. Por ejemplo: Ejemplo se representa como Ejemplo).
  - : división de la página. Se recomienda, junto con css, en vez de <table> cuando se desea alinear contenido.
  - <img>: imagen. Requiere del atributo *src*, que indica la ruta en la que se encuentra la imagen. Por ejemplo: . Es conveniente, por accesibilidad, poner un atributo alt="texto alternativo".
  - : etiquetas para listas.
  - <b>: texto en negrita (*etiqueta desaprobada. Se recomienda usar la etiqueta *).
  - <i>: texto en cursiva (*etiqueta desaprobada. Se recomienda usar la etiqueta *).
  - <s>: texto tachado (*etiqueta desaprobada. Se recomienda usar la etiqueta <del>*).
  - <u>: Antes texto subrayado. A partir de HTML 5 define porciones de texto diferenciadas o destacadas del resto, para indicar correcciones por ejemplo. (etiqueta desaprobada en HTML 4.01 y redefinida en HTML 5)

La mayoría de etiquetas deben cerrarse como se abren, pero con una barra («/») tal como se muestra en los siguientes ejemplos:

- <table><tr><td>Contenido de una celda</td></tr></table>.
- <script>Código de un script integrado en la página</script>

JQuery

jQuery tiene una sintaxis demasiada sencilla ya que todos los comandos se reconocen de manera fácil, porque comienzan con el símbolo $.

Sintaxis

La sintaxis básica es: **$(selector).action()**

Dónde:

- $ define jQuery
- (selector) para encontrar elementos html
- Action() para manipular los elementos

El evento Document Ready

La mayoría de métodos están dentro de un evento document:

```
ready$(document).ready(function(){
    //métodos
});
```

ANEXO C

PRUEBAS DE LA METODOLOGÍA PROPUESTA "XP TIC´S"

a) Pruebas Unitarias

Caso de Prueba Unitaria para un Módulo:

Caso Prueba Unitaria					
Id Caso de Prueba:	**Nombre Módulo:**				
Iteración:					
Descripción de la prueba:					
Nombre del Tester:					
Funcionalidad: Descripción	**Método Utilizado**	**Recibe**	**Resultado esperado del método**	**Resultado esperado de la prueba**	**Resultado real de la prueba**
Fun1: descrip.				Ok/No ok	Ok/No ok
Fun2: descrip.				Ok/No ok	Ok/No ok
…					

Reporte de errores del Caso de Prueba Unitaria del Módulo:

Reporte de Prueba No.:			
Errores Encontrados:			
Id Caso de Prueba:			
Funcionalidad Probada	**Error Encontrado**	**Clasificación del Error**	**Observación**
Fun1		Alta/Media/Baja	
Fun3		Alta/Media/Baja	
…			

b) Pruebas de Aceptación

Caso Prueba de Aceptación	
Código:	**Historia de Usuario (Nro. y Nombre):**
Nombre:	
Descripción:	
Condiciones de Ejecución: 1. 2. 3. …	
Entrada / Pasos de ejecución: 1. 2. 3. …	
Resultado Esperado:	
Evaluación de la Prueba:	

ANEXO D

INSTRUMENTOS DE MEDICIÓN DE LA METODOLOGÍA DE DESARROLLO ÁGIL IMPLEMENTADA AL DEPARTAMENTO TIC

De acuerdo al criterio que más se adapte sobre la Metodología propuesta (XP TIC), marque con una equis (X) según corresponda, tomando en consideración la siguiente escala de valoración:

1. Ocasionalmente (en escasas ocasiones)
2. Ordinariamente (en ocasiones puntuales)
3. Frecuentemente (en la mayor parte)
4. Muy Frecuentemente (en casi todas las ocasiones)
5. Siempre

ENCUESTA No.1

PREVIA APLICACIÓN DE LA METODOLOGÍA AL DEPARTAMENTO TIC

Parámetros referentes al proceso de desarrollo software del Departamento TIC.

1. Cree usted que la metodología propuesta engloba todos los aspectos que forman parte del proceso de desarrollo del sistema software que cotidianamente se desarrollan para la ETFA?

Valor				
1	**2**	**3**	**4**	**5**

2. Cree usted que aplicando la metodología propuesta, se optimice el proceso de desarrollo del sistema software?

Valor				
1	**2**	**3**	**4**	**5**

3. Cree usted que aplicando la metodología propuesta se cumplirán todas las fases del proceso de desarrollo del sistema software (análisis, diseño, implementación, pruebas, etc.)?

Valor				
1	**2**	**3**	**4**	**5**

4. Cree usted que la metodología propuesta optimice la gestión de la calidad del sistema software?

Valor				
1	**2**	**3**	**4**	**5**

5. Cree usted que se lograría definir claramente el alcance del proyecto software aplicando la metodología propuesta?

Valor				
1	**2**	**3**	**4**	**5**

6. Cree usted que se entregaría el sistema software satisfactoriamente al cliente (personal de los diferentes Departamentos de la ETFA), aplicando la metodología propuesta?

Valor				
1	**2**	**3**	**4**	**5**

7. Cree usted que la metodología propuesta cumple con los objetivos establecidos del área de desarrollo software del Departamento TIC?

Valor				
1	**2**	**3**	**4**	**5**

8. Cree usted que se puede solventar las falencias de producción de software ocurridos hasta el momento, aplicando la metodología propuesta?

Valor				
1	**2**	**3**	**4**	**5**

9. Considera usted que la metodología propuesta es necesaria en el Departamento TIC para trabajar como un marco estándar a seguir?

Valor				
1	**2**	**3**	**4**	**5**

Parámetros para la implementación de la metodología de desarrollo ágil.

10. Cree usted que la metodología propuesta es simple para ser entendida por el equipo de trabajo del Departamento TIC?

Valor				
1	**2**	**3**	**4**	**5**

11. Cree usted que la metodología propuesta contiene principios de desarrollo ágil?

Valor				
1	**2**	**3**	**4**	**5**

12. Considera usted que la metodología propuesta promueve el trabajo en equipo, no solo dentro del equipo del Departamento TIC, sino también el trabajo con el cliente (personal de otros Departamentos de la ETFA)?

Valor				
1	**2**	**3**	**4**	**5**

13. Cree usted que la metodología propuesta es fácil de utilizar?

Valor				
1	**2**	**3**	**4**	**5**

14. Escriba una opinión sobre cómo le pareció la metodología propuesta para el Departamento TIC?

ENCUESTA No.2

SEGÚN LA EXPERIENCIA OBTENIDA CON LA APLICACIÓN DE LA METODOLOGÍA

Parámetros referentes al proceso de desarrollo software del Departamento TIC.

1. Cree usted que la metodología propuesta se adapta a las necesidades de las etapas del proceso de desarrollo del sistema software?

Valor				
1	**2**	**3**	**4**	**5**

2. Cree usted que la metodología propuesta engloba todos los aspectos que forman parte del proceso de desarrollo del sistema software que cotidianamente desarrollan para la ETFA?

Valor				
1	**2**	**3**	**4**	**5**

3. Cree usted que aplicando la metodología propuesta, se optimiza el proceso de desarrollo del sistema software?

Valor				
1	2	3	4	5

4. Cree usted que los tiempos se ajustan de acuerdo a la planificación establecida en el proyecto, utilizando la metodología propuesta?

Valor				
1	2	3	4	5

5. Cree usted que se entrega el sistema software satisfactoriamente al cliente (personal de los diferentes Departamentos de la ETFA), aplicando la metodología propuesta?

Valor				
1	2	3	4	5

6. Cree usted que la metodología propuesta cumple con los objetivos establecidos del área de desarrollo software del Departamento TIC?

Valor				
1	2	3	4	5

7. Cree usted que la metodología propuesta cumple con la planificación establecida por el equipo de trabajo del Departamento TIC?

Valor				
1	2	3	4	5

8. Cree usted que se puede solventar las falencias de producción de software ocurridos hasta el momento, aplicando la metodología propuesta?

Valor				
1	2	3	4	5

9. Cree usted que la metodología propuesta sigue una estructura lógica, organizada y estructurada que satisface el ciclo de vida del sistema software?

Valor				
1	2	3	4	5

10. Considera usted adecuado el esfuerzo del equipo de trabajo aplicando la metodología propuesta?

Valor				
1	2	3	4	5

Parámetros para la implementación de la metodología de desarrollo ágil.

11. Cree usted que la metodología propuesta se adapta ágilmente a cualquier proyecto software que se desarrolla en el Departamento TIC?

Valor				
1	2	3	4	5

12. Considera usted que la metodología propuesta adapta los requisitos cambiantes, incluso si llegan tarde al desarrollo?

Valor				
1	2	3	4	5

13. Considera usted que la metodología propuesta promueve el trabajo en equipo, no solo dentro del equipo del Departamento TIC, sino también el trabajo con el cliente (personal de otros Departamentos de la ETFA)?

Valor				
1	2	3	4	5

14. Considera usted satisfactoria la entrega a tiempo de los productos generados con la aplicación de la metodología propuesta?

Valor				
1	2	3	4	5

15. Encuentra usted satisfactorio el empleo de la metodología propuesta?

Valor				
1	2	3	4	5

16. Seleccione los principios ágiles más relevantes que considere usted que posee la metodología.

Entrega temprana y continua de software de valor		El software que funciona es la principal medida del progreso	
Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo		Los procesos ágiles promueven el desarrollo sostenido	

Entregas con frecuencia software que funcione		La atención continua a la excelencia técnica enaltece la agilidad	
Las personas del negocio y los desarrolladores deben trabajar juntos		La simplicidad como arte de maximizar la cantidad de trabajo que no se hace, es esencial.	
Construcción de proyectos en torno a individuos motivados		Las mejores arquitecturas, requisitos y diseños emergen de equipos que se auto-organizan	
Comunicación cara a cara es clave dentro del equipo de desarrollo		En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta en consecuencia	

17. Escriba una opinión sobre cómo le pareció la metodología propuesta para el Departamento TIC?

ENCUESTA No.3

DE LOS GESTORES DEL PROYECTO

Parámetros referentes al proceso de desarrollo software del Departamento TIC.

1. Cree usted que la metodología propuesta cumple con la funcionalidad adecuada para el desarrollo del sistema software?

Valor				
1	**2**	**3**	**4**	**5**

2. Cree usted que la metodología propuesta se adapta a las necesidades de las etapas del proceso de desarrollo del sistema software en el Departamento TIC?

Valor				
1	**2**	**3**	**4**	**5**

3. Cree usted que la metodología propuesta engloba todos los aspectos que forman parte del proceso de desarrollo del sistema software que cotidianamente desarrollan para la ETFA?

Valor				
1	**2**	**3**	**4**	**5**

4. Cree usted que los tiempos se ajustan de acuerdo a la planificación establecida en el proyecto, utilizando la metodología propuesta?

Valor				
1	**2**	**3**	**4**	**5**

5. Cree usted que la metodología propuesta optimice la gestión de la calidad del sistema software?

Valor				
1	**2**	**3**	**4**	**5**

6. Cree usted que se lograría definir claramente el alcance del proyecto software aplicando la metodología propuesta?

Valor				
1	**2**	**3**	**4**	**5**

7. Cree usted que la metodología propuesta cumple con los objetivos establecidos del área de desarrollo software del Departamento TIC?

Valor				
1	**2**	**3**	**4**	**5**

8. Cree usted que la metodología propuesta cumple con la planificación establecida por el equipo de trabajo del Departamento TIC?

Valor				
1	**2**	**3**	**4**	**5**

9. Cree usted que la metodología propuesta plantea un marco de trabajo óptimo que permite realizar el seguimiento y control respectivo del proceso de desarrollo software?

Valor				
1	**2**	**3**	**4**	**5**

10. Cree usted que la metodología propuesta plantea herramientas de pruebas del software (testing) adecuadas que permita la verificación y validación de resultados?

Valor				
1	**2**	**3**	**4**	**5**

Parámetros para la implementación de la metodología de desarrollo ágil.

11. Cree usted que la metodología propuesta es simple para ser entendida por el equipo de trabajo del Departamento TIC?

Valor				
1	**2**	**3**	**4**	**5**

12. Cree usted que la metodología propuesta se adapta ágilmente a cualquier proyecto software que se desarrolla en el Departamento TIC?

Valor				
1	**2**	**3**	**4**	**5**

13. Considera usted que la metodología propuesta adapta los requisitos cambiantes, incluso si llegan tarde al desarrollo?

Valor				
1	**2**	**3**	**4**	**5**

14. Cree usted que la metodología propuesta plantea una comunicación cara a cara dentro del equipo de desarrollo?

Valor				
1	**2**	**3**	**4**	**5**

15. Considera usted que la metodología propuesta promueve el trabajo en equipo, no solo dentro del equipo del Departamento TIC, sino también el trabajo con el cliente (personal de otros Departamentos de la ETFA)?

Valor				
1	**2**	**3**	**4**	**5**

16. Considera usted que la metodología propuesta plantea la evolución de los requerimientos y soluciones mediante la colaboración de grupos auto organizados y multidisciplinarios?

Valor				
1	**2**	**3**	**4**	**5**

17. Escriba una opinión sobre cómo le pareció la metodología propuesta para el Departamento TIC?

ANEXO E

VALORACIÓN DE LOS INSTRUMENTOS (ENCUESTAS) SOBRE LA METODOLOGÍA ÁGIL "XP TIC´S"

Organización de las variables de la hipótesis

a) Encuesta No.1 previa aplicación de la Metodología "XP TIC´S"

- **Variable Independiente:** Metodología de desarrollo ágil.

