UNIVERSIDAD LAICA ELOY ALFARO DE MANABI

FACULTAD DE CIENCIAS INFORMATICAS

TESIS DE GRADO
PREVIO A LA OBTENCIÓN DEL TÍTULO DE:
INGENIERO EN SISTEMAS

"ESTUDIO E IMPLEMENTACIÓN DE BOTS PARA LA AUTOMATIZACIÓN DE LA ADMINISTRACIÓN DE CÓDIGO FUENTE MEDIANTE EL MODELO DE INTEGRACIÓN CONTINUA PARA EL ÁREA DE DESARROLLO DE LA FACULTAD DE CIENCIAS INFORMÁTICAS DE LA UNIVERSIDAD LAICA ELOY ALFARO".

PRESENTADO POR:

BARBERAN LEÓN JULIO VICENTE

DIRECTOR:

Ing. Freddy Xavier Alarcón Villamar

2015

MANTA – MANABÍ – ECUADOR

CERTIFICACIÓN.

En mi calidad de Director del Proyecto de Grado de la Facultad de Ciencias Informáticas de la ULEAM, certifico:

Haber dirigido y revisado el proyecto de grado sobre el tema: "ESTUDIO E IMPLEMENTACIÓN DE BOTS PARA LA AUTOMATIZACIÓN DE LA ADMINISTRACIÓN DE CÓDIGO FUENTE MEDIANTE EL MODELO DE INTEGRACIÓN CONTINUA PARA EL ÁREA DE DESARROLLO DE LA FACULTAD DE CIENCIAS INFORMÁTICAS DE LA UNIVERSIDAD LAICA ELOY ALFARO" del estudiante egresado Barberan León Julio Vicente

Considero que el mencionado trabajo de investigación reúne los requisitos y méritos suficientes para ser sometido a la evaluación del jurado examinador que las autoridades designen.

_____

Ing. Freddy Xavier Alarcón Villamar
Director del Proyecto de Grado.

“ESTUDIO E IMPLEMENTACIÓN DE BOTS PARA LA AUTOMATIZACIÓN DE LA ADMINISTRACIÓN DE CÓDIGO FUENTE MEDIANTE EL MODELO DE INTEGRACIÓN CONTINUA PARA EL ÁREA DE DESARROLLO DE LA FACULTAD DE CIENCIAS INFORMÁTICAS DE LA UNIVERSIDAD LAICA ELOY ALFARO”.

TRIBUNAL DE SUSTENTACIÓN

Nombres	**Firmas**
Ing. _____	_____
Ing. _____	_____
Ing. _____	_____

Calificación Trabajo de Graduación

Calificación Trabajo Escrito: _____

Calificación Sustentación de Proyecto de Grado: _____

Nota Final de Trabajo de Graduación: _____

Lo certifico,

Lic. María Esperanza Molina
SECRETARIA DE LA FACULTAD
DE CIENCIAS INFORMÁTICAS

DECLARACIÓN EXPRESA DE AUTORÍA DE PROYECTO DE GRADO.

Yo, BARBERAN LEÓN JULIO VICENTE con Cédula Nacional de Identidad N° 131473849-3, reconozco como único titular del contenido de este Proyecto de Grado, cuyo tema es “ESTUDIO E IMPLEMENTACIÓN DE BOTS PARA LA AUTOMATIZACIÓN DE LA ADMINISTRACIÓN DE CÓDIGO FUENTE MEDIANTE EL MODELO DE INTEGRACIÓN CONTINUA PARA EL ÁREA DE DESARROLLO DE LA FACULTAD DE CIENCIAS INFORMÁTICAS DE LA UNIVERSIDAD LAICA ELOY ALFARO”, y derechos patrimoniales a la Universidad Laica “Eloy Alfaro de Manabí”, en virtud de lo dispuesto en el Art. 15 de la Ley de Propiedad Intelectual.

Asimismo, autorizamos a la ULEAM para que realice la digitalización y publicación de esta tesis de posgrado en el repositorio digital de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Finalmente, la responsabilidad del contenido de esta Tesis de Grado corresponde exclusivamente al autor.

Lo certifica,

Barberan León Julio Vicente

131273849-3

DEDICATORIA.

En primer lugar, a ti Dios padre porque gracias a ti hice este sueño realidad por todo el amor con el que me rodeas y porque me tienes en tus manos.

Madre, no me equivoco si digo que eres la mejor mamá del mundo gracias por todo tu esfuerzo, tu apoyo y por la confianza que depositaste en mí. Gracias porque siempre has estado a mi lado Te quiero mucho.

Abuelitas son como mi segunda madre este logro quiero compartirlo con ustedes gracias por creer en mi ocupan un lugar muy especial en mi corazón.

A mi Hermano Junior, te dedico este trabajo como muestra de dedicación y convicción se puedan logran tus metas sigue adelante.

A mi novia Betsabet Bastidas me has apoyado en todo el transcurso de este trabajo animando me cuando no podía avanzar y siendo inquebrantable en las buenas y en las malas.

A mi gran amigo Conforme Ariel en los años de carrera fuiste de grana apoyo en momentos difíciles de estudio y salimos abantes en muchas gracias por estar ahí.

A Mendoza Julio fuiste de mucha ayuda y me enseñaste a apreciar cada momento que se vive a batallar por lo que uno quiere y nunca rendirse has sido de gran inspiración en esta tesis Gracias amigo.

A todo el curso del 5A mil gracias por todos los momentos que he pasado junto al curso, aunque sea solo por dar lata y molestar solo puedo decir gracias por formar parte de mi vida estudiantil.

A todos mis profesores no solo de la carrera sino de toda la vida mil gracias porque algunas maneras forman parte de lo que ahora soy. Especialmente al Ing. Freddy Alarcón Villamar que ha estado en el transcurso de este trabajo.

Julio Barberan L.

AGRADECIMIENTO

"Lo que importa verdaderamente en la vida no son los objetivos que nos marcamos, sino los caminos que seguimos para lograrlos"

Peter Bamn

Son a tantas las personas a quien le debo el agradecimiento respectivo, durante la etapa de desarrollo del presente trabajo de titulación por medio del cual se llevó con éxito su realización. A mi familia por brindarme todo el apoyo incondicional durante mi etapa de estudio y estar presente en todos los momentos tanto buenos como malos de mi vida estudiantil a mis profesores y compañeros de estudios fue una base importante en el transcurso de los años de estudio.

Quiero hacer presente un profundo agradecimiento al Ing. Walter Colon García y al Ing. José Bazurto Roldan mentores en vida estudiantil y modelos a seguir como personas y al Ing. Freddy Alarcón Villamar Mi director de tesis, profesor y amigo, por su importante aporte y participación activa en el desarrollo de este trabajo de grado.

Destacando, por encima de todo, su disponibilidad y paciencia tanto a nivel científico como personal.

RESUMEN EJECUTIVO.

Actualmente la catedra que se imparte en las Áreas de Desarrollo no se están orientando a las metodologías y modelos informáticos que hacen énfasis en la automatización del proceso de construcción, análisis y pruebas de las aplicaciones, en donde el enfoque está orientado a la administración de las aplicaciones en cualquier fase del ciclo de vida del software que se implemente además servirá como apoyo a la sustentación de la cátedras del Área de Desarrollo en donde el alumnado está capacitado para las etapas de desarrollo de software pero no aplica modelos informáticos que puedan ser de apoyo a la administración y más que todo a la fiabilidad e integridad de las aplicaciones cuando se presenten fallos y errores.

El presente estudio se basara en la investigación y la implementación de un sistema operativo para servidores que permita unir un entorno de desarrollo integrado para la utilización de un modelo informático como es la Integración Continua y el desarrollo de procesos para realizar integraciones en la versión actual de una aplicación en un repositorio de código fuente mediante la creación y ejecución de BOTS, todo esto para la automatización en la administración del código fuente bajo una plataforma OSX que permitirá incursionar en nuevos campos o tendencias con el fin de aumentar la calidad del presente trabajo.

ABSTRACT

Currently the chair that is taught in the Areas of Development is not being directed to the methodologies and computer models that emphasize the construction process automation, analysis and testing of applications, where the focus is oriented administration applications at any stage of the software life cycle to be implemented will also serve to support the sustainability of the chairs of the Development Area where students are trained for the stages of software development but not applied computer models that can be supportive administration and above all the reliability and integrity of applications when failures and errors occur.

This study was based on research and implementation of a server operating system that allows attaching an integrated development environment for the use of a computer model as Continuous Integration and development of processes for integration in the current version an application on a source code repository by creating and implementing BOTS, all this automation in the management of source code under OSX platform to enter new fields or trends in order to increase the quality of this work.

CONTENIDO

CONTENIDO DE ILUSTRACIONES

CONTENIDO DE TABLAS

CAPITULO I

CONTEXTUALIZACIÓN DE LA INVESTIGACIÓN

1.1. INTRODUCCIÓN

La incursión de nuevas tecnologías que permitan a metodologías de software o modelos informáticos optimizar tiempos de respuesta a pruebas unitarios, aceptación o funcionales a los programadores en etapas finales de metodologías agiles y de esta manera crear un mejor entorno de trabajo sin entrar en dudas por fallas en el momento de pruebas y ser mucho más confiable en la producción de software bajo programas centralizados en un servidor que permita entrar en repositorios de código fuente para de esa manera jalar el código.

Mediante un esquema de trabajo que es diseñado por los programadores revisen la codificación y envíen una cantidad parametrizada de reportes vía web a las personas que realizan el desarrollo de la aplicación bajo técnicas de integración continúa implementadas mediante buenas prácticas de desarrollo de software el programador ganara tiempo en etapas de desarrollo de software.

Las aplicaciones de esta tecnología no solo permiten la optimización de código, también el rendimiento de la aplicación ya que incluye simulación de tiempos de ejecución en dispositivos móviles ejecutándose bajo la plataforma IOS de esa manera ahorramos tiempo en la configuración de los dispositivos para la ejecución de cualquier aplicación.

El presente trabajo de titulación está organizado de la siguiente manera:

- Capítulo 1 enmarca todos los elementos que sirvieron para la contextualización del problema y la definición de objetivos a desarrollar.
- Capítulo 2 corresponde a la fundamentación teórica que se investigó para formar una base sólida para el desarrollo de los posteriores capítulos.
- Capítulo 3 específica y marca los distintos tipos de métodos y metodologías científicas que se usaran.
- Capítulo 4 establece la propuesta que se va a llevar a cabo para el cumplimiento de los objetivos que se establecieron para el desarrollo de este trabajo de titulación.
- Capítulo 5 evidencia los resultados arrojados, que se llevó a cabo en el trascurso que se desarrolló este trabajo de titulación.

1.2. PRESENTACIÓN DEL TEMA

“ESTUDIO E IMPLEMENTACIÓN DE BOTS PARA LA AUTOMATIZACIÓN DE LA ADMINISTRACIÓN DE CÓDIGO FUENTE MEDIANTE EL MODELO DE INTEGRACIÓN CONTINUA PARA EL ÁREA DE DESARROLLO DE LA FACULTAD DE CIENCIAS INFORMÁTICAS DE LA UNIVERSIDAD LAICA ELOY ALFARO”.

1.3. SITUACIÓN PROBLEMÁTICA

1.3.1. Ubicación Y Contextualización.

El presente proyecto de Grado se lo llevará a cabo en el Universidad Laica “Eloy Alfaro” de Manabí bajo el área de desarrollo de software de la Facultad de Ciencias Informáticas

INSTITUCIÓN PÚBLICA

Universidad Laica “Eloy Alfaro” de Manabí

Facultad de Ciencias Informáticas

Ilustración 1: Ubicación y Contextualización

Elaborado por: Barberan León Julio Vicente

1.3.2. Planteamiento Del Problema.

En el presente trabajo de titulación se pretende abordar sobre la escasa incursión de modelos informáticos para las etapas de desarrollo como apoyo para las metodologías de software que se planten para aplicar al desarrollo de aplicaciones y como sustento para las cátedras relacionadas al Área de Desarrollo, el cual puede ser un punto de influencia en la integridad y fiabilidad del software ya que en etapas de producción donde se requiere el mayor grado de efectividad al realizar cada una de las actividades de dicha etapa para esto se plantea las siguientes preguntas relacionadas al grado de beneficio al utilizar este modelo:

¿Cuáles son las metodologías de software aplicables para la implantación de modelos informáticos para la automatización de la gestión y control de código fuente de una aplicación?, la cuestión planteada trata de ver en qué grado de eficiencia y rendimiento pueda afectar a una aplicación en cualquier etapa del ciclo de vida del software. ¿La capacitación acerca de la integración continua puede mejorar la calidad de software, así como mejores

profesionales en el campo de desarrollo?, esta pregunta relaciona cuanto puede afectar en la vida profesional este tipo de modelos en el desarrollo de aplicaciones como la calidad de estas.

¿Las plataformas utilizadas para la enseñanza en el Área de Desarrollo no permiten trabajos bajo estas nuevas formas de modelación informática?, este factor tiene que ver con la poca incursión bajo estas plataformas, para que el estudiantado sea mucho más versátil a la hora de manejar varios entornos de programación por lo cual puede mejorar el ámbito profesional del estudiantado.

1.4. DIAGRAMA CAUSA-EFECTO DEL PROBLEMA.

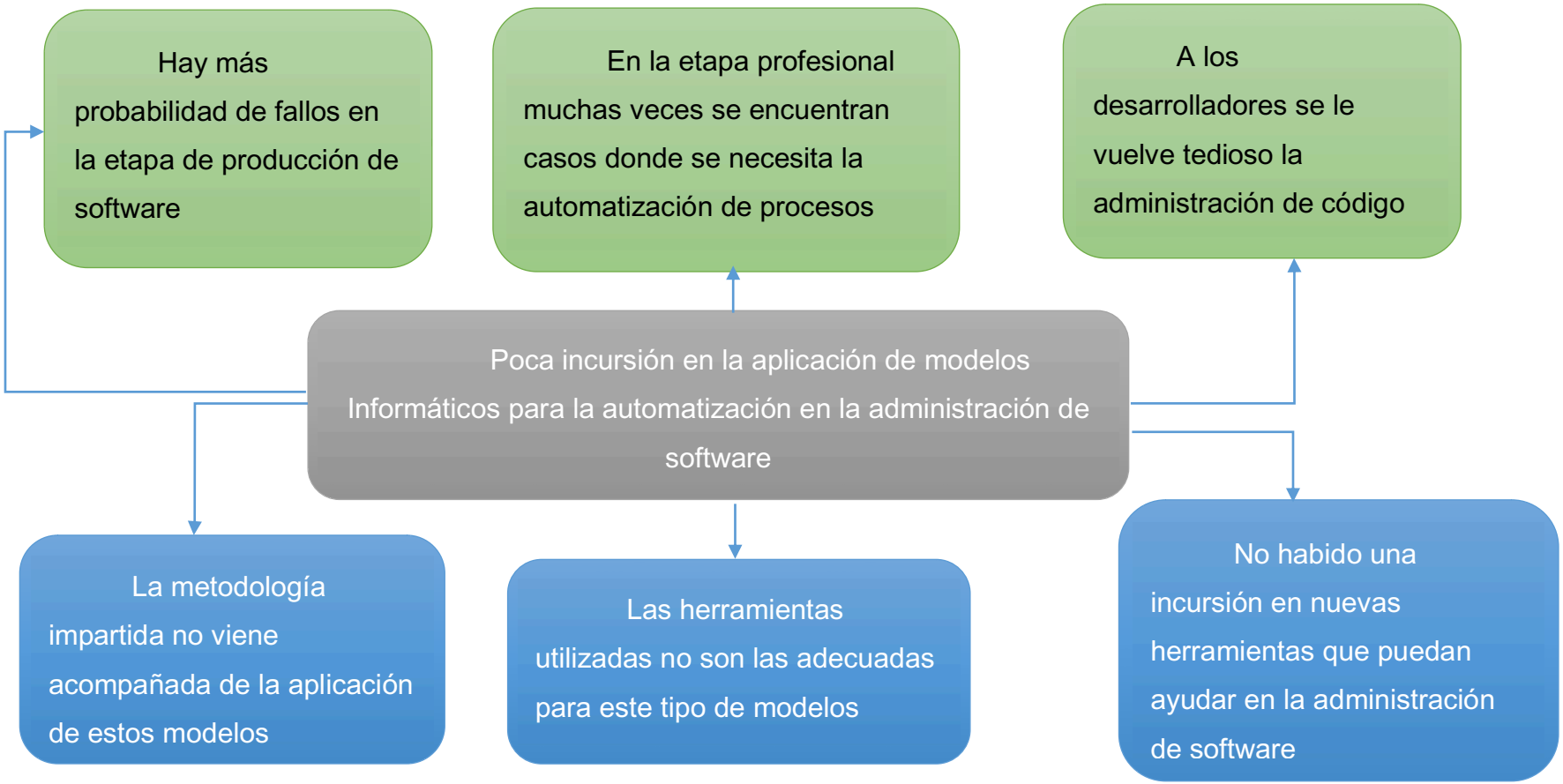

Ilustración 2: Diagrama causa-efecto

Fuente: Barberan León Julio Vicente

Elaborado por: Barberan León Julio Vicente

1.5. OBJETIVOS.

1.5.1. Objetivo General.

Implementar BOTS para la optimización y automatización en el proceso de construcción, análisis y pruebas de aplicaciones bajo el modelo de Integración continúa mediante la plataforma OSX.

1.5.2. Objetivos Específicos.

- Investigar las funciones que brinda Xcode Server para el soporte del modelo de Integración Continua
- Realizar un análisis para establecer los servicios que ofrecerá la ejecución de BOTS bajo el modelo de integración continua
- Implementar la ejecución de BOTS en un sistema operativo de servidor bajo el modelo de integración continua
- Evaluar la implementación de los BOTS en la administración del código fuente de una aplicación

1.6. JUSTIFICACIÓN.

La incursión de nuevas herramientas para la automatización de la gestión del control de código fuente es uno de los tópicos que no se topan en la Área de Desarrollo, con nuevas tendencias y mejoras por parte de modelos informáticos los cuales nos brindan soporte en la etapa de producción de una aplicación lo cual suele ser un momento en que raramente la efectividad del software es completa lo cual indica que no se imparten estos modelos los cuales puedan aumentar la efectividad e integridad del software. El presente trabajo de titulación se justifica por las siguientes razones:

- Contribuir al mejoramiento académico del estudiantado en la facultad
- Incursión de nuevas herramientas orientadas al Área de Desarrollo
- Brindar apoyo didáctico tecnológico a las cátedras que se imparten
- Reducir la tasa de estudiantes con déficit académico en área de programación
- Mejorar en nivel de productividad de los desarrolladores en etapas de producción de software.

1.7. IMPACTOS ESPERADOS.

1.7.1. Impacto Tecnológico.

Mediante tecnologías que permitan unir conceptos que se distinguían en la optimización y rendimiento de código fuente alojados en repositorios remotos los cuales se necesitan de intervención de desarrolladores para que un trabajo colaborativo siguiera un flujo continuo y sin obstáculos consumía un tiempo considerable en etapas de producción.

La aplicación llega a un punto donde se necesita las pruebas funcionales sean unitarios o de aceptación con el usuario y mediante esta tecnología se optimizan tiempos y rendimiento de la aplicación haciéndola mucho más versátil en etapas de producción de software.

1.7.2. Impacto Social.

Los desarrolladores a nivel general no ejercen buenas prácticas de programación bajo modelo informáticos que permitan optimizar tiempo o mejorar el rendimiento de una aplicación teniendo en cuenta la utilización de metodologías de software que rigen estos comportamientos en cuanto se refiere al desarrollo de software utilizando técnicas que permitan a un grupo de desarrolladores quitarle cargas que por sí misma son evaluativas, a lo que respecta etapas de producción mas no desarrollo aumentaría la colaboración en grupo bajo etapas críticas en el desarrollo de una aplicación .

CAPITULO II

MARCO TEORICO REFERENCIAL

2.1. INTRODUCCIÓN

En el presente capítulo se detalla la base teórica, legal y los principios fundamentales para el "Estudio e implementación de BOTS para la automatización de la administración de código fuente mediante el modelo de Integración Continua para el Área de Desarrollo de la Facultad de Ciencias Informáticas de la Universidad Laica Eloy Alfaro".

2.2. ANTECEDENTES

En esta sección se destaca la colaboración e investigación del Ing. MARTIN FOWLER para el desarrollo involucrándolo con metodologías ágiles de programación como se describe a continuación:

Una parte importante de cualquier proceso de desarrollo de software es que cada vez sea más fiable la aplicación en etapas de producción donde el código es puesto a prueba y refleja la aplicación que esto pueda brindar. A pesar de su importancia, a menudo nos sorprendemos cuando esto no se hace. Esto permite que cada desarrollador para integrar todos los días lo que reduce los problemas de integración. El término "integración continua" se originó con la Programación Extrema como una de sus originales prácticas. Aunque Integración Continua es una práctica que no requiere de herramientas particular, hemos encontrado que es útil el uso de un servidor de integración continua.

La forma más fácil para mí de explicar lo que es Integración Continua como es y cómo funciona es mostrar un rápido ejemplo con el desarrollo de una pequeña función. Supongamos que tengo que hacer algo para un módulo de software, no importa realmente que tarea es, por el momento voy a suponer que es pequeño y se puede hacer en un par de horas.

Comienzo tomando una copia de la fuente integrada en mi desarrollo local, hago esto mediante el uso de un sistema de gestión de código fuente

por el control de una copia de trabajo desde la línea principal. Un sistema de control de código fuente se queda con todo el código fuente de un proyecto en un repositorio. El estado actual del sistema que normalmente se conoce como "BRANCH". Cada vez que un desarrollador puede hacer una copia controlada de la línea principal en su propia máquina, esto se llama 'CHECK OUT'. Ahora me tomo mi copia de trabajo y hacer lo que tengo que hacer para completar mi tarea. De Esta consistirá en tanto alterar el código de producción, y también agregar o cambiar la fuente dependiendo de las necesidades del desarrollador.

2.3. DEFINICIONES CONCEPTUALES

2.3.1 **Software**

Es la agrupación lógica o conocido como base sistemática lógica de uno o varios componentes de una solución informática, que consta de un conjunto de módulos o componentes que permiten la resolución de actividades y tareas para fines que se implementan como objetivo de la aplicación. Cada módulo tiene un objetivo que se define de cierta manera que la agrupación de cada uno de esos módulos vuelve el software siendo compilados en una base que permita leer en un lenguaje máquina que sirva para ser ejecutado y manipulado por un ente que interactúe de manera directa o indirecta esto se puede definir en varias rutinas que son visibles o invisibles para el usuario. El software se construye en sentido genérico diferenciando que existen varias maneras de diseñar tanto la parte abstracta o física de un ordenador (Hardware o Software) la factibilidad y por tanto calidad se alcanza optimizando recursos mediante utilizando diseños, practicas. Los sistemas informáticos se componen de una agrupación de programas que codificados mediante un lenguaje que permita servir a otros módulos. Varios módulos de las aplicaciones de sistemas compilan y

construyen las estructuras de información compleja pero determinada (Jane Price Laudon, 2004).

El software se ha convertido en el elemento clave de la evolución de los sistemas y productos informáticos. En los pasados 50 años, ha pasado de ser una resolución de problemas especializada y una herramienta de análisis de información, a ser una industria por sí misma. Pero la temprana cultura e historia de la programación ha creado un conjunto de problemas que persisten todavía hoy. El software se ha convertido en un factor que limita la evolución de los sistemas informáticos. El software se compone de programas, datos y documentos que se relacionan con otros indistintos o bien relacionados a este mediante la vinculación de datos. Desde el principio del proyecto se puede aplicar uno de los mecanismos más efectivos para garantizar la calidad del software: la revisión técnica de software que incluye muchos elementos. La documentación proporciona el fundamento para un buen desarrollo y, lo que es más importante, proporciona guías para la tarea de mantenimiento del software (Candela, García, Quesada, & Santana, 2007).

Los sistemas que incorporan software dentro sus componentes sean módulos rutinas son resistentes al ambiente, pero en un gran porcentaje se descubren fallas o defectos en las primeras fases de desarrollo del software siendo capas hasta esa etapa ser libre de corrección en los mejores casos sin tener que agregar más errores al flujo de la aplicación y la interacción entre las partes del software. Los componentes del software son diseñados e implementados de manera que se realice un reutilización y diferentes subrutinas procesos módulos o programas distintos permitiendo maneras de programar como la encapsulación para cualquier tipo manera de comunicación con los distintos componentes del software mediante este siguiente ejemplo: Las interfaces actuales con el usuario se construyen con componentes reutilizables que permiten la creación de ventanas gráficas,

menús desplegables y una amplia variedad de mecanismos de interacción. Las estructuras de datos y los detalles de procesamiento requeridos para construir la interfaz están contenidos en una librería de componentes reutilizables para la construcción de la interfaz (Pressman, 2010).

2.3.2 **Tipos de software**

Una aplicación de cualquier índole es una secuencia de instrucciones detalladas que operan y manipulan el flujo y la interacción entre la parte lógica y física de un sistema de cómputo. Escoger el software indicado para una organización o centro de cómputo denota una toma de decisión importante para el rendimiento y optimización de los procesos que se requieren para las transacciones. El software del sistema y el software de aplicaciones. Cada uno sin importar el lenguaje de escritura o el método de compilación o plataforma realiza una función diferente (Pressman, 2010).

El software del sistema es una agrupación de sub rutinas o programas que se encargara de la manipulación control y administración de todos los recursos empleados por programas secundarios o hardware que necesita de la interacción con algún módulo de un programa o rutina dependiendo de la necesidad, como por ejemplo el cálculo que necesito llevarse dentro del procesador a cargar una aplicación en memoria, los canales de enlaces de comunicación para interacción con el hardware de salida. Las clases de software se relaciones y pueden coger un grupo de listas enlazadas los cuales deben interactuar con otros tipo de listas dependiendo de lo que se realice o la tarea que se procese, generalmente el software del sistema controlan y administran este tipo de lista que son aplicaciones y sabiendo que pueden tener sub rutinas del sistema interactuando como enlace de comunicación con una de estas listas o un recurso del computador o hardware, para operar este tipo de software cada una rutina debe pasar por

el software de sistemas y teniendo en base a o que pide el un usuario o el sistema realiza una acción para esa rutina en la cola de procesos y de esto se deduce que siempre los usuarios manejan software de aplicaciones y con esto el sistema y los procesos deben estar creados y diseñados justamente para la plataforma que va albergar estos procesos (Muñoz, Piattini, & Rubia, 2010).

2.3.2.1 **Software del sistema y sistemas operativos de pc**

El sistema que manipula la interacción ente los distintos componentes lógicos y físicos dentro del computador dirige componentes del sistema de cómputo y es mediador entre software de aplicaciones y el hardware de cómputo. El software de sistema que controla y administra todas las actividades del computador se denomina Sistema Operativo. Otro tipo de software se compone de compiladores y aplicaciones y rutinas de traducción a un lenguaje comprensible para la maquina los cuales por convención se trasforman en lenguajes de programación, en lenguajes de computador que recepten de alguna forma las entradas de usuario y arrojen resultados de acuerdo a lo que la maquina recepte y procese además de esto se encuentra otro tipo de aplicaciones llamada programas de utilería que realizan la ejecución de tareas simples de procesamiento (Candela, García, Quesada, & Santana, 2007).

El sistema operativo es el administrador, contextualizando se lo denominaría como el jefe de sistemas de cómputo como función más relevante ubica y envía responsabilidades a cada recurso del sistema, programa la utilización de recursos mediante políticas dependiendo de la necesidad que se efectúe en el momento más precisamente por los procesos ya que si se otorga una cantidad mínima de recursos por el manejo de sistema operativo esto influye a que cause estragos en las listas de procesos ya que el usuario notaria que

no llevan a cabo las salida respetivas que desea por lo tanto además de programar brinda vigilancia en momento de procesamiento ya que variedad de procesamiento cambia con el tiempo y se otorga recursos dinámicamente tanto para procesos como para hardware de entrada y salida, enlaces de comunicación y como ultima dependencia coordina la programación de recursos en distintas áreas de la computadora. El seguimiento de cada procesos cada recurso cada enlace lo hace mediante un seguimiento autónomo y también lo puede realizar de quien lo está utilizando porque independientemente sea un usuario o programa este lo pueden utilizar indistintamente, de que programas se están utilizando y ejecutando y sea de cualquier acceso no permitido al sistema mediante rutinas que detectan entradas de este tipo (Flynn, 2011).

