UNIVERSIDAD HISPANOAMERICANA

INGENIERÍA INFORMÁTICA

PROPUESTA DE MEJORA EN LA METODOLOGÍA DE DESARROLLO DE *SOFTWARE*, EN LA DIVISIÓN INFRAESTRUCTURA DEL SECTOR TELECOMUNICACIONES DEL INSTITUTO COSTARRICENSE DE ELECTRICIDAD, EN EL PERIODO 2017-2018

Estudiante: Oscar Rojas Salas

Directora: Yenory Rojas Hernández

Segundo Semeste 2017, Primer Trimestre 2018

DECLARACIÓN JURADA.

DECLARACIÓN JURADA

Yo Oscar Rojas Salas, mayor de edad, portador de la cédula de identidad número 1-1385-0640 egresado de la carrera de Ingeniería Informática de la Universidad Hispanoamericana, hago constar por medio de éste acto y debidamente apercibido y entendido de las penas y consecuencias con las que se castiga en el Código Penal el delito de perjurio, ante quienes se constituyen en el Tribunal Examinador de mi trabajo de tesis para optar por el título de Licenciatura, juro solemnemente que mi trabajo de investigación titulado: Propuesta de mejora en la metodología de desarrollo de software, en la División Infraestructura del Sector Telecomunicaciones del Instituto Costarricense de Electricidad, en el período 2017-2018, es una obra original que ha respetado todo lo preceptuado por las Leyes Penales, así como la Ley de Derecho de Autor y Derecho Conexos número 6683 del 14 de octubre de 1982 y sus reformas, publicada en la Gaceta número 226 del 25 de noviembre de 1982; incluyendo el numeral 70 de dicha ley que advierte; articulo 70. Es permitido citar a un autor, transcribiendo los pasajes pertinentes siempre que éstos no sean tantos y seguidos, que puedan considerarse como una producción simulada y sustancial, que redunde en perjuicio del autor de la obra original. Asimismo, quedo advertido que la Universidad se reserva el derecho de protocolizar este documento ante Notario Público.

En fe de lo anterior, firmo en la ciudad de San José, a los catorce días del mes de febero del año dos mil dieciocho.

Firma del estudiante
Cédula: 1-1385-0640

CARTA DEL TUTOR.

CARTA DEL TUTOR

San José, 13 de febrero del 2018.

Ing. Yenory Rojas Hernández PhD.
Facultad de Computación
Universidad Hispanoamericana

Estimada señora:

El estudiante Oscar Rojas Salas, cédula de identidad número 113850640, me ha presentado, para efectos de revisión y aprobación, el trabajo de investigación denominado "Propuesta de mejora en la metodología de desarrollo de software, en la División Infraestructura del Sector Telecomunicaciones del Instituto Costarricense de Electricidad, en el período 2017-2018", el cual ha elaborado para optar por el grado académico de Licenciatura en Ingeniería Informática.

En mi calidad de tutor, he verificado que se han hecho las correcciones indicadas durante el proceso de tutoría y he evaluado los aspectos relativos a la elaboración del problema, objetivos, justificación; antecedentes, marco teórico, marco metodológico, tabulación, análisis de datos; conclusiones y recomendaciones.

De los resultados obtenidos por el postulante, se obtiene la siguiente calificación:

a)	ORIGINAL DEL TEMA	10%	9
b)	CUMPLIMIENTO DE ENTREGA DE AVANCES	20%	20
C)	COHERENCIA ENTRE LOS OBJETIVOS, LOS INSTRUMENTOS APLICADOS Y LOS RESULTADOS DE LA INVESTIGACION	30%	30
d)	RELEVANCIA DE LAS CONCLUSIONES Y RECOMENDACIONES	20%	20
e)	CALIDAD, DETALLE DEL MARCO TEORICO	20%	19
	TOTAL	100	98

En virtud de la calificación obtenida, se avala el traslado al proceso de lectura.