No.	Preguntas	Valoración				
		1	**2**	**3**	**4**	**5**
10	Cree usted que la metodología propuesta es simple para ser entendida por el equipo de trabajo del Departamento TIC?				2	
11	Cree usted que la metodología propuesta contiene principios de desarrollo ágil?				1	1
12	Considera usted que la metodología propuesta promueve el trabajo en equipo, no solo dentro del equipo del Departamento TIC, sino también el trabajo con el cliente (personal de otros Departamentos de la ETFA)?					2
13	Cree usted que la metodología propuesta es fácil de utilizar?					2
	SUMA	0	0	0	3	5

- **Variable dependiente:** Optimización del análisis, diseño e implementación del sistema software.

No.	Preguntas	Valoración				
		1	**2**	**3**	**4**	**5**
1	Cree usted que la metodología propuesta engloba todos los aspectos que forman parte del proceso de desarrollo del sistema software que cotidianamente se desarrollan para la ETFA?			1		1
2	Cree usted que aplicando la metodología propuesta, se optimice el proceso de desarrollo del sistema software?			1		1
3	Cree usted que aplicando la metodología propuesta se cumplirán todas las fases del proceso de desarrollo del sistema software (análisis, diseño, implementación, pruebas, etc.)?			1		1

4	Cree usted que la metodología propuesta optimice la gestión de la calidad del sistema software?			1		1
5	Cree usted que se lograría definir claramente el alcance del proyecto software aplicando la metodología propuesta?			1		1
6	Cree usted que se entregaría el sistema software satisfactoriamente al cliente (personal de los diferentes Departamentos de la ETFA), aplicando la metodología propuesta?			1		1
7	Cree usted que la metodología propuesta cumple con los objetivos establecidos del área de desarrollo software del Departamento TIC?				1	1
8	Cree usted que se puede solventar las falencias de producción de software ocurridos hasta el momento, aplicando la metodología propuesta?				1	1
9	Considera usted que la metodología propuesta es necesaria en el Departamento TIC para trabajar como un marco estándar a seguir?				1	1
	SUMA	0	0	6	3	9

b) Encuesta No. 2 según la experiencia obtenida con la aplicación de la Metodología "XP TIC´S"

- **Variable Independiente:** Metodología de desarrollo ágil.

No.	**Preguntas**	**Valoración**				
		1	**2**	**3**	**4**	**5**
11	Cree usted que la metodología propuesta se adapta ágilmente a cualquier proyecto software que se desarrolla en el Departamento TIC?					2
12	Considera usted que la metodología propuesta adapta los requisitos cambiantes, incluso si llegan tarde al desarrollo?				2	
13	Considera usted que la metodología propuesta promueve el trabajo en equipo, no solo dentro del equipo del Departamento TIC, sino también el trabajo con el cliente (personal de otros Departamentos de la ETFA)?				1	1
14	Considera usted satisfactoria la entrega a tiempo de los productos generados con la aplicación de la metodología propuesta?					2
15	Encuentra usted satisfactorio el empleo de la metodología propuesta?					2
	SUMA	0	0	0	3	7

- **Variable dependiente:** Optimización del análisis, diseño e implementación del sistema software.

No.	**Preguntas**	**Valoración**				
		1	**2**	**3**	**4**	**5**
1	Cree usted que la metodología propuesta se adapta a las necesidades de las etapas del proceso de desarrollo del sistema software?					2
2	Cree usted que la metodología propuesta engloba todos los aspectos que forman parte del proceso de desarrollo del sistema software que cotidianamente desarrollan para la ETFA?					2
3	Cree usted que aplicando la metodología propuesta, se optimiza el proceso de desarrollo del sistema software?				1	1
4	Cree usted que los tiempos se ajustan de acuerdo a la planificación establecida en el proyecto, utilizando la metodología propuesta?					2
5	Cree usted que se entrega el sistema software satisfactoriamente al cliente (personal de los diferentes Departamentos de la ETFA), aplicando la metodología propuesta?				1	1
6	Cree usted que la metodología propuesta cumple con los objetivos establecidos del área de desarrollo software del Departamento TIC?				1	1
7	Cree usted que la metodología propuesta cumple con la planificación establecida por el equipo de trabajo del Departamento TIC?				2	
8	Cree usted que se puede solventar las falencias de producción de software ocurridos hasta el momento, aplicando la metodología propuesta?				1	1
9	Cree usted que la metodología propuesta sigue una estructura lógica, organizada y estructurada que satisface el ciclo de vida del sistema software?				2	
10	Considera usted adecuado el esfuerzo del equipo de trabajo aplicando la metodología propuesta?				1	1
	SUMA	0	0	0	9	11

c) Encuesta No.3 de los Gestores del Proyecto de la Metodología "XP TIC´S"

- **Variable Independiente:** Metodología de desarrollo ágil.

No.	Preguntas	Valoración				
		1	**2**	**3**	**4**	**5**
11	Cree usted que la metodología propuesta es simple para ser entendida por el equipo de trabajo del Departamento TIC?					3
12	Cree usted que la metodología propuesta se adapta ágilmente a cualquier proyecto software que se desarrolla en el Departamento TIC?				2	1
13	Considera usted que la metodología propuesta adapta los requisitos cambiantes, incluso si llegan tarde al desarrollo?				1	2
14	Cree usted que la metodología propuesta plantea una comunicación cara a cara dentro del equipo de desarrollo				1	2
15	Considera usted que la metodología propuesta promueve el trabajo en equipo, no solo dentro del equipo del Departamento TIC, sino también el trabajo con el cliente (personal de otros Departamentos de la ETFA)?					3
16	Considera usted que la metodología propuesta plantea la evolución de los requerimientos y soluciones mediante la colaboración de grupos auto organizados y multidisciplinarios?				1	2
	SUMA	0	0	0	5	13

- **Variable dependiente:** Optimización del análisis, diseño e implementación del sistema software.

No.	Preguntas	Valoración				
		1	**2**	**3**	**4**	**5**
1	Cree usted que la metodología propuesta cumple con la funcionalidad adecuada para el desarrollo del sistema software?					3
2	Cree usted que la metodología propuesta se adapta a las necesidades de las etapas del proceso de desarrollo del sistema software en el Departamento TIC?					3

3	Cree usted que la metodología propuesta engloba todos los aspectos que forman parte del proceso de desarrollo del sistema software que cotidianamente desarrollan para la ETFA?					3
4	Cree usted que los tiempos se ajustan de acuerdo a la planificación establecida en el proyecto, utilizando la metodología propuesta?				1	2
5	Cree usted que la metodología propuesta optimice la gestión de la calidad del sistema software?					3
6	Cree usted que se lograría definir claramente el alcance del proyecto software aplicando la metodología propuesta?					3
7	Cree usted que la metodología propuesta cumple con los objetivos establecidos del área de desarrollo software del Departamento TIC?				2	1
8	Cree usted que la metodología propuesta cumple con la planificación establecida por el equipo de trabajo del Departamento TIC?				2	1
9	Cree usted que la metodología propuesta plantea un marco de trabajo óptimo que permite realizar el seguimiento y control respectivo del proceso de desarrollo software?				2	1
10	Cree usted que la metodología propuesta plantea herramientas de pruebas del software (testing) adecuadas que permita la verificación y validación de resultados?					3
	SUMA	0	0	0	7	23

ANEXO F

CASO DE ESTUDIO PRÁCTICO

Proyecto
Sistema Inteligente de Control Disciplinario para Aspirantes a Tropa de la ETFA [SICODIAT]

Metodología de Desarrollo Ágil de Sistemas Software "XP TIC´s" del Departamento TIC de la ETFA

Versión 1.0

Historial de Revisiones

Fecha	Versión	Descripción	Autor
01/09/2013	1.0	Primera versión con los apartados y contenidos básicos	Juan Játiva, Luis Egas

DESCRIPCIÓN DE LA METODOLOGÍA DE TRABAJO

I. INTRODUCCIÓN

Este documento describe la implementación de la metodología de trabajo XP TIC y ciertas metodologías ágiles en el Departamento TIC de la ETFA para la gestión del desarrollo del proyecto **Sistema Inteligente de Control Disciplinario para Aspirantes a Tropa de la ETFA [SICODIAT]**.

Incluye junto con la descripción de este ciclo de vida iterativo e incremental para el proyecto, los artefactos o documentos con los que se gestionan las tareas de adquisición y suministro: requisitos, monitorización y seguimiento del avance, así como las responsabilidades y compromisos de los participantes en el proyecto.

I.1 PROPÓSITO DE ESTE DOCUMENTO

Facilitar la información de referencia necesaria a las personas implicadas en el desarrollo del **Sistema Inteligente de Control Disciplinario para Aspirantes a Tropa de la ETFA [SICODIAT]**.

I.2 ALCANCE

El sistema software está dirigido al Departamento Cuerpo de Alumnos, Académico, evaluación y Tic’s de la ETFA.

II. DESCRIPCIÓN GENERAL DE LA METODOLOGÍA

II.1 FUNDAMENTACIÓN

Las principales razones del uso de un ciclo de desarrollo iterativo e incremental de tipo ágil, el cual se basa nuestra metodología, para la ejecución de este proyecto son:

- Sistema modular. Las características del **Sistema Inteligente de Control Disciplinario para Aspirantes a Tropa de la ETFA [SICODIAT]** permiten desarrollar una base funcional mínima y sobre ella ir incrementando las funcionalidades o modificando el comportamiento o apariencia de las ya implementadas.
- Entregas frecuentes y continuas al cliente de los módulos terminados, de forma que puede disponer de una funcionalidad básica en un tiempo mínimo y a partir de ahí un incremento y mejora continua del sistema.
- Previsible inestabilidad de requisitos.
  - Para el cliente resulta difícil precisar cuál será la dimensión completa del sistema, y su crecimiento puede continuarse en el tiempo, suspenderse o detenerse.
- Complicaciones en el canal de comunicación con el usuario final del sistema el cual no se puede centrar pruebas debido a la disponibilidad de tiempo de la parte interesada.

II.2 VALORES DE TRABAJO

Los valores que deben ser practicados por todos los miembros involucrados en el desarrollo y que hacen posible que la metodología XP tenga éxito son:

- Simplicidad
- Comunicación
- Retroalimentación (Feedback)
- Coraje o valentía
- Respeto.

III. PERSONAS Y ROLES DEL PROYECTO

Persona	Contacto	Rol
Juan Xavier Játiva	0984659882	Programador, Coach
Luis Miguel Egas	0998444215	Programador, Tester
Lucas Garcés		Consultor
ETFA		Cliente

IV. RESPONSABILIDADES

Programador

- Produce el código del sistema.
- Escribe las pruebas unitarias.

Cliente

- Escribe las historias de usuario y las pruebas funcionales para validar su implementación.
- Asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar el mayor valor de negocio.

Tester

- Ayuda al cliente a escribir las pruebas funcionales.
- Ejecuta pruebas regularmente y difunde los resultados en el equipo.
- Es responsable de las herramientas de soporte para pruebas.

Entrenador (coach)

- Responsable del proceso global.
- Guía a los miembros del equipo para seguir el proceso correctamente.

Consultor

- Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto.
- Ayuda al equipo a resolver un problema específico.

V. FASES DE LA METODOLOGÍA SELECCIONADA "XP TIC".

1ª Fase: Exploración

1.1 RECOLECCIÓN DE REQUERIMIENTOS GENERALES

Nombre	Departamento	Requerimiento/Historia Usuario	Prioridad estimada
Aspirantes	Personal	Gestión de Aspirantes	Alta
Instructores	Personal	Gestión de instructores	Media
Faltas	Dpto. Cuerpo Atros.	Gestión de faltas	Alta
Sanciones	Dpto. Cuerpo Atros.	Gestión de sanciones	Alta
Perfiles	Dpto. Cuerpo Atros.	Perfiles de Usuario	Media
Registro	Dpto. Cuerpo Atros.	Gestión de sanciones Aspirantes	Media
Registro	Dpto. Cuerpo Atros.	Gestión de méritos Aspirantes	Media
Reportes	Dpto. Cuerpo Atros.	Gestión de reportes	Media

1.1.5 Supuestos Generales

El sistema de control disciplinario ETFA permitirá brindar una gestión eficiente y organizada de los Aspirantes sancionados diariamente de acuerdo al Reglamento de Sanciones y recompensas para Aspirantes a Oficiales y tropa de las FFAA.

1.1.6 Principales procesos a generarse

No.	Nombre del Proceso	Descripción del Proceso	Actor	Funciones del Actor
1	Gestión de Aspirantes	El sistema permitirá dar de alta, baja y modificaciones de los Aspirantes.	Administrador	Alta, baja, modificaciones y búsquedas.

2	Gestión de instructores	El sistema permitirá dar de alta, baja y modificaciones de los instructores.	Administrador	Alta, baja y modificaciones.
3	Gestión de faltas	El sistema permitirá dar de alta y modificaciones de las faltas.	Administrador	Alta y modificaciones
4	Gestión de Sanciones	El sistema permitirá dar de alta y modificaciones de las sanciones.	Administrador	Alta y modificaciones
5	Gestión de perfiles de usuario	Usuario administrador, regulador y secretario.	Sistema	Perfiles con permisos.
6	Gestión de sanciones de Aspirantes	Registro de sanciones	Administrador, Regulador, Secretario	Registro
7	Gestión de méritos de Aspirantes	Registro de méritos	Administrador, Regulador, Secretario	Registro
8	Gestión de reportes	Reportes por Aspirante, fecha, promoción.	Administrador, Regulador, Secretario	Reportes.