2.3.2.2 **Software propietario**

La definición de software como una aplicación informática que permite varias formas de utilización ayudando en la ejecución de distintas formas de procesamiento de información. Propietario es aquel que posee legítimamente un derecho de propiedad. Mediante la unión de estas definiciones se recibe que el software propietario es aquel que su poseedor tiene derechos legítimos para acciones que afectan directamente en el uso o utilización que se da al software por parte directa e indirecta, la alteración de su estructura como parte de futras versiones que se puedan distribuir o redistribuir con su contenido sea alterado e original.

Las disposiciones que generalmente goza el software propietario son de disponer por parte del cliente un soporte que garantiza la funcionalidad plena de la adquisición del software a diferencia de lo que se podría involucrar con el software libre ya que este brinda a disposición de cualquier persona una redistribución gratuita del software para su alteración o modificación sea estructural o funcional (Valenzuela, 2013).

2.3.2.2.1 Características de software propietario

El software y consecuentemente la ingeniería del software, permite examinar las características del software que lo diferencian de otras cosas que los hombres pueden construir. Cuando se construye hardware, el proceso humano (análisis y diseño, construcción, prueba) se traduce finalmente en una forma física. Si construimos una nueva computadora, nuestro boceto inicial, diagramas formales de diseño y prototipo de prueba, evolucionan hacia un producto físico (chips, tarjetas de circuitos impresos, fuentes de potencia, etc.). El software es un elemento del sistema que es lógico, en lugar de físico. Por tanto, el software tiene unas características considerablemente distintas a las del hardware (Sommerville, 2011).

EL software independientemente del tipo o de su privacidad para la fundación del software libre (FSF) las definiciones de aplican sin distinción de software que no sea libre o mitad libre y privativo debido a que la utilización, alteración, distribución o redistribución esta ilícita legalmente y se necesita de autorización previa del autor que tiene los derechos legítimos.

- El software pertenece exclusivamente a una o varias entidades legalmente inscritas bajo la respectiva propiedad intelectual, sin acción alguna previa con una debida autorización no se podrá realizar ninguna alteración al código fuente del software.
- El software queda cerrado en el sentido alguno de distribuciones o redistribuciones sea la versión en que se encuentre sin previa autorización del propietario.
- Los derechos y propiedad y decisión de utilización quedan exclusivamente del propietario.

- Mayormente el software privativo pose una mayor calidad y acabado en las prestaciones que brinda el software.

- Las prestaciones de software no brindan la capacidad de tener enlaces de accesos para que los usuarios finales puedan añadir características a las copias que tienes en máquinas locales haciendo imposible la alteración de su contenido (Flynn, 2011).

2.3.3 **Procesos de software**

Un Proceso de software es un conjunto de actividades relacionadas que conduce a la producción de un software específico. Estas tareas pueden incluir el desarrollo de software desde cero en un lenguaje de programación estándar, Sin embargo, las aplicaciones no se desarrollan necesariamente de esta manera. El software se encuentra a menudo desarrollado mediante la ampliación y modificación de los sistemas existentes o mediante la configuración y la integración de los componentes del sistema. Hay muchos procesos de software diferentes, pero todos deben incluir cuatro actividades que son fundamentales para la ingeniería de software:

- Especificación del software. - La funcionalidad del software y las limitaciones en su operación debe ser definido en el extracto contractual de adquisición, aparte las limitaciones respecto a ciertas funcionalidades extrañas tanto como hardware y software.

- Diseño e implementación del software. - El software para cumplir con la especificación debe ser producido y elaborado mediante las necesidades que explícitamente o implícitamente el usuario especifica en la entrevista que se desarrollan por parte del desarrollador.

- Validación del software El software debe ser validado para asegurarse de que hace lo que el cliente quiere y responda a los posibles

fallos y contingencias que se puedan presentar en las fases del ciclo de vida del software (Sommerville, 2011).

2.3.3.1 **Modelos de procesos de software.**

Para resolver los problemas reales de una industria, un ingeniero del software o un equipo de ingenieros debe incorporar una estrategia de desarrollo que acompañe al proceso, métodos y capas de herramientas y fases genérica. Los modelos de proceso o paradigmas de ingeniería del software se seleccionan en base a un modelo de proceso para la ingeniería del software según la naturaleza del proyecto y de la aplicación, los métodos y las herramientas a utilizarse, y los controles y entregas que se requieren. Todo el desarrollo del software se puede caracterizar como bucle de resolución de problemas en el que se encuentran cuatro etapas distintas: status quo, definición de problemas, desarrollo técnico e integración de soluciones (Sommerville & Galipienso, 2011).

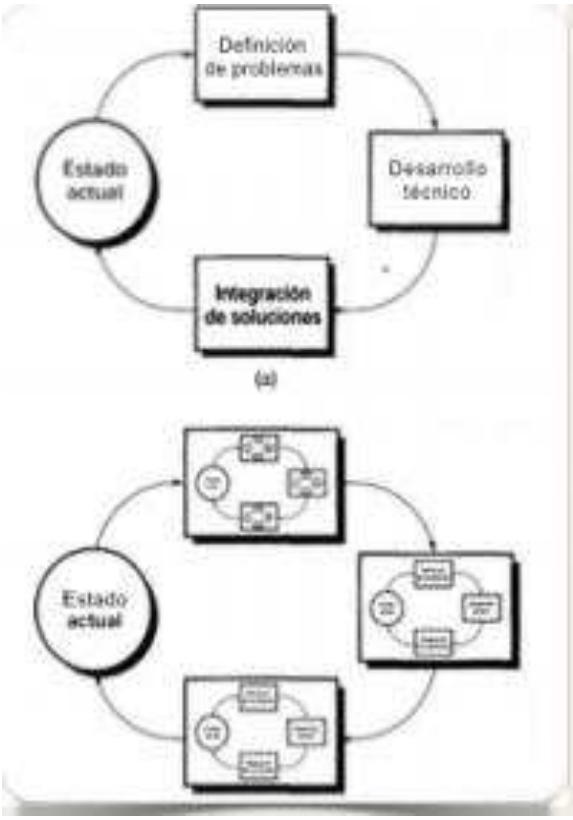

Ilustración 3: Modelado de proceso de software

Fuente: Ingeniería del software

Elaborado por: Ian Sommerville, María Isabel Alfonso Galipienso

2.3.4 **Metodologías de desarrollo de software**

Se ha establecido que las metodologías de desarrollo de software se basan alrededor de tres pilares en los cuales se fundamente estas metodologías básicamente de la siguiente manera: que es lo que se debe hacer y el orden, como se deben realizar los procesos y que materiales pueden ser útiles para llevar a cabo estas tareas.

Esto representa los ciclos de vida de acuerdo a las diferentes etapas, procesos y tareas que deben realizar de acuerdo a una sistematización por parte de una metodología sabiendo con precisión del tipo de ciclo a empleas se deben tener en cuenta las técnicas a realizar con sus respectivas actividades y herramientas de software a emplear para cada uno de los casos. Los sistemas informáticos se conciben y fomentan mediante un

mismo desarrollo conforme a una misma manera. Existen una variedad de paradigmas acerca del ciclo de vida del software que dependiendo de las características y necesidades de lo que se va a desarrollar en algunos casos la aplicación se da de manera múltiple utilizando más de un paradigma aplicando lo mejor de cada uno haciéndolo de manera hibrida. Debido a esto se han desarrollado sistemáticamente paradigmas según un ciclo de vida que mayormente se encuentra como secuencial: es centra en realizar y cumplir un o unas actividades y no empezar otras relacionadas, espiral: se despliega un análisis basado en realizar un desarrollo con el mínimo de riego posible que se puedan presentar a lo largo del ciclo de vida, evolutivo: se puede empezar con un mínimo de riesgo y mediante el transcurso del desarrollo evoluciones a tal punto de variar los requisitos añadiéndole y forman una estructura (Pressman, 2010).

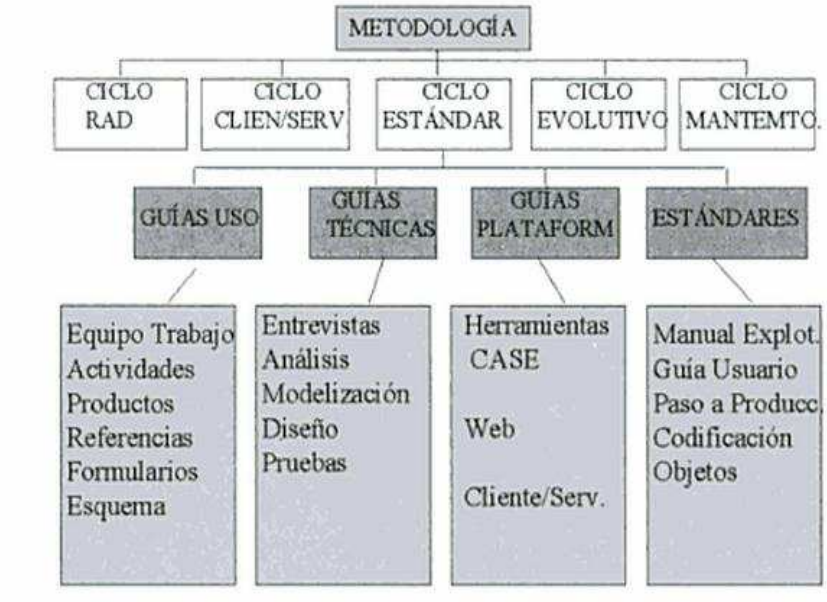

Ilustración 4: Metodologías de desarrollo de software

Fuente: Ingeniería del software: Un enfoque práctico

Elaborado por: Roger Pressman

Sim embargo todas estas etapas de software, ciclos, etapas de análisis, desarrollo, pruebas e implantación son indicadores en una o varias estas del ciclo de vida que se adopte a pesar de esto entra en juego técnicas que permitan la recolección de información por parte del desarrollador a los clientes mediante esta técnica se llevan a cabo reuniones que permiten conocer los principales objetivos que se necesitan cumplir para que el proyecto funciones por contraparte se van desglosando los requisitos que se dan la base para el desarrollo y el ciclo de vida que se necesite implantar. Sim embargo se realizan actividades específicas de análisis y diseño mediante un modelado los procesos y funciones que se contemplan a lo largo del desarrollo y la información recopilada hasta el momento de su elaboración por consiguiente esto produce el diseño propio del software que se quiere elaborar y las bases de datos o ficheros (Román & Tuya, 2007).

2.3.4.1 **Metodologías de desarrollo de software clásicas**

Las diferentes etapas que atraviesa el software en ciclo de vida desde procesos sintetizados también conocido o denominados como metodología se concentran en una agrupación de modelos de análisis y procesos que pueden ser (cascada, incremental, espiral, evolutivo). Basado que los modelos consecuentemente poseen y definen los artefactos, actividades, roles acompañado de una fase de técnicas seguidas de buenas prácticas que se relaciona a lo largo de una metodología para llegar a un objetivo planeado para todo el ciclo de vida de software.

Las metodologías clásicas poseen un enfoque lineal o casi lineal, sistemático, secuencial, con lo cual se inicia con la especificación de los requisitos en las entrevista de trabajo por parte del desarrollador y cliente pasando a una planeación con lo cual se deben especificar la forma en que se desarrollara el software y continua con el modelado lo cual pasa por técnicas de diagramación para llegar a la fase de construcción donde se

emplean técnicas de programación o herramientas que permitan transformar la modelación en código para sus posteriores prueba y ensayos para dar a ejecutar el despliegue que forma la entrega al cliente y su respectivo soporte y retroalimentación a pesar que todas estas fases de ciclo de vida de software.

En la actualidad, el trabajo de software esta acelerado y sujeto a una cadena infinita de cambios (de características, funciones y contenido de la información). Con frecuencia, el modelo en cascada no es apropiado para dicho trabajo. Sin embargo, puede servir como un modelo de proceso útil en situaciones donde los requerimientos están fijos y donde el trabajo se realiza, hasta su conclusión de una manera lineal (Pressman, 2010).

2.3.4.2 **Metodologías de desarrollo de software agiles**

Los métodos ágiles enfatizan las comunicaciones cara a cara en vez de la documentación. La mayoría de los equipos ágiles están localizados en una simple oficina abierta, a veces llamadas “plataformas de lanzamiento". La oficina debe incluir revisores, escritores de documentación y ayuda, diseñadores de iteración y directores de proyecto. Los métodos ágiles también enfatizan que el software funcional es la primera medida del progreso. Combinado con la preferencia por las comunicaciones cara a cara, generalmente los métodos ágiles son criticados y tratados como "indisciplinados" por la falta de documentación técnica.

La idea detrás de estos procesos es que los requisitos de una aplicación no siempre pueden definirse complementa mente antes de comenzar la implementación y es necesario un ciclo de vida basado en iteraciones cortas y una comunicación muy fluida con el cliente para que el

proyecto vaya bien encaminando desde el principio y no se desvíe en fecho y coste (Brito, 2009).

El desarrollo de un buen software y la respectiva entrega cumpliendo todos los aspectos contractuales, requieren de un marco de creación, desarrollo, implementación y depuración sistemático, pero a la vez que no sea rígido. Alrededor de los desarrolladores de software siempre se consta de una frase que tiene muchas adiciones a lo largo de la trayectoria que pueda tener el ciclo de vida de software como la complejidad inherente al software mediante esta frase siempre se esconde los posibles fracasos, fallos, resultados no deseados tan real como lo que es el mismo software, las dimensiones que pueden tener un sistema de información, un componente son enormes y que pueden contraer tareas de procesamiento críticas en el funcionamiento integrado que se pueda dar, con sus respectivos comportamientos los cuales deben ser tratados particularmente pero también de forma general. La respectiva unión o integración de un conjunto de componentes que puedan formar un sistema corresponden generalmente a un proceso de desarrollo de software. Estos sistemas utilizan las siguientes características:

La complejidad del dominio del problema es enfoca en situaciones del desarrollo donde se traslada eventos o escenarios basados en las entrevistas sostenidas con el cliente y que mayormente se encuentran limitantes que coinciden con una infinidad de requerimientos que no se complementan o contradicen. Las expresiones como simplicidad de uso, coste de fiabilidad y adaptación a cambios son componentes externos que se añadirán al proyecto en el transcurso del desarrollo se podría asumir la dependencia a elaborar proyectos con clientes y usuarios que generalmente dominan el problema en un ambiente real, pero con muy pocos conocimientos sobre el software a implementar y desarrollar, de aquí se introducen sea para un metodología o modelo informático representación

visual que traten de introducir al cliente en análisis del dominio del problema (Muñoz, Piattini, & Rubia, 2010).

2.3.4.2.1 Metodología XP

El desarrollo de esta metodología conocida como programación extrema (XP) se basa en un enfoque para la creación, diseño, construcción mediante la utilización de buenas prácticas para el desarrollo la cual propone enfocarse en extremos. Su enfoque es trasladar el desarrollo mediante alternativas a metodologías tradicionales como cascada prototipo y evolutivos proyectos que poseen un enfoque estructurado que haya fallado, o por la forma de cultura organizacional que posee hace que sea mucho más difícil que el software sea propenso a utilizar un método alternativo (Pressman, 2010).

En fases culminativas del software donde se encuentra en versiones que son para las pruebas e implantación se requerirá con frecuencia realizar recambios en la parte administrativa que pueda contener, la programación extrema constituye un garante de terminación exitosa de un proyecto informático siempre que haya un margen de error mínimo o ajuste de recursos esenciales como puede ser el tiempo requerida con respecto a módulos internos, el coste, el rendimiento y fiabilidad a contingencias que pueda llevar a tener mediante estas variables que se mencionan se podrá tener una adecuada planeación del proyecto a desarrollar y sobre todo conseguir un balance de lo que se constara en las fases de desarrollo como son los recursos y actividades necesarios para terminar el proyecto.

El desarrollo de prototipos se conoce como un complemento muy utilizado o como métodos alternativos surgen de la necesidad que tenían en presentación que se extendían por mucho tiempo lo cual mantenía con cierta incertidumbre a los usuarios por lo cual se deicidio basado en un enfoque iterativo que permita entregables mediante el proyecto se va desarrollando a

efecto de esto surgieron las denominadas metodologías agiles siendo el modelo de prototipo pionero en esta forma.

La programación extrema se iniciaría basada en casos donde lo más común es que una institución que se basa en la creación de software se cumplen normalmente roles para cada colaborador en el desarrollo de un software casos como lo que corresponde al diseño esto permite ver que la mayoría de veces la relación que se tiene por parte de los clientes y el equipo de desarrollo varía mucho de las metodologías tradicionales a las agiles sobre todo con los requisitos ya que estos se capturaban al momentos de las entrevistas de trabajo con la diferencia que generalmente nunca quedaba toda la información receptada y entendida (Rubén, 2014).

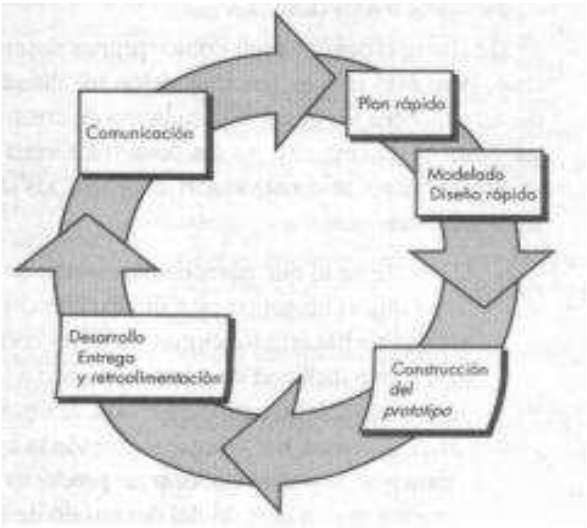

Ilustración 5: Etapas de la Metodología XP

Fuente: Desarrollo de Software ágil

Elaborado por: Lainez Fuentes José Rubén

El desarrollo en que se basa la programación extrema toma un gran punto de inflexión con los actores que van a ser parte de los eventos del software con el equipo de trabajo que se establecerá con el equipo de trabajo ya que su importancia es tal que si en el momento de desarrollar no sean han planificado reuniones para tratar de avances se puede caer en un

pozo donde la comunicación no se pueda dar y haya problemas en el futuro con los requisitos que se necesitan y no se tomaron en cuenta se requiere además iteraciones con el equipo de trabajo cuando se necesitan realizar cambios o después de uno además de todo posible fallo localizando mostrando niveles de prioridad.

Para que todas las interacciones con el equipo de programadores surjan efecto se necesita que en XP se establezca como lo dictan sus buenas prácticas la realización de pruebas de aceptación con los posibles actores potencias para los eventos descritos de esa manera se aprovechara eficientemente cada rol. Bajo los paradigmas del desarrollo XP realiza pequeños historiales de acuerdo a la capacidad con que se han dado las reuniones de trabajo a esto se lo denomina Historias de usuario la cual corresponde a los requisitos que el usuario final requiere en el software tanto en versiones finales como el final. Este proceso se basa en dos fases:

- En la fase inicial el usuario explica de acuerdo a su manera de expresar e interpretar su necesidad lo que él desea que software tenga o haga de manera que el encargado del equipo deberá comunicarse con el cliente de manera de hacerle llegar cualquier por menor que surja con fallas de cualquier tipo relacionado a las fases de XP que se requieran hacer y con su respectivo coste debido a esto el usuario dejara un historial y mediante el orden de prioridades que se requiera en función de esto se definirán fechas fijas que cumplirán los avances que se realicen.

- Se basa en una política propia de XP que pide que cada dos o tres semanas se deberán realizar entregables de manera que se agruparan de acuerdo a las funciones de cada rol que puede que varía con el tiempo y finalmente las historias se vuelven cartas que le facilitara al desarrollador el trabajo catalogándolas de manera que se tenga una estructura previa para la planificación del proyecto

- Cuando se recopilan todas las historias de usuarios que se realizado se deberán programar una planificación por etapas la cuales se dirigirán distintas iteraciones en donde cada una de las partes interesadas por etapa deberá estén realmente implicadas en el desarrollo del proyecto

La programación extrema consta de características permisibles tanto para el equipo de trabajo como para los actores potenciales vinculados en los eventos que se recopilaron mediante las entrevistas de trabajo como son:

- La programación extrema dicta que por cada entregable el proyecto deberá aportar mejoras continuas.

- Por cada fase cumplida en versiones finales se deberá realizar pruebas unitarias que consigan ser de entregables para el cliente pro lo cual es muy casual que sean iterativas en el sentido de repeticiones automatizadas que permitan también las pruebas de unitarias de regresión por lo general se desarrolla antes que el desarrollo en sí.

- El desarrollo como lo dicta la política de la programación extrema se basa en que los avances desde la fase de análisis y construcción se realizan en parejas que generalmente son dos personas del mismo puesto. Se realiza de la manera que uno está pendiente y revisa lo que el otro desarrollador realiza y corrigen de manera sincrónica así se realiza una eficiente colaboración para evitar los posibles fallos de productividad que se puedan dar.

- Otra política que se realiza con la programación extrema es que del equipo de trabajo se necesita siempre que uno de los clientes este continuamente supervisando y si es posible retroalimentado el trabajo de los desarrollado así permitiendo integraciones continuas y haciendo una detección de errores sin margen de error para los requisitos que en el futro puedan cambiar en el ciclo de vida de cada módulo en equipos de trabajo distintos, esta metodología ayuda a que todos los involucrados tenga voz y voto en el desarrollo del software (Pressman, 2010).

2.3.4.2.1.1 Fases de la metodología XP

- **1ª Fase: Planeación del Software**: La iniciación que comprende que la metodología XP es establecer mediante reuniones con los involucrados en el desarrollo del software y los actores que serán parte los casos de uso que se definirán con la finalidad de establecer las historias de usuario con los usuarios. Basado en esta política se infiere que tanto historias de usuario como los casos de uso que se desarrollaron previamente tienen un cierto parecido pero ciertas diferencias como la involucración del usuario en estas historias ya que usualmente el colabora con sus experiencias y funciones que se reflejan de manera escrita en esta historia de una manera casual.

  En esta fase se detalla de una manera no técnica las características y funcionalidades que se requiere que lleve el software, pero de esta manera no se hace ninguna referencia a las posibles estructuras de bases de datos a implementar ni de codificación todas estas historias de usuarios que son utilizados para la estimación de los tiempos de desarrollo otra utilización son para pruebas unitarias que se necesitaran para verificar si la aplicación funciona correctamente cuando una historia está completa y se la va a implementar, las 2 partes deben realizar una reunión que indique lo que se debe concretar y cumplir en dicha historia. Cuando se tienen todas las historias de usuarios ya definidas se realiza un planeamiento de entregables donde es establece que cada historias de usuario que se realiza deberá está relacionada o vinculada una versiona fija del software con su respectiva fecha de entrega, cuando se llega a este punto se definen 4 factores que son los más influyentes en esta etapa como son los objetivos que son referentes al cumplimiento de las historias de usuario, el tiempo que se llevara el desarrollo y los entregables con sus respectivas fechas y el número de personas que se necesitara para cada realizable cuando se tiene los 4 factores se

pasa a evaluar la calidad del software que se va implementar por cada versión.

Cualquier software bajo la metodología XP se parte en interacciones muy cercanas a las dos o tres semanas de duración cuando inician una interacción los usuarios escogen las historias establecidas en la planeación de entregables además se realizan las historias que no pasaron las pruebas de aceptación que se han hecho interacciones pasadas.

- **2ª Fase: Diseño**: Bajo esta metodología XP se persigue que en la fase de codificación la carga bajo la construcción sea menos pesada ya que se realizan diseños simples y sencillos basado esto se tiene en XP una política basada en un diccionario de termino y especificaciones que se basan en que la función principal es identificar y ayudar a facilitar la lectura de funciones métodos clases y una mayor reutilización del código. Cuando se presentar fallos que infieren causa potencial se establece que una pareja del equipo de desarrolladores busque en el problema siempre teniendo en cuenta que la funcionalidad operar sobre todo ya que esta solo debe ser pedida para lo que fue diseñado sin agregar más funcionalidad extra, aquí se parte de refactorizaciones continuas.
  Bajo esta etapa surge la utilidad de tarjeta CRS las cuales permiten que la codificación se oriente bajo la programación orientada a objeto dejando atrás la tradicional programación estructurada mediante la vinculación con objetos o clases las cuales representadas mediante propiedades y atributos realizan las acciones de las historias y las propias características que puedan contener.

- **3ª Fase: Desarrollo y Pruebas**: El usuario es fundamental para la parte de codificación o desarrollo; se necesita que se encuentre presente ya que este aportara en todas fases de desarrollo su experiencia para que las características del software sean correctas.

Cuando se necesita desarrollar las historias de usuario juegan un papel muy importante ya que los usuarios crean estas historias y son ellos mismo quienes tienen puesto el tiempo en que serán realizables y entregables. Antes de que se llegue al desarrollo el cliente deberá realizar una revisión de todas las historias previamente probadas y estructuradas.

El desarrollo debe ser de manera que se esté bajo políticas y estándares que provén un diseño reusable, la creación de pruebas unitarias y de características de cualquier extensión o directamente consigo de componentes es la parte más crítica en esta fase ya que pone no solo el código a prueba sino la misma ingeniería de requisitos que se desarrolló para el proyecto (Torres, 2013).

2.3.4.2.2 Metodología SCRUM

SCRUM posee un esquema basado en la gestión y desarrollo de software ágil mediante tareas y procesos iterativos que se aplican a modelos incrementales, esta metodología de software puede ser utilizada mediante equipos de desarrollo completos o de mantenimiento o un acercamiento a gestiones de programas.

Esta metodología representa una alternativa para los softwares que llevan conformada parte o completamente su estructura metodologías que siguen patrones fijos clásicos como cascada, como dictan algunas referencias acerca de la creación de software complejo bajo este marco de referencia que es especifica la creación de software complejo con esto SCRUM realiza el desarrollo denominados equipos SCRM a quienes se les asigna.

El desarrollo ágil de software refiere a métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan mediante la colaboración de grupos auto organizado y disciplinario. Existen muchos métodos de desarrollo ágil; la mayoría minimiza riesgos desarrollando software en lapsos cortos. El

software desarrollado en una unidad de tiempo es llamado una iteración, 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 una DEMO (sin errores) al final de cada iteración. Al final de cada iteración el equipo vuelve a evaluar las prioridades del proyecto (Chin - Gary, 2004).

2.3.4.2.2.1 Características SCRUM

SCRUM es un modelo de referencia que define un conjunto de prácticas y roles, y que puede tomarse como punto de partida para definir el proceso de desarrollo que se ejecutará durante un proyecto. Los roles principales en SCRUM son el SCRUM MASTER, que mantiene los procesos y trabaja de forma similar al director de proyecto, los interesados externos o internos, y el equipo que incluye a los desarrolladores. Durante cada sprint, un periodo entre una y cuatro semanas (la magnitud es definida por el equipo), el equipo crea un incremento de software potencialmente entregable (utilizable). El conjunto de características que forma parte de cada sprint viene del Product Backlog, que es un conjunto de requisitos de alto nivel de prioridades que definen el trabajo a realizar. Los elementos del Product Backlog que forman parte del sprint se determinan durante la reunión de Sprint (García, Dedo, & Gómez, 2012).