Atentamente,

Agustín Francesa Alfaro
Cédula identidad 113470859

CARTA DEL LECTOR.

CARTA DE LECTOR

Universidad Hispanoamericana
Sede Llorente
Escuela de Ingeniería Informática

Estimados señores

La estudiante **Rojas Salas Oscar,** cédula de identidad: **1-1385-0640**, me ha presentado para efectos de revisión y aprobación, el trabajo de investigación denominado " PROPUESTA DE MEJORA EN LA METODOLOGÍA DE DESARROLLO DE SOFTWARE, EN LA DIVISIÓN INFRAESTRUCTURA DEL SECTOR TELECOMUNICACIONES DEL INSTITUTO COSTARRICENSE DE ELECTRICIDAD, EN EL PERIODO 2017-2018", el cual ha elaborado para obtener su grado de **Licenciatura** en **Ingeniería Informática.**

He revisado y he hecho las observaciones relativas al contenido analizado, particularmente lo relativo a la coherencia entre el marco teórico y análisis de datos, la consistencia de los datos recopilados y la coherencia entre éstos y las conclusiones; asimismo, la aplicabilidad y originalidad de las recomendaciones, en términos de aporte de la investigación.

Por consiguiente, este trabajo cuenta con mi aval para ser presentado en la defensa pública.

Atte.

Firma: 

Ing. Roberto Romero Poveda

Cédula: 1-0996-0505

CARTA DE REVISIÓN DEL FILÓLOGO.

CARTA DE REVISIÓN DEL FILÓLOGO

San José, 06 de abril del 2018

SEÑORES
UNIVERSIDAD HISPANOAMERICANA.

Hago constar que he revisado el trabajo de tesis del estudiante OSCAR ROJAS SALAS denominado PROPUESTA DE MEJORA EN LA METODOLOGÍA DE DESARROLLO DE SOFTWARE, EN LA DIVISIÓN INFRAESTRUCTURA DEL SECTOR TELECOMUNICACIONES DEL INSTITUTO COSTARRICENSE DE ELECTRICIDAD, EN EL PERIODO 2017-2018, para optar por el grado académico de LICENCIATURA EN INGENIERÍA INFORMÁTICA.

Dada esta revisión se considera que dicho trabajo cumple con los requisitos establecidos por la Universidad, para ser presentado como requerimiento final de graduación.

Atentamente

Licda. Norma Patricia Segura Herrera.
Carné Colegio de Licenciados y Profesores 032847
Cédula 106440550

DEDICATORIA.

Este trabajo lo dedico, principalmente a Dios, sin la ayuda de Él en el día a día no hubiera podido lograr esta meta en mi formación profesional.

También dedico esta tesis a mis padres, quienes durante todo el proceso de estudio me han apoyado dando palabras sabias y aliento para no caer o levantarme.

AGRADECIMIENTO.

Agradezco a Dios que me permitió cumplir este sueño, porque me dio la salud, el deseo y la sabiduría para afrontar todo el proceso de mis estudios, y poder concluir este proyecto.

A mi familia porque durante todo el proceso de estudio estuvieron a mi lado, brindándome apoyo, motivación y valores fundamentales que me permitieron desarrollar como persona, estudiante y profesional.

Aquellas personas que hoy no están a mi lado, pero que fueron pieza importante en el engranaje de mi vida, siempre me brindaron el apoyo, aliento y sus opiniones para seguir adelante en mis estudios.

A todas las personas de la universidad que estuvieron inmersas en el proceso de estudio y en el proyecto de graduación, quienes tuvieron la dedicación de compartir el conocimiento y guiarme en el camino.

Al Instituto Costarricense de Electricidad (ICE), especialmente a mis compañeros de trabajo quienes me dieron sus aportes durante mis estudios y proyecto de graduación. Contribuciones significativas para combinar la teoría del estudio universitario con la práctica por su experiencia.

Contenido.

Figuras.

Tablas.

CAPITULO I: PLANTEAMIENTO DEL TEMA.