1.1.7 Estimación General del Proyecto

HISTORIA	TIEMPO ESTIMADO HORAS
Gestión de Aspirantes	125
Gestión de instructores	125
Gestión de faltas	125
Gestión de sanciones	50
Perfiles de Usuario	25
Registro sanciones	110
Registro de méritos	25
Gestión de reportes	100
Tiempo Estimado Total Horas	685

1.2 PLAN GENERAL DE DESARROLLO

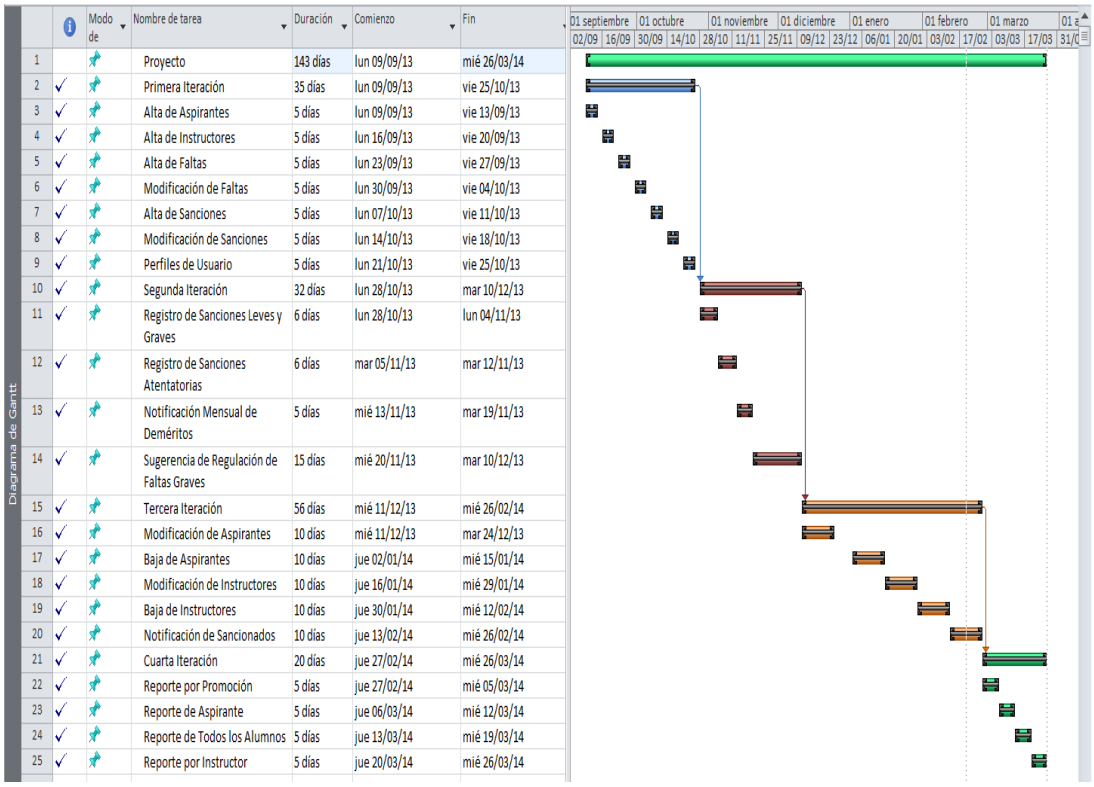


Diagrama de Gantt

	Modo de	Nombre de tarea	Duración	Comienzo	Fin
1		Proyecto	143 días	lun 09/09/13	mié 26/03/14
2	✓	Primera Iteración	35 días	lun 09/09/13	vie 25/10/13
3	✓	Alta de Aspirantes	5 días	lun 09/09/13	vie 13/09/13
4	✓	Alta de Instructores	5 días	lun 16/09/13	vie 20/09/13
5	✓	Alta de Faltas	5 días	lun 23/09/13	vie 27/09/13
6	✓	Modificación de Faltas	5 días	lun 30/09/13	vie 04/10/13
7	✓	Alta de Sanciones	5 días	lun 07/10/13	vie 11/10/13
8	✓	Modificación de Sanciones	5 días	lun 14/10/13	vie 18/10/13
9	✓	Perfiles de Usuario	5 días	lun 21/10/13	vie 25/10/13
10	✓	Segunda Iteración	32 días	lun 28/10/13	mar 10/12/13
11	✓	Registro de Sanciones Leves y Graves	6 días	lun 28/10/13	lun 04/11/13
12	✓	Registro de Sanciones Atentatorias	6 días	mar 05/11/13	mar 12/11/13
13	✓	Notificación Mensual de Deméritos	5 días	mié 13/11/13	mar 19/11/13
14	✓	Sugerencia de Regulación de Faltas Graves	15 días	mié 20/11/13	mar 10/12/13
15	✓	Tercera Iteración	56 días	mié 11/12/13	mié 26/02/14
16	✓	Modificación de Aspirantes	10 días	mié 11/12/13	mar 24/12/13
17	✓	Baja de Aspirantes	10 días	jue 02/01/14	mié 15/01/14
18	✓	Modificación de Instructores	10 días	jue 16/01/14	mié 29/01/14
19	✓	Baja de Instructores	10 días	jue 30/01/14	mié 12/02/14
20	✓	Notificación de Sancionados	10 días	jue 13/02/14	mié 26/02/14
21	✓	Cuarta Iteración	20 días	jue 27/02/14	mié 26/03/14
22	✓	Reporte por Promoción	5 días	jue 27/02/14	mié 05/03/14
23	✓	Reporte de Aspirante	5 días	jue 06/03/14	mié 12/03/14
24	✓	Reporte de Todos los Alumnos	5 días	jue 13/03/14	mié 19/03/14
25	✓	Reporte por Instructor	5 días	jue 20/03/14	mié 26/03/14

2.1 HISTORIAS DE LOS USUARIOS

Historia de Usuario	
Número: 1	**Usuario:** Desarrollo
Nombre historia: Alta de Aspirantes	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 5	**Iteración asignada: 1**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: • Ingreso de Alumnos con los siguientes atributos: nombres, apellidos, cedula, dirección, teléfono, teléfono móvil, edad, fecha de nacimiento, lugar de nacimiento, mail, promoción. • Los alumnos de primer año dispondrán de 120 méritos. • Los alumnos de segundo año tendrán 80 méritos.	
Observaciones: Los méritos son la cantidad límite de puntos que posee un aspirante durante un año según el Reglamento de Sanciones y Recompensas para Aspirantes a Oficiales y Tropa de las FFAA; los puntos se van disminuyendo mediante son sancionados.	

Historia de Usuario	
Número: 2	**Usuario:** Desarrollo
Nombre historia: Modificación de Aspirantes	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 5	**Iteración asignada: 3**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: • Modificación de Aspirantes de los siguientes atributos: dirección, teléfono, teléfono móvil, mail, promoción.	
Observaciones: Se podrá visualizar los datos de cada Aspirante	

Historia de Usuario	
Número: 3	**Usuario:** Desarrollo
Nombre historia: Baja de Aspirantes	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 5	**Iteración asignada: 3**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: • Baja de Aspirantes. En cualquiera de los casos que el Aspirante deje de pertenecer a la ETFA sea alta, baja voluntaria, baja médica, baja académica o baja disciplinaria; pasara a una inactividad en el sistema.	
Observaciones: Únicamente lo podrá realizar el Administrador en los casos que el Reglamento de Sanciones y Recompensas para Aspirantes a Oficiales y Tropa de las FFAA lo avale. El sistema no realizará ningún proceso sino será el administrador quien con la previa documentación legal ponga al Aspirante en estado inactivo, pudiendo obtener únicamente reportes específicos del mismo.	

Historia de Usuario	
Número: 4	**Usuario:** Desarrollo
Nombre historia: Alta de instructores	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 5	**Iteración asignada: 1**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: • Ingreso de instructores con los siguientes atributos: grado, cargo, nombres, apellidos, cedula, dirección, teléfono, teléfono móvil, edad, fecha de nacimiento, lugar de nacimiento, mail.	
Observaciones: el grado de cada instructor podrá ser escogido por el usuario entre los siguientes: Soldado (Sldo.), Cabo Segundo (Cbos.), Cabo Primero (Cbop.), Sargento Segundo (Sgos.), Sargento Primero (Sgop.), Suboficial Segundo (Subs.), Suboficial Primero (Subp.), Suboficial Mayor (Subm.), Subteniente (Subt.), Teniente (Tnte.), Capitán (Capt.), Mayor (Mayo.), Teniente Coronel (Tcrn.), Coronel (Crnl.), Brigadier General (Brig.), Teniente General (Tgra.) y General del Aire(Graa). El atributo grado será seleccionado usando las abreviaturas.	

Historia de Usuario	
Número: 5	**Usuario:** Desarrollo
Nombre historia: Modificación de Instructores	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 5	**Iteración asignada: 3**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: • Modificación de instructores únicamente de los siguientes atributos: grado, cargo, dirección, teléfono, teléfono móvil, mail.	
Observaciones: La modificación de mencionados atributos se podrá realizar únicamente por el administrador en caso que se hayan modificado sus datos y en el caso especial del grado y cargo se lo realizará cuando el instructor haya ascendido o haya sido cambiado de cargo respectivamente.	

Historia de Usuario	
Número: 6	**Usuario:** Desarrollo
Nombre historia: Baja de instructores	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 5	**Iteración asignada: 3**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: • Pasará a una inactividad en el sistema.	
Observaciones: la baja de un instructor del sistema será realizada por el administrador en caso que mencionado instructor haya sido dado el pase o pase al servicio pasivo de las FFAA, es sistema no intervendrá con ningún proceso, únicamente se dará de baja el registro del instructor, los registros de sanciones y méritos en que se encuentre involucrado no se eliminarán.	

Historia de Usuario	
Número: 7	**Usuario:** Desarrollo
Nombre historia: Alta de faltas	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 5	**Iteración asignada: 1**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: • Ingreso de faltas con los siguientes atributos: Articulo, literal. Las faltas disciplinarias serán: Leves, Graves y Atentatorias y cada una de ellas serán clasificadas por: Contra la subordinación, abuso de facultades, contra los deberes y obligaciones militares, contra la puntualidad y asistencia, contra el decoro personal y compostura militar, contra la propiedad, contra la salubridad e higiene, contra la moral, contra la seguridad de las operaciones militares.	
Observaciones: Las faltas ingresadas serán las que constan en el Reglamento de Sanciones y Recompensas para Aspirantes a Oficiales y Tropa de las FFAA	

Historia de Usuario	
Número: 8	**Usuario:** Desarrollo
Nombre historia: Modificación de Faltas	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 5	**Iteración asignada: 1**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: • Se realizará para el atributo descripción.	
Observaciones: Se podrá modificar únicamente en el caso que el Reglamento de Sanciones y Recompensas para Aspirantes a Oficiales y Tropa de las FFAA haya sido legalmente actualizado o modificado.	

Historia de Usuario	
Número: 9	**Usuario:** Desarrollo
Nombre historia: Alta de sanciones	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 5	**Iteración asignada: 1**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: • Ingreso de sanciones con los siguientes atributos: Descripción, deméritos. • Las faltas leves tendrán las siguientes sanciones y la resta del siguiente número de deméritos: Censura (2 DEMÉRITOS), 30’ de mantenimiento de instalaciones (3 DEMÉRITOS), 30’ de ejercicios físicos (4 DEMÉRITOS), ½ día de perdida de franquicia (5 DEMÉRITOS), 1 día de perdida de franquicia (6 DEMÉRITOS).	

• Las faltas graves tendrán las siguientes sanciones y la resta del siguiente número de deméritos: Perdida de franquicia de 2 días (7 DEMÉRITOS), Perdida de franquicia de 3 días (8 DEMÉRITOS), Perdida de franquicia de 4 días (10 DEMÉRITOS), Perdida de franquicia de 5 días (12 DEMÉRITOS), Perdida de franquicia de 6 días (14 DEMÉRITOS), rutina disciplinaria de 5 días (16 DEMÉRITOS), rutina disciplinaria de 7 días (18 DEMÉRITOS), rutina disciplinaria de 7 días (20 DEMÉRITOS), • Las faltas atentatorias tendrán las siguientes sanciones y la resta del siguiente número de deméritos: 8 día de perdida de franquicia (21 DEMÉRITOS), rutina disciplinaria de 10 días (27 DEMÉRITOS), rutina disciplinaria de 15 días (34 DEMÉRITOS), rutina disciplinaria de 15 días (40 DEMÉRITOS),
Observaciones: El demerito es el número de puntos a descontar a cada aspirante cada vez que se registre una nueva sanción. Las sanciones para las faltas leves están relacionadas directamente y únicamente con las faltas leves, las sanciones graves con las faltas graves y las sanciones atentatorias con las faltas atentatorias.

Historia de Usuario	
Número: 10	**Usuario:** Desarrollo
Nombre historia: Modificación de sanciones	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 5	**Iteración asignada: 1**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: • Modificación de sanciones en los campos descripción y deméritos.	
Observaciones: Se podrá modificar sanciones únicamente por el usuario administrador únicamente en caso que el Reglamento de Sanciones y Recompensas para Aspirantes a Oficiales y Tropa de las FFAA haya sido legalmente modificado.	

Historia de Usuario	
Número: 11	**Usuario:** Desarrollo
Nombre historia: Perfiles de usuario	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 5	**Iteración asignada: 1**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: • Existirán 3 tipos de usuarios: Administrador, regulador, secretario. **Secretario:** podrá realizar el registro únicamente de faltas leves y graves según sea el caso, podrá sacar reportes según sea la necesidad e imprimirlos. **Regulador:** tendrá acceso a una ventana especial y única para el donde el sistema le permitirá regular las faltas graves y su función estará en escoger la sanción a imponer a un Aspirante según la falta que se haya cometido, podrá imprimir reportes. **Administrador:** podrá realizar la gestión de aspirantes, instructores, faltas, sanciones, imprimir reportes, excepto regular sanciones**.**	
Observaciones: El sistema permitirá que si un alumno no ha sido sancionado antes con la misma falta, sugerir mediante un pop up al regulador la menor sanción, en caso que la falta sea cometida por segunda vez el sistema sugerirá la siguiente sanción, sucesivamente por reincidencia.	