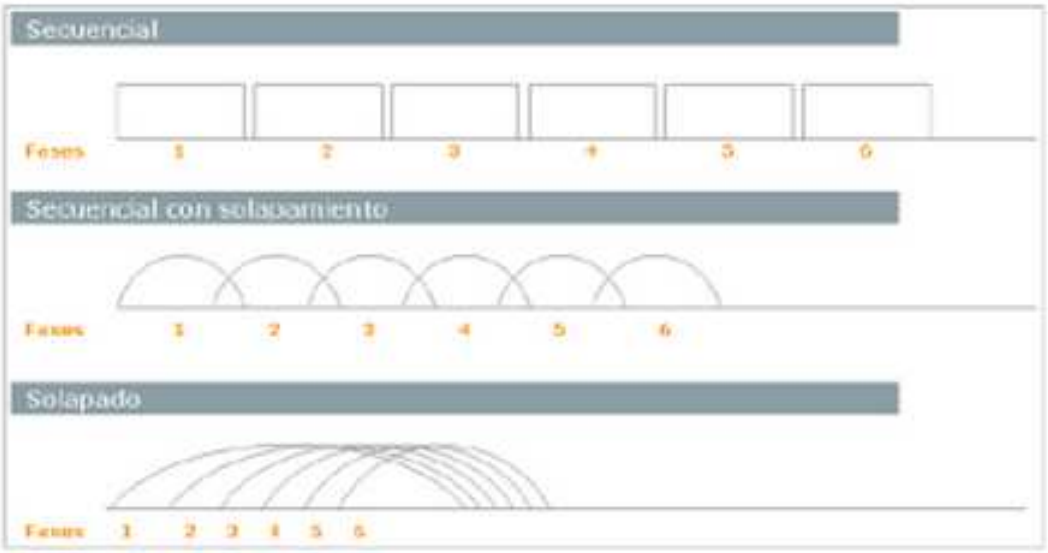

Ilustración 6: Velocidades basadas en Sprint

Fuente: Métodos Ágiles y Scrum

Elaborado por: Alonso Álvarez García, Rafael de las Heras del Dedo, Carmen Lasa Gómez

Durante esta reunión, el Product Owner identifica los elementos del Product Backlog que quiere ver completados y los hace del conocimiento del equipo. Entonces, el equipo determina la cantidad de ese trabajo que puede comprometerse a completar durante el siguiente sprint. Durante el sprint, nadie puede cambiar el Sprint Backlog, lo que significa que los requisitos están congelados durante el sprint.

Scrum permite la creación de equipos auto organizado impulsando la con localización de todos los miembros del equipo, y la comunicación verbal entre todos los miembros y disciplinas involucrados en el proyecto.

Un principio clave de Scrum es el reconocimiento de que durante un proyecto los clientes pueden cambiar de idea sobre lo que quieren y que los desafíos impredecibles no pueden ser fácilmente enfrentados de una forma predictiva y planificada. Por lo tanto, Scrum adopta una aproximación pragmática, aceptando que el problema no puede ser completamente

entendido o definido, y centrándose en maximizar la capacidad del equipo de entregar rápidamente y responder a requisitos emergentes.

Las características más marcadas que se logran notar en Scrum serían: gestión regular de las expectativas del cliente, resultados anticipados, flexibilidad y adaptación, retorno de inversión, mitigación de riesgos, productividad y calidad, alineamiento entre cliente y equipo, por último, equipo motivado. Cada uno de estos puntos mencionados hace que el Scrum sea utilizado de manera regular en un conjunto de buenas prácticas para el trabajo en equipo y de esa manera obtener resultados posibles. Existen varias implementaciones de sistemas para gestionar el proceso de Scrum, que van desde notas amarillas y pizarras hasta paquetes de software. Una de las mayores ventajas de Scrum es que es muy fácil de aprender, y requiere muy poco esfuerzo para comenzarse a utilizar (Dimes, 2015).

2.3.5 **Modelos informáticos**

Un software se indistintamente de un proyecto de ingeniería, se componen de un conjunto de etapas que dan vida al ciclo de desarrollo. En el caso de ser un proyecto basado en el desarrollo informático está constituido por actividades concretamente dirigidos por paradigmas informáticos de esta manera de acuerdo a las necesidades se incorporan a la manera en que se va organizar el proyecto como clásicos, prototipos o ya sea orientados a objetos.

Mediante este enfoque en que una importante fundamental del desarrolla determina cual es la metodología para el ciclo de vida más adecuado a las necesidades del software condicionando el direccionamiento

y razón en que se desarrollan las actividades definidas en las metodologías de desarrollo debido a 4 razones:

- El coste que pueda implicar hace la diferencia para que sea el elemento más costoso de un sistema informático.

- Las generaciones de hoy en día ya sea de ordenadores y redes de telecomunicación permite una gran disminución en la complejidad que se tenía al desarrollar software ya que tenía que incorporarse esta tecnología de manera que se ajusten a la necesidad para pequeño o un solo proyecto.

- Las bajas que se tuvieron cuando se daba formas de desarrollo siguiendo metodologías clásicas que impedían cumplir con todos los plazos fijados en las entrevistas con los clientes (Guérin, 2012).

2.3.6 **Sistema de control de versiones (S.C.V)**

El sistema de control de versión es corresponde a una herramienta que almacena cualquier cambio en varios proyectos indistintamente relacionado a la vez, de esta manera se crean historiales de versión en cualquier fase de construcción que se esté implementando. Las versiones son estáticas ya que en registro de versión se guarda estáticamente todos los cambios que originó el producto de software además de todas las integraciones de forma acumulada.

Esta manera no implica la automatización completa de integraciones futuras porque su proceso es de forma manual, pero de manera que facilitan

los problemas que surjan al momento de que un equipo de desarrolladores necesita la integración de módulos a uno o varios proyectos existentes debido el producto va cambiando y lo va haciendo mediante el sistema de control de versiones que posee permitiendo mayor cohesión (Solsona & Viso, 2007).

2.3.7 **Desarrollo dirigido por tests (T.D.D)**

El Desarrollo Dirigido por Tests (T.D.D), está constituida como parte de las buenas prácticas que ofrece XP precisamente cuando se habla de diseño y técnicas de desarrollo debido esto tiene tres razones para ser considerado como técnica para reducir considerablemente los posibles fallos que se asomen en la fase construcción del software utilizando mediante el modelo de orientación objeto se logra una alta reutilización de código y preparado para el cambio si eso llegara con los requisitos previamente escogidos. La refactorización como esta expresado en un sin número de conceptos es la forma de volver a escribir código; se trata de modificar ciertas partes del código, pero sin alterar el propio comportamiento. Para esto se realiza la búsqueda de códigos duplicados mediante las técnicas de la TDD incidiendo en que esto debería ser eliminado de manera que no haya código duplicado y sea mucho más eficiente el código (Minoli, García, & Garzás, 2010).

2.3.8 **Tipos de test**

La relación entre las distintas pruebas que se presentan en el desarrollo de aplicaciones es susceptible a interpretaciones caóticas. Principalmente no se han categorizado los distintos tipos de pruebas de desarrollo de acuerdo a las necesidades y componentes que se llevan en torno a la programación que se realice. Los equipos generalmente realizan tipos de adaptación o presentan sus propias convenciones ya que en las pruebas que se desarrollan varían de acuerdo a las necesidades de los

equipos ya que se puede llegar hasta interconectar pruebas para verificar si un componente es homogéneo a otro y de esa manera se presentan distintos tipos de pruebas.

Cada equipo desarrolla terminologías propias de desarrollo para sus pruebas, pero sobre todo lo hacen porque no hay una generalización universal de cómo escribir las pruebas de manera similar a otros desarrolladores, para ser concisos al momento de desarrollar una prueba (Minoli, García, & Garzás, 2010).

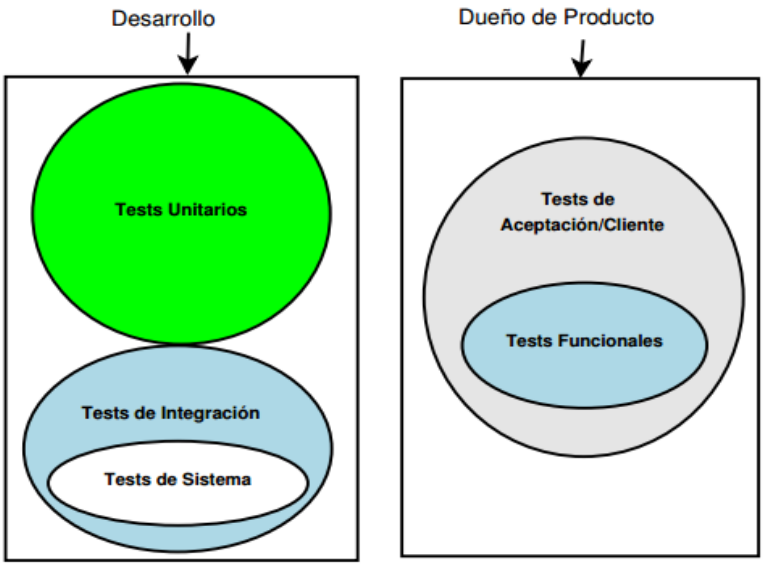

Ilustración 7: Escenarios para el desarrollo de Test

Fuente: Cómo implantar un proceso de integración continua

*Elaborado por: **Mariano Minoli**, **Ana María García**, **Javier Garzás***

2.3.8.1 **Desarrollo dirigido por tests de aceptación (A.T.D.D)**

Estas pruebas de aceptación se realizan para que el cliente certifique que el sistema es válido para él. La planificación detallada de estas pruebas debe haberse realizado en etapas tempranas de desarrollo del proyecto, con el objetivo de utilizar los resultados como indiciador de su validez; si se ejecuta las pruebas documentadas a satisfacción del cliente, el producto se considera correcto y, por tanto, adecuado para su puesta en producción.

Este enfoque basado en una implementación para volver más eficiente y eficaz ciertos paradigmas agiles cabe destacar que su enfoque se hace de manera más cerca con el cliente y su experiencias con las característica recogidas en fases de análisis, toda historia de usuario es utilizada como los requisitos que se vinculan a esta pruebas sus diseños se realizan tanto en fase de análisis como construcción por esto entran muy temprano a fases de desarrollo e implementación para realizar las pruebas respectivas ya que esto asegurara la calidad del producto. Es una forma de afrontar la implementación de una manera totalmente distinta a las metodologías tradicionales. El trabajo del analista de negocio se transforma para reemplazar páginas y páginas de requisitos escritos en lenguaje natural (nuestro idioma), por ejemplos ejecutables surgidos del consenso entre los distintos miembros del equipo, incluido por supuesto el cliente (Jurado, 2010).

2.3.8.2 **Desarrollo dirigido por pruebas unitarias**

Las pruebas unitarias constituyen el primer paso para detectar herreros en el código, pues se centran en la menor unidad de diseño del software el modulo o un método de una clase el objetivo principal de estas pruebas es el de la detección de errores en cada uno de los módulos del software a ser ejecutados independientemente del resto de componentes. Se suele ser ejecutadas por el programador que construye el modulo sobre sus aspectos estructurales, intentando asegurar algún tipo de cobertura de algunos aspectos funcionales, para comprobar la integridad de la información en las estructuras de información locales y la interfaz del módulo. No obstante realizar la integración con otros módulos deberá revisarse de nuevo la interfaz.

Aunque para estas pruebas de unidad se considera cada módulo de manera independiente, durante la ejecución de los casos de prueba se debe

proporcionar un entorno que simule la comunicación con el resto del sistema. En concreto, será necesario un conductor que simule la llamada al módulo, pasándole al componente los datos precisos para realizar la prueba. En muchos casos los conductores se completan con funcionalidad para informar al probar sobre los resultados obtenidos en la prueba.

Los resguardos, por su parte harán las veces de los módulos que son llamadas por el componente bajo prueba. Pueden utilizarse versiones preliminares de esos módulos se pueden codificar resguardos que emiten siempre la misma respuesta ante la llamada del módulo bajo prueba o simplemente pueden no realizar operación alguna. La necesidad de utilizar estos elementos hace que las pruebas se vean en ocasiones adulteradas por detección de falsos errores debidos a la implementación de estos módulos complementarios. En entornos de desarrollo orientado a objetos se utilizan objetos simulados para realizar la labor de los conductores y resguardos.

Sim embargo, los objetos simulados son elementos activos en el proceso de prueba puesto que permiten al probador definir un conjunto de reglas que se deben cumplir en la iteración de los objetos bajo la prueba con los simulados. En caso de que un objeto simulado, durante la ejecución de una prueba, detecte el incumplimiento de alguna de las reglas de interacción, provocara su interrupción e informara de la anomalía detectada (Tuya, Román, & Cosín, 2007).

2.3.8.3 **Desarrollo dirigido por pruebas entregables**

Las pruebas de entregas son el proceso de probar una entrega del sistema componente o modulo que será distribuido a los clientes. El principal objetivo de estos procesos es incrementar la confianza del suministrador en que el sistema satisface sus requerimientos. Si es así, este puede entregar

como un producto o ser entregado al cliente. Para demostrar que el sistema satisface sus requerimientos, tiene que mostrar que esta entrega la funcionalidad especificada, rendimiento y confiabilidad, y que no falla durante su uso normal. Las pruebas de entregas son normalmente un proceso de pruebas de caja negra en las que las prueban se derivan a partir de las especificaciones del sistema. El sistema se trata como una caja negra cuyo comportamiento solo puede ser determinado estudiando sus entradas y salidas relacionadas, Otro nombre para esto es de pruebas funcionales, debido a que el probador solo le interesa la funcionalidad y no la implementación del software. Se han recogido experiencia de pruebas en conjunto de guías que incrementan la probabilidad que las pruebas de defectos tengan éxitos. Algunos ejemplos de estas guías son:

- Elegir entradas que fuerzan a que el sistema genere todos los mensajes de error
- Diseñar entradas que hacen que los búferes de entrada se desborden
- Repetir las misma entrada o series de entradas varias veces
- Forzar a que se generen las salidas invalidas

Forzar los resultados de los cálculos para que sean demasiado grandes o demasiado pequeño (Jurado, 2010).

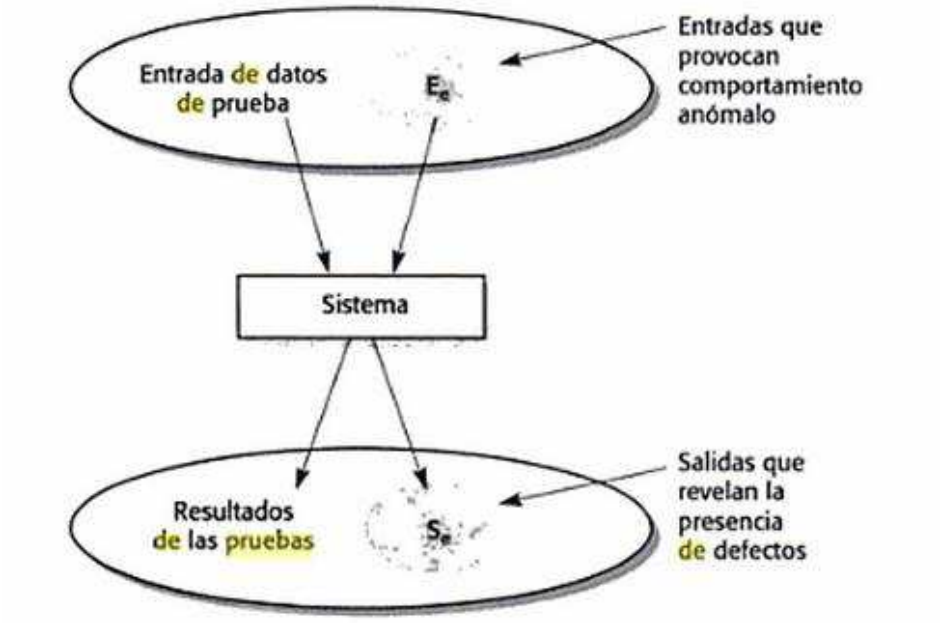

Ilustración 8: Etapas de pruebas de aceptación

Fuente: Diseño Ágil con TDD

Elaborado por: Carlos Blé Jurado

2.3.8.4 **Pruebas de implantación**

Las pruebas de implantación se realizan con el software ya instalado en su entorno de operación real y por tanto se centran en 2 aspectos por una parte los errores derivados de la integración de hardware y software del propio sistema; por otra la interacción del sistema con resto de sistemas de la instalación. Se analizan primordialmente requisitos no funcionales, pero también funciones que dependan fuertemente del entorno de operación real o requieren la interacción con sistemas o componentes que no se encontraron disponibles hasta la fase de implantación. En concreto se realizan pruebas de seguridad para verificar si existen errores en los mecanismos de protección del entorno de operación. También deben realizarse pruebas de rendimiento reales puestos que en las pruebas del sistema solo habrán podido comprobarse rendimientos teóricos sobre el entorno de desarrollo. Por último, se examinan los mecanismos de

recuperación y copia de seguridad de modo que se detecten errores en caso de caídas en los servicios de software o hardware (Tuya, Román, & Cosín, 2007).

2.3.8.5 **Pruebas de sistema**

Es una de las pruebas que permiten realizar la integración entre cualquier componentes modulo o aplicación misma o externa a nivel de sistemas de información ya que como tal integra varias partes de un sistema a nivel tanto funcional como de integración ya que puede causar o provocar automatización de bases de datos o inclusión de procesos en un servidor de cualquier índole de servicio además estas pruebas se realizan a nivel de rendimiento ya que para llegar a este punto se debe contar con pruebas unitarias y de implantación para que sean probadas. La realización de este tipo de pruebas se puede considerar por dos tipos pruebas de sistemas funcionales y de integración considerando el último tipo de pruebas son pequeñas pruebas de sistema a escala de módulos y componentes fuertemente unidos o dependientes de funcionalidad ya que la diferencia radica en dependencia de modulo a componente y de la misma forma como la funcionalidad misma de cualquier componente (Tuya, Román, & Cosín, 2007).

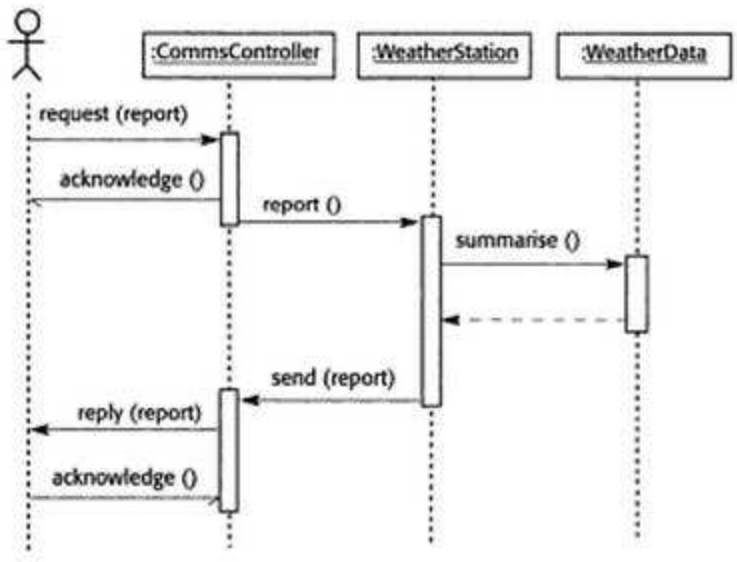

Ilustración 9: Etapas de pruebas sistema

Fuente: Técnicas cuantitativas para la gestión en la Ingeniería del Software

Elaborado por: Isabel Ramos Román, José Javier Dolado Cosín, Javier Tuya

2.3.8.6 **Desarrollo dirigido por pruebas de rendimiento**

Cuando un sistema se ha integrado completamente, es posible probar las propiedades emergentes de un sistema tales como rendimiento y fiabilidad. Las pruebas de rendimiento tienen que diseñarse para asegurar que un sistema pueda procesar su carga esperada esto normalmente implica planificar una serie de pruebas en las que carga se va incrementando regularmente hasta que el rendimiento del sistema se hace inaceptable. Como sucede con otros tipos de pruebas, las pruebas de rendimiento como de descubrir problemas y defectos en el sistema. Para probar si los requerimientos de rendimientos son alcanzados, se debe construir un perfil operacional el cual es un conjunto de pruebas que reflejan la combinación real de trabajo que debería ser maneja por un sistema debido a que si un 90% de transacciones son de tipo A y un 5% de tipo B entonces como regla

se debería crear un perfil operacional para que la amplia mayoría operacional de un sistema (Jurado, 2010).

2.3.8.7 **Pruebas de regresión**

Estas pruebas son utilizadas cuando un equipo de desarrollo detecta un problema y precisamente corregido se volverá a enviar al entorno de pruebas aquel o aquellos componentes afectados y corregidos, para su verificación. Dado que al subsanar un error de programación pueda cometer otros, suelen volver a repetir pruebas donde intervienen componentes, para verificar el funcionamiento (Tuya, Román, & Cosín, 2007).

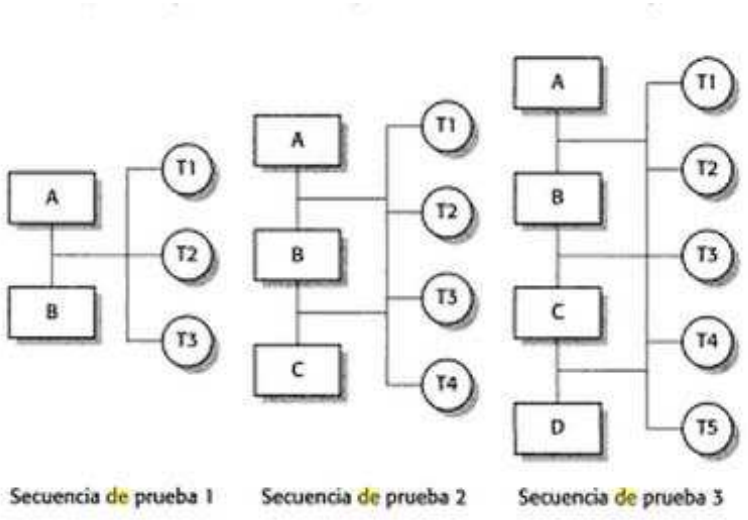

Ilustración 10: Ejemplo de los flujos de una prueba de regresión

Fuente: Técnicas cuantitativas para la gestión en la Ingeniería del Software

Elaborado por: Isabel Ramos Román, José Javier Dolado Cosín, Javier Tuya

2.3.9 **Integración continua (I.C)**

Mediante este modelo informática se logra ser mucho más transparente y visible con el entorno de trabajo en quipos para lo cual independiente de escogerse o no una mitología ágil para el desarrollo para

tener un historial basado desde la fases de análisis hasta construcción de esta manera todo el equipo ve lo que va cambiando y de manera sincrónica ve la evolución del proyecto de forma más dinámica ósea versiones que son estáticas se van compilando y formando de manera dinámica en un ordenar local de un miembro del equipo de trabajo.

La integración continua busca sobre todo la detección de errores de manera muy temprana cuando se inicie fases de construcción, de esta manera se logra la creación de varias pruebas que realizan en fase de análisis haciendo que cualquier versión sea revisada y aprobada independientemente del tipo de paradigma que se llega a tomar mediante este tipo el control de versión se tendrá un mayor acceso de calidad para todas las subidas que se realicen. La metodología de desarrollo no es determinante siempre y cuando se cumplan una serie de buenas prácticas. La IC es independiente del tipo de metodología de gestión (ágil o predictiva), sin embargo, por los beneficios que aporta, proporciona gran valor a las metodologías ágiles, ayudando a tener un producto funcional en todo momento. Los beneficios de hacer IC son varios, sin embargo, se entenderán y apreciarán mejor una vez que conozcamos las prácticas que conllevan (Jurado, 2010).

2.3.9.1 **Construcción**

Construcción tiene en el ámbito de desarrollo de software varios significados incluso se puede distinguir como la forma de que se compila el código fuente de un aplicación o consistir en la automatización de pruebas y código fuente, además estos pueden ser ejecutados en segundo plano como parte de scrips para formar lotes de código que se ejecuten dentro de un servidor o en un mismo repositorio de código fuente (Jurado, 2010).

2.3.9.2 **Prácticas de integración continua**

Automatización en fase de construcción

Divide el script de construcción en diferentes comandos para que cualquiera pueda lanzar una parte de forma aislada (por ejemplo, lanzar las pruebas), sin que pierda tiempo en realizar el proceso completamente. Normalmente, algunos desarrolladores usan un IDE para la construcción, estás herramientas son de gran utilidad para nuestra productividad, pero es esencial poder construir nuestro software sin IDE alguno. Nos deben facilitar la vida, pero no debemos caer en la dependencia absoluta.

La construcción está compuesta por pruebas unitarias

La automatización de pruebas unitarias se necesita como requisito funcional en fases de construcción para que al momento de llegar a fases de implementación se torne mucho más fácil al momento de empezar ya que estas prácticas están siempre asociadas a metodologías agiles se necesita que las pruebas se den de manera temprana y rápida ya que como dicta ciertas políticas de XP esto podría influir en los tiempos de entrega que se realicen a los clientes potenciales.

Los cambios se reflejan automáticamente

Es una de las prácticas más comunes dentro de CI ya que establece políticas entre desarrolladores cuando y como se debe realizar una subida de código al repositorio de código fuente esto implica que el código solo debe ser subido cuando haya sido revisado por los colaboradores del equipo de trabajo y haya pasado todas las pruebas de aceptación o unitarios que se tengan planeadas para esto (Humble & Farley, 2010).

2.3.10 **Sistema de control de versiones (V.C.S)**

Se trata de la gestión de una cantidad importante de archivos que son modificables a lo largo del tiempo, es posible para un conjunto de archivos sea necesario volver a una versión anterior como por ejemplo un fallo de diseño pueda implicar que el código que no es escalable si es posible volver a un estado original conocido, el tiempo invertido en revertir los cambios es menor, brinda la capacidad de tener equipos de desarrolladores de manera paralela de forma que exista una versión estable de todo el proyecto y otra más experimental del mismo donde se probaran diferentes algoritmos y diseño (Fernández & Martin, 2012).

2.3.10.1 **Commit**

La operación Commit representa cuando se produce una alteración del código y se da la necesidad de subir estos cambios a un repositorio local que brinda cada usuario o cliente que este tiene alojado, cada Commit representa un conjunto de alteraciones o cambios que han sufrido el código desde su última baja contemplados mediante un único identificador ya que cada Commit solicita una breve descripción de que parte del código se ha modificado (Jurado, 2010).

2.3.10.2 **Pull**

La operación Pull representa cualquier cambio que se haya subido al repositorio de código de versiones y se necesita descargar cualquier parte de estos cambios a un directorio local de trabajo o repositorio local para aplicarlo al código sim embargo es accesible a elección de partes del código alojados remotamente (Kawalerowicz & Berntson, 2011).

2.3.10.3 **Push**

La operación Push representa cualquier cambio que se convertido en Commit se solicitara cargarlo como ante paso a subirlo al sistema de control de versiones para adjuntarlo con cualquier cambio paralelo que se haya realizado (Kawalerowicz & Berntson, 2011).

2.3.11 **Entorno de desarrollo integrado (I.D.E)**

El Entornó de desarrollo integrado representa una aplicación en forma de grupos de herramientas de software dedicadas al desarrollo de aplicaciones pero esto no infunde en la manera que ha de compilar y depurar ya que se puede establecer para un solo lenguaje o como para varios, este viene como un solo programa que consta de un editor de código, compilador o compiladores dependiendo si se cuenta con más de un lenguaje de programación un depurador de código fuente y el diseñador gráfico para las GUI que se deseen realizar (Humble & Farley, 2010).

2.3.12 **Sistema operativo servidor**

Un nivel más abajo está los sistemas operativos de servidor. Estos se ejecutan en servidores que son computadores personales muy grandes, estación de trabajo o incluso mainframes, y dan servicio a múltiples usuarios a través de una red, permitiéndoles compartir recursos de hardware y software. Los servidores pueden prestar servicios de cualquier índole (Candela, García, Quesada, & Santana, 2007).

2.3.1. Servidor dedicado

Un servidor dedicado es utilizado como un prestador de servicios de almacenamiento dividido, así como parte de préstamos para cualquier servicio de almacenamiento fijo y dedicado, estos son administrables por el usuario o la organización que provee de estos servidores. Lo que corresponde a mantenimiento del servidor está comprendido por la organización que provee el propio servicio de servidor que viene incluido dentro del servidor o por quien provee el internet para todos los puntos de acceso que están dentro del rango del servidor (Candela, García, Quesada, & Santana, 2007).