1.1. DEFINICIÓN DEL PROBLEMA.

El Sector de Telecomunicaciones confiere a la División Infraestructura la responsabilidad del desarrollo, implementación de la red y configuración de las soluciones competitivas de los usuarios de telecomunicaciones, para lo cual la División establece como objetivo general: “Desarrollar e implementar una solución integral de la infraestructura de la red de telecomunicaciones, realizando una gestión eficiente en la infraestructura actual para optimizar e incorporar nuevos servicios y soluciones de telecomunicaciones.” (Instituto Costarricence de Electricidad, 2017). Para alcanzar este objetivo el área Tecnologías de la Información desempeña un rol primordial, brindando las herramientas tecnológicas, la infraestructura de red y el soporte requerido por los procesos adscritos a la División.

El dinamismo del mercado de telecomunicaciones y por ende de la División Infraestructura requieren de soluciones informáticas cada vez más eficientes y oportunas, que permitan contar con insumos para la toma de decisiones y soluciones que faciliten la integración de nuevas tecnologías.

Actualmente, los desarrollos de las soluciones informáticas tienen un porcentaje importante de satisfacción y aceptación por parte de los usuarios finales, además las implementaciones de estos han permitido que los objetivos planteados por la División

Infraestructura se cumplan; no obstante, los tiempos para la puesta en producción de los sistemas no se implementa con la rapidez que requiere el negocio, esta situación genera ralentización en los procesos de la cadena y provoca que se torne lenta la comercialización de los servicios ofrecidos por la institución; con lo cual se corre el riesgo de sufrir pérdidas económicas y de clientes.

Aunado a lo anterior, se ha constatado que, pese a realizar el proceso de análisis de requerimientos de forma correcta y el diseño de la solución sea la adecuada, durante la etapa de implementación algunos de los requerimientos distan en relación a la necesidad real del negocio, generando retrocesos en el proceso de desarrollo de *software*. Esto es debido al tiempo transcurrido entre la solicitud, por parte del área que requiere el desarrollo del *software*, hasta la implementación y entrega del mismo, que no necesariamente sea producto de una duración extensa en el proceso de desarrollo de *software*, sino que obedece al dinamismo propio de la organización e incluso del avance de la tecnología y de otros factores externos.

Por otra parte, el método actual utilizado para el análisis de requerimientos parece no facilitar las actividades de este proceso, dado a que se realizan retrocesos en las etapas de: análisis de requerimientos, diseño y desarrollo, definidas en el proceso de desarrollo del *software*; esto se lleva acabo para asegurar la interpretación de los requerimientos entre el usuario experto (intermediario), analista y desarrollador. Esta situación provoca un aumento en los tiempos transcurridos en cada una de las etapas;

lo cual impacta los tiempos de entrega del sistema; considerando que la metodología de desarrollo de *software* actual consiste en la entrega del aplicativo con todos sus componentes desarrollados, es decir, hasta ese momento se realiza la puesta en producción.

Otra situación presente es la carencia “parcial” de métricas a lo largo del proceso de desarrollo de *software*, a modo de ejemplo, la métrica establecida para estimar la duración total del desarrollo del proyecto se calcula considerando: 1) la experiencia obtenida en otros proyectos y 2) la codificación (programación) de los requerimientos; esta estimación es realizada por parte del programador y la coordinadora de Tecnologías de la Información, siendo esto algo subjetivo, no incorrecto, pero tampoco basado en estándares utilizados en la industria del *software*. Según Pressman, “La medición permite ganar comprensión acerca del proceso y del proyecto, al proporcionar un mecanismo de evaluación objetiva”. (2010, pág. 571). Además, menciona que “La medición es una herramienta administrativa. Si se realiza adecuadamente, ofrece entendimiento al gerente de un proyecto. Y, como resultado, lo auxilia a él y al equipo de software para tomar decisiones que conducirán hacia un proyecto exitoso”. (2010, pág. 572). De esta manera, la intensión de las métricas es proporcionar un conjunto de indicadores de proceso que conduzca a mejorar el proceso de *software* para garantizar, en la medida de lo posible, un proyecto exitoso.