Historia de Usuario	
Número: 12	**Usuario:** Desarrollo
Nombre historia: Registro de sanciones leves y graves	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 5	**Iteración asignada: 2**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: Se guardará con los siguientes campos: Grado (Atro.), nombres, Articulo, literal, sanción, sancionador, deméritos restados y méritos disponibles. El registro de sanciones lo realizará el secretario seleccionando el nombre del aspirante de la lista desplegada en el sistema o a su vez pondrá las primeras letras del nombre para que se vaya mostrando en el sistema los nombres con el filtro ingresado, de la misma forma seleccionará el tipo de falta si es leve o grave y seleccionará la sanción. El secretario solo podrá ingresar faltas leves y graves.	

La sanción a escoger por el usuario se reflejará en pantalla para selección del usuario dependiendo del tipo de falta que este haya escogido. En caso que el secretario haya ingresado una falta leve, esta se guardará, imprimirá y notificará al sancionado. (**ver H15**). Si el usuario ingreso una falta grave, el sistema enviará la información de la falta grave cometida por un Aspirante al regulador el mismo que tendrá en pantalla los campos: grado, nombres, artículo, literal, sanción, deméritos, sancionador y el motivo, el regulador en pantalla podrá aumentar la sanción o disminuir, es sistema le permitirá recibir una sugerencia en base a la experiencia. (**ver H17**) Una vez que el regulador haya puesto la sanción el sistema guardará la información y notificará (ver H16) al sancionado.
Observaciones: Esto se lo realiza en base al Reglamento de Sanciones y Recompensas para Aspirantes a Oficiales y Tropa de las FFAA.

Historia de Usuario	
Número: 13	**Usuario:** Desarrollo
Nombre historia: Registro de sanciones atentatorias	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 5	**Iteración asignada: 2**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: Se podrá ingresar faltas atentatorias únicamente por el regulador con los siguientes campos: grado, nombres del Aspirante, artículo, literal, deméritos y méritos disponibles.	
Observaciones: Esto se lo realiza en base al Reglamento de Sanciones y Recompensas para Aspirantes a Oficiales y Tropa de las FFAA.	

Historia de Usuario	
Número: 14	**Usuario:** Desarrollo
Nombre historia: Notificación de sancionados	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 5	**Iteración asignada: 3**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: El sistema enviará un e mail al aspirante sancionado, si es de una falta leve lo enviará directamente luego del registro del secretario. Si es una falta grave lo enviará posterior a la ratificación y moderación de la sanción por parte del regulador.	

Si es de una falta atentatoria se enviará una vez que el regulador haya ingresado la falta y sanción. El email se enviará con los siguientes campos: Grado, nombres, Articulo Literal, sancionador, deméritos, méritos disponibles.
Observaciones: Se usara los emails de los aspirantes que se ingreso en el alta de los mismos.

Historia de Usuario	
Número: 15	**Usuario:** Desarrollo
Nombre historia: Notificación mensual de deméritos.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 5	**Iteración asignada: 2**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: El sistema enviará un email el primer día de cada mes a cada aspirante con los siguientes campos: Grado, nombres, méritos disponibles.	
Observaciones:	

Historia de Usuario	
Número: 16	**Usuario:** Desarrollo
Nombre historia: Sugerencia de regulación de faltas graves	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 5	**Iteración asignada: 2**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: El sistema sugerirá al regulador que sanción es la más adecuada para cada Aspirante en las faltas graves de acuerdo a las veces que es Aspirante haya sido sancionado con el mismo artículo y literal, según las siguientes consideraciones: Si es la primera vez que el Aspirante es sancionado en base a una misma falta (artículo y literal), el sistema sugerirá que sea sancionado con 2 días de perdida de franquicia**.** Si es la segunda vez que el Aspirante es sancionado en base a una misma falta (artículo y literal), el sistema sugerirá que sea sancionado con 4 días de perdida de franquicia**.**	

Si es la tercera vez que el Aspirante es sancionado en base a una misma falta (artículo y literal), el sistema sugerirá que sea sancionado con 6 días de perdida de franquicia**.** Si es la cuarta vez que el Aspirante es sancionado en base a una misma falta (artículo y literal), el sistema sugerirá que sea sancionado con rutina disciplinaria de 5 días. Si es la quinta vez en adelante que el Aspirante es sancionado en base a una misma falta (artículo y literal), el sistema sugerirá que sea sancionado con rutina disciplinaria de 9 días.

Historia de Usuario	
Número: 17	**Usuario:** Administrador
Nombre historia: Reporte por Promoción	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Puntos estimados: 5	**Iteración asignada: 4**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: • Mostrará en un tiempo seleccionado (diario, semanal, mensual, anual) la lista de toda la promoción sancionada con fecha, falta, sancionadora, deméritos bajados, deméritos disponibles.	
Observaciones: El reporte será un pdf y con el encabezado proporcionado por la ETFA.	

Historia de Usuario	
Número: 18	**Usuario:** Administrador
Nombre historia: Reporte Aspirante	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Puntos estimados: 5	**Iteración asignada: 4**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: Mostrará las faltas cometidas por un mismo aspirante en un periodo determinado, tomando en cuenta los siguientes campos: Faltas leves: fecha, articulo, literal, sanción, deméritos, sancionador. Novedad. Faltas graves: fecha, articulo, literal, sanción, deméritos, sancionador. Novedad. Faltas atentatorias: fecha, articulo, literal, sanción, deméritos, sancionador. Novedad.	

Historia de Usuario	
Número: 19	**Usuario:** Administrador
Nombre historia: Reporte de todos los Alumnos	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Puntos estimados: 8	**Iteración asignada: 4**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: Mostrará un listado de todos los Aspirantes con sus méritos disponibles a la fecha.	
Observaciones: Ninguna.	

Historia de Usuario	
Número: 20	**Usuario:** Administrador
Nombre historia: Reporte por Instructor	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Puntos estimados: 8	**Iteración asignada: 4**
Programador responsable: Xavier Játiva – Luis Egas	
Descripción: Mostrará el listado de Aspirantes sancionador por un mismo instructor.	
Observaciones: Ninguna.	

2.3 PRIORIDAD

Historia N°	Descripción	Prioridad	Iteración
1	Alta de Aspirantes	Alta	1
2	Modificación de Aspirantes	Baja	3
3	Baja de Aspirantes	Baja	3
4	Alta de Instructores	Alta	1
5	Modificación Instructores	Baja	3
6	Baja de Instructores	Baja	3
7	Alta de Faltas	Alta	1
8	Modificación de Faltas	Alta	1
9	Alta de Sanciones	Alta	1
10	Modificación de Sanciones	Alta	1
11	Perfiles de Usuario	Alta	1
12	Registro de sanciones leves y graves	Media	2
13	Registro de sanciones atentatorias	Media	2
14	Notificación de sancionados	Baja	3
15	Notificación mensual de deméritos	Media	2
16	Sugerencia de regulación de faltas graves	Media	2
17	Reporte por Promoción	Media	4
18	Reporte Aspirante	Media	4
19	Reporte de todos los Alumnos	Media	4
20	Reporte por Instructor	Media	4

3ª FASE: ITERACIONES

3.1. DIVISIÓN EN ITERACIONES

De acuerdo con los interesados del sistema se desarrollara en 4 iteraciones detalladas a continuación, las cuales en cada reunión en fechas elegidas entre ambos interesados con el fin de poder realizar correcciones en el sistema.

Plan de iteraciones (Release Planning)

Iteración	Historias	Fecha Inicio	Fecha Reunión	Fecha Entrega
Primera Iteración	H1, H4, H7, H8, H9, H10, H11	09-09-2013	23-09-2013	18-10-2013
Segunda Iteración	H12, H13, H15, H16	21-10-2013	29-10-2013	26-11-2013
Tercera Iteración	H2, H3, H5, H6 H14	27-11-2013	02-01-2014	12-02-2014
Cuarta Iteración	H17, H18, H19, H20	13-02-2014	20-03-2014	09-04-2014

3.2. VELOCIDAD DEL PROYECTO

Horario de Trabajo Semanal				
Lunes	**Martes**	**Miércoles**	**Jueves**	**Viernes**
08:00pm	**08:00pm**	**08:00pm**	**08:00pm**	**08:00pm**
13:00pm	**13:00pm**	**13:00pm**	**13:00pm**	**13:00pm**

	ITERACIÓN 1	ITERACIÓN 2	ITERACIÓN 3	ITERACIÓN 4
Horas	200	135	250	100
Semanas	7	5,4	6	4
horas semanales	29	34	42	25
H .de usuario	7	4	5	4

3.4. REUNIONES DIARIAS

Sistema Inteligente de Control Disciplinario Atros. ETFA [SICODIAT]

Control Diario

N° ……
Iteracción

FECHA DEFINICIÓN DE LA HISTORIA			PROCESO	DESCRIPCIÓN TAREA	TIPO DE TAREA	RESPONSABLE	FECHA SEGUIMIENTO			ESTADO	OBSERVACIONES
DIA	MES	AÑO					DIA	MES	AÑO		
09	**09**	**2013**	Aspirantes	Diseño de Formulario con campos requeridos	Diseño	*Javier Játiva Luis Égas*	10	09	2013	**CUMPLIDA**	Ninguna
10	09	2013	Aspirantes	Validación de campos	Programación	*Javier Játiva Luis Égas*	11	09	2013	**CUMPLIDA**	Ninguna
11	09	2013	Aspirantes	Script para el ingreso de nuevo aspirante	Programación	*Javier Játiva Luis Égas*	12	09	2013	**CUMPLIDA**	Ninguna
12	09	2013	Aspirantes	Validación de ingreso	Programación	*Javier Játiva Luis Égas*	13	09	2013	**CUMPLIDA**	Ninguna
13	09	2013	Aspirantes	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	16	09	2013	**CUMPLIDA**	Ninguna
16	**09**	**2013**	Instructores	Diseño de Formulario con campos requeridos	Diseño	*Javier Játiva Luis Égas*	17	09	2013	**CUMPLIDA**	Ninguna
17	09	2013	Instructores	Validación de campos	Programación	*Javier Játiva Luis Égas*	18	09	2013	**CUMPLIDA**	Ninguna
18	09	2013	Instructores	Script para el ingreso de nuevo instructor	Programación	*Javier Játiva Luis Égas*	19	09	2013	**CUMPLIDA**	Ninguna
19	09	2013	Instructores	Validación de ingreso	Programación	*Javier Játiva Luis Égas*	20	09	2013	**CUMPLIDA**	Ninguna
20	09	2013	Instructores	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	23	09	2013	**CUMPLIDA**	Ninguna
23	**09**	**2013**	Faltas	Diseño de Formulario con campos requeridos	Diseño	*Javier Játiva Luis Égas*	24	09	2013	**CUMPLIDA**	Ninguna
24	09	2013	Faltas	Validación de campos	Programación	*Javier Játiva Luis Égas*	25	09	2013	**CUMPLIDA**	Ninguna
25	09	2013	Faltas	Script para el ingreso de nueva falta	Programación	*Javier Játiva Luis Égas*	26	09	2013	**CUMPLIDA**	Ninguna
26	09	2013	Faltas	Validación de ingreso	Programación	*Javier Játiva Luis Égas*	27	09	2013	**CUMPLIDA**	Ninguna

27	09	2013	Faltas	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	30	09	2013	**CUMPLIDA**	Ninguna
30	**09**	**2013**	Faltas	Re-Diseño de formulario para modificación de falta	Diseño	*Javier Játiva Luis Égas*	01	09	2013	**CUMPLIDA**	Ninguna
01	10	2013	Faltas	Implementación de variables get, rellenando campos	Programación	*Javier Játiva Luis Égas*	02	09	2013	**CUMPLIDA**	Ninguna
02	10	2013	Faltas	Script para la modificación de falta	Programación	*Javier Játiva Luis Égas*	03	09	2013	**CUMPLIDA**	Ninguna
03	10	2013	Faltas	Validación de modificación	Programación	*Javier Játiva Luis Égas*	04	09	2013	**CUMPLIDA**	Ninguna
04	10	2013	Faltas	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	07	09	2013	**CUMPLIDA**	Ninguna
07	**10**	**2013**	Sanción	Diseño de Formulario con campos requeridos	Diseño	*Javier Játiva Luis Égas*	08	09	2013	**CUMPLIDA**	Ninguna
08	10	2013	Sanción	Validación de campos	Programación	*Javier Játiva Luis Égas*	09	10	2013	**CUMPLIDA**	Ninguna
09	10	2013	Sanción	Script para el ingreso de nueva sanción	Programación	*Javier Játiva Luis Égas*	10	10	2013	**CUMPLIDA**	Ninguna
10	10	2013	Sanción	Validación de ingreso	Programación	*Javier Játiva Luis Égas*	11	10	2013	**CUMPLIDA**	Ninguna
11	10	2013	Sanción	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	14	10	2013	**CUMPLIDA**	Ninguna
14	**10**	**2013**	Sanción	Re-Diseño de formulario para modificación de Sanción	Diseño	*Javier Játiva Luis Égas*	15	10	2013	**CUMPLIDA**	Ninguna
15	10	2013	Sanción	Implementación de variables get, rellenando campos	Programación	*Javier Játiva Luis Égas*	16	10	2013	**CUMPLIDA**	Ninguna
16	10	2013	Sanción	Script para la modificación de Sanción	Programación	*Javier Játiva Luis Égas*	17	10	2013	**CUMPLIDA**	Ninguna
17	10	2013	Sanción	Validación de modificación	Programación	*Javier Játiva Luis Égas*	18	10	2013	**CUMPLIDA**	Ninguna
18	10	2013	Sanción	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	21	10	2013	**CUMPLIDA**	Ninguna
21	**10**	**2013**	Perfil	Diseño de interfaz súper-administrador	Diseño	*Javier Játiva Luis Égas*	22	10	2013	**CUMPLIDA**	Ninguna