2.3.13 **Robot (BOT)**

Se conoce como BOT aun programa informática que simula el comportamiento humano en ciertas tareas que realiza el computador de manera directa o indirecta haciendo esto simular eventos que se lanzan ya sea de manera servidor cliente o cliente servidor para lanzar aplicaciones interactuando dentro del computador o con usuarios mediante una acción, ya que este se ejecuta de acuerdo al tiempo o acciones que sucedan adentro en su configuración ya que el lenguaje que este maneja es indistinto (González, 2010).

2.4. CONCLUSIÓN

El presente marco teórico me dio la apertura para encontrar nuevas bases que fundamenten el presente trabajo de titulación bajo esto fui planteando conceptos que serán aplicables para el estudio que se quiere realizar sobre todo en la parte que corresponde a las metodologías de software que existen y como se pueden mejorar y por consiguiente automatizar procesos y en desarrollo haciéndolo más autónomo y eficiente. Bajo la premisa tecnológica la cual es sobre los avances que se dan cada día y como va evolucionando y mejorando en este capítulo me dio pautas para buscar nuevas formas de complementación con tecnologías que no se han profundizado lo suficiente para abarca en un proyecto de software y ofrecerán nuevas formas de automatización de procesos que abarcan desde paradigmas agiles para el desarrollo de software hasta modelos informáticos que puedan compenetrarse para seguir un proyecto en donde el equipo de trabajo y bajo la metodología que se siga haga énfasis en lo ágil.

CAPITULO III

DIAGNÓSTICO O ESTUDIO DE CAMPO

3.1. INTRODUCCIÓN

En el presente capítulo se explica la metodología investigativa que se utilizó, las técnicas y herramientas para la respectiva recolección de datos, y los resultados arrojados mediante un análisis de muestreo que se siguió para el "Estudio e implementación de BOTS para la automatización de la administración de código fuente mediante el modelo de Integración Continua para el Área de Desarrollo de la Facultad de Ciencias Informáticas de la Universidad Laica Eloy Alfaro".

3.2. TIPOS DE INVESTIGACIÓN

El presenta trabajo de titulación aplica la investigación descriptiva y exploratorio:

Se considera como investigación descriptiva aquella que reseña las características o rasgos de un objeto o momento de estudio. Consiste en "buscar especificar propiedades, características y rasgos importantes de cualquier fenómeno que se analice. Describe tendencias de un grupo o población. El tipo de investigación será descriptivo porque se someterá a un análisis en el que se mide y evalúa diversos aspectos o componentes tales como cuerpos legales y normativas vigentes del problema a investigar, teniendo esto como premisa se aplicará este tipo de investigación para cuantificar tiempo y medir la optimización del código que se pueda brindar con el desarrollo de este trabajo.

Los estudios exploratorios se realizan cuando el objetivo es examinar un tema o problema de investigación poco estudiado, del cual se tienen muchas dudas o no se ha abordado antes. Es decir, cuando la revisión de la literatura revelo que tan solo hay guías no investigadas e ideas vagamente relacionadas con el problema de estudio, o bien, si deseamos indagar sobre temas y áreas desde nuevas perspectivas, utilizando estas características de la investigación exploratoria se estudiara los efectos de emplear el modelo

de integración continua con el apoyo de BOTS para mejorar el desarrollo de software ágil bajo la metodología XP (Sampieri & Fernández Collado, 2006**).**

3.3. MÉTODO DE INVESTIGACIÓN

El presenta trabajo de titulación aplica el método de investigación deductivo:

Con este método se utiliza el razonamiento para obtener conclusiones que parten de hechos particulares aceptados como válidos, para llegar a conclusiones, cuya aplicación sea de carácter general. El método se inicia con un estudio individual de los hechos y se formulan conclusiones universales que se postulan como leyes, principios o fundamentos de una teoría (Torres & Augusto, 2006).

Contemplando la función que cumple este método de investigación se partirá del hecho de medir mediante el tiempo que una aplicación es probada en etapas de pruebas para analizar si se puede reducir el tiempo y mediante esta manera optimizar la productividad del desarrollador de software.

3.4. HERRAMIENTAS DE RECOLECCIÓN DE DATOS

Se lo puede definir como el proceso de vincular conceptos abstractos con indicadores empíricos mediante un plan explícito y organizado para poder clasificar los datos disponibles, en función del concepto del investigador. Un instrumento de medición adecuado es aquel que registra datos observables que representan verdaderamente los conceptos o las variables que el investigador tiene en mente (Gómez, 2006).

3.4.1. Observación

La observación consiste en el registro sistemático, valido y confiable de comportamientos o conductas manifiestas de los objetos de estudios

Puede Utilizarse como instrumentos de medición en muy diversas circunstancias. Es el método más usado por quienes están orientados a observaciones de la conducta. (Gómez, 2006)

3.4.1.1 **Definición de eventos a observar**

EVENTO A OBSERVAR	DEFINICIÓN
• **PRUEBAS UNITARIAS**	Se requiere observar las pruebas unitarias que se realizan cada vez que un grupo de desarrollo realice pruebas unitarias y las suba a un repositorio de código fuente lo cual se tendrá que bajar y observar los procesos de verificación de pruebas que se realicen en cada integración.
• **INTEGRACIONES CON UN REPOSITORIO DE CODIGO FUENTE**	Las integraciones se requieren realizar de manera continua mediante un plan o esquema previamente realizado.
• **DISPARADORES**	Estos disparadores representaran procesos de segundo plan que se ejecutan de manera muy parecida a lo que se entiende por disparadores en base de datos después de acciones específicas que se realicen dentro de la base de datos, en el caso de estos disparadores se harán mediante integraciones de manera que puede ser que se ejecuten antes o después.

Tabla 1: Definición de eventos

Fuente: Introducción a la metodología de la investigación científica

Elaborado por: Barberan León Julio Vicente

3.4.1.2 **Definición de aspectos**

EVENTO A OBSERVAR	DEFINICIÓN	ASPECTOS
• **PRUEBAS UNITARIAS**	Se requiere observar las pruebas unitarias que se realizan cada vez que un grupo de desarrollo realice pruebas unitarias y las suba a un repositorio de código fuente lo cual se tendrá que bajar y observar los procesos de verificación de pruebas que se realicen en cada integración.	• Clases • Vistas • Métodos
• **INTEGRACIONES CON UN REPOSITORIO DE CODIGO FUENTE**	Las integraciones se requieren realizar de manera continua mediante un plan o esquema previamente realizado.	• Automático • Manual
• **DISPARADORES**	Estos disparadores representaran procesos de segundo plan que se ejecutan de manera muy parecida a lo que se entiende por disparadores en base de datos después de acciones específicas que se realicen dentro de la base de datos, en el caso de estos disparadores se harán mediante integraciones de manera que puede ser que se ejecuten antes o después.	• Antes • Durante • Después

Tabla 2: Definición de aspectos

Fuente: Introducción a la metodología de la investigación científica

Elaborado por: Barberan León Julio Vicente

3.4.1.3 Definición de unidades de observación

Evento	Unidades	Esquema
Integraciones	• Horas • Minutos • Días	Cada integración se realizara de manera automática mediante un esquema de integraciones por hora lo cual realizara mediante un repositorio de código fuente

Tabla 3: Definición de unidades de observación

Fuente: Introducción a la metodología de la investigación científica

Elaborado por: Barberan León Julio Vicente

3.4.1.4 Hojas de codificación (Pruebas de software)

OBSERVADOR					
TIEMPO	**EVENTOS**				
APLICACIÓN MOVIL	NUMERO	PRUEBAS DE ÉXITO	PRUEBAS DE FALLO	INTEGRACIÓN MANUAL	INTEGRACIÓN AUTOMATICA
PRUEBAS UNITARIAS					
VISTAS					
METODOS					

Tabla 4: Hojas de codificación

Fuente: Introducción a la metodología de la investigación científica

Elaborado por: Barberan León Julio Vicente

3.4.1.5 **Hojas de codificación (simulación de dispositivos)**

OBSERVADOR							
	IPHONE 4S	IPHONE 5S	IPAD 2	IPAD RETINA	IPAD AIR	IPHONE 6	IPHONE 6 PLUS
CLASE (PRUEBA UNITARIA)							
TIEMPO							

Tabla 5: Hojas de codificación(dispositivos)

Fuente: Introducción a la metodología de la investigación científica

Elaborado por: Barberan León Julio Vicente

3.5. FUENTES DE INFORMACIÓN DE DATOS

Bajo la información que se ha considera de vital importancia para el presente trabajo se ha divido en 2 secciones una primaria como los datos que se han visto directamente en os eventos previamente estudiados y secundario como eventos desde otro contexto u observador lo haya realizado.

3.5.1. Fuentes de información de datos primaria

Corresponde a toda información directa que se ha recuperado o adquirido de primera mano de manera técnica realizando una recolección de datos como es la observación cuantitativa que se aplicó para adquirir este tipo de información para el presente trabajo.

3.5.2. Fuentes de información de datos secundaría

Corresponde a toda información que sirvió de base y apoyo las cuales están descritas en referencias bibliográficas acerca de tecnologías sobre el modelo informático de integración continua bajo automatización de código fuente en repositorios que permitan tal técnica y sean factibles a este trabajo

3.6 INSTRUMENTAL OPERACIONAL

Bajo una perspectiva cuantitativa, la recolección de datos es equivalente a medir. De acuerdo con definiciones el término es ampliamente difundido como la asignación de números a diversos eventos o fenómenos que se realicen en etapas de recolección de datos por consiguiente permiten evaluar de manera precisa.

3.6.1. Estructura y características de lo(s) instrumento(s) de recolección de datos

La observación consiste en el registro sistemático, valido, y confiable de comportamientos o conductas manifiestas de los objetos de estudio. Puede utilizarse como instrumento de medición es muy diversas circunstancias.

Evento	Definición
Definición de la precisión de aspectos, eventos o conductos a observar	Se compone como el conjunto de comportamientos o eventos a investigar y desarrollar una síntesis de un estudio.
Extraer una muestra representativa de aspectos, eventos o conductas a observar	Representa las características asociadas a los eventos que se han de estudiar.
Establecer y definir las unidades de observación	Se definen los intervalos en que se realizara los estudios de los eventos categorizando las respectivas unidades las cuales representaran el suceso en que se podrán diferenciar.
Establecer y definir las categorías y subcategorías de la observación	Se establece la posible cadena de eventos después de los sucesos a estudiar y establecer cuáles de esos eventos serán esenciales para la progresión de la investigación.
Selección de los observadores	Se establece el reportorio de observadores que realizaran las respectivas observaciones de los eventos
Elegir el medio de observación	Representará las condiciones en que se encuentre el medio para el posible estudio siempre que sea factible
Elaborar las hojas de codificación	Representan la hoja de ruta y como el observador registrara los eventos a estudiar

Tabla 6: Estructura y características de los instrumento de recolección de datos

Fuente: Introducción a la metodología de la investigación científica

Elaborado por: Barberan León Julio Vicente

3.7 ESTRATEGIA OPERACIONAL PARA LA RECOLECCIÓN Y TABULACIÓN DE DATOS

Se aplicó la técnica de recolección de datos basada en la observación cuantitativa directa apoyándose en todos los eventos que se pudieron captar mediante la investigación de la misma y poder interpretarlos mediante métodos de investigación para poder extraer todas las características más distintivas del presente trabajo.

3.7.1. Plan de recolección

Bajo el presente trabajo se captó y recolecto el mayor número de variables directas e indirectas que pueda estar orientado al tema de automatización de procesos bajo el modelo informático de integración continua dentro de los procesos que se tienen contemplados bajo el presente modelo se ha tomado en cuento los posibles errores en tiempo de ejecución que se han podido presentar.

En la presente investigación se realizó bajo una metodología orientada descriptiva y exploratoria la cual se procedió a obtener el mayor contenido de información posible para posterior armar una base fija que me permita utilizar la metodología exploratoria para analizar los procesos que se puedan dar.

El proceso de ejecución que se llevó bajo la técnica de recolección basada en la observación cuantitativa permitió formar una base tanto contextual como numérica de los eventos que sucedieron en el tiempo que se desarrolló el presente trabajo y captar como procesos que se han dado de manera manual se pueden transformar de manera mucho más eficaz en procesos automáticos.

3.7.2. Plan de tabulación

Mediante la utilización de la estadística probabilística se desarrolló el presente punto captando toda la información posible uniéndola con frecuencias absolutas y relativas para los eventos observados y dependiendo de los posibles resultados arrojados, se pasó elaborar las frecuencias acumuladas para conocer el porcentaje sucesos y fallos que se presente en el presente trabajo.

3.7.3. Análisis e interpretación de los datos

Se basó este punto con la utilización de gráficos estadísticos mediante barras representando los sucesos y fallos que se realizaron en la etapa de observación y midiendo la eficiencia que tiene el modelo de integración continua con los tiempos que se demoró en el suceso de tiempos, Ver (3.9.2).

3.8 PLAN DE MUESTREO

Para el presente punto se analizó todos los aspectos cualitativos que se tomaron de las observaciones cuantitativas muéstrales, con esto me permite exponer el tamaño maestral de manera más exacta y correcta y añadiendo credibilidad fiabilidad y credibilidad en los resultados estadísticos y probabilísticos basado en la presente formula:

$$n = \frac{k^2 pqN}{(e^2(N-1)) + k^2 pq}$$

$$n = \frac{1{,}95^2(0{,}9)(0{,}1)(12)}{(0{,}05^2(12-1)) + 1{,}95^2(0{,}9)(0{,}1)}$$

$$n = 11{,}10$$

Variable	*Definición*	*Valor*	*Relación*
N	Tamaño de la población o de la muestra	12	Una aplicación constituida mediante 2 clases para pruebas unitarias con un total de 12 métodos para pruebas.
k	Nivel de confianza	1,95	Se define como una constante que funciona de acuerdo al nivel de confianza que el observador le brinda. Lo que nos da un 95% con un 0,95 más un 100% que presenta la unidad equivalente a un 1,95.
e	Error muestral deseado	0,05	El error muestral representa la falta de diferencia de la constante K para lo cual del 100% de confiablidad que se da se obtiene un error muestral 0,05.
p	Proporción de observaciones conceptuales	0,9	Representa el tamaño de las observaciones que poseen las características propias de las observaciones que se produjeron en cada evento.
q	Proporción de observaciones fijas	0,1	Representa el tamaño de las observaciones que no poseen las características propias de las observaciones que se produjeron en cada evento.
n	Tamaño de la muestra		Representa el resultado de la fórmula que se presenta en este punto

Tabla 7: Definición de variables bajo el plan de muestreo

Fuente: Introducción a la metodología de la investigación científica

3.8.1. Segmentación

En el presente punto se detalla mediante las hojas de codificación que se utilizaron para realizar las presentes observaciones para el respectivo estudio de los eventos que se plantearon en el transcurso del desarrollo del presente trabajo.

3.8.2. Técnica de muestreo

Para este trabajo se utilizó la técnica de muestreo intencional ya que como el observador selecciona una muestra y trata que esta sea de la manera más representativa posible para la propuesta planteada y de acuerdo a criterio que este sujeto puede llegar hacer subjetivo y representativo.

Implica seleccionar una muestra considerada como típica o representativa de la población tanto el muestreo intencional como el casual están sujetos al sesgo del investigador y, lo que es más importante, conducen a estimadores cuyas propiedades no pueden evaluarse. Por tanto, ninguna de estas técnicas genera una muestra aleatoria. (Scheaffer & Mendenhall)

3.8.3. Tamaño de la muestra

Representa "la totalidad de fenómenos a estudiar en donde las unidades poseen una característica común, la cual se estudia y da origen a los datos de la investigación".

Basado en lo anterior expuesto puedo establecer que el tamaño de la muestra va estar orientado en una aplicación móvil la cual será testada mediante los archivos que estén compuesto y evaluar cualquier comportamiento que surjan en cada observación.

3.9 PRESENTACIÓN Y ANÁLISIS DE LOS RESULTADOS

En el presente trabajo se realizó el estudio bajo una aplicación móvil empleado el uso de BOTS para las presentes pruebas y resultados que se obtuvieron empleando esta técnica.

3.9.1. Presentación y descripción de los resultados obtenidos

- Integración 1

OBSERVADOR	JULIO BARBERAN LEÓN		
OBJETOS A EVALUAR TIEMPO 0,40 SEG	EVENTOS		
APLICACIÓN MOVIL	TIEMPO	ÉXITO	FALLO
Commit	0,00 SEG		
Push	0,00 SEG		
Pull	0,20 SEG	✓	
Verificación de pruebas	0,20 SEG	✓	

Tabla 8: Resultado de I.C.M 1

Fuente: Introducción a la metodología de la investigación científica

Elaborado por: Barberan León Julio Vicente

- Integración 2

OBSERVADOR	JULIO BARBERAN LEÓN		
OBJETOS A EVALUAR TIEMPO 0,30 SEG	EVENTOS		
APLICACIÓN MOVIL	TIEMPO	ÉXITO	FALLO
Commit	0,00 SEG		
Push	0,00 SEG		
Pull	0,20 SEG	✓	
Verificación de pruebas	0,10 SEG	✓	

Tabla 8: Resultado I.C.M 2

Fuente: Introducción a la metodología de la investigación científica

Elaborado por: Barberan León Julio Vicente

- Integración 3

OBSERVADOR	JULIO BARBERAN LEÓN		
OBJETOS A EVALUAR TIEMPO 0,65	EVENTOS		
APLICACIÓN MOVIL	TIEMPO	ÉXITO	FALLO
Commit	0,15 SEG	✓	
Push	0,20 SEG	✓	
Pull	0,20 SEG	✓	
Verificación de pruebas	0,10 SEG	✓	

Tabla 9: Resultado I.C.M 3

Fuente: Introducción a la metodología de la investigación científica

Elaborado por: Barberan León Julio Vicente

- Integración 4

OBSERVADOR	JULIO BARBERAN LEÓN		
OBJETOS A EVALUAR TIEMPO 0,65	EVENTOS		
APLICACIÓN MOVIL	TIEMPO	ÉXITO	FALLO
Commit	0,15 SEG	✓	
Push	0,20 SEG	✓	
Pull	0,20 SEG	✓	
Verificación de pruebas	0,10 SEG	✓	

Tabla 10: Resultado I.C.M 4

Fuente: Introducción a la metodología de la investigación científica

Elaborado por: Barberan León Julio Vicente

- Integración 5

OBSERVADOR	JULIO BARBERAN LEÓN		
OBJETOS A EVALUAR TIEMPO 0,85	EVENTOS		
APLICACIÓN MOVIL	TIEMPO	ÉXITO	FALLO
Commit	0,15 SEG	✓	
Push	0,20 SEG	✓	
Pull	0,20 SEG	✓	
Verificación de pruebas	0,30 SEG	✓	

Tabla 11: Resultado I.C.M 5

Fuente: Introducción a la metodología de la investigación científica

Elaborado por: Barberan León Julio Vicente

- Integración 6

OBSERVADOR	JULIO BARBERAN LEÓN		
OBJETOS A EVALUAR TIEMPO 2 MIN – 15 SEG	EVENTOS		
APLICACIÓN MOVIL	TIEMPO	ÉXITO	FALLO
Commit	0,15 SEG	✓	
Push	0,20 SEG	✓	
Pull	0,20 SEG	✓	
Verificación de pruebas	1 MIN – 20 SEG	✓	

Tabla 12: Resultado I.C.M 6

Fuente: Introducción a la metodología de la investigación científica

Elaborado por: Barberan León Julio Vicente

- Integración 7

OBSERVADOR	JULIO BARBERAN LEÓN		
OBJETOS A EVALUAR TIEMPO 2 MIN – 15 SEG	EVENTOS		
APLICACIÓN MOVIL	TIEMPO	ÉXITO	FALLO
Commit	0,15 SEG	✓	
Push	0,20 SEG	✓	
Pull	0,20 SEG	✓	
Verificación de pruebas	1 MIN – 20 SEG	✓	

Tabla 13: Resultado I.C.M 7

Fuente: Introducción a la metodología de la investigación científica

Elaborado por: Barberan León Julio Vicente

- Integración 8

OBSERVADOR	JULIO BARBERAN LEÓN		
OBJETOS A EVALUAR TIEMPO 2 MIN – 15 SEG	EVENTOS		
APLICACIÓN MOVIL	TIEMPO	ÉXITO	FALLO
Commit	0,15 SEG	✓	
Push	0,20 SEG	✓	
Pull	0,20 SEG	✓	
Verificación de pruebas	1 MIN – 20 SEG	✓	

Tabla 14: Resultado I.C.M 8

Fuente: Introducción a la metodología de la investigación científica

Elaborado por: Barberan León Julio Vicente

- Integración 9

OBSERVADOR	JULIO BARBERAN LEÓN		
OBJETOS A EVALUAR TIEMPO 2 MIN – 15 SEG	EVENTOS		
APLICACIÓN MOVIL	TIEMPO	ÉXITO	FALLO
Commit	0,15 SEG	✓	
Push	0,20 SEG	✓	
Pull	0,20 SEG	✓	
Verificación de pruebas	1 MIN – 20 SEG	✓	

Tabla 15: Resultado I.C.M 9

Fuente: Introducción a la metodología de la investigación científica

Elaborado por: Barberan León Julio Vicente

3.9.2. Informe final del análisis de los resultados

TIPO APLICACIÓN	TIEMPO	OPERACIÓN
MOVIL SWIFT	40 segundos	• Pull • Verificación de pruebas
	30 segundos	• Pull • Verificación de pruebas
	1 minuto 5 segundos	• Pull • Verificación de pruebas • Push • Commit
	1 minutos 5 segundos	• Pull • Verificación de pruebas • Push • Commit
	1 minutos 25 segundos	• Pull • Verificación de pruebas • Push • Commit
	2 minutos 15 segundos	• Pull • Verificación de pruebas • Push • Commit
	2 minutos 15 segundos	• Pull • Verificación de pruebas • Push • Commit

	2 minutos 15 segundos	• Pull • Verificación de pruebas • Push • Commit
	2 minutos 15 segundos	• Pull • Verificación de pruebas • Push • Commit

Tabla 16: Análisis de resultados I.C.M

Fuente: Introducción a la metodología de la investigación científica

Elaborado por: Barberan León Julio Vicente

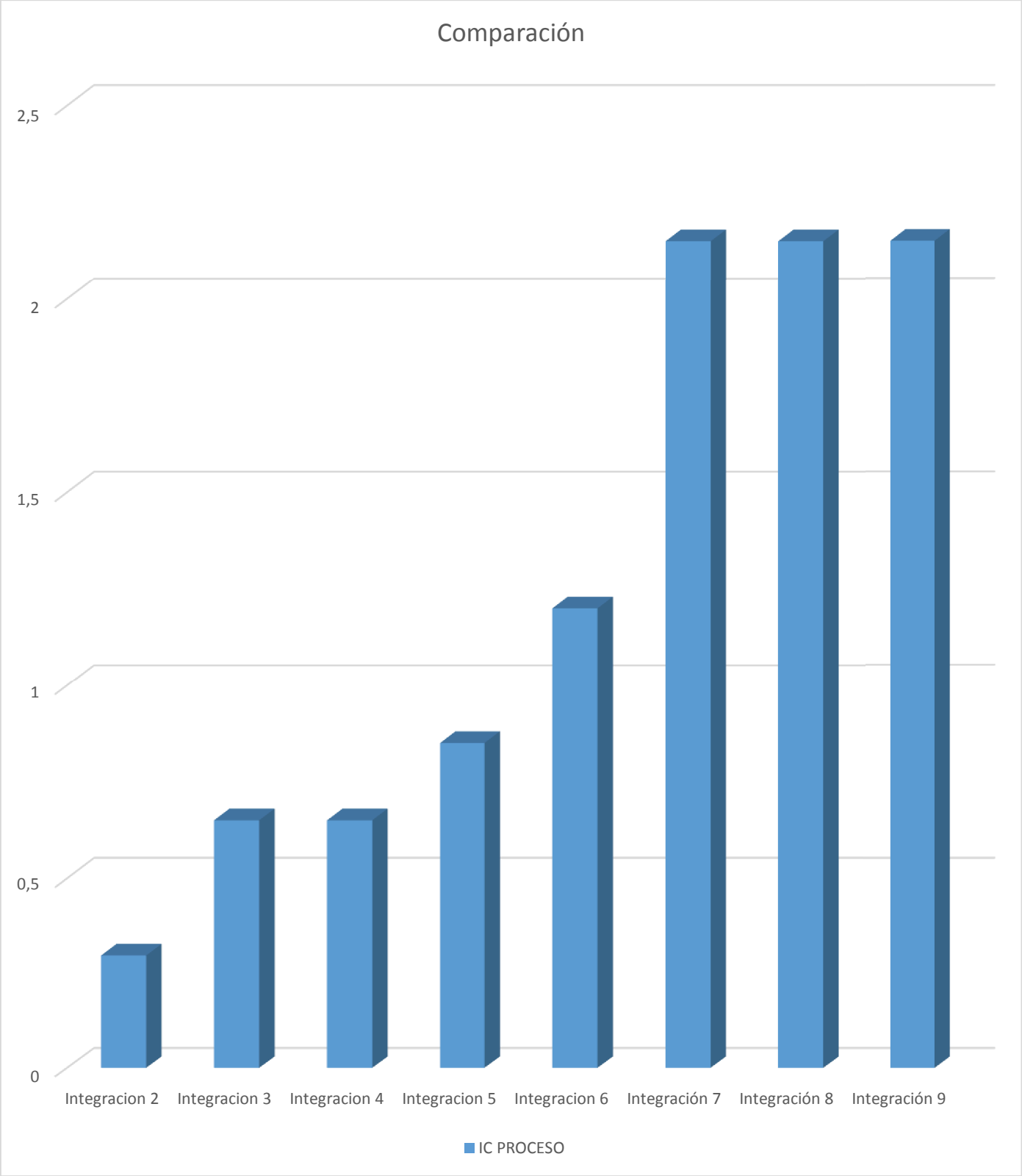

Ilustración 11: Comparativa de integraciones continuas

Fuente: Informe final del análisis de los resultados (3.9.2)

Elaborado por: Barberan León Julio Vicente

CAPITULO IV

DISEÑO DE LA PROPUESTA

4.1. INTRODUCCIÓN

En el actual capítulo se presentara los pasos que se realizaron para elaborar el presente trabajo de titulación siguiendo una metodología de software para probar su eficiencia mediante las mejoras de integración continua y comparar el tiempo que el grupo de desarrollo se demoraría haciendo pruebas mediante la implementación de esta tecnología que utiliza procesos de segundo planos para estar probando y optimizando el código fuente desde un repositorio de código fuente y servidor central para alojar estos procesos y presentarlos en forma de reportes mediante un entorno web o de una manera mucho más detalla mediante su entorno integrado de desarrollo.

4.2. DESCRIPCIÓN DE LA PROPUESTA

La presente propuesta brindara a los desarrolladores una nueva forma para que en etapas de producción bajo una metodología de software ágil las pruebas que se realicen al código se hagan de una manera automática reduciendo cargas y haciendo el desarrollo mucho más admisible.

Mediante la tecnología a utilizar se aumentará la eficiencia de desarrollo en la etapa de pruebas cumpliendo con los objetivos que se generan mediante este modelo informático mejorando la calidad y fiabilidad del software, con diferentes formas de lograrlo:

Los problemas o errores que se presentan en la colaboración del grupo de trabajo que puedan crear estos procesos de segundo plano mediante disparadores a cualquier evento que suceda durante la integración y de esta manera detectando cualquier fallo y notificándolo al equipo de trabajo de quien cometió el error.

Cuando la ejecución de pruebas se lo realiza en un repositorio local y se necesita probar la aplicación en varios dispositivos con distintas

configuraciones el proceso se vuelve arduo y por consiguiente consume mucho tiempo, pero utilizando procesos en segundo plano (BOTS) el flujo de integración continúa lo resuelve mediante la programación de estos procesos para volverlo automático

La determinación de gráficos estadísticos para evaluar la evolución del software y conocer detalladamente las personas que colaboraron en el trascurso de implementación bajo el modelo de integración continua.