La metodología de desarrollo de *software* aplicada en la División Infraestructura no se ajusta al entorno del negocio, ya que la puesta en operación de sus sistemas

informáticos desarrollados no responde en forma oportuna y eficiente a los tiempos requeridos tanto por la División como por el Sector de Telecomunicaciones, provocando deficiencias y ralentización en el flujo de los procesos internos y consecuentemente en la comercialización de los servicios que ofrece la institución.

Aunado a lo anterior, el área de Tecnologías de la información cuenta con poco personal para atender las solicitudes de desarrollo de *software* de la División Infraestructura, situación que impacta en la respuesta oportuna a los tiempos requeridos por la División.

El siguiente diagrama representa el conjunto de causas o factores que contribuyen a generar el efecto o problemática en estudio.

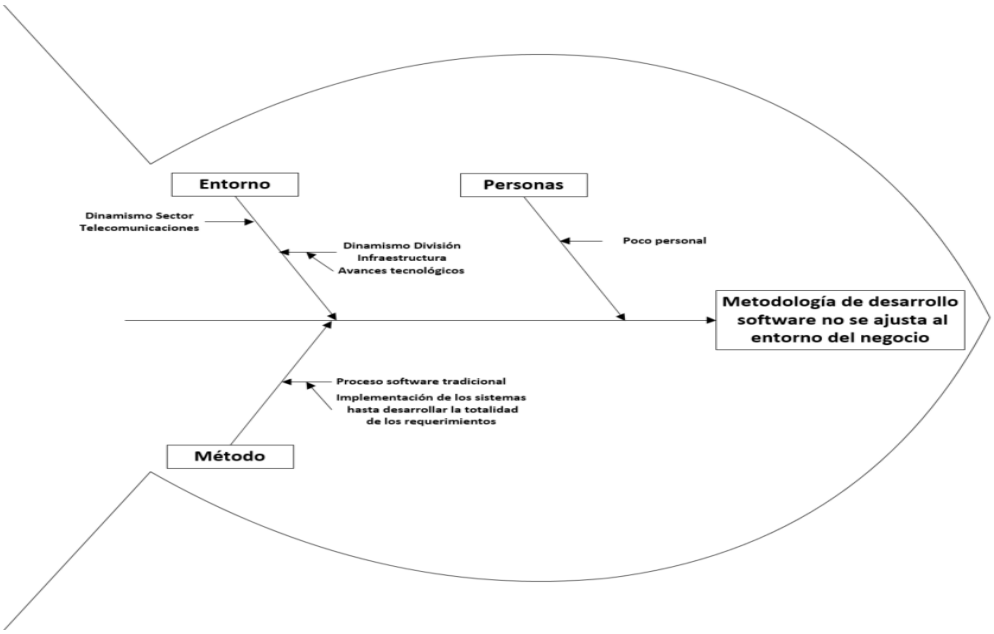


Figura 1. Diagrama Ishikawa.

Fuente: Elaboración propia.

1.2. JUSTIFICACIÓN DEL PROYECTO.

El panorama actual del Sector de Telecomunicaciones se encuentra en un proceso de apertura de servicios, lo cual implica una reacción inmediata en brindar una mejora significativa de los servicios móviles y de banda ancha, y para esto es fundamental contar con el inventario y control de la infraestructura de la red de telecomunicaciones que permite la prestación de estos servicios.

La División Infraestructura tiene el propósito de brindar soluciones de info-comunicaciones innovadoras, integrales, oportunas, y competitivas; su fin es desarrollar e implementar una solución integral de la infraestructura de la red de telecomunicaciones, realizando una gestión eficiente en la infraestructura actual para optimizar e incorporar nuevos servicios y soluciones de telecomunicaciones.