22	10	2013	Perfil	Diseño de interfaz regulador y administrador	Diseño	*Javier Játiva Luis Égas*	16	10	2013	**CUMPLIDA**	Ninguna
23	10	2013	Perfil	Diseño de interfaz de ingreso a perfil de Usuario	Diseño	*Javier Játiva Luis Égas*	17	10	2013	**CUMPLIDA**	Ninguna
24	10	2013	Perfil	Validación de ingreso a perfil de usuario	Programación	*Javier Játiva Luis Égas*	25	10	2013	**CUMPLIDA**	Ninguna
25	10	2013	Perfil	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	25	10	2013	**CUMPLIDA**	Ninguna
28	**10**	**2013**	Sanción	Diseño de Formulario con campos requeridos	Diseño	*Javier Játiva Luis Égas*	29	10	2013	**CUMPLIDA**	Ninguna
29	10	2013	Sanción	Validación de campos	Diseño	*Javier Játiva Luis Égas*	30	10	2013	**CUMPLIDA**	Ninguna
30	10	2013	Sanción	Script para el registro de falta leve	Programación	*Javier Játiva Luis Égas*	31	10	2013	**CUMPLIDA**	Ninguna
31	10	2013	Sanción	Script para el registro de falta grave	Programación	*Javier Játiva Luis Égas*	01	11	2013	**CUMPLIDA**	Ninguna
01	11	2013	Sanción	Validación de ingreso	Pruebas	*Javier Játiva Luis Égas*	04	11	2013	**CUMPLIDA**	Ninguna
04	11	2013	Sanción	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	05	11	2013	**CUMPLIDA**	Ninguna
05	**11**	**2013**	Sanción	Diseño de Formulario con campos requeridos	Diseño	*Javier Játiva Luis Égas*	06	11	2013	**CUMPLIDA**	Ninguna
06	11	2013	Sanción	Validación de campos	Diseño	*Javier Játiva Luis Égas*	07	11	2013	**CUMPLIDA**	Ninguna
07	11	2013	Sanción	Script para el registro de falta atentatoria	Diseño	*Javier Játiva Luis Égas*	08	11	2013	**CUMPLIDA**	Ninguna
08	11	2013	Sanción	Validación de ingreso	Pruebas	*Javier Játiva Luis Égas*	11	11	2013	**CUMPLIDA**	Ninguna
11	11	2013	Sanción	Script para la regulación de la falta atentatoria	Programación	*Javier Játiva Luis Égas*	12	11	2013	**CUMPLIDA**	Ninguna
12	11	2013	Sanción	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	13	11	2013	**CUMPLIDA**	Ninguna
13	**11**	**2013**	Méritos	Script para detectar inicio de mes	Programación	*Javier Játiva Luis Égas*	14	11	2013	**CUMPLIDA**	Ninguna

14	11	2013	Méritos	Script para obtener campos requeridos de aspirantes	Programación	*Javier Játiva Luis Égas*	15	11	2013	**CUMPLIDA**	Ninguna
15	11	2013	Méritos	Creación del cuerpo del email	Diseño	*Javier Játiva Luis Égas*	18	11	2013	**CUMPLIDA**	Ninguna
18	11	2013	Méritos	Script para envió de email a las direcciones de los aspirantes	Programación	*Javier Játiva Luis Égas*	19	11	2013	**CUMPLIDA**	Ninguna
19	11	2013	Méritos	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	20	11	2013	**CUMPLIDA**	Ninguna
20	**11**	**2013**	Regulación	Diseño de tabla para visualizar sugerencias	Diseño	*Javier Játiva Luis Égas*	21	11	2013	**CUMPLIDA**	Ninguna
21	11	2013	Regulación	Script para obtener número de veces que ha sido sancionado	Programación	*Javier Játiva Luis Égas*	22	11	2013	**CUMPLIDA**	Ninguna
22	11	2013	Regulación	Script para el cálculo de la sugerencia de la sanción de acuerdo a al número de veces que ha sido sancionado	Programación	*Javier Játiva Luis Égas*	25	11	2013	**CUMPLIDA**	Ninguna
25	10	2013	Regulación	Script para el registro de la sanción sugerida	Programación	*Javier Játiva Luis Égas*	26	11	2013	**CUMPLIDA**	Ninguna
26	11	2013	Regulación	Validación de actualización de sanción	Pruebas	*Javier Játiva Luis Égas*	27	11	2013	**CUMPLIDA**	Ninguna
27	11	2013	Regulación	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	28	11	2013	**CUMPLIDA**	Ninguna
28	11	2013	Regulación	Script para el cálculo de la sugerencia de la sanción de acuerdo a al número de veces que ha sido sancionado	Programación	*Javier Játiva Luis Égas*	29	11	2013	**CUMPLIDA**	Ninguna
29	11	2013	Regulación	Script para el registro de la sanción sugerida	Programación	*Javier Játiva Luis Égas*	02	12	2013	**CUMPLIDA**	Ninguna
02	12	2013	Regulación	Validación de actualización de sanción	Pruebas	*Javier Játiva Luis Égas*	03	12	2013	**CUMPLIDA**	Ninguna
03	12	2013	Regulación	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	04	12	2013	**CUMPLIDA**	Ninguna
04	12	2013	Regulación	Script para el cálculo de la sugerencia de la sanción de acuerdo a al número de veces que ha	Programación	*Javier Játiva Luis Égas*	05	12	2013	**CUMPLIDA**	Ninguna

				sido sancionado							
05	12	2013	Regulación	Script para el registro de la sanción sugerida	Programación	*Javier Játiva Luis Égas*	06	12	2013	**CUMPLIDA**	Ninguna
06	12	2013	Regulación	Validación de actualización de sanción	Pruebas	*Javier Játiva Luis Égas*	06	12	2013	**CUMPLIDA**	Ninguna
09	12	2013	Regulación	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	10	12	2013	**CUMPLIDA**	Ninguna
10	12	2013	Regulación	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	10	12	2013	**CUMPLIDA**	Ninguna
11	**12**	**2013**	Aspirantes	Re-Diseño de formulario para modificación de Aspirante	Diseño	*Javier Játiva Luis Égas*	12	12	2013	**NO CUMPLIDO**	Error estilo visual CSS
12	12	2013	Aspirantes	Re-Diseño de formulario para modificación de Aspirante	Diseño	*Javier Játiva Luis Égas*	13	12	2013	**CUMPLIDA**	Ninguna
13	12	2013	Aspirantes	Implementación de variables get, rellenando campos	Programación	*Javier Játiva Luis Égas*	16	12	2013	**NO CUMPLIDO**	Error variables get
16	12	2013	Aspirantes	Implementación de variables get, rellenando campos	Programación	*Javier Játiva Luis Égas*	17	12	2013	**CUMPLIDA**	Ninguna
17	12	2013	Aspirantes	Script para la modificación de Aspirante	Programación	*Javier Játiva Luis Égas*	18	12	2013	**NO CUMPLIDO**	Error script
18	12	2013	Aspirantes	Script para la modificación de Aspirante	Programación	*Javier Játiva Luis Égas*	19	12	2013	**CUMPLIDA**	Ninguna
19	12	2013	Aspirantes	Validación de modificación	Programación	*Javier Játiva Luis Égas*	20	12	2013	**NO CUMPLIDO**	Error en BD
20	12	2013	Aspirantes	Validación de modificación	Programación	*Javier Játiva Luis Égas*	23	12	2013	**CUMPLIDA**	Ninguna
23	12	2013	Aspirantes	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	24	12	2013	**NO CUMPLIDO**	50 %
24	12	2013	Aspirantes	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	24	12	2013	**CUMPLIDA**	Ninguna
02	**01**	**2014**	Aspirantes	Re-Diseño de formulario para baja de Aspirante	Diseño	*Javier Játiva Luis Égas*	03	01	2014	**NO CUMPLIDO**	Error estilo visual CSS
03	01	2014	Aspirantes	Re-Diseño de formulario para baja de Aspirante	Diseño	*Javier Játiva Luis Égas*	06	01	2014	**CUMPLIDA**	Ninguna

06	01	2014	Aspirantes	Implementación de variables get, rellenando campos	Programación	*Javier Játiva Luis Égas*	07	01	2014	**NO CUMPLIDO**	Error variables get
07	01	2014	Aspirantes	Implementación de variables get, rellenando campos	Programación	*Javier Játiva Luis Égas*	08	01	2014	**CUMPLIDA**	Ninguna
08	01	2014	Aspirantes	Script para la baja de Aspirante	Programación	*Javier Játiva Luis Égas*	09	01	2014	**NO CUMPLIDO**	Error script
09	01	2014	Aspirantes	Script para la baja de Aspirante	Programación	*Javier Játiva Luis Égas*	10	01	2014	**CUMPLIDA**	Ninguna
10	01	2014	Aspirantes	Validación de baja	Programación	*Javier Játiva Luis Égas*	13	01	2014	**NO CUMPLIDO**	Error en BD
13	01	2014	Aspirantes	Validación de baja	Programación	*Javier Játiva Luis Égas*	14	01	2014	**CUMPLIDA**	Ninguna
14	01	2014	Aspirantes	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	16	01	2014	**NO CUMPLIDO**	50 %
15	01	2014	Aspirantes	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	16	01	2014	**CUMPLIDA**	Ninguna
16	**01**	**2013**	Aspirantes	Re-Diseño de formulario para modificación de instructor	Diseño	*Javier Játiva Luis Égas*	17	01	2014	**NO CUMPLIDO**	Error estilo visual CSS
17	01	2014	Aspirantes	Re-Diseño de formulario para modificación de instructor	Diseño	*Javier Játiva Luis Égas*	20	01	2014	**CUMPLIDA**	Ninguna
20	01	2014	Aspirantes	Implementación de variables get, rellenando campos	Programación	*Javier Játiva Luis Égas*	21	01	2014	**NO CUMPLIDO**	Error variables get
21	01	2014	Aspirantes	Implementación de variables get, rellenando campos	Programación	*Javier Játiva Luis Égas*	22	01	2014	**CUMPLIDA**	Ninguna
22	01	2014	Aspirantes	Script para la modificación de instructor	Programación	*Javier Játiva Luis Égas*	23	01	2014	**NO CUMPLIDO**	Error script
23	01	2014	Aspirantes	Script para la modificación de instructor	Programación	*Javier Játiva Luis Égas*	24	01	2014	**CUMPLIDA**	Ninguna
24	01	2014	Aspirantes	Validación de modificación	Programación	*Javier Játiva Luis Égas*	27	01	2014	**NO CUMPLIDO**	Error en BD
27	01	2014	Aspirantes	Validación de modificación	Programación	*Javier Játiva Luis Égas*	28	01	2014	**CUMPLIDA**	Ninguna
28	01	2014	Aspirantes	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	29	01	2014	**NO CUMPLIDO**	50 %

29	01	2014	Aspirantes	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	30	01	2014	**CUMPLIDA**	Ninguna
30	**01**	**2014**	Aspirantes	Re-Diseño de formulario para baja de instructor	Diseño	*Javier Játiva Luis Égas*	31	01	2014	**NO CUMPLIDO**	Error estilo visual CSS
31	01	2014	Aspirantes	Re-Diseño de formulario para baja de instructor	Diseño	*Javier Játiva Luis Égas*	03	02	2014	**CUMPLIDA**	Ninguna
03	02	2014	Aspirantes	Implementación de variables get, rellenando campos	Programación	*Javier Játiva Luis Égas*	04	02	2014	**NO CUMPLIDO**	Error variables get
04	02	2014	Aspirantes	Implementación de variables get, rellenando campos	Programación	*Javier Játiva Luis Égas*	05	02	2014	**CUMPLIDA**	Ninguna
05	02	2014	Aspirantes	Script para la baja de instructor	Programación	*Javier Játiva Luis Égas*	06	02	2014	**NO CUMPLIDO**	Error script
06	02	2014	Aspirantes	Script para la baja de instructor	Programación	*Javier Játiva Luis Égas*	07	02	2014	**CUMPLIDA**	Ninguna
07	02	2014	Aspirantes	Validación de baja	Programación	*Javier Játiva Luis Égas*	10	02	2014	**NO CUMPLIDO**	Error en BD
10	02	2014	Aspirantes	Validación de baja	Programación	*Javier Játiva Luis Égas*	11	02	2014	**CUMPLIDA**	Ninguna
11	02	2014	Aspirantes	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	12	02	2014	**NO CUMPLIDO**	50 %
12	02	2014	Aspirantes	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	13	02	2014	**CUMPLIDA**	Ninguna
13	**02**	**2014**	Sanción	Creación del cuerpo del email	Diseño	*Javier Játiva Luis Égas*	14	02	2014	**CUMPLIDA**	Ninguna
14	02	2014	Sanción	Script para envió de email al ser falta leve	Programación	*Javier Játiva Luis Égas*	17	02	2014	**CUMPLIDA**	Ninguna
17	02	2014	Sanción	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	18	02	2014	**CUMPLIDA**	Ninguna
18	02	2014	Sanción	Script para envió de email al ser falta grave	Programación	*Javier Játiva Luis Égas*	19	02	2014	**CUMPLIDA**	Ninguna
19	02	2014	Sanción	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	20	02	2014	**CUMPLIDA**	Ninguna
20	02	2014	Sanción	Script para envió de email al ser falta atentatoria	Programación	*Javier Játiva Luis Égas*	21	02	2014	**CUMPLIDA**	Ninguna