Mediante esta propuesta se reducirá el tiempo en que el desarrollador en etapas de producción del software necesite comprobar o verificar ciertas partes o la totalidad de la aplicación haciendo mucho más productivo la forma en que desarrolla.

4.3 ANÁLISIS DE LAS TECNOLOGÍAS DE APLICACIÓN

Tecnologías	Descripción	Análisis
• **IDE: XCODE 6.0** • **LENGUAJE DE PROGRAMACIÓN: SWIFT**	Diseño y construcción de la aplicación móvil y pruebas unitarias.	Conexión como un servicio interno dentro de OSX SERVER para el almacenamiento de repositorios y procesos de segundo plano "BOTS".
• **OSX SERVER 3.0**	Programa de servidor que permite implementar repositorios de código fuente para el alojamiento de BOTS y configuración vía WEB.	Permite integrar varios servicios conectándolos con el repositorios vía SSH
• **FRAMEWORKS:** ○ **MAP KIT**	Permite anotar mapas a entornos móviles como web incrustados de distintas maneras o vistas para la visualización	Mediante MAP KIT se realiza la incrustación de mapas mediante mapas de modelado móvil
• **REPOSITORIOS:** ○ **GITHUB** ○ **BITBUCKET**	Repositorio de alojamiento de forma pública o privada de código fuente mediante desarrollos colaborativos se software con un sistema de control de versiones	Estos repositorios se conectan al IDE y servidor para ser el canal de flujo de pruebas y administración del código que se almacena en el repositorio

Tabla 17: Análisis de las tecnologías de aplicación

Fuente: Introducción a la metodología de la investigación científica

4.4 ETAPAS DE LA PROPUESTA

La propuesta se desarrolló en base a requerimientos funcionales tanto para la parte del desarrollo del software a implementar utilizando la metodología de desarrollo de software programación extrema o XP, así como la parte del desarrollo para la creación de los procesos de segundo plano o "BOTS" que estarían ejecutándose en el servidor que realizará la administración de código, ya que estos son las prestaciones o funciones que brindará el sistema como el servidor, por otra parte se tendrán los requerimientos que no están vinculados directamente , sino a la parte lógica que se implementó para cumplir con los objetivos que se planean en los requisitos funcionales tanto para el desarrollo de software como para la implementaciones en el servidor.

4.4.1. Plan de entrega

La presente planificación basada en las historias que se realizan al principio del proyecto después del estudio del proyecto a emprender mediante los requerimientos de los usuarios mediante esto se establece una planificación que a lo largo del desarrollo van cambiando o adaptándose de acuerdo a cada iteración que se realiza con la idea de que se vaya acoplando a los objetivos que el cliente quiere.

4.4.2. Historias de usuario

Historia de Usuario -01	
Nombre historia: Vista de un mapa geográfico usando la ubicación actual	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** (Alta / Media / Baja)
Puntos estimados: 3	**Iteración asignada: 1**
Programador responsable: BARBERAN LEÓN JULIO	
Descripción: **Cualquier iteración con el mapa se lo debe realizar usado la ubicación actual del cliente y de acuerdo a eso podrá efectuar cualquier operación sobre la misma.**	
Observaciones: ***CONFIRMADO con el cliente***	

Tabla 18: Historia de usuario -01

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

Historia de Usuario -02	
Nombre historia: Ingreso de datos para una ubicación	
Prioridad en negocio: Media	**Riesgo en desarrollo:** (Alta / Media / Baja)
Puntos estimados: 2	**Iteración asignada: 1**
Programador responsable: BARBERAN LEÓN JULIO	
Descripción: **Para el almacenamiento de una ubicación geográfica sobre el mapa se necesitará de 2 campos con los cuales se procede a guardar y sobre todo la posibilidad para poder ubicar la ubicación moviéndolo a disposición del cliente:** • **Radio de longitud** • **Entradas** • **Salidas** • **Nota** • **Iniciando la aplicación mostrando la ubicación actual de manera geográfica** • **Mostrando las ubicaciones en una viñeta y de manera geográfica**	
Observaciones: ***CONFIRMADO con el cliente***	

Tabla 19: Historia de usuario -02

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

Historia de Usuario -03	
Nombre historia: La aplicación abarque radios de longitud	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** (Alta / Media / Baja)
Puntos estimados: 1	**Iteración asignada: 1**
Programador responsable: BARBERAN LEÓN JULIO	
Descripción: • **Un dato que se necesitara indispensable al momento de almacenar una ubicación será el radio de longitud**	
Observaciones: ***CONFIRMADO con el cliente***	

Tabla 20: Historia de usuario -03

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

Historia de Usuario -04	
Nombre historia: Visualización geográfica de los datos	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** (Alta / Media / Baja)
Puntos estimados: 1	**Iteración asignada: 1**
Programador responsable: BARBERAN LEÓN JULIO	
Descripción: • **Cada ubicación será visualizada de manera geográfica mediante la implementación de MAP KIT y CORE DATA para el almacenamiento y carga de datos**	
Observaciones: ***CONFIRMADO con el cliente***	

Tabla 21: Historia de usuario -04

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

Historia de Usuario -05	
Nombre historia: Marcado geográfico de ubicaciones almacenadas	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** (Alta / Media / Baja)
Puntos estimados: 1	**Iteración asignada: 2**
Programador responsable: BARBERAN LEÓN JULIO	
Descripción: • **Cada ubicación tendrá un marcado donde se desplegara los datos que contienen**	
Observaciones: ***CONFIRMADO con el cliente***	

Tabla 22: Historia de usuario -05

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

Historia de Usuario -06	
Nombre historia: La aplicación me permita dar una ubicación	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** (Alta / Media / Baja)
Puntos estimados: 3	**Iteración asignada: 2**
Programador responsable: BARBERAN LEÓN JULIO	
Descripción: • **La aplicación podrá implementar mediante un ruta singular un punto geográfico referenciado e implementado desde el código fuente**	
Observaciones: ***CONFIRMADO con el cliente***	

Tabla 23: Historia de usuario -06

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

Historia de Usuario -07	
Nombre historia: Selección de marcado geográfico	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** (Alta / Media / Baja)
Puntos estimados: 1	**Iteración asignada: 1**
Programador responsable: BARBERAN LEÓN JULIO	
Descripción: • **Cada ubicación podrá ser eliminado o ver su radio de longitud y notas seleccionando el marcado de ubicación que se implementara en el mapa**	
Observaciones: ***CONFIRMADO con el cliente***	

Tabla 24: Historia de usuario -07

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

Historia de Usuario -08	
Nombre historia: Almacenar de la información contenida de la ubicación	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** (Alta / Media / Baja)
Puntos estimados: 2	**Iteración asignada: 2**
Programador responsable: BARBERAN LEÓN JULIO	
Descripción: • **Cada ubicación será almacenado dentro de un CORE DATA abriendo la pestaña de añadir ubicación**	
Observaciones: ***CONFIRMADO con el cliente***	

Tabla 25: Historia de usuario -08

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

Historia de Usuario -09	
Nombre historia: Eliminación de la información contenida de la ubicación	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** (Alta / Media / Baja)
Puntos estimados: 3	**Iteración asignada: 2**
Programador responsable: BARBERAN LEÓN JULIO	
Descripción: • **Cada ubicación podrá ser eliminada del CORE DATA abriendo la ubicación de marcado**	
Observaciones: ***CONFIRMADO con el cliente***	

Tabla 26: Historia de usuario -09

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

Nº	Nombre	prioridad	Riesgo	Esfuerzo	Iteración
01	**Vista de un mapa geográfico usando la ubicación actual**	Alta	Medio	2	1
02	**Ingreso de datos para una ubicación**	Medio	Medio	2	1
03	**La aplicación abarque radios de longitud**	Bajo	Bajo	1	1
04	**Visualización geográfica de los datos**	Bajo	Bajo	1	1
05	**Marcado geográfico de ubicaciones almacenadas**	Bajo	Bajo	1	2
06	**La aplicación me permita dar una ubicación**	Alta	Alta	3	2
07	**Selección de marcado geográfico**	Bajo	Bajo	1	2
08	**Almacenar de la información contenida de la ubicación**	Alta	Medio	2	3
09	**Eliminación de la información contenida de la ubicación**	Alta	Bajo	3	3

Tabla 27: Asignación de tarea

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

4.4.3. Plan de entrega

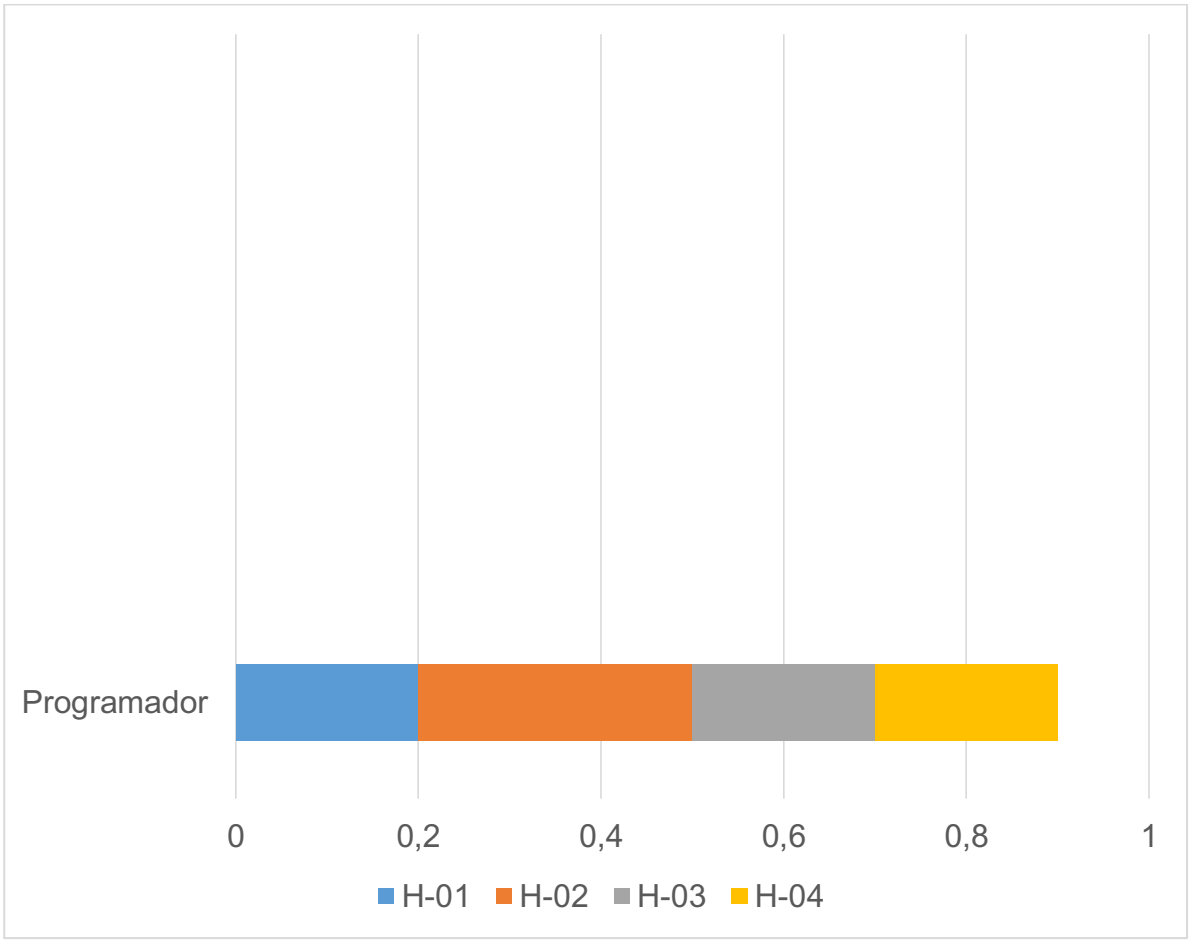

Ilustración 12: Plan de entrega 1

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

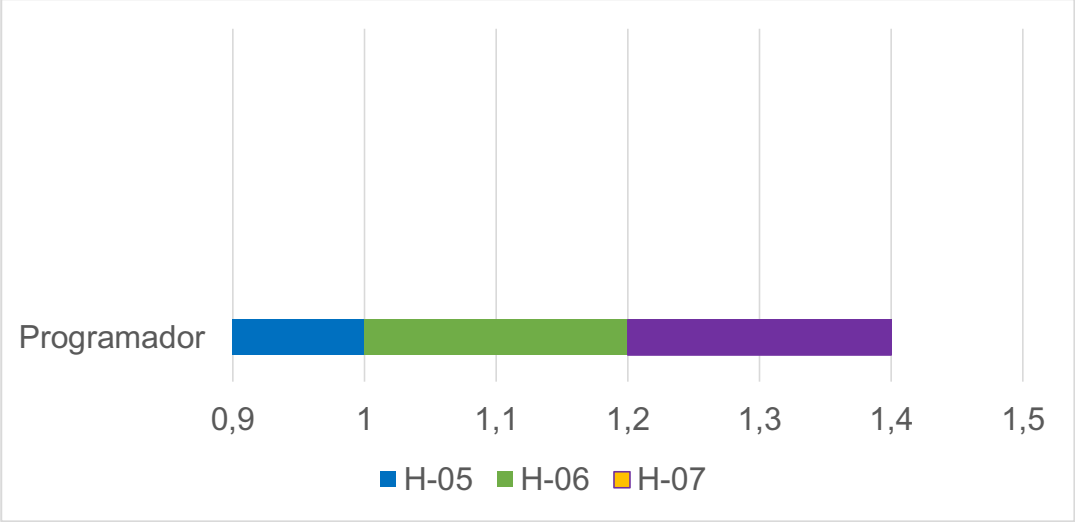

Ilustración 13: Plan de entrega 2

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

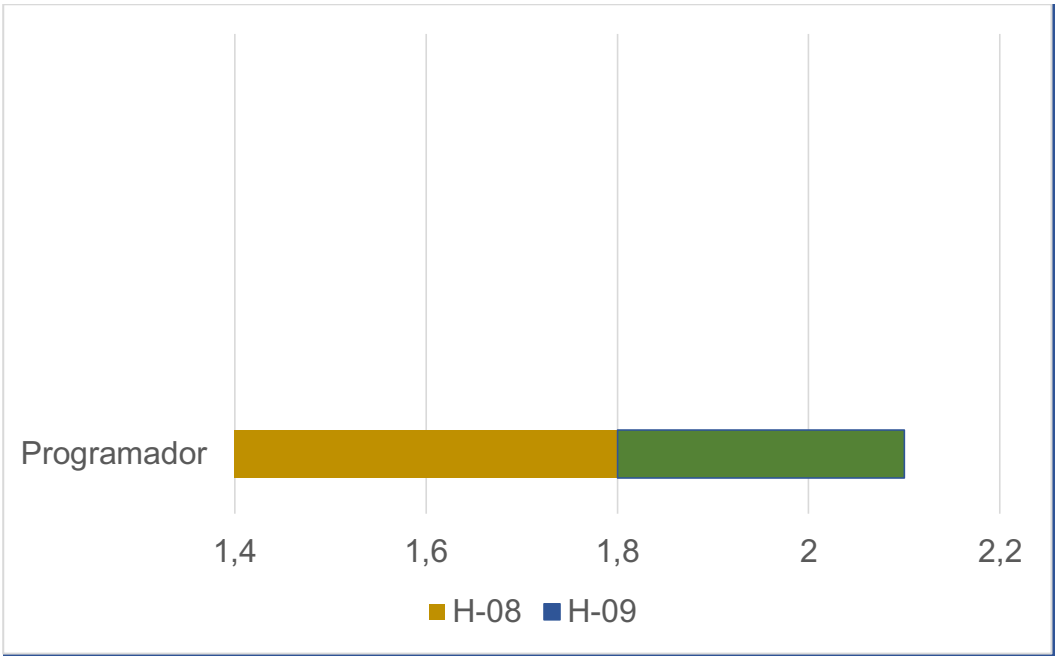

Ilustración 14: Plan de entrega 3

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

4.4.4. Iteración 1

En cada iteración se va a presentar un registro de las historias de usuarios que se manejen en la presente iteración junto con cada uno de sus pruebas funcionales, recortes de pantalla.

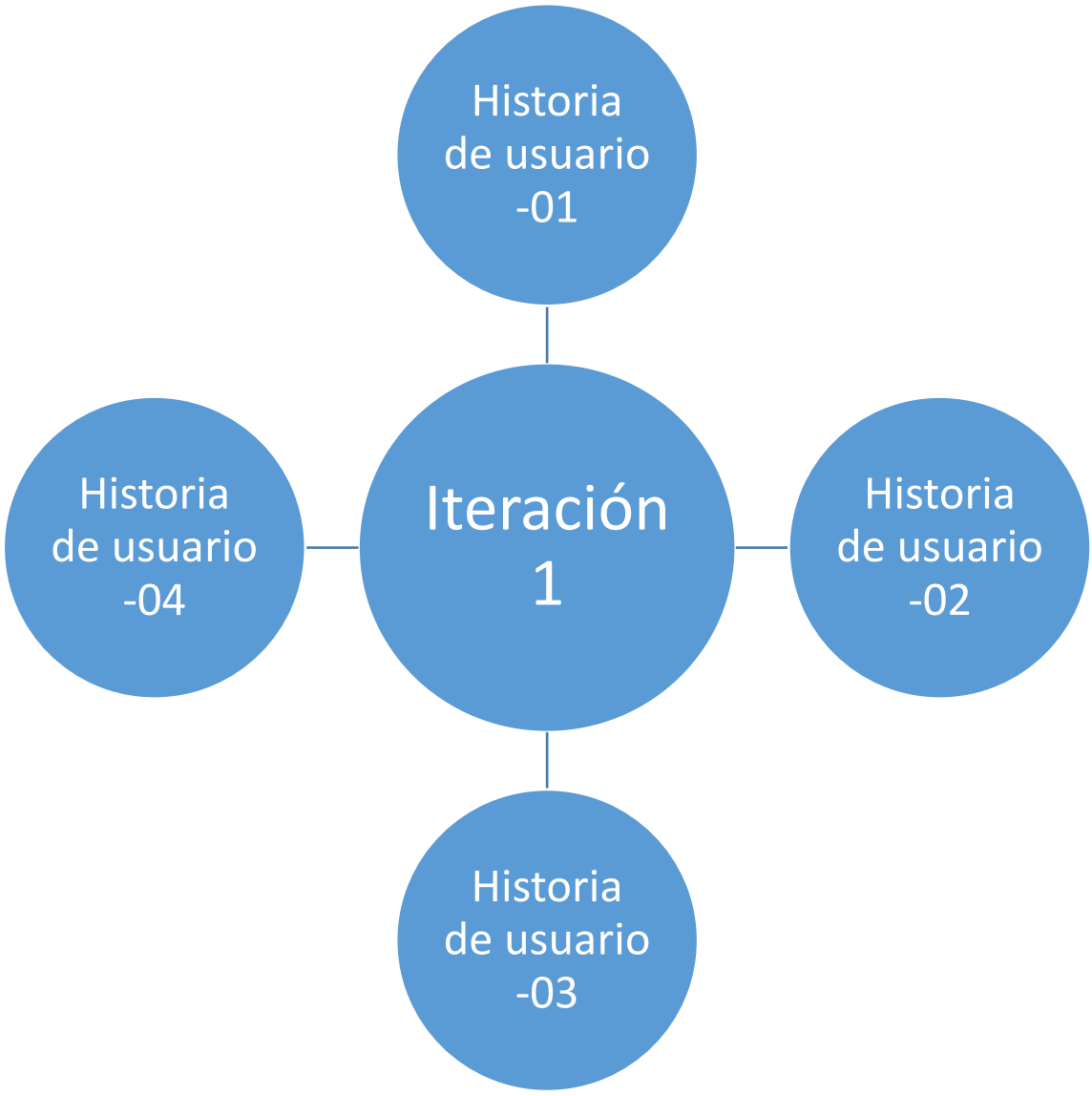

Ilustración 15: Relación Iteración 1 con historias de usuario

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

4.4.4.1. Tareas de las historias

	Tarea
Número tarea: 1	**Número historia: 01**
Nombre tarea: Creación del modelo de datos (CORE DATA)	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 20 Mayo del 2015	**Fecha fin: 21 Mayo del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Creación de los campos necesarios que almacenara el CORE DATA dentro de la estructura a desarrollar	

Tabla 28: Tarea de historia -01

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

	Tarea
Número tarea: 2	**Número historia: 01**
Nombre tarea: Creación del Navegador y controlador para la vista geográfica	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 21 Mayo del 2015	**Fecha fin: 23 Mayo del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Creación del controlador y navegador que cargara las funciones que se enlazan con la vista del mapa que se cargue inicialmente.	

Tabla 29: Tarea de historia -02

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

	Tarea
Número tarea: 3	**Número historia: 01**
Nombre tarea: Creación de la vista y enlace con el controlador	
Tipo de tarea: Diseño	**Puntos estimados: 1**
Fecha inicio: 23 Mayo del 2015	**Fecha fin: 25 Mayo del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Creación de la vista utilizando MAP KIT y enlenzándola con el controlador a cada control añadido en la vista.	

Tabla 30: Tarea de historia -03

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

	Tarea
Número tarea: 4	**Número historia: 02**
Nombre tarea: Creación del segundo controlador y navegador para el ingreso de datos al CORE DATA	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 25 Mayo del 2015	**Fecha fin: 26 Mayo del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Creación del segundo controlador y navegador ya que este se enlaza con la viñeta de añadir que se añade en la primera vista ya que este controlador se compone del modelo de datos del CORE DATA: • Radio de longitud • Entradas • Salidas • Nota	

Tabla 31: Tarea de historia -04

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

	Tarea
Número tarea: 5	**Número historia: 02**
Nombre tarea: Creación de la vista y enlace con el controlador	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 26 Mayo del 2015	**Fecha fin: 26 Mayo del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Creación de la vista utilizando MAP KIT y enlenzándola con el controlador a cada control añadido en la vista.	

Tabla 32: Tarea de historia -05

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

	Tarea
Número tarea: 6	**Número historia: 02**
Nombre tarea: Creación y desarrollo de las operaciones de enlace entre los 2 controladores y operaciones de carga de las vistas	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 26 Mayo del 2015	**Fecha fin: 27 Mayo del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Esta clase servirá como enlace hacia el CORE DATA y replicará en las vistas todo tipo de acción que se realizará tanto sobre el mapa y el CORE DATA	

Tabla 33: Tarea de historia -06

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

Tarea	
Número tarea: 8	**Número historia: 04**
Nombre tarea: Creación de objetos estáticos para la carga de ubicación de datos	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 28 Mayo del 2015	**Fecha fin: 28 Mayo del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Se crearon datos estáticos que sin necesidad de llamar a datos contenidos dentro de un CORE DATA se puedan visualizar por el momento de manera estática	

Tabla 34: Tarea de historia -08

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

Tarea	
Número tarea: 9	**Número historia: 04**
Nombre tarea: Carga de ubicaciones (estáticas) en el mapa	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 28 Mayo del 2015	**Fecha fin: 29 Mayo del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Se creó un evento de carga para poder visualizar al momento de abrir la aplicación los datos que se encuentran dentro de la clase de cálculo para poderlos visualizar	

Tabla 35: Tarea de historia -09

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

4.4.4.2. Demos de las historias

- Historias 1-2

Ilustración 16: Demo de historias 1-2

Fuente: Core Location Framework Reference - Apple Developer

Elaborado por: Barberan León Julio Vicente

- Historia 3

Ilustración 17: Demo de historia 3

Fuente: XCode - Apple Developer

Elaborado por: Barberan León Julio Vicente

- **Historia 4**

Ilustración 18: Demo de historia 4

Fuente: Core Location Framework Reference - Apple Developer

Elaborado por: Barberan León Julio Vicente

4.4.4.3. Pruebas de aceptación

- Historia 1
  - La carga de la ubicación actual usando MAP KIT se realizó de manera correcta produciendo la vista deseada por el cliente
- Historia 2-3-4
  - La vista que se enlaza con los campos que producirán el almacenamiento de la ubicación tomando un radio de longitud y una nota se realizó de manera correcta.
  - La carga de datos fue correcta mediante los parámetros ingresados se pudo visualizar de manera correcta

4.4.5. Iteración 2

Ilustración 19: Relación iteración 2 con historias de usuario

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

4.4.5.1. Tareas de las historias

	Tarea
Número tarea: 10	**Número historia: 05**
Nombre tarea: Añadir la ubicación al marcado del mapa	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 28 Mayo del 2015	**Fecha fin: 29 Mayo del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Cada ubicación (Clase) se añadirá al controlador el cual lo presentará de manera geográfica en el mapa	

Tabla 36: Tarea de historia 10

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

	Tarea
Número tarea: 11	**Número historia: 05**
Nombre tarea: Carga de marcado al mapa	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 29 Mayo del 2015	**Fecha fin: 30 Mayo del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: La carga de las ubicaciones se las realizara añadiendo al constructor los parámetros que se componen con la clase que realiza el cálculo y crea el objeto y añado en el evento de cargo de la vista del mapa el nuevo objeto (marcado) para su visualización.	