21	02	2014	Sanción	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	24	02	2014	**CUMPLIDA**	Ninguna
24	02	2014	Sanción	Script para envió de email a la direcciones del aspirante sancionado	Programación	*Javier Játiva Luis Égas*	25	02	2014	**CUMPLIDA**	Ninguna
25	02	2014	Sanción	Validación de aspirante sancionado	Programación	*Javier Játiva Luis Égas*	26	02	2014	**CUMPLIDA**	Ninguna
26	02	2014	Sanción	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	26	02	2014	**CUMPLIDA**	Ninguna
27	**02**	**2014**	Reporte	Diseño de interfaz para selección de Promoción	Diseño	*Javier Játiva Luis Égas*	28	02	2014	**CUMPLIDA**	Ninguna
28	02	2014	Reporte	Creación del documento PDF	Programación	*Javier Játiva Luis Égas*	03	03	2014	**CUMPLIDA**	Ninguna
03	03	2014	Reporte	Script para obtener datos de los aspirantes de la promoción	Programación	*Javier Játiva Luis Égas*	04	03	2014	**CUMPLIDA**	Ninguna
04	03	2014	Reporte	Ubicar datos de aspirantes en documento PDF	Programación	*Javier Játiva Luis Égas*	05	03	2014	**CUMPLIDA**	Ninguna
05	03	2014	Reporte	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	06	03	2014	**CUMPLIDA**	Ninguna
06	**03**	**2014**	Reporte	Diseño de interfaz para selección de los aspirantes	Diseño	*Javier Játiva Luis Égas*	07	03	2014	**CUMPLIDA**	Ninguna
07	03	2014	Reporte	Creación del documento PDF	Programación	*Javier Játiva Luis Égas*	10	03	2014	**CUMPLIDA**	Ninguna
10	03	2014	Reporte	Script para obtener datos de los aspirantes	Programación	*Javier Játiva Luis Égas*	11	03	2014	**CUMPLIDA**	Ninguna
11	03	2014	Reporte	Ubicar datos de aspirantes en documento PDF	Programación	*Javier Játiva Luis Égas*	12	03	2014	**CUMPLIDA**	Ninguna
12	03	2014	Reporte	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	13	03	2014	**CUMPLIDA**	Ninguna
13	**03**	**2014**	Reporte	Diseño de interfaz para selección de todos los alumnos	Diseño	*Javier Játiva Luis Égas*	14	03	2014	**CUMPLIDA**	Ninguna
14	03	2014	Reporte	Creación del documento PDF	Programación	*Javier Játiva Luis Égas*	17	03	2014	**CUMPLIDA**	Ninguna
17	03	2014	Reporte	Script para obtener datos de los alumnos	Programación	*Javier Játiva Luis Égas*	18	03	2014	**CUMPLIDA**	Ninguna

18	03	2014	Reporte	Ubicar datos de aspirantes en documento PDF	Programación	*Javier Játiva Luis Égas*	19	03	2014	**CUMPLIDA**	Ninguna
19	03	2014	Reporte	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	20	03	2014	**CUMPLIDA**	Ninguna
20	**03**	**2014**	Reporte	Diseño de interfaz para selección de instructor	Diseño	*Javier Játiva Luis Égas*	21	03	2014	**CUMPLIDA**	Ninguna
21	03	2014	Reporte	Creación del documento PDF	Programación	*Javier Játiva Luis Égas*	24	03	2014	**CUMPLIDA**	Ninguna
24	03	2014	Reporte	Script para obtener datos de los alumnos que han sido sancionados por el instructor	Programación	*Javier Játiva Luis Égas*	25	03	2014	**CUMPLIDA**	Ninguna
25	03	2014	Reporte	Ubicar datos de aspirantes en documento PDF	Programación	*Javier Játiva Luis Égas*	26	03	2014	**CUMPLIDA**	Ninguna
26	03	2014	Reporte	Pruebas de funcionamiento	Pruebas	*Javier Játiva Luis Égas*	26	03	2014	**CUMPLIDA**	Ninguna

3.4 DISEÑO

Modelo Entidad-Relación:

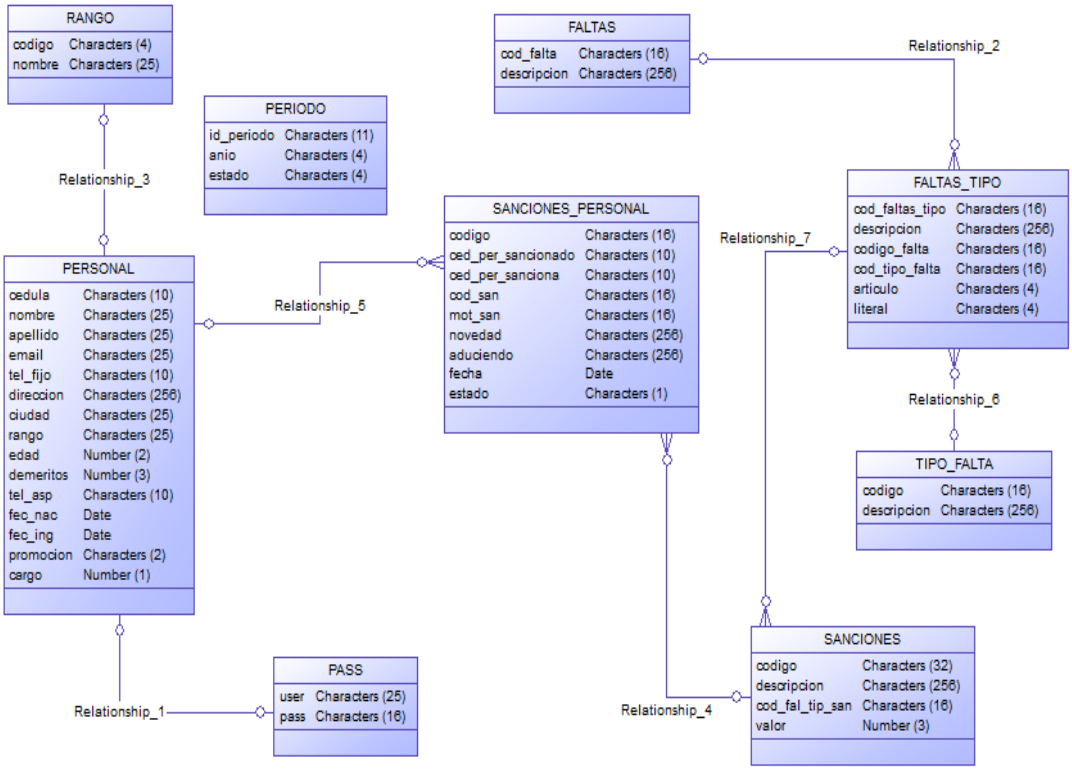
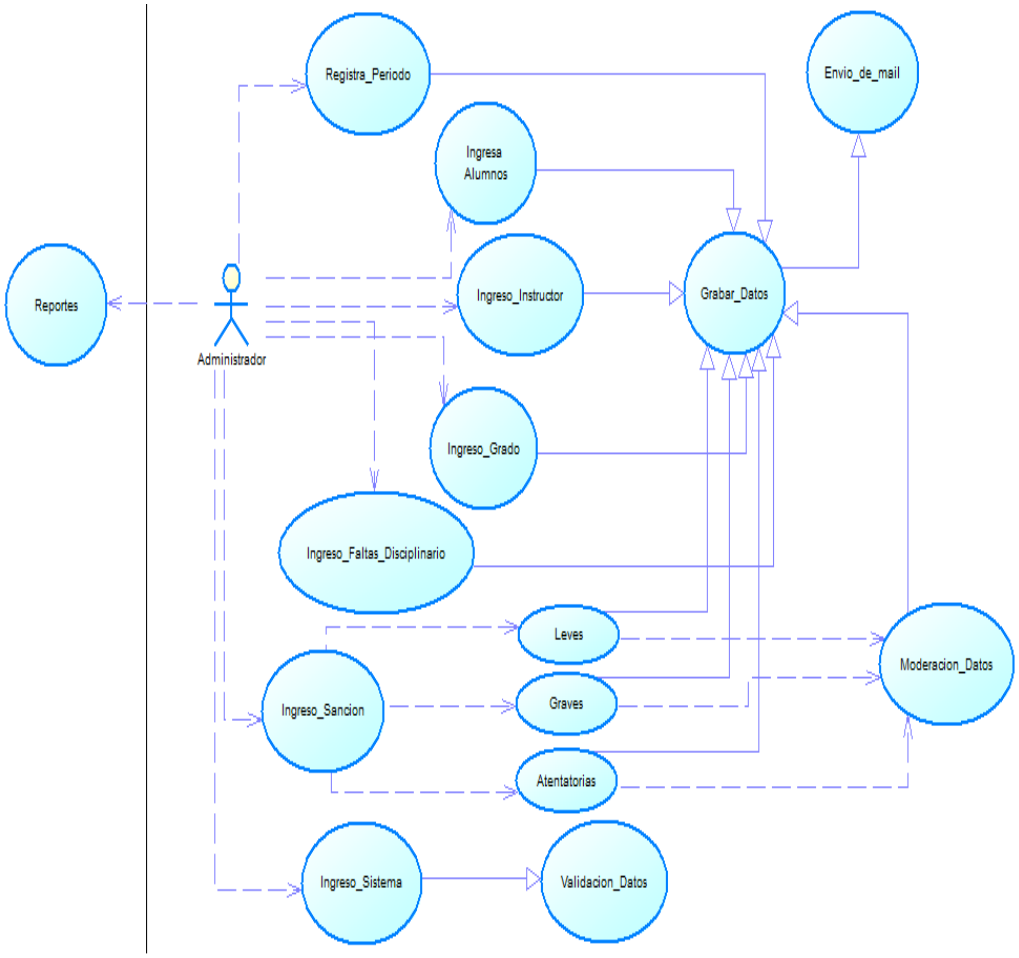


Diagrama de casos de uso del negocio

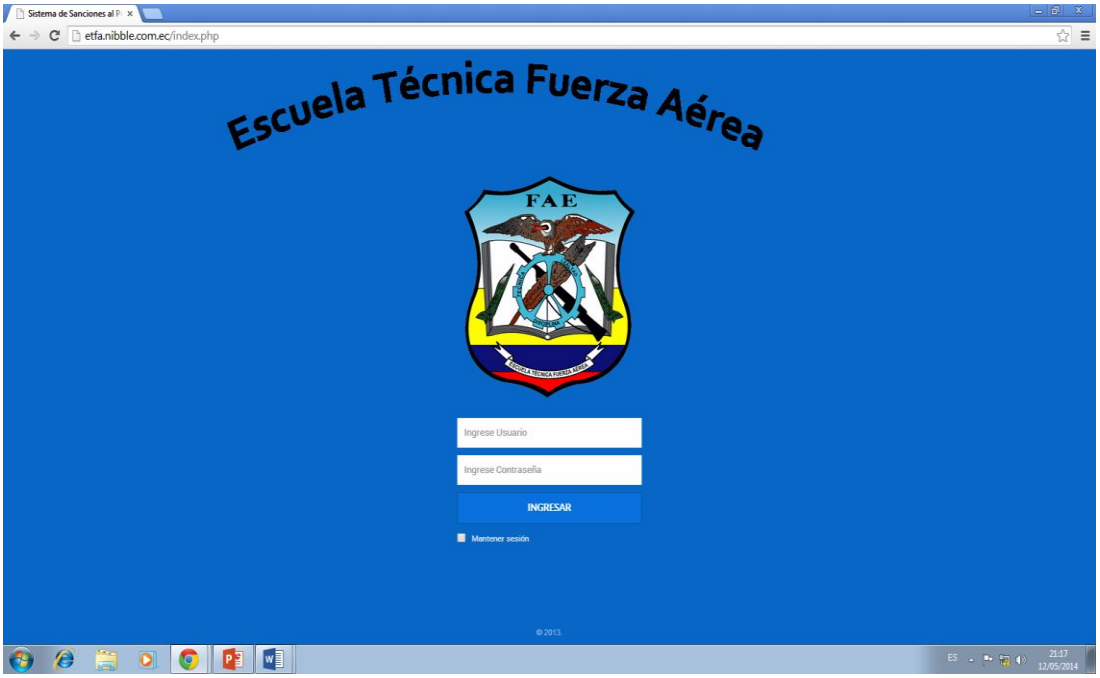


3.4.3 Glosario de Términos

Término	Significado
SICODIAT	Sistema Inteligente de Control Disciplinario de Aspirantes a Tropa de la ETFA
TIC	Tecnologías de Información y Comunicación
HU	Historia de Usuario
DFD	Diagrama de Flujo de Datos
Nómina	Listado en general de artículos
Atro	Aspirante a Tropa

3.6.5.1. Interfaz de Ingreso al Sistema

En el momento de ingresar al sistema software, se encontrará con la siguiente interfaz:

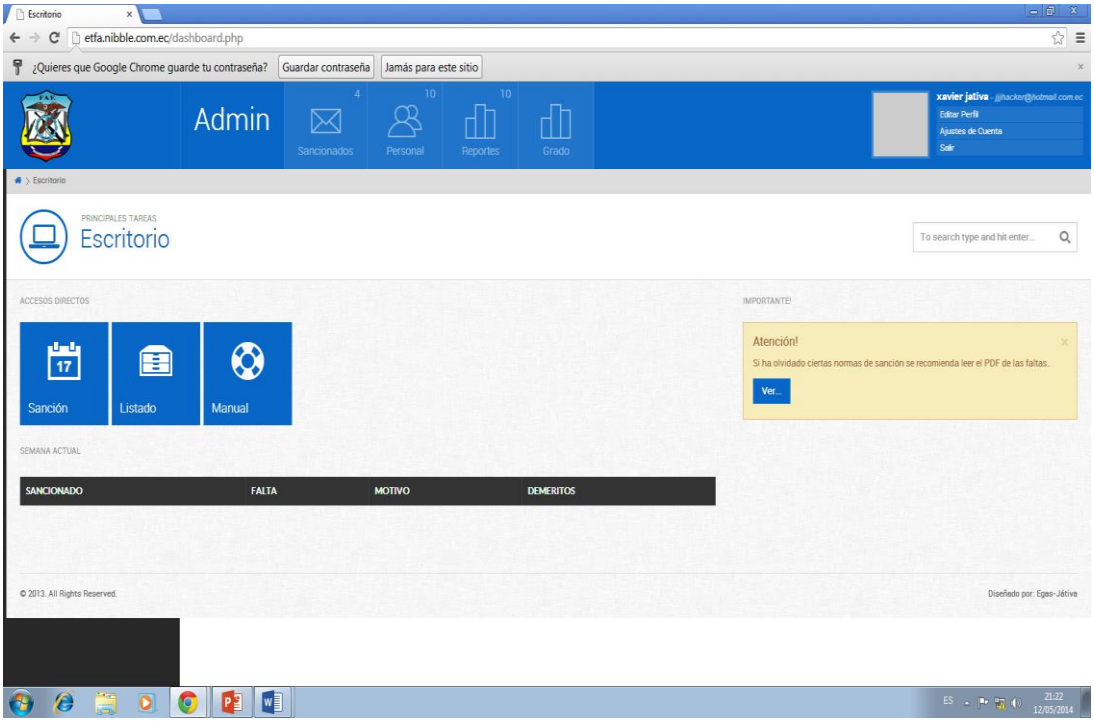


Se ha definido el siguiente diseño para la interfaz de ingreso para la figura anterior:

- Color fondo interfaz ingreso: #0866C6
- Texto encabezado: fuente Verdana, tamaño 34 puntos, color #000000
- Cajas de Texto: fuente texto Verdana, tamaño 14 puntos, color texto #999DC3, fondo cajas #FFFFFF, dimensión cajas 6 cm x 1cm.
- Botones de comando: fuente texto Verdana, tamaño 15 puntos, color texto #FFFFFF, color fondo #1E82E, dimensión botones 6 cm x 1cm
- Check Box: fuente texto Verdana, tamaño 10 puntos, color texto #FFFFFF.
- Logotipo: Imagen logotipo ETFA genérico para toda aplicación. Tamaño 10cm x 8cm.

3.6.5.2. Interfaz de Inicio

En el momento de ingresar al sistema software, se encontrará con la siguiente interfaz:



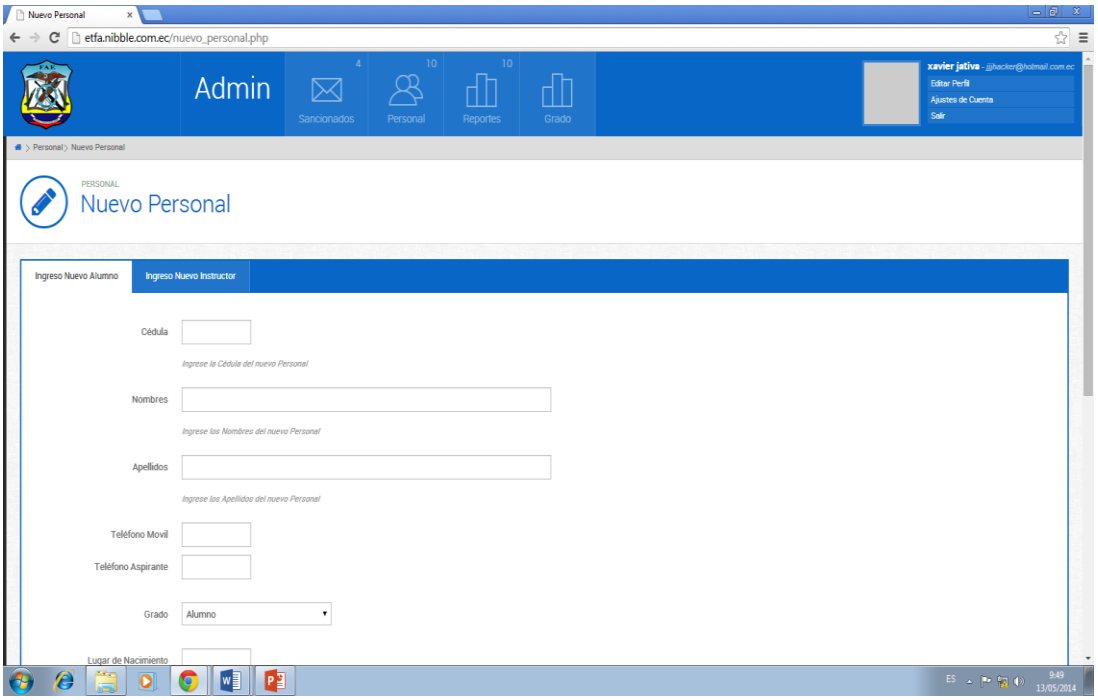
Se definió el siguiente diseño para la interfaz de inicio en la Figura anterior:

ESTRUCTURA DE DISEÑO	CARACTERÍSTICAS	SUB-ESTRUCTURA	ELEMENTOS Y CARACTERÍSTICAS
Encabezado	Color fondo: #0866C6 Texto: fuente Verdana, tamaño 32 puntos, color #000000	Área de navegación	• Menús • Imagen usuario • Información de usuario
Menú de posicionamiento	Color fondo: #000000 Texto: fuente Verdana, tamaño 14 puntos, color #000000	Caja de búsqueda	Color fondo interfaz ingreso: #0866C6 Cajas de Texto: fuente texto Verdana, tamaño 12 puntos, color texto #0866C7, fondo cajas #FFFFFF, dimensión cajas 4 cm x 0,6 cm. Botones de comando: fuente texto Verdana, tamaño 12 puntos, color texto #FFFFFF, color fondo #1E82E, dimensión botones 4 cm x0,6cm
		Imagen	Check Box: fuente texto Verdana, tamaño 10 puntos, color texto #FFFFFF. Logotipo: Imagen logotipo ETFA genérico para toda

			aplicación. Tamaño 10cm x 8cm.
Cuerpo	Color fondo: #000000 Texto: fuente Verdana, tamaño 14 a 20 puntos, color #00012E	Elementos auxiliares	Color fondo interfaz: #0866C6 Texto encabezado: fuente Verdana, tamaño 34 puntos, color #000000 Cajas de Texto: fuente texto Verdana, tamaño 14 puntos, color texto #999DC3, fondo cajas #FFFFFF, dimensión cajas 6 cm x 1cm. Botones de comando: fuente texto Verdana, tamaño 15 puntos, color texto #FFFFFF, color fondo #1E82E, dimensión botones 6 cm x 1cm Check Box: fuente texto Verdana, tamaño 10 puntos, color texto #FFFFFF.

3.6.5.3. Interfaz para Gestión de Procesos

En el momento de ingresar al sistema software, se encontrará con la siguiente interfaz:



Se definió el siguiente diseño para la interfaz de gestión de acuerdo a la Figura anterior:

ESTRUCTURA DE DISEÑO	CARACTERÍSTICAS	SUB-ESTRUCTURA	ELEMENTOS Y CARACTERÍSTICAS
Encabezado	Color fondo: #0866C6 Texto: fuente Verdana, tamaño 32 puntos, color #000000	Área de navegación	• Menús • Imagen usuario • Información de usuario
Cuerpo	Color fondo: #000000 Texto: fuente Verdana, tamaño 14 a 20 puntos, color #00012E	Elementos auxiliares	Color fondo interfaz: #0866C6 Texto encabezado: fuente Verdana, tamaño 34 puntos, color #000000 Cajas de Texto: fuente texto Verdana, tamaño 14 puntos, color texto #999DC3, fondo cajas #FFFFFF, dimensión cajas 6 cm x 1cm. Botones de comando: fuente texto Verdana, tamaño 15 puntos, color texto #FFFFFF, color fondo #1E82E, dimensión botones 6 cm x 1cm Check Box: fuente texto Verdana, tamaño 10 puntos, color texto #FFFFFF.

3.7. CODIFICACIÓN

3.7.1. Requerimientos Tecnológicos

Software requerido para la ejecución del sistema software:

- Servidor de aplicaciones web
  - Apache,
- Sistema gestor de base de datos relacional
  - MySQL
- Navegador web
  - Mozilla,
  - Internet Explorer,
  - Google Chrome.
- Tecnologías
  - PHP,
  - HTML
  - jQuery.

3.8. PRUEBAS

- **Pruebas unitarias**

Caso Prueba Unitaria					
Id Caso de Prueba: 01		**Nombre Módulo: Gestión de Aspirantes**			
Iteración: 1-2-3					
Descripción de la prueba: Alta, Modificación y baja de Aspirantes					
Nombre del Tester: Luis Egas					
Funcionalidad: Descripción	**Método Utilizado**	**Recibe**	**Resultado esperado del método**	**Resultado esperado de la prueba**	**Resultado real de la prueba**
Envío de información para el alta del Aspirante en la Base de datos	**POST**	**Función de alta de aspirante programada en PHP**	**Return 1**	Ok	Datos de aspirante ingresados correctamente
Envío de información para la modificación del Aspirante a la Base de datos	**POST**	**Función de modificación de aspirante programada en PHP**	**Return 1**	Ok	Datos de aspirante modificados correctamente
Envío de información para la baja del Aspirante a la Base de datos	**POST**	**Función de baja de aspirante programada en PHP**	**Return 1**	Ok	Aspirante inactivo.

Caso Prueba Unitaria					
Id Caso de Prueba: 02		**Nombre Módulo: Gestión de Instructores**			
Iteración: 1-2-3					
Descripción de la prueba: Alta, Modificación y Baja de Instructores					
Nombre del Tester: Juan Játiva					
Funcionalidad: Descripción	**Método Utilizado**	**Recibe**	**Resultado esperado del método**	**Resultado esperado de la prueba**	**Resultado real de la prueba**

Envío de información para el alta del Instructor en la Base de datos	**POST**	**Función de alta de instructor programada en PHP**	**Return 1**	Ok	Datos de instructor ingresados correctamente
Envío de información para la modificación del Instructor a la Base de datos	**POST**	**Función de modificación de instructor programada en PHP**	**Return 1**	Ok	Datos de instructor modificados correctamente
Envío de información para la baja del instructor a la Base de datos	**POST**	**Función de baja de instructor programada en PHP**	**Return 1**	Ok	Instructor inactivo.

Caso Prueba Unitaria					
Id Caso de Prueba: 03	**Nombre Módulo: Gestión de Faltas**				
Iteración: 1					
Descripción de la prueba: Alta, Modificación de Faltas					
Nombre del Tester: Juan Játiva					
Funcionalidad: Descripción	**Método Utilizado**	**Recibe**	**Resultado esperado del método**	**Resultado esperado de la prueba**	**Resultado real de la prueba**
Envío de información para el alta de una falta en la Base de datos	**POST**	**Función de alta de una falta programada en PHP**	**Return 1**	Ok	Datos de falta ingresada correctamente
Envío de información para la modificación de una falta en la Base de datos	**POST**	**Función de modificación de una falta programada en PHP**	**Return 1**	Ok	Datos de falta modificados correctamente

Caso Prueba Unitaria					
Id Caso de Prueba: 04		**Nombre Módulo: Gestión de Sanciones**			
Iteración: 1					
Descripción de la prueba: Alta, Modificación de Sanciones					
Nombre del Tester: Luis Egas					
Funcionalidad: Descripción	**Método Utilizado**	**Recibe**	**Resultado esperado del método**	**Resultado esperado de la prueba**	**Resultado real de la prueba**
Envío de información para el alta de una sanción en la Base de datos	**POST**	**Función de alta de una sanción programada en PHP**	**Return 1**	Ok	Datos de sanción ingresados correctamente
Envío de información para la modificación de una sanción en la Base de datos	**POST**	**Función de modificación de una sanción programada en PHP**	**Return 1**	Ok	Datos de sanción modificados correctamente

Caso Prueba Unitaria					
Id Caso de Prueba: 05		**Nombre Módulo: Gestión Perfiles de Usuario**			
Iteración: 1					
Descripción de la prueba: Perfiles de usuario para cada tipo de usuario					
Nombre del Tester: Juan Játiva					
Funcionalidad: Descripción	**Método Utilizado**	**Recibe**	**Resultado esperado del método**	**Resultado esperado de la prueba**	**Resultado real de la prueba**
Envío de información a la Base de datos para el logueo de usuario tipo Administrador	**SESION**	**Sesión programada en PHP para usuario tipo administrador**	**Return 1**	Ok	Logueo tipo de usuario administrador
Envío de información a la Base de datos para el logueo de usuario tipo Regulador	**SESION**	**Sesión programada en PHP para usuario tipo regulador**	**Return 1**	Ok	Logueo tipo de usuario regulador

Envío de información a la Base de datos para el logueo de usuario tipo Secretario	**SESION**	**Sesión programada en PHP para usuario tipo secretario**	**Return 1**	Ok	Logueo tipo de usuario secretario

Caso Prueba Unitaria					
Id Caso de Prueba: 06	**Nombre Módulo: Gestión de sanciones de Aspirantes.**				
Iteración: 1-2-3					
Descripción de la prueba: Registro de sanciones leves, graves, atentatorias y notificación de sancionados.					
Nombre del Tester: Luis Egas					
Funcionalidad: Descripción	**Método Utilizado**	**Recibe**	**Resultado esperado del método**	**Resultado esperado de la prueba**	**Resultado real de la prueba**
Envío de información del aspirante, falta leve, sanción, y sancionador a la Base de datos	**POST**	**Función de alta de una falta leve a un aspirante programada en PHP**	**Return 1**	Ok	Sanción de falta leve ingresada correctamente al aspirante
Envío de información del aspirante, falta grave, sanción, y sancionador a la Base de datos	**POST**	**Función de alta de una falta grave a un aspirante programada en PHP**	**Return 1**	Ok	Sanción de falta grave ingresada correctamente al aspirante
Envío de información del aspirante, falta atentatoria, sanción, y sancionador a la Base de datos	**POST**	**Función de alta de una falta atentatoria a un aspirante programada en PHP**	**Return 1**	Ok	Sanción de falta atentatoria ingresada correctamente al aspirante
Envío de información del aspirante, sobre la falta cometida por email	**MAIL PHP**	**Función de envió de mail para el aspirante sancionado**	**Return 1**	Ok	Email enviado correctamente.
Diseño de algoritmo para la sugerencia de falta grave	**Consulta Sql**	**Datos e interpretación para sugerencia**	**Return 1**	Ok	Sugerencias para faltas graves.