Tabla 37: Tarea de historia -11

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

	Tarea
Número tarea: 12	**Número historia: 06**
Nombre tarea: Añadir la ubicación GPS al marcado del mapa	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 30 Mayo del 2015	**Fecha fin: 1 Junio del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Cada ubicación (Clase) sea modo GPS, tramo de ubicaciones, o una simple ubicación se añadirá al controlador el cual lo presentara de manera geográfica en el mapa	

Tabla 38: Tarea de historia -12

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

	Tarea
Número tarea: 13	**Número historia: 06**
Nombre tarea: Carga de ubicación predefinida en modo en GPS	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 1 Junio del 2015	**Fecha fin: 3 Junio del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Las diferentes formas que se pueden presentar mediante MAP KIT están como ubicaciones fijas, rutas de 2 o más puntos y GPS y se procedió a cargar añadiendo un delegado al método cargar de la vista del mapa para las ubicaciones y en modo GPS	

Tabla 39: Tarea de historia -13

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

	Tarea
Número tarea: 14	**Número historia: 07**
Nombre tarea: Añadido de evento para la selección de marcaje	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 3 Junio del 2015	**Fecha fin: 3 Junio del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Cada vez que se seleccione una ubicación geográfica se desplegara la información contenida de dicha ubicación y el evento de eliminar añadido en forma de un icono	

Tabla 40: Tarea de historia -14

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

	Tarea
Número tarea: 15	**Número historia: 07**
Nombre tarea: Añadido de notación al marcaje de ubicación	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 3 Junio del 2015	**Fecha fin: 3 Junio del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Se desplegará al momento de hacer clic sobre cualquier ubicación las opciones de vista información de la ubicación y la opción de eliminación	

Tabla 41: Tarea de historia -15

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

4.4.5.2. Demos de las historias

- Historia 5

Ilustración 20: Demo de historia 5

Fuente: Core Location Framework Reference - Apple Developer

Elaborado por: Barberan León Julio Vicente

- Historia 6

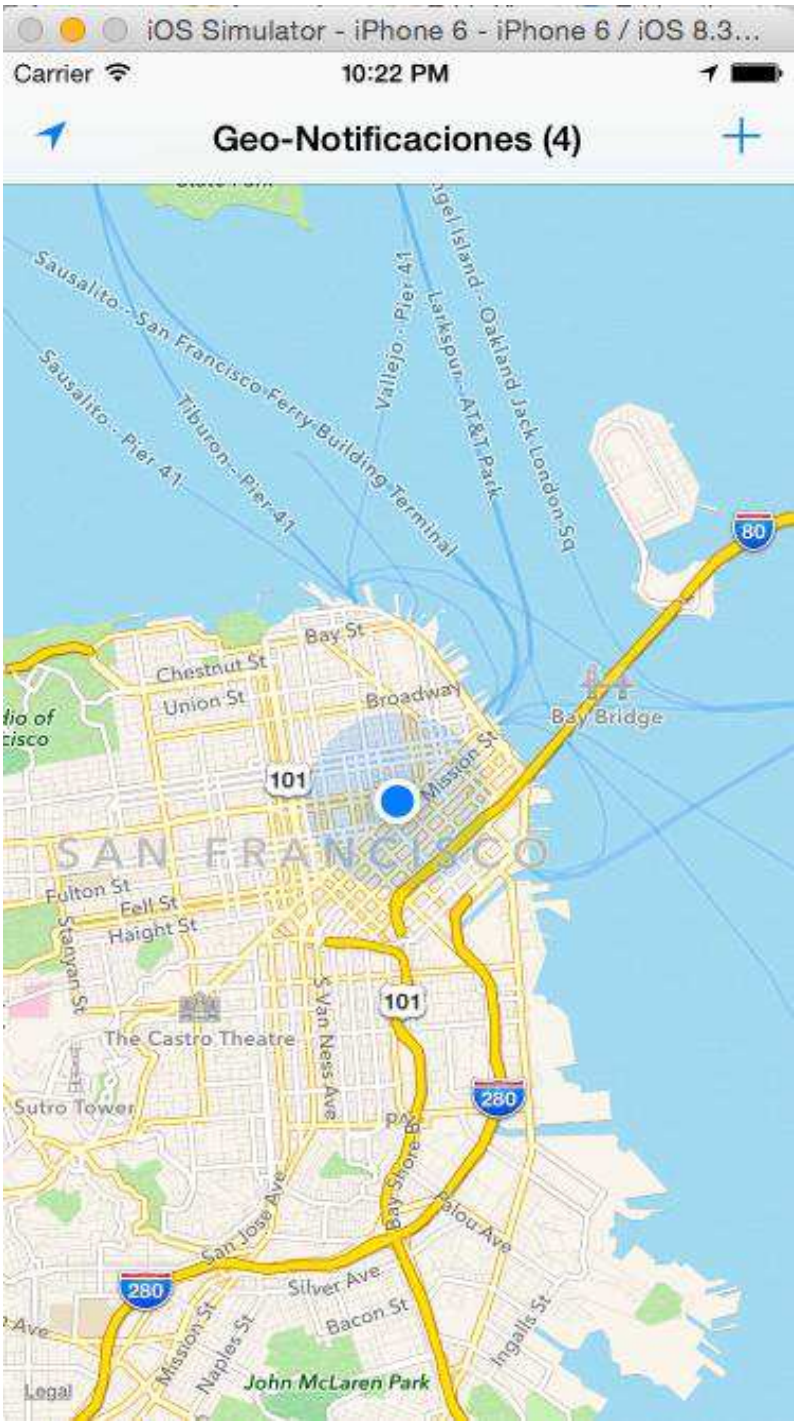

Ilustración 21: Demo de historia 6

Fuente: Core Location Framework Reference - Apple Developer

Elaborado por: Barberan León Julio Vicente

- Historia 7

Ilustración 22: Demo de historia 7

Fuente: Core Location Framework Reference - Apple Developer

Elaborado por: Barberan León Julio Vicente

4.4.5.3. Pruebas de aceptación

- Historia 5
  - Cada ubicación mostro una marca al momento de cargarse dependiendo su tipo lo mostraba con un marcado estático y dinámico
- Historia 6-7
  - La vista del mapa marcaba notaciones tanto cuando se representaba de manera normal o GPS mostrando las opciones deseadas

4.4.6. Iteración 3

Ilustración 23: Iteración 3 con historias de usuario

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

4.4.6.1. Tareas de las historias

Tarea	
Número tarea: 16	**Número historia: 08**
Nombre tarea: Añadido de ubicación al modelo de CORE DATA	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 3 Junio del 2015	**Fecha fin: 5 Junio del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Todas las ubicaciones serán añadidos al modelo CORE DATA realizado en la primera iteración y consecuentemente ser almacenado	

Tabla 42: Tarea de historia -16

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

	Tarea
Número tarea: 17	**Número historia: 08**
Nombre tarea: Carga de ubicación del modelo CORE DATA al mapa	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 3 Junio del 2015	**Fecha fin: 5 Junio del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Todas las ubicaciones almacenadas dentro del CORE DATA serán mostradas en el mapa utilizando funciones nativas de MAP KIT	

Tabla 43: Tarea de historia -17

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

Tarea	
Número tarea: 18	**Número historia: 09**
Nombre tarea: Eliminación de la ubicación dentro del modelo CORE DATA	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 3 Junio del 2015	**Fecha fin: 6 Junio del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Al momento de enlazar la notación de vista con el evento de eliminar se procederá a eliminar el dato contenido dentro del CORE DATA	

Tabla 44: Tarea de historia -18

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

Tarea	
Número tarea: 19	**Número historia: 09**
Nombre tarea: Enlace del método eliminar con la notación de marcado	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 3 Junio del 2015	**Fecha fin: 6 Junio del 2015**
Programador responsable: BARBERAN LEÓN JULIO VICENTE	
Descripción: Se llama dentro del controlador de la vista de marcado al método eliminar contenido dentro la clase de operaciones	

Tabla 45: Tarea de historia -19

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

4.4.6.2. Demos de las historias

- Historia 8

Ilustración 24: Demo de historia 8

Fuente: Core Location Framework Reference - Apple Developer

Elaborado por: Barberan León Julio Vicente

- Historia 9

Ilustración 25: Demo de historia 9

Fuente: Core Location Framework Reference - Apple Developer

Elaborado por: Barberan León Julio Vicente

4.4.6.3. Pruebas de aceptación

- Historia 8
    - La carga de las ubicaciones y el almacenamiento se produjo de manera correcta
- Historia 9
    - Los datos que fueron eliminados dentro del modelo pasaron correctamente las pruebas

4.5. Conexión entre repositorios y aplicación

- Conexión con GITHUB y aplicación
- Se establece la dirección de la aplicación mediante líneas de comando

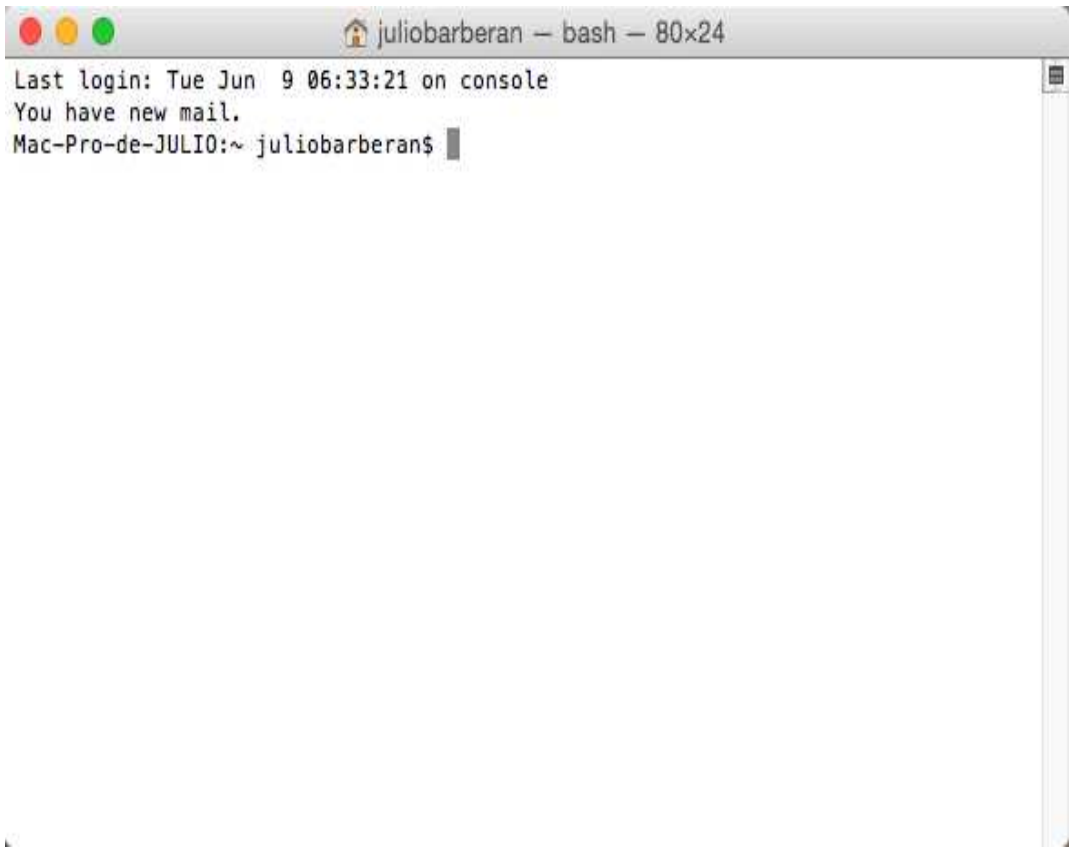

Ilustración 26: Terminal OSX

Fuente: Apple OSX 10.11 CMD

Elaborado por: Barberan León Julio Vicente

Ilustración 27: Ubicación de la carpeta del proyecto

Fuente: Apple OSX 10.11 CMD

Elaborado por: Barberan León Julio Vicente

- Se conecta y crea un origen tipo Git para la conexión entre el repositorio y la aplicación

Ilustración 28: Creación y conexión con gIthub

Fuente: Apple OSX 10.11 CMD

Elaborado por: Barberan León Julio Vicente

- Se realiza el primer Push para enviar la aplicación al repositorio y este alojado

Ilustración 29: Primer Push enviado al repositorio

Fuente: Apple OSX 10.11 CMD

Elaborado por: Barberan León Julio Vicente

- Conexión con bitbucket y aplicación

Ilustración 30: Terminal OSX

Fuente: Apple OSX 10.11 CMD

Elaborado por: Barberan León Julio Vicente

- Se conecta y crea un origen tipo Git para la conexión entre el repositorio y la aplicación

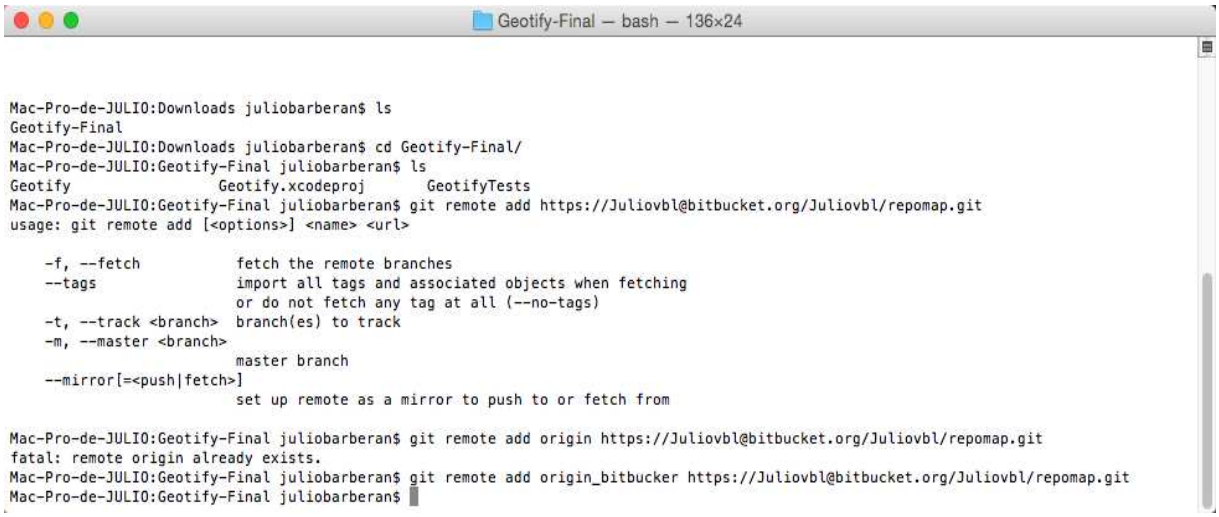

Ilustración 31: Creación y conexión con Bitbucket

Fuente: Apple OSX 10.11 CMD

Elaborado por: Barberan León Julio Vicente

- Se realiza el primer Push para enviar la aplicación al repositorio y este alojado

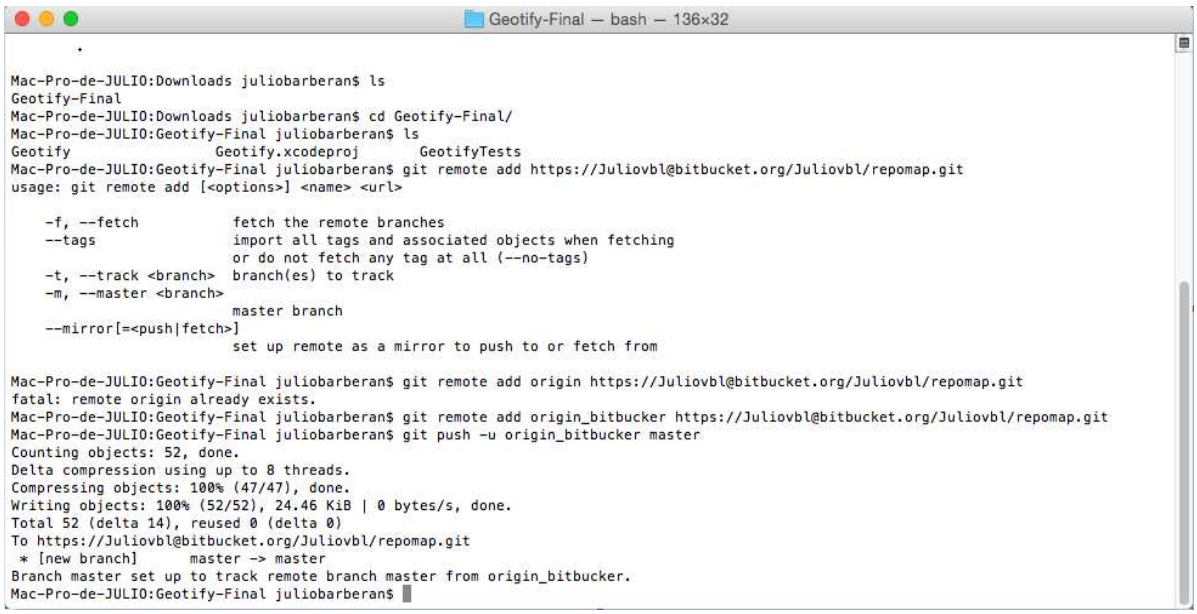

Ilustración 32: primer Push con el repositorio

Fuente: Apple OSX 10.11 CMD

Elaborado por: Barberan León Julio Vicente

4.6. Implementación de Xcode al servicio OSX SERVER

- Selección del servidor que alojara los "BOTS"

Ilustración 33: Selección de servidor

Fuente: Apple OSX 10.11 xcode

Elaborado por: Barberan León Julio Vicente

- Importando el servicio XCODE server a OSX server

Ilustración 34: Importando Xcode al servidor

Fuente: Apple OSX 10.11 xcode

Elaborado por: Barberan León Julio Vicente

- Activando servicio XCODE server

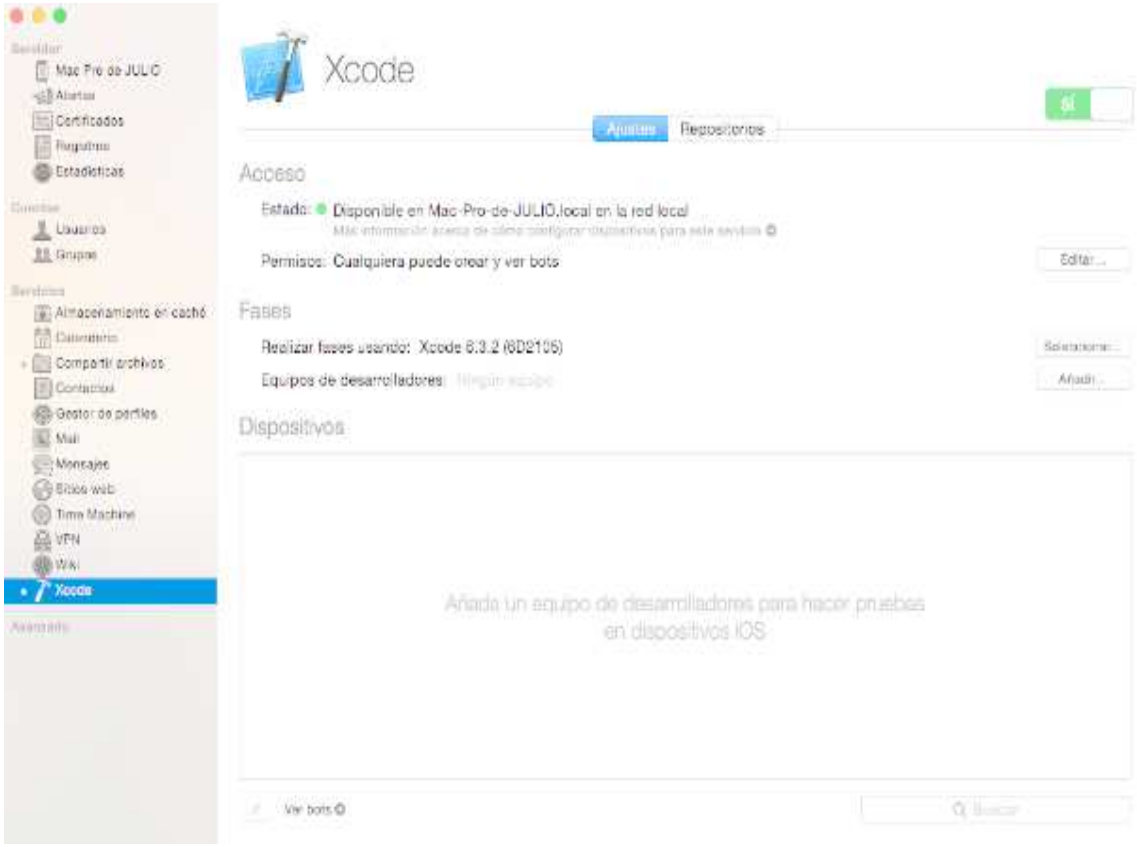

Ilustración 35: Activación del servicio Xcode

Fuente: Apple OSX 10.11 xcode

Elaborado por: Barberan León Julio Vicente

4.7. Creación de “BOTS” en Xcode

- Se selecciona la pestaña PRODUCT Y CREATE BOTS
- Se escoge el esquema
- Se establece el nombre del BOT
- Y el servidor el cual va estar alojado

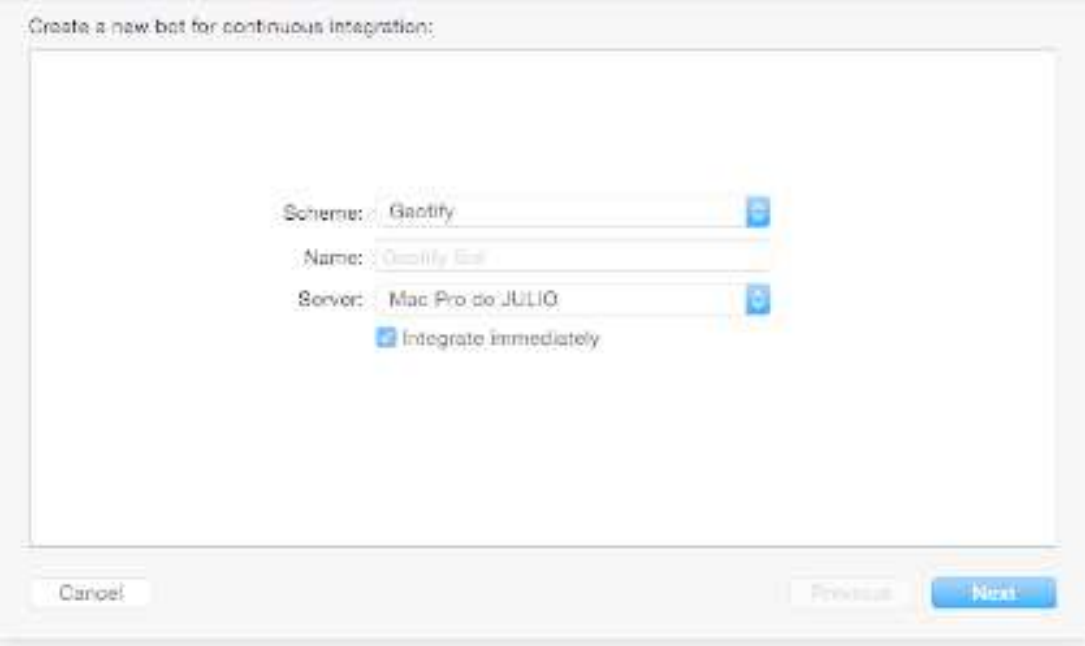

Ilustración 36: Creación del esquema de trabajo

Fuente: Apple OSX 10.11 xcode

Elaborado por: Barberan León Julio Vicente

- Se selecciona el repositorio que va enlazarse entre los BOTS y la aplicación
- Dependiendo de los repositorios se pueden enlazar mediante llaves de SSH

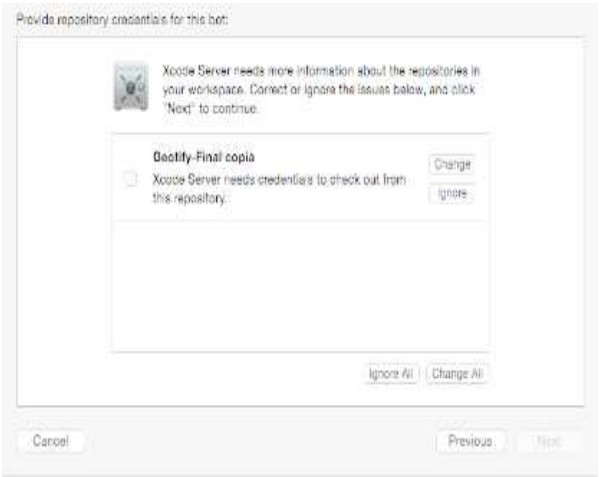

Ilustración 37: Opción a inserción de SSH

Fuente: Apple OSX 10.11 xcode

Elaborado por: Barberan León Julio Vicente

- Se establece el esquema que integrara la aplicación desde el repositorio hasta el servidor el cual puede ser:
    - Periodo de tiempo
    - Periodo de Push
    - Periodo de Commit
    - Acciones como el análisis de la aplicación, pruebas de funcionalidad y la opción de limpiar los BOTS

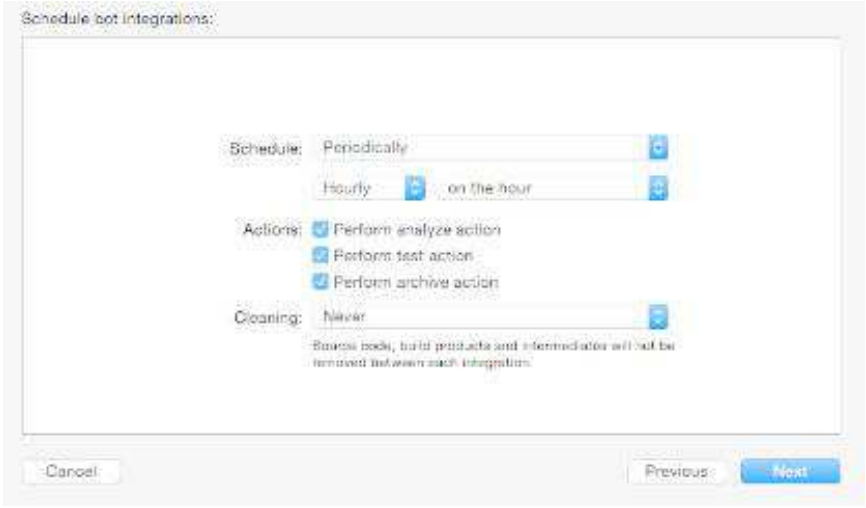

Ilustración 38: Esquema de Horarios de BOTS

Fuente: Apple OSX 10.11 xcode

Elaborado por: Barberan León Julio Vicente

- Se establece el sistema operativo y los despóticos que simularan el análisis y las pruebas que se hayan almacenado en repositorio

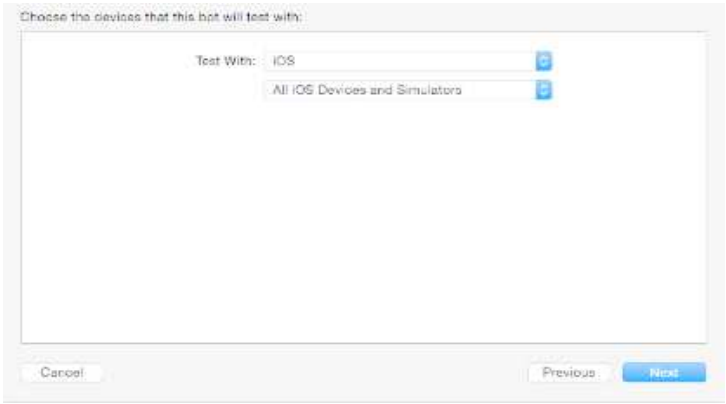

Ilustración 39: Plataformas y dispositivos donde correran BOTS

Fuente: Apple OSX 10.11 xcode

Elaborado por: Barberan León Julio Vicente

- Se programa disparadores los cuales realizan acciones que afectan a cualquier ambiente del sistema operativo antes o después de una integración

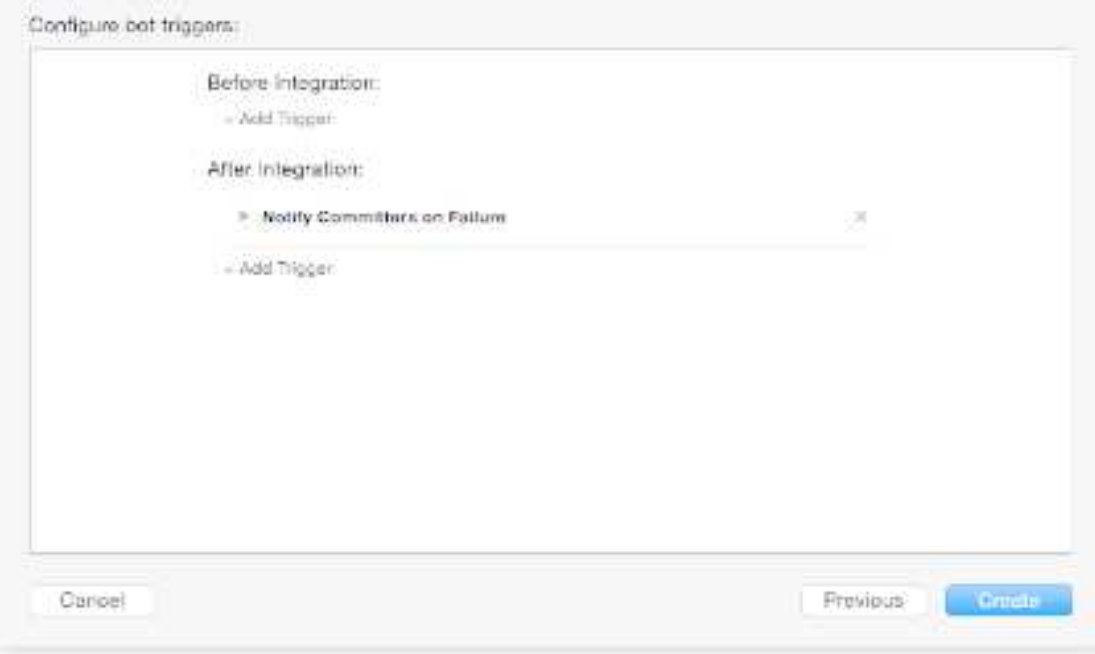

Ilustración 40: Adquisición de disparadores

Fuente: Apple OSX 10.11 xcode

Elaborado por: Barberan León Julio Vicente

- En el presente se utilizó el siguiente script para lanzar integraciones automáticas de acuerdo al esquema de trabajo que se haya escogido
- Script GITHUB

```
#!/usr/bin/env ruby


      $repository_url = "https://github.com/APPBOTS/KITMAPS.git"
      $repository_mode = "git"

      $server_host = " Mac-Pro-de-JULIO.local"



      require 'net/http'

      def kick(branch)
        theURL = URI("http://#{$server_host}/xcs/kick-commit-bots")
        if branch.nil?
          Net::HTTP.post_form(theURL, 'repository' => $repository_url)
        else
          Net::HTTP.post_form(theURL,  'repository'  =>  $repository_url,
'branch' => branch)
        end
      end

      if __FILE__ == $0

        branches = []

        if $repository_mode == "git"
          $stdin.each_line do |line|
            oldrev, newrev, ref = line.strip.split
            if ref =~ %r{^refs/heads/(.+)$}
              branches.push($~[1])
            end
```

```
    end
  elsif $repository_mode == "svn" and ARGV.length >= 2
    repository = ARGV[0]
    revision = ARGV[1]
    modifiedDirs = `svnlook dirs-changed -r #{revision} #{repository}`.lines.map { |line| line.chomp }
    modifiedDirs.each do |d|
      if d =~ %r{branches/([^/]+)}
        branches.push($~[1])
      end
    end
  end

  if branches.empty?
    kick(nil)
  else
    branches.each do |branch|
      kick(branch)
    end
  end
end
```

Script BITBUCKET

```
#!/usr/bin/env ruby


$repository_url = "https://Juliovbl@bitbucket.org/Juliovbl/repomap.git"
$repository_mode = "git"

$server_host = "Mac-Pro-de-JULIO.local"



require 'net/http'
```

```
def kick(branch)
  theURL = URI("http://#{$server_host}/xcs/kick-commit-bots")
  if branch.nil?
    Net::HTTP.post_form(theURL, 'repository' => $repository_url)
  else
    Net::HTTP.post_form(theURL, 'repository' => $repository_url, 'branch' => branch)
  end
end

if __FILE__ == $0

  branches = []

  if $repository_mode == "git"
    $stdin.each_line do |line|
      oldrev, newrev, ref = line.strip.split
      if ref =~ %r{^refs/heads/(.+)$}
        branches.push($~[1])
      end
    end
  elsif $repository_mode == "svn" and ARGV.length >= 2
    repository = ARGV[0]
    revision = ARGV[1]
    modifiedDirs = `svnlook dirs-changed -r #{revision} #{repository}`.lines.map { |line| line.chomp }
    modifiedDirs.each do |d|
      if d =~ %r{branches/([^/]+)}
        branches.push($~[1])
      end
    end
  end

  if branches.empty?
    kick(nil)
  else
```

```
    branches.each do |branch|
      kick(branch)
    end
  end
end
```

- BOTS ejecutándose vía IDE

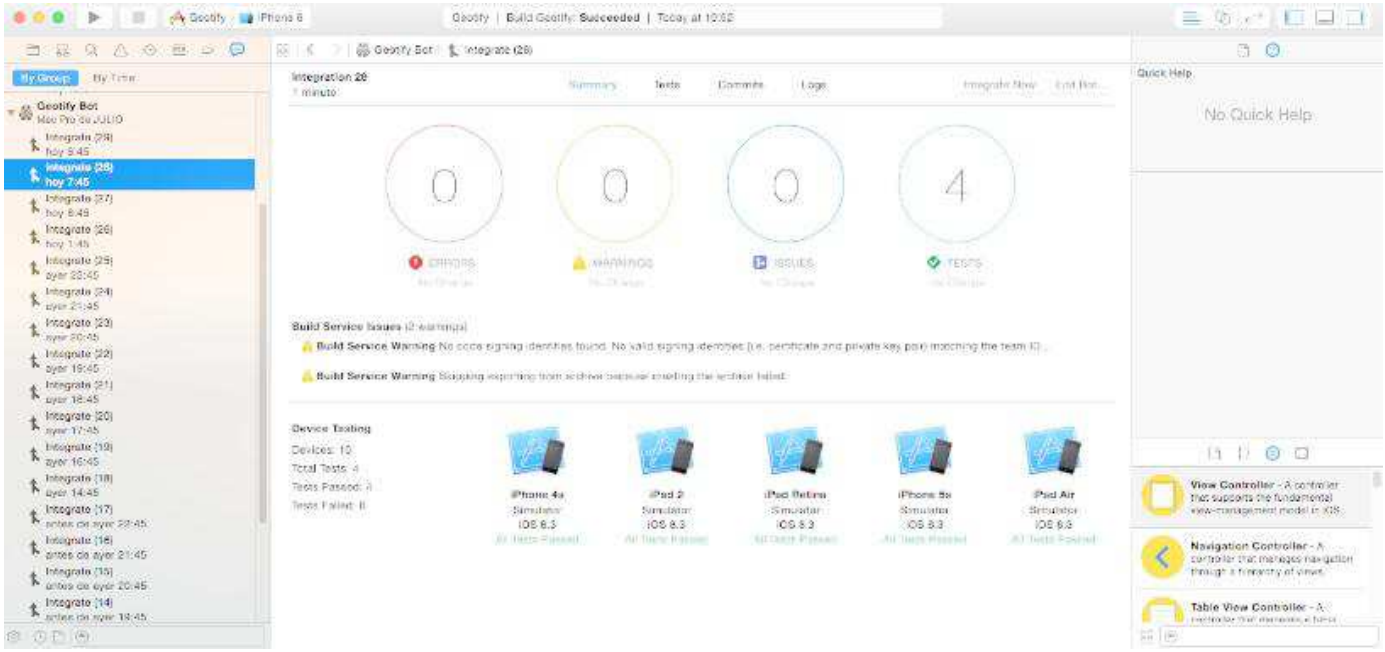

Ilustración 41: BOTS corriendo vía IDE

Fuente: Apple OSX 10.11 xcode

Elaborado por: Barberan León Julio Vicente

- BOTS ejecutándose vía WEB

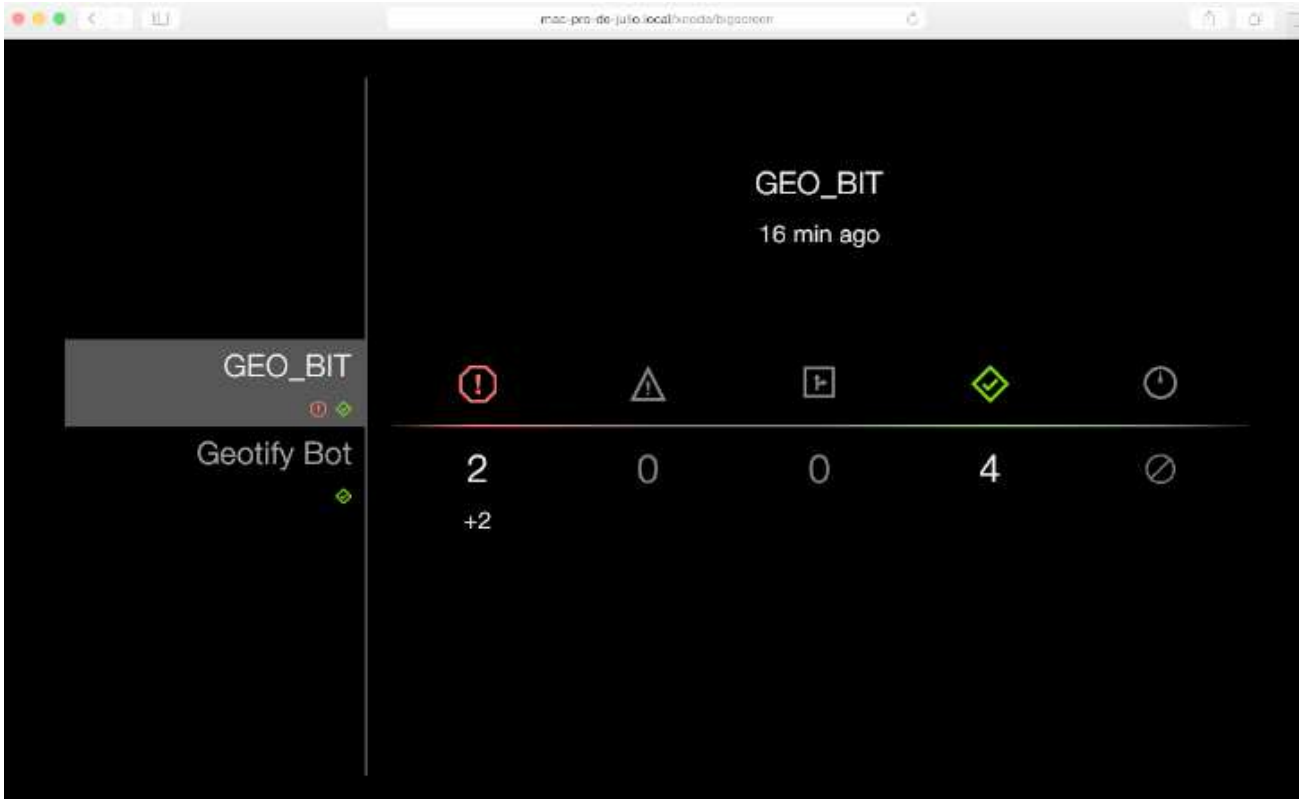

Ilustración 42: BOTS ejecutando vía WEB

Fuente: Apple OSX 10.11 xcode

Elaborado por: Barberan León Julio Vicente

CAPITULO V

RESULTADOS: PRESENTACIÓN Y ANÁLISIS

5.1. INTRODUCCIÓN

En el presente capítulo se mostrará los resultados obtenidos mediante el uso de un modelo informático automatizado con procesos de segundo plano que permitirán que el tiempo de desarrollo que conllevo realizar mediante el uso de metodologías agiles de software puede ser mucho más eficiente con esta tecnología.

5.2. SEGUIMIENTO Y MONITOREO DE RESULTADOS

En el presente punto se utilizó los resultados arrojados en el capítulo 3 acerca de las observaciones que se realizan para establecer una comparación en base al tiempo que se demoró en realizar las pruebas del código desde el repositorio.

FECHA	DESCRIPCIÓN	CARACTERISTICAS
01/02/2015 – 09:00	• Primera conexión con el servidor y el repositorio de código fuente • Creación del primer esquema de trabajo	• Pruebas de éxito • Pruebas de Integración • Integración Manual • Integración Automática
11/02/2015 – 09:00	• Primera integración entre el repositorio de código fuente y el servidor	• Pruebas de éxito • Pruebas de Integración • Integración Manual • Integración Automática
11/02/2015 – 11:00	• Segunda integración entre el repositorio de código fuente y el servidor	• Pruebas de éxito • Pruebas de Integración • Integración Manual • Integración Automática

	• Fallo perdido de conexión entre el repositorio de código fuente y el servidor	
14/02/2015 – 08:00	• Tercera conexión con el servidor y el repositorio de código fuente • Creación del segundo esquema de trabajo realizando simulaciones con dispositivos móviles	• Pruebas de éxito • Pruebas de Integración • Integración Manual • Integración Automática
14/02/2015 – 09:00	• Cuarta conexión con el servidor y el repositorio de código fuente • Creación del tercer esquema de trabajo realizando integraciones automáticas	• Pruebas de éxito • Pruebas de Integración • Integración Manual • Integración Automática

14/02/2015 – 10:00	• Quinta conexión automática con el servidor y el repositorio de código fuente	• Pruebas de éxito • Pruebas de Integración • Integración Manual • Integración Automática
14/02/2015 – 11:00	• Sexta conexión automática con el servidor y el repositorio de código fuente	• Pruebas de éxito • Pruebas de Integración • Integración Manual • Integración Automática
14/02/2015 – 12:00	• Séptima conexión automática con el servidor y el repositorio de código fuente	• Pruebas de éxito • Pruebas de Integración • Integración Manual • Integración Automática
14/02/2015 – 13:00	• Octava conexión automática con el servidor y el repositorio de código fuente	• Pruebas de éxito • Pruebas de Integración • Integración Manual • Integración Automática
14/02/2015 – 14:00	• Novena conexión automática con el servidor y el repositorio de código fuente	• Pruebas de éxito • Pruebas de Integración • Integración Manual • Integración Automática

Tabla 46: Seguimiento de I.C

Fuente: Plan de entrega (4.1.1)

Elaborado por: Barberan León Julio Vicente

FECHA: 07/06/2015

HORA: 09:00

CARACTERISTICAS:

- Primera conexión con el servidor y el repositorio de código fuente
- Creación del primer esquema de trabajo

OBSERVADOR	JULIO BARBERAN LEÓN				
OBJETOS A EVALUAR			**EVENTOS**		
TIEMPO 1 MIN – 30 SEG					
APLICACIÓN MOVIL	NUMERO	PRUEBAS DE ÉXITO	PRUEBAS DE FALLO	INTEGRACIÓN MANUAL	INTEGRACIÓN AUTOMATICA
PRUEBAS UNITARIAS					
VISTAS					
METODOS	**3**	✓		✓	

Tabla 47: Resultado I.C.A 1

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

OBSERVADOR	JULIO BARBERAN LEÓN						
	IPHONE 4S	IPHONE 5S	IPAD 2	IPAD RETINA	IPAD AIR	IPHONE 6	IPHONE 6 PLUS
CLASE (PRUEBA UNITARIA)	•	•	•	•	•	•	•
TIEMPO	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Tabla 48: Resultado I.C.A 1 (dispositivo)

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

En el presente grafico se ilustra la conexión con el repositorio y la aplicación, con lo que se puede apreciar se realizó con satisfacción

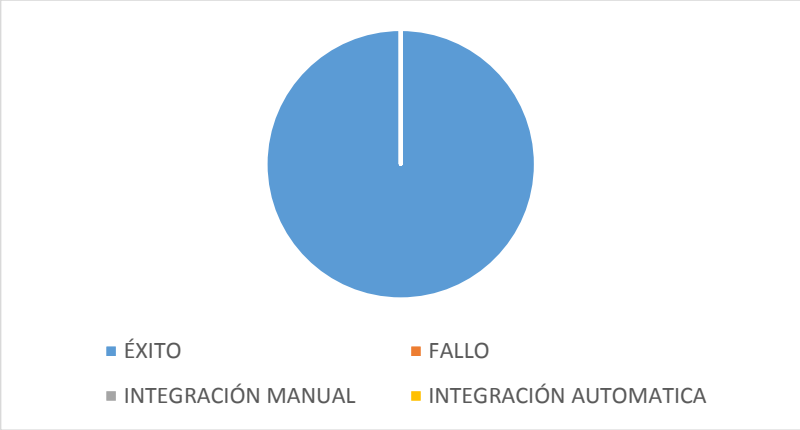

Ilustración 43: Representación de conexión entre repositorio y aplicación

Fuente: Seguimiento y monitoreo de resultados (5.2)

Elaborado por: Barberan León Julio Vicente

FECHA: 07/06/2015

HORA: 10:00

CARACTERISTICAS:

OBSERVADOR JULIO BARBERAN LEÓN					
OBJETOS A EVALUAR			**EVENTOS**		
TIEMPO 1 MIN – 30 SEG					
APLICACIÓN MOVIL	NUMERO	PRUEBAS DE ÉXITO	PRUEBAS DE FALLO	INTEGRACIÓN MANUAL	INTEGRACIÓN AUTOMATICA
PRUEBAS UNITARIAS					
VISTAS					
METODOS	**3**	✓		✓	

- Primera integración entre el repositorio de código fuente y el servidor

Tabla 49: Resultado I.C.A 2

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

OBSERVADOR	JULIO BARBERAN LEÓN						
	IPHONE 4S	IPHONE 5S	IPAD 2	IPAD RETINA	IPAD AIR	IPHONE 6	IPHONE 6 PLUS
CLASE (PRUEBA UNITARIA)	•	•	•	•	•	•	•
TIEMPO	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Tabla 50: Resultado I.C.A 2 (dispositivo)

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

En el presente grafico se ilustra la primera integración manual que se realizó la cual afecto a tres métodos de la aplicación a probar mediante el modelo de integración continua.

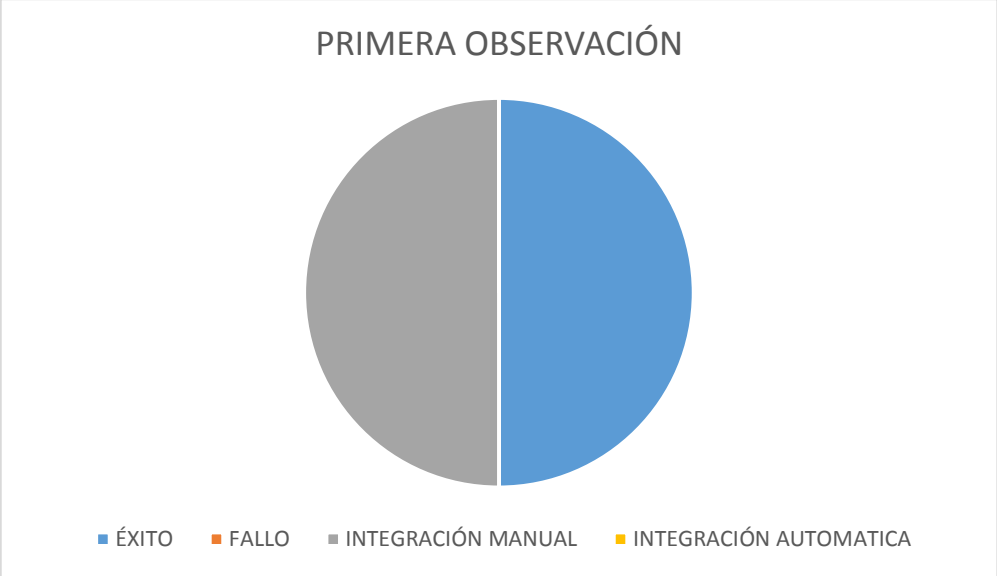

Ilustración 44: Comparativa I.C 1

Fuente: Seguimiento y monitoreo de resultados (5.2)

Elaborado por: Barberan León Julio Vicente

FECHA: 07/06/2015

HORA: 11:00

CARACTERISTICAS:

- Segunda integración entre el repositorio de código fuente y el servidor
- Fallo de conexión entre el repositorio de código fuente y el servidor

OBSERVADOR	JULIO BARBERAN LEÓN				
OBJETOS A EVALUAR / **TIEMPO 2 MIN**	EVENTOS				
APLICACIÓN MOVIL	NUMERO	PRUEBAS DE ÉXITO	PRUEBAS DE FALLO	INTEGRACIÓN MANUAL	INTEGRACIÓN AUTOMATICA
PRUEBAS UNITARIAS					
VISTAS					
METODOS	**3**		✓	✓	

Tabla 51: Resultado I.C.A 3

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

OBSERVADOR	JULIO BARBERAN LEÓN						
	IPHONE 4S	IPHONE 5S	IPAD 2	IPAD RETINA	IPAD AIR	IPHONE 6	IPHONE 6 PLUS
CLASE (PRUEBA UNITARIA)	•	•	•	•	•	•	•
TIEMPO	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Tabla 52: Resultado I.C.A 3 (dispositivo)

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

En el presente grafico se ilustra la segunda integración manual que se realizó la cual fallo al momento de la conexión con el repositorio lo cual causo que no se llevara a cabo la respectiva integración.

Ilustración 45: Comparativa I.C 2

Fuente: Seguimiento y monitoreo de resultados (5.2)

Elaborado por: Barberan León Julio Vicente

FECHA: 07/06/2015

HORA: 12:00

CARACTERISTICAS:

- Tercera conexión con el servidor y el repositorio de código fuente
- Creación del segundo esquema de trabajo realizando simulaciones con dispositivos móviles

OBSERVADOR	JULIO BARBERAN LEÓN				
OBJETOS A EVALUAR	**EVENTOS**				
TIEMPO 1 MIN – 10 SEG					
APLICACIÓN MOVIL	NUMERO	PRUEBAS DE ÉXITO	PRUEBAS DE FALLO	INTEGRACIÓN MANUAL	INTEGRACIÓN AUTOMATICA
PRUEBAS UNITARIAS	**2**	✓		✓	
VISTAS	**1**	✓		✓	
METODOS	**3**	✓		✓	

Tabla 53: Resultado I.C.A 4

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

OBSERVADOR	JULIO BARBERAN LEÓN						
	IPHONE 4S	IPHONE 5S	IPAD 2	IPAD RETINA	IPAD AIR	IPHONE 6	IPHONE 6 PLUS
CLASE (PRUEBA UNITARIA)	•	•	•	•	•	•	•
TIEMPO	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Tabla 54: Resultado I.C.A 4 (dispositivo)

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

En el grafico respectivo se puede ver como la aplicación ha afectado a otro número de elementos de la cual está compuesta como son vistas en donde se lo ha realizado utilizando integración manual.

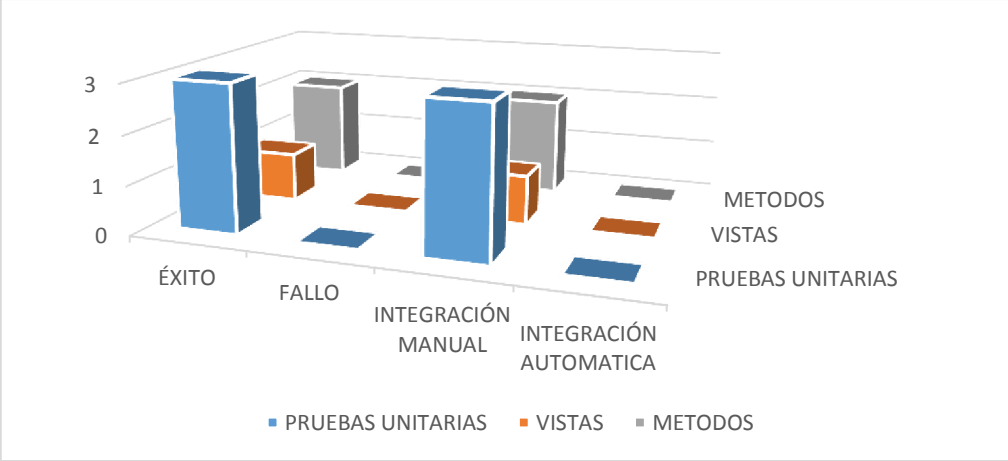

Ilustración 46: Comparativa I.C 3

Fuente: Seguimiento y monitoreo de resultados (5.2)

Elaborado por: Barberan León Julio Vicente

FECHA: 07/06/2015

HORA: 13:00

CARACTERISTICAS:

- Cuarta conexión con el servidor y el repositorio de código fuente
- Creación del tercer esquema de trabajo realizando integraciones automáticas

OBSERVADOR	**JULIO BARBERAN LEÓN**				
OBJETOS A EVALUAR	EVENTOS				
TIEMPO 0 MIN – 0 SEG					
APLICACIÓN MOVIL	NUMERO	PRUEBAS DE ÉXITO	PRUEBAS DE FALLO	INTEGRACIÓN MANUAL	INTEGRACIÓN AUTOMATICA

PRUEBAS UNITARIAS	**2**	✓			✓
VISTAS	**1**	✓			✓
METODOS	**3**	✓			✓

Tabla 55: Resultado I.C.A 5

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

OBSERVADOR	**JULIO BARBERAN LEÓN**						
	IPHONE 4S	IPHONE 5S	IPAD 2	IPAD RETINA	IPAD AIR	IPHONE 6	IPHONE 6 PLUS
CLASE (PRUEBA UNITARIA)	•	•	•	•	•	•	•
TIEMPO	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Tabla 56: Resultado I.C.A 5 (dispositivo)

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

En el grafico respectivo se aprecia el uso de la integración automática mediante el servicio IDE que se implementó junto con los BOTS.

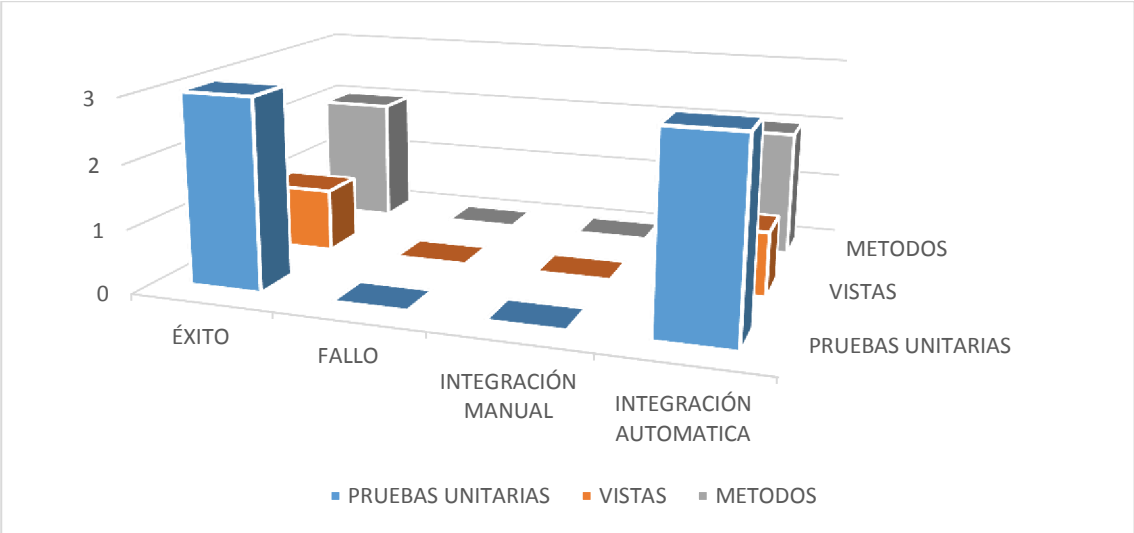

Ilustración 47: Comparativa I.C 4

Fuente: Seguimiento y monitoreo de resultados (5.2)

Elaborado por: Barberan León Julio Vicente

FECHA: 07/06/2015

HORA: 14:00

CARACTERISTICAS:

- Quinta conexión automática con el servidor y el repositorio de código fuente

OBSERVADOR	JULIO BARBERAN LEON				
OBJETOS A EVALUAR	**EVENTOS**				
TIEMPO 0 MIN – 0 SEG					
APLICACIÓN MOVIL	NUMERO	PRUEBAS DE ÉXITO	PRUEBAS DE FALLO	INTEGRACIÓN MANUAL	INTEGRACIÓN AUTOMATICA
PRUEBAS UNITARIAS	**2**	✓			✓
VISTAS	**1**	✓			✓
METODOS	**3**	✓			✓

Tabla 57: Resultado I.C.A 6

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

OBSERVADOR	JULIO BARBERAN LEÓN						
	IPHONE 4S	IPHONE 5S	IPAD 2	IPAD RETINA	IPAD AIR	IPHONE 6	IPHONE 6 PLUS
CLASE (PRUEBA UNITARIA)	•	•	•	•	•	•	•
TIEMPO	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Tabla 58: Resultado I.C.A 6 (dispositivo)

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

En el grafico respectivo se aprecia el uso de la integración automática mediante el servicio IDE que se implementó junto con los BOTS incorporando un mayor número de métodos para la comprobación de los resultados.