Caso Prueba Unitaria					
Id Caso de Prueba: 07	**Nombre Módulo: Gestión de méritos de aspirantes.**				
Iteración: 2					
Descripción de la prueba: Notificación de méritos mensuales					
Nombre del Tester: Luis Egas					
Funcionalidad: Descripción	**Método Utilizado**	**Recibe**	**Resultado esperado del método**	**Resultado esperado de la prueba**	**Resultado real de la prueba**
Envío de información al aspirante, sobre los méritos disponibles por email	**MAIL PHP**	**Función de envió de mail para el aspirante con los méritos disponibles**	**Return 1**	Ok	Email enviado correctamente.

Caso Prueba Unitaria					
Id Caso de Prueba: 08	**Nombre Módulo: Gestión de Reportes.**				
Iteración: 4					
Descripción de la prueba: Visualización de Reportes					
Nombre del Tester: Juan Játiva					
Funcionalidad: Descripción	**Método Utilizado**	**Recibe**	**Resultado esperado del método**	**Resultado esperado de la prueba**	**Resultado real de la prueba**
Envío de información de promoción	**POST**	**Datos en función programada para interpretación y generación de reporte de promoción**	**Return 1**	Ok	Reporte en PDF.
Envío de información de aspirante	**POST**	**Datos en función programada para interpretación y generación de**	**Return 1**	Ok	Reporte en PDF.

		reporte de aspirante			
Envío de información de todos los alumnos	**POST**	**Datos en función programada para interpretación y generación de reporte de todos los alumnos**	**Return 1**	Ok	Reporte en PDF.
Envío de información de instructor	**POST**	**Datos en función programada para interpretación y generación de reporte por instructor**	**Return 1**	Ok	Reporte en PDF.

- **Pruebas de aceptación**

Caso Prueba de Aceptación	
Código: 01	**Historia de Usuario: 1 – Alta de Aspirantes**
Nombre: Alta de Aspirantes	
Descripción: Ingreso de Aspirante.	
Condiciones de Ejecución: 1. Todos los campos del Aspirante deben estar llenos, son obligatorios. 2. Los campos de texto (nombre, apellido, ciudad) se permite solo letras. 3. Los campos numéricos (cedula, teléfono, edad, fecha de nacimiento) se permite solo números.	
Entrada / Pasos de ejecución: 1. Validación de campos. 2. Ingreso no duplicado de aspirante. 3. Ingreso de datos del aspirante.	
Resultado Esperado: Datos ingresados correctamente.	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 02	**Historia de Usuario: 2 – Modificación de Aspirantes**
Nombre: Modificación de Aspirantes	
Descripción: Modificación de Aspirante.	
Condiciones de Ejecución: 1. Los campos de texto (nombre, apellido, ciudad) se permite solo letras. 2. Los campos numéricos (cédula, teléfono, edad, fecha de nacimiento) se permite solo números. 3. La cédula no se permitirá modificar. 4. Todos los campos son obligatorios.	
Entrada / Pasos de ejecución: 1. Selección de aspirante. 2. Validación de campos. 3. Actualización de datos de aspirante. 4. No modificar cédula.	
Resultado Esperado: Datos actualizados correctamente.	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 03	**Historia de Usuario: 3 – Baja de Aspirantes**
Nombre: Baja de Aspirantes	
Descripción: Baja de Aspirante.	
Condiciones de Ejecución: 1. Campo aspirante obligatorio. 2. Campo de falta obligatorio. 3. Campo de sancionador obligatorio. 4. Campo de descripción de baja obligatorio.	
Entrada / Pasos de ejecución: 1. Selección de aspirante. 2. Selección de falta atentatoria. 3. Selección de sancionador. 4. Descripción de la baja del aspirante. 5. Actualización de aspirante.	
Resultado Esperado: Aspirante dado de baja correctamente.	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 04	**Historia de Usuario: 4 – Alta de Instructores**
Nombre: Alta de Instructores	
Descripción: Ingreso de Instructor.	
Condiciones de Ejecución: 1. Todos los campos del Instructor deben estar llenos, son obligatorios. 2. Los campos de texto (nombre, apellido, ciudad) se permite solo letras. 3. Los campos numéricos (cedula, teléfono, edad, fecha de nacimiento) se permite solo números.	
Entrada / Pasos de ejecución: 1. Validación de campos. 2. Ingreso no duplicado de instructor. 3. Ingreso de datos del instructor.	
Resultado Esperado: Datos ingresados correctamente.	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 05	**Historia de Usuario: 5 – Modificación de Instructores**
Nombre: Modificación de Instructores	
Descripción: Modificación de Instructores.	
Condiciones de Ejecución: 1. Los campos de texto (nombre, apellido, ciudad) se permite solo letras. 2. Los campos numéricos (cédula, teléfono, edad, fecha de nacimiento) se permite solo números. 3. La cédula no se permitirá modificar. 4. Todos los campos son obligatorios.	
Entrada / Pasos de ejecución: 1. Selección de Instructores. 2. Validación de campos. 3. Actualización de datos de Instructores. 4. No modificar cédula.	
Resultado Esperado: Datos actualizados correctamente.	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 06	**Historia de Usuario: 6 – Baja de Instructor**
Nombre: Baja de Instructor	
Descripción: Baja de Instructor.	
Condiciones de Ejecución: 1. Campo Instructor obligatorio. 2. Campo de descripción de baja obligatorio.	
Entrada / Pasos de ejecución: 1. Selección de Instructor. 4. Descripción de la baja del Instructor. 5. Actualización de Instructor.	
Resultado Esperado: Instructor dado de baja correctamente.	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 07	**Historia de Usuario: 7 – Alta de faltas**
Nombre: Alta de Faltas	
Descripción: Ingreso de Faltas.	
Condiciones de Ejecución: 1. Todos los campos deben estar llenos, son obligatorios. 2. Los campos de texto (literal) se permite solo letras. 3. Los campos numéricos (articulo) se permite solo números. 4. El campo tipo de falta (Leve, Grave, Atentatoria).	
Entrada / Pasos de ejecución: 1. Validación de campos. 2. Selección de tipo de Falta. 3. Ingreso de datos nueva falta.	
Resultado Esperado: Datos ingresados correctamente.	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 08	**Historia de Usuario: 8 – Modificación de Faltas**
Nombre: Modificación de Faltas	
Descripción: Modificación de Faltas.	
Condiciones de Ejecución: 1. Todos los campos deben estar llenos, son obligatorios. 2. Los campos de texto (literal) se permite solo letras. 3. Los campos numéricos (articulo) se permite solo números. 4. El campo tipo de falta (Leve, Grave, Atentatoria).	
Entrada / Pasos de ejecución: 1. Validación de campos. 2. Selección de tipo de Falta. 3. Actualización de datos nueva falta.	
Resultado Esperado: Datos actualizados correctamente.	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 09	**Historia de Usuario: 9 – Alta de Sanciones**
Nombre: Alta de Sanciones	
Descripción: Ingreso de Sanciones	
Condiciones de Ejecución: 1. Todos los campos deben estar llenos, son obligatorios. 2. Los campos de texto (descripción) se permite solo letras. 3. Los campos numéricos (deméritos) se permite solo números. 4. El campo tipo de falta (Leve, Grave, Atentatoria).	
Entrada / Pasos de ejecución: 1. Validación de campos. 2. Selección de tipo de Falta. 3. Ingreso de datos nueva Sanción.	
Resultado Esperado: Datos ingresados correctamente.	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 10	**Historia de Usuario: 10 – Modificación de Sanciones**
Nombre: Modificación de Sanciones	
Descripción: Modificación de Sanciones	
Condiciones de Ejecución: 1. Todos los campos deben estar llenos, son obligatorios. 2. Los campos de texto (descripción) se permite solo letras. 3. Los campos numéricos (deméritos) se permite solo números. 4. El campo tipo de falta (Leve, Grave, Atentatoria).	
Entrada / Pasos de ejecución: 1. Validación de campos. 2. Selección de tipo de Falta. 3. Actualización de datos nueva Sanción.	
Resultado Esperado: Datos actualizados correctamente.	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 11	**Historia de Usuario: 11 – Perfiles de Usuario**
Nombre: Perfiles de Usuario	
Descripción: Perfiles de usuario para cada uno.	
Condiciones de Ejecución: 1. Todos los campos deben estar llenos, son obligatorios en formulario de logueo (User y Pass).	
Entrada / Pasos de ejecución: 1. Ingresamos los datos de User y Pass 2. Interfaz según tipo de Usuario	
Resultado Esperado: Logueo correcto	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 12	**Historia de Usuario: 12 – Registro de Sanciones Leves y Graves**
Nombre: Registro de Sanciones Leves y Graves	
Descripción: Perfiles de usuario para cada uno.	
Condiciones de Ejecución: 1. Campo de Aspirante obligatorio 2. Campo de Falta obligatorio. 3. Campo de Sanción obligatorio 4. Campo de méritos obligatorio 5. Campo de Sancionador obligatorio	
Entrada / Pasos de ejecución: 1. Selección de Aspirante. 2. Selección de Falta. 3. Selección de Sanción. 4. Selección de méritos. 5. Selección de Sancionador.	
Resultado Esperado: Sanción ingresada correctamente	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 13	**Historia de Usuario: 13 – Registro de Sanciones Atentatoria**
Nombre: Registro de Sanciones Atentatoria	
Descripción: Perfiles de usuario para cada uno.	
Condiciones de Ejecución: 1. Campo de Aspirante obligatorio 2. Campo de Falta Atentatoria obligatorio. 3. Campo de Sanción obligatorio 4. Campo de méritos obligatorio 5. Campo de Sancionador obligatorio	
Entrada / Pasos de ejecución: 1. Selección de Aspirante. 2. Selección de Falta Atentatoria. 3. Selección de Sanción. 4. Selección de méritos. 5. Selección de Sancionador.	
Resultado Esperado: Email enviado correctamente	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 15	**Historia de Usuario: 15 – Notificación mensual de deméritos**
Nombre: Notificación mensual de deméritos	
Descripción: Notificación mensual de deméritos	
Condiciones de Ejecución: 1. Cambio de mes	
Entrada / Pasos de ejecución: 1. Envía mail con detalle de deméritos restantes, fecha.	
Resultado Esperado: Email enviado correctamente	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 16	**Historia de Usuario: 16 – Sugerencia de regulación de faltas graves**
Nombre: Sugerencia de regulación de faltas graves	
Descripción: Sugerencia de regulación de faltas graves	
Condiciones de Ejecución: 1. Consulta de sanciones anteriores	
Entrada / Pasos de ejecución: 1. Selección de falta a regular. 2. Selección de sugerencia de falta grave generada por sistema.	
Resultado Esperado: Falta regulada correctamente	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 17	**Historia de Usuario: 17 – Reporte por Promoción**
Nombre: Reporte por Promoción	
Descripción: Reporte por Promoción	
Condiciones de Ejecución: 1. Campo de Promoción	
Entrada / Pasos de ejecución: 1. Selección de campo promoción. 2. Visualización de datos de promoción.	
Resultado Esperado: PDF generado correctamente	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 18	**Historia de Usuario: 18 – Reporte por Aspirante**
Nombre: Reporte por Aspirante	
Descripción: Reporte por Aspirante	
Condiciones de Ejecución: 1. Campo de Aspirante	
Entrada / Pasos de ejecución: 1. Selección de aspirante 2. Visualización de datos de aspirante.	
Resultado Esperado: PDF generado correctamente	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 19	**Historia de Usuario: 19 – Reporte por todos los Aspirantes**
Nombre: Reporte por todos los Aspirantes	
Descripción: Reporte por todos los Aspirantes	
Condiciones de Ejecución: 1. Campo de todos los Aspirantes	
Entrada / Pasos de ejecución: 1. Selección de todos los aspirantes 2. Visualización de datos de todos los aspirantes.	
Resultado Esperado: PDF generado correctamente	
Evaluación de la Prueba: Ok	

Caso Prueba de Aceptación	
Código: 20	**Historia de Usuario: 20 – Reporte por Instructor**
Nombre: Reporte por Instructor	
Descripción: Reporte por Instructor	
Condiciones de Ejecución: 1. Campo de instructor.	
Entrada / Pasos de ejecución: 1. Selección de instructor 2. Visualización de datos de todos los aspirantes sancionados por instructor.	
Resultado Esperado: PDF generado correctamente	
Evaluación de la Prueba: Ok	

UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN SOFTWARE

CERTIFICACIÓN

Se certifica que el presente trabajo fue desarrollado por los señores: Luis Miguel Egas Robalino y Juan Xavier Játiva Álvarez, bajo nuestra supervisión.

_____

ING. LUCAS GARCÉS

DIRECTOR DEL PROYECTO

_____

ING. LUIS GUERRA

CODIRECTOR DEL PROYECTO

_____

ING. LUIS GUERRA

DIRECTOR DE CARRERA

_____

DR. RODRIGO VACA

SECRETARIO