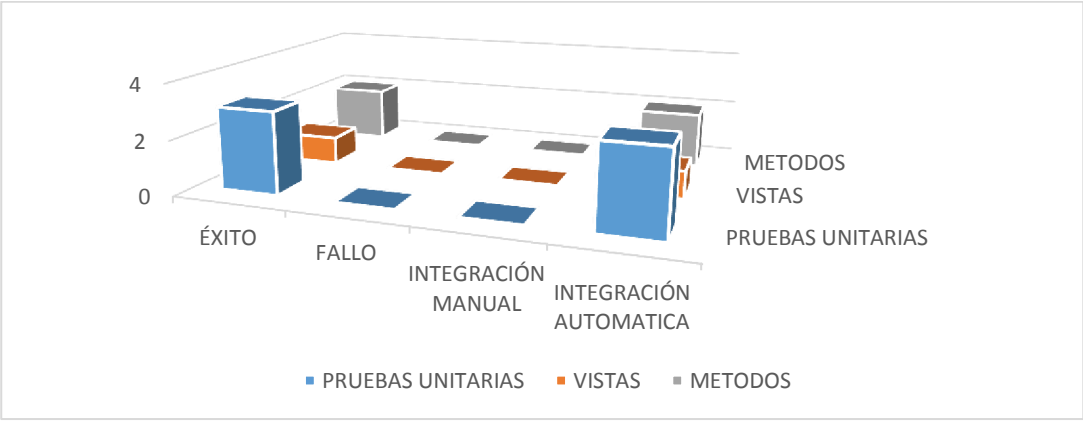

Ilustración 48: Comparativa I.C 5

Fuente: Seguimiento y monitoreo de resultados (5.2)

Elaborado por: Barberan León Julio Vicente

FECHA: 07/06/2015

HORA: 11:00

CARACTERISTICAS:

- Sexta conexión automática con el servidor y el repositorio de código fuente

OBSERVADOR	JULIO BARBERAN LEÓN				
OBJETOS A EVALUAR	**EVENTOS**				
TIEMPO 0 MIN – 0 SEG					
APLICACIÓN MOVIL	NUMERO	PRUEBAS DE ÉXITO	PRUEBAS DE FALLO	INTEGRACIÓN MANUAL	INTEGRACIÓN AUTOMATICA
PRUEBAS UNITARIAS	**2**	✓			✓
VISTAS	**3**	✓			✓
METODOS	**12**	✓			✓

Tabla 59: Resultado I.C.A 7

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

OBSERVADOR	JULIO BARBERAN LEÓN						
	IPHONE 4S	IPHONE 5S	IPAD 2	IPAD RETINA	IPAD AIR	IPHONE 6	IPHONE 6 PLUS
CLASE (PRUEBA UNITARIA)	•	•	•	•	•	•	•
TIEMPO	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Tabla 60: Resultado I.C.A 7 (dispositivo)

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

En las siguientes pruebas se va a seguir con un esquema igual al de la integración número cinco con los mimos parámetros a evaluar

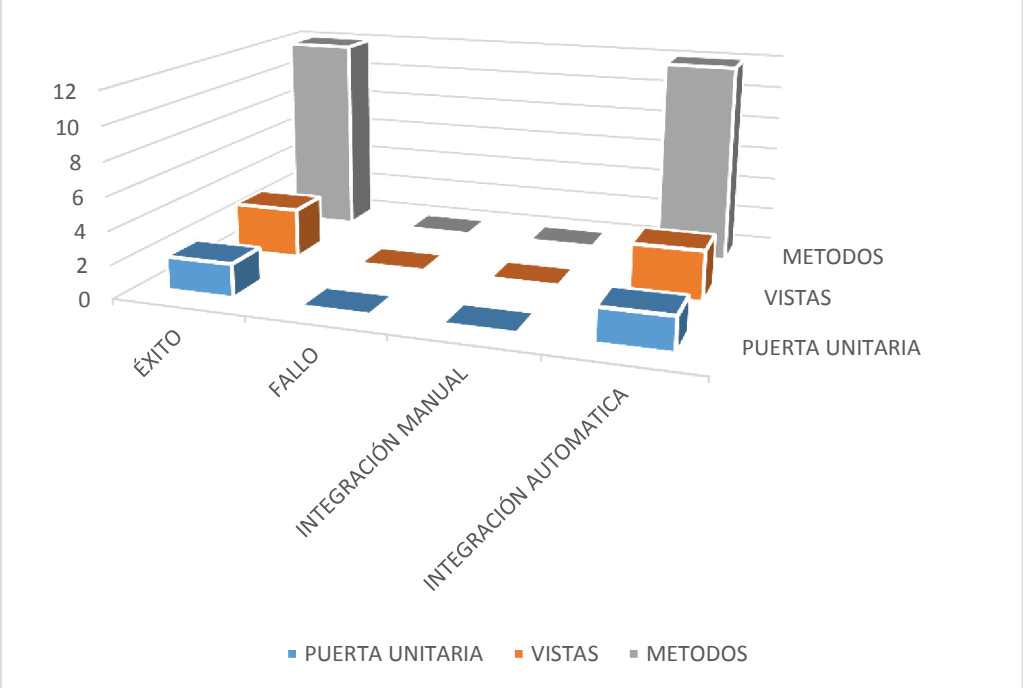

Ilustración 49: Comparativa I.C 6

Fuente: Seguimiento y monitoreo de resultados (5.2)

Elaborado por: Barberan León Julio Vicente

FECHA: 14/02/2015

HORA: 12:00

CARACTERISTICAS:

- Séptima conexión automática con el servidor y el repositorio de código fuente

OBSERVADOR	**JULIO BARBERAN LEÓN**				
OBJETOS A EVALUAR	**EVENTOS**				
TIEMPO 0 MIN – 0 SEG					
APLICACIÓN MOVIL	NUMERO	PRUEBAS DE ÉXITO	PRUEBAS DE FALLO	INTEGRACIÓN MANUAL	INTEGRACIÓN AUTOMATICA
PRUEBAS UNITARIAS	**2**	✓			✓
VISTAS	**3**	✓			✓
METODOS	**12**	✓			✓

Tabla 61: Resultado I.C.A 8

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

OBSERVADOR	JULIO BARBERAN LEÓN						
	IPHONE 4S	IPHONE 5S	IPAD 2	IPAD RETINA	IPAD AIR	IPHONE 6	IPHONE 6 PLUS
CLASE (PRUEBA UNITARIA)	•	•	•	•	•	•	•
TIEMPO	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Tabla 62: Resultado I.C.A 8 (dispositivo)

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

En el grafico respectivo se aprecia el uso de la integración automática mediante el servicio IDE que se implementó junto con los BOTS

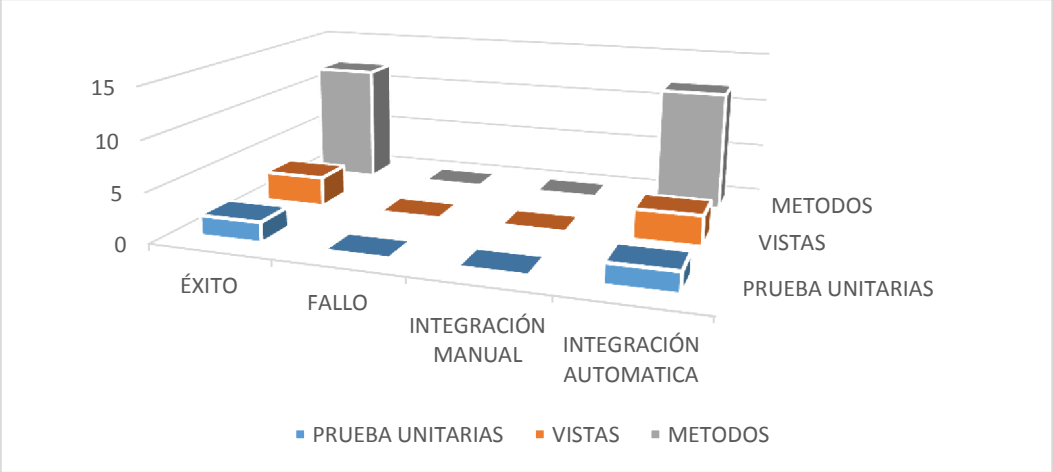

Ilustración 50: Comparativa I.C 7

Fuente: Seguimiento y monitoreo de resultados (5.2)

Elaborado por: Barberan León Julio Vicente

FECHA: 14/02/2015

HORA: 13:00

CARACTERISTICAS:

- Octava conexión automática con el servidor y el repositorio de código fuente

OBSERVADOR	JULIO BARBERAN LEÓN				
OBJETOS A EVALUAR	**EVENTOS**				
TIEMPO 0 MIN - 0 SEG					
APLICACIÓN MOVIL	NUMERO	PRUEBAS DE ÉXITO	PRUEBAS DE FALLO	INTEGRACIÓN MANUAL	INTEGRACIÓN AUTOMATICA
PRUEBAS UNITARIAS	**2**	✓			✓
VISTAS	**3**	✓			✓
METODOS	**12**	✓			✓

Tabla 63: Resultado I.C.A 9

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

OBSERVADOR	JULIO BARBERAN LEÓN						
	IPHONE 4S	IPHONE 5S	IPAD 2	IPAD RETINA	IPAD AIR	IPHONE 6	IPHONE 6 PLUS
CLASE (PRUEBA UNITARIA)	•	•	•	•	•	•	•
TIEMPO	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Tabla 64: Resultado I.C.A 9 (dispositivo)

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

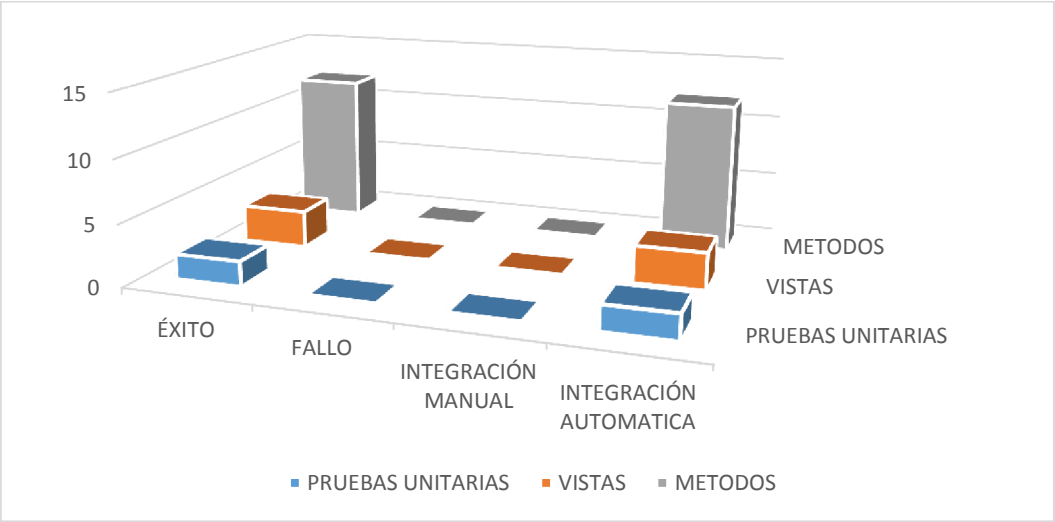

Ilustración 51: Comparativa I.C 8

Fuente: Seguimiento y monitoreo de resultados (5.2)

Elaborado por: Barberan León Julio Vicente

FECHA: 14/02/2015

HORA: 14:00

CARACTERISTICAS:

- Novena conexión automática con el servidor y el repositorio de código fuente

OBSERVADOR	JULIO BARBERAN LEÓN				
OBJETOS A EVALUAR	EVENTOS				
TIEMPO 0 MIN – 0 SEG					
APLICACIÓN	NUMER	PRUEBA	PRUEBA	INTEGRACIÓ	INTEGRACIÓ

MOVIL	O	S DE ÉXITO	S DE FALLO	N MANUAL	N AUTOMATICA
PRUEBAS UNITARIAS	**2**	✓			✓
VISTAS	**3**	✓			✓
METODOS	**12**	✓			✓

Tabla 65: Resultado I.C.A 10

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

OBSERVADOR	JULIO BARBERAN LEÓN						
	IPHONE 4S	IPHONE 5S	IPAD 2	IPAD RETINA	IPAD AIR	IPHONE 6	IPHONE 6 PLUS
CLASE (PRUEBA UNITARIA)	•	•	•	•	•	•	•
TIEMPO	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Tabla 66: Resultado I.C.A 10 (dispositivo)

Fuente: Desarrollo de Software ágil

Elaborado por: Barberan León Julio Vicente

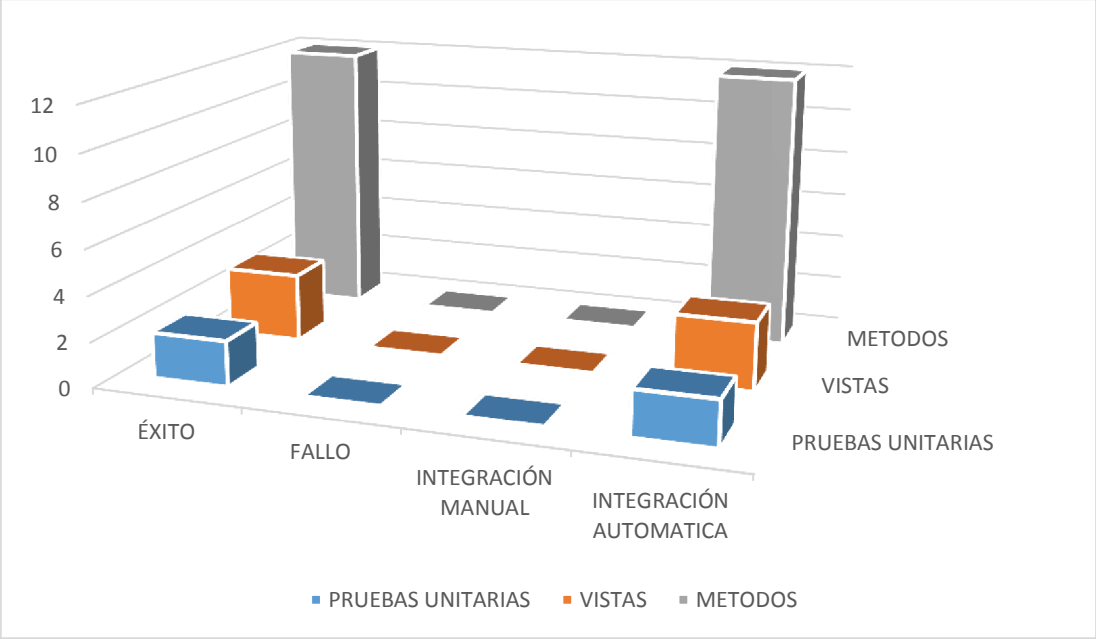

Ilustración 52: Comparativa I.C 9

Fuente: Seguimiento y monitoreo de resultados (5.2)

Elaborado por: Barberan León Julio Vicente

En el presente grafico se comparan los tiempos que se tomaron por cada integración que se realizó de manera manual y automática como resultados arroja que un ahorro considerable del tiempo que un programador desea evaluar el código fuente.

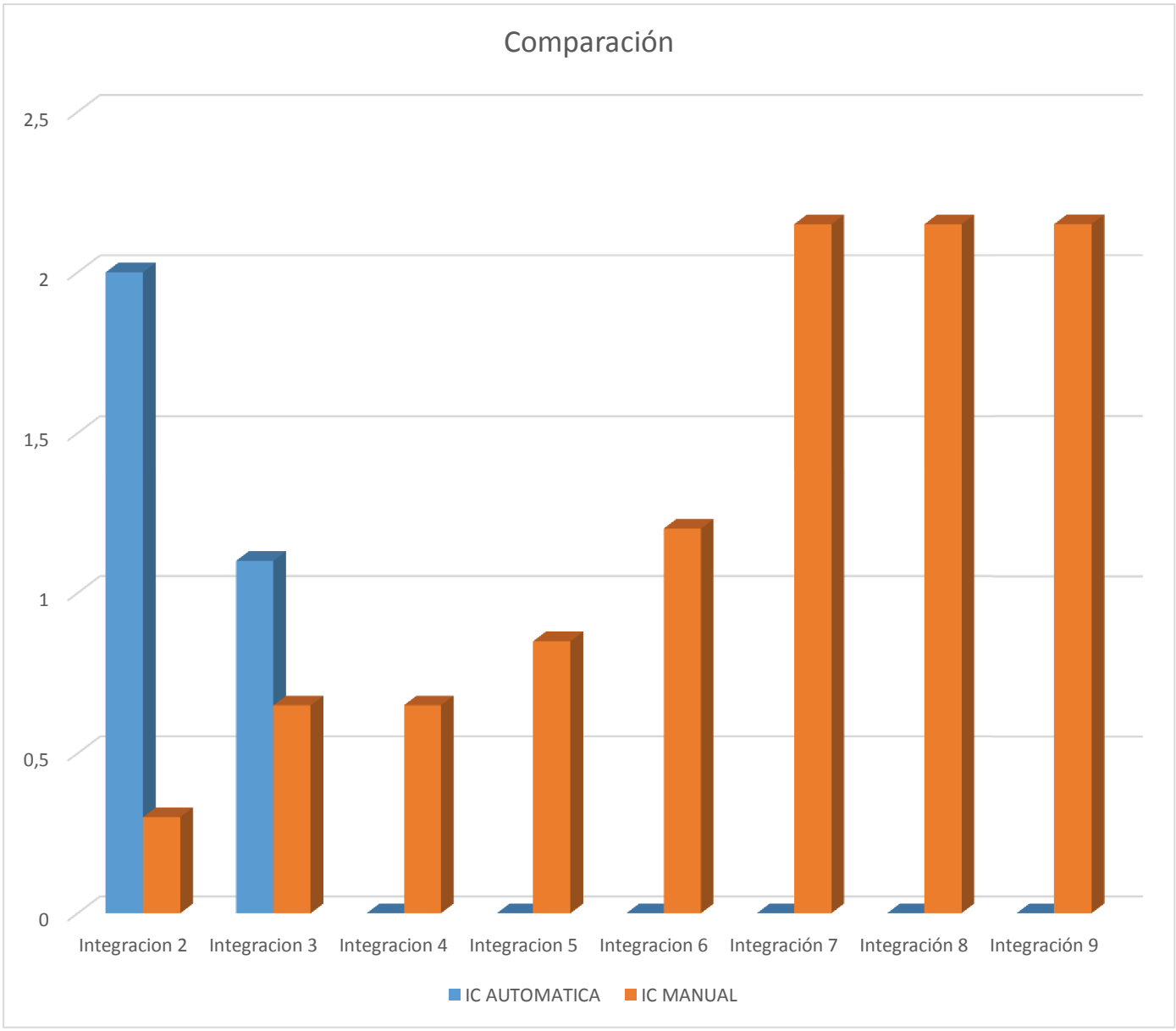

Ilustración 53: Comparativa I.C automática y I.C manual

Fuente: Seguimiento y monitoreo de resultados (5.2)

Elaborado por: Barberan León Julio Vicente

El siguiente grafico se comparan los tiempos que se tomaron por todas las integraciones sumándolos y haciéndolo un total con un equivalente a 4,4 segundos mediante un proceso de integración automática contra un 11,54 segundo mediante un proceso de integración manual con un ahorro del 38,4% que se hubiera requerido haciéndolo con el modelo típico de integración manual.

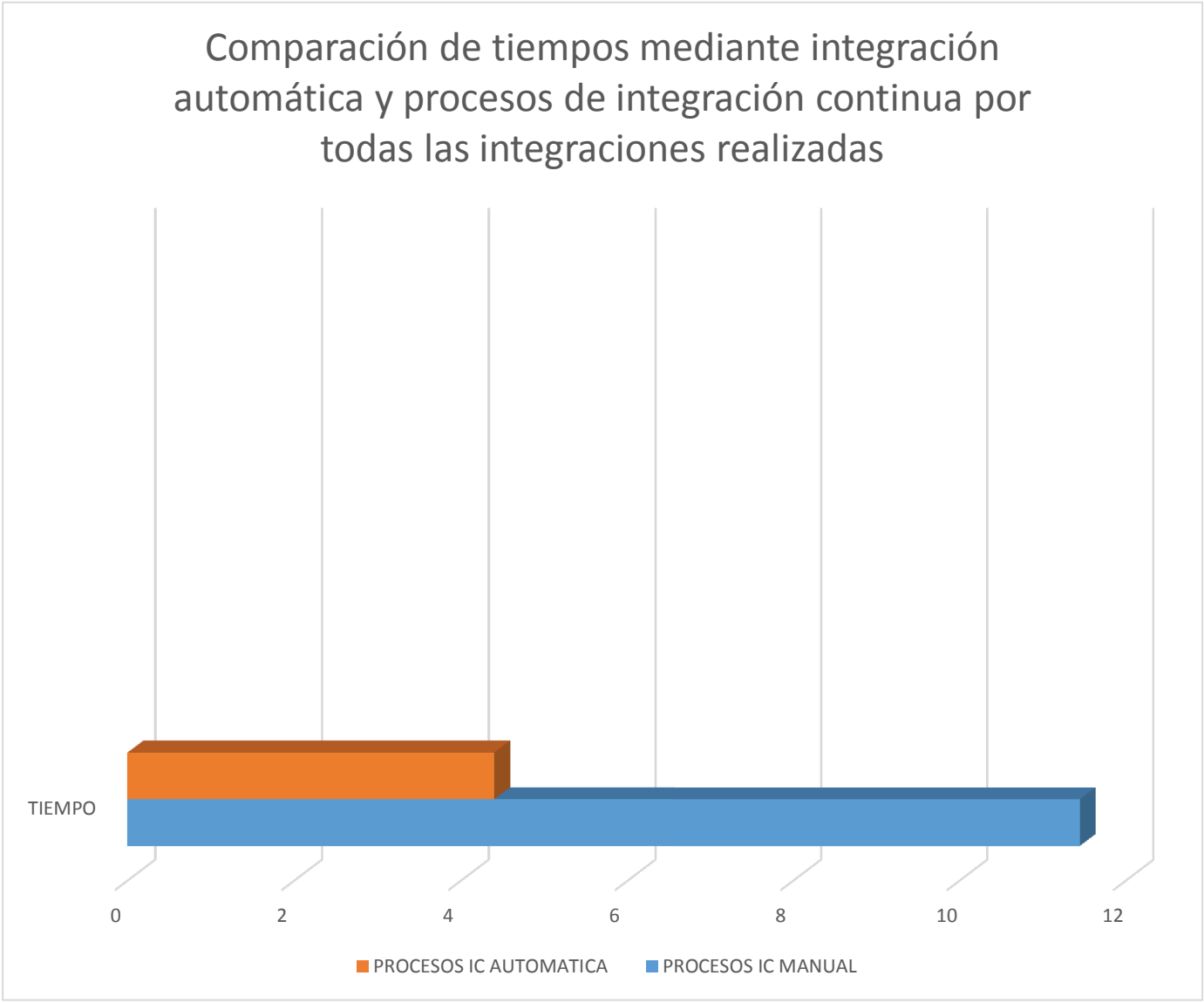

Ilustración 54: Comparativa de tiempos I.C manual y I.C automática

Fuente: Seguimiento y monitoreo de resultados (5.2)

Elaborado por: Barberan León Julio Vicente

5.3. CONCLUSIONES

En este punto se cierran cada una de las colusiones que se produjeron mediante el desarrollo de este trabajo lo cual está directamente ligado con los objetivos que se realizaron para cumplir con todas las demandas del trabajo por consiguiente cumpliendo con cada una de las metas y el conocimiento adquirido.

OBJETIVOS	CONCLUSIONES
• Investigación de las funciones que brinda Xcode Server para el soporte del modelo de Integración Continua	• Conociendo los distintos tipos de repositorios de almacenamiento de código fuente se procedió a investigar las funciones que brinda la herramienta Xcode Server para la administración del código fuente que se encuentra en cualquier repositorio.
• Realizar un análisis para establecer los servicios que ofrecerá la ejecución de BOTS bajo el modelo de integración continua	• Se procedió a establecer los servicios que ofrecerán los BOTS los cuales estaban regidos mediante el modelo de integración continua como el análisis de pruebas del proyecto, esquemas de integración mediante un esquema de trabajo, simulación de dispositivos móviles.

• Implementar la ejecución de BOTS en un sistema operativo de servidor bajo el modelo de integración continua	• La implementación se la realizo mediante el uso de ruby un lenguaje programación capaz de interactuar con el sistema operativo de acuerdo a los eventos que sucedieran por cada integración que se realizaba y configurando el servidor OSX para alojarlos y presentarlos mediante una página web.
• Evaluar la implementación de los BOTS en la administración del código fuente de una aplicación	• Se tomó los tiempos en que los BOTS realizaban las pruebas y simulaban en dispositivos móviles su funcionalidad utilizando la metodología XP se alcanzó a reducir el tiempo de pruebas en un 38,4 % por ciento del tiempo que se demoró utilizando el proceso de integración continua normal

Tabla 67: Conclusiones

Fuente: Objetivos Específicos (1.5.2)

Elaborado por: Barberan León Julio Vicente

5.4. RECOMENDACIONES

Con la base anterior de conclusiones y el respectivo desarrollo del presente trabajo de titulación, se presenta una recopilación se sugerencias y recomendaciones para el área enfocada en este trabajo como es el área de desarrollo y poder implementarlo en futuros trabajos para el estudiantado de la facultad y posibles aplicaciones a las metodologías de software para incrementar su nivel de productividad.

- Introducir modelos informáticos y realizarlos mediante la incursión de metodólogas de software agiles para promover la productividad en fases de pruebas de funcionalidad o aceptación.

- Soporte para que los desarrolladores o grupos de desarrollo de software se centran mucho más en partes criticas como la etapa de la funcionalidad misma del software

- Introducir tecnologías para la enseñanza en materias de desarrollo de software que permitan automatizar ciertos procesos en el desarrollo de software

- Implementar la programación a nivel de sistema operativo en catedra correspondientes al área de programación para la mejor captación de los diferentes procesos que se pueden llevar mediante cualquier evento

- Tomar como apoyo este trabajo de titulación para futuros trabajos que el estudiantado quiera realizar en ase a nuevas formas de implementación

5.5. BIBLIOGRAFÍA

Candela, S., García, C., Quesada, A., & Santana, F. (2007). *Fundamentos de sistemas operativos: teoría y ejercicios* . Madrid: Thomson.

Dimes, T. (2015). *Conceptos Básicos De Scrum: Desarrollo De Software Agile Y Manejo De Proyectos* . Babelcube.

Fernandez, D. V., & Martin, C. (2012). *Desarrollo de Videojuegos: Arquitectura del Motor de VideoJuegos.* Universidad de Castilla la Mancha.

Flynn, I. M. (2011). *Sistemas Operativos.* Cengage Learning.

Fuentes, J. R. (2014). *Desarrollo de Software agil.* Createspace Independent .

García, A. Á., Dedo, R. d., & Gómez, C. L. (2012). *Métodos Ágiles y Scrum.* Anaya Multimedia-Anaya Interactiva.

Gomez, M. M. (2006). *Introducción a la metodología de la Investigación Científica.* Cordoba - Argentina: Brujas.

González, M. Á. (2010). *Evolución tecnológica y cibermedios.*

Guérin, B. A. (2012). *Gestión de proyectos informáticos: Desarrollo, análisis y control.*

Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation* .

Jurado, C. B. (2010). *Diseño Ágil con TDD.*

Kawalerowicz, M., & Berntson, C. (2011). *Continuous Integration in .NET.* Manning.

Minoli, M., García, A. M., & Garzás, J. (2010). *Cómo implantar un proceso de integración continua.* Madrid.

Muñoz, C. C., Piattini, M. G., & Rubia, M. Á. (2010). *Calidad del producto y proceso software.* Madrid : RA-MA.

Pressman, R. S. (s.f.). *INGENIEÍA DEL SOFTWARE.* McGraw-Hill.

Román, I. R., & Tuya, J. J. (2007). *Técnicas cuantitativas para la gestión en la ingeniería del software.* La Coruña: NETBIBLO.

Sampieri, R. H., & Fernández Collado, C. (2006). *Metodología de la Investigación.* Mexico D.F.

SCHEAFFER, R. L., & MENDENHALL, W. (s.f.). *ELEMENTO DE MUESTREO.* THOMSON.

Solsona, F., & Viso, E. (2007). *Manual de Supervivencia en Linux .* Mexico.

SOMMERVILLE, I., & ALFONSO, M. I. (s.f.). *Ingeniería del Software .* MADRID - ESPAÑA: PEARSON EDUCATION.

Torres, A. B. (2013). *La dirección de proyectos: Una nueva visión.* Diaz Santos.

TORRES, B., & AUGUSTO, C. (2006). *Metodología de la Investigación. Para administración, economía, humanidades y ciencias sociales.* México: PEARSON.

Tuya, J., Román, I. R., & Cosín, J. D. (2007). *TÉCNICAS CUANTITATIVAS PARA LA GESTIÓN EN LA INGENIERÍA DEL SOFTWARE.* La Coruña - España: NETBIBLO.

Valenzuela, D. P. (2013). *Software Libre y Software Propietario: Impacto Jurídico, Económico y Cultiral en Colombia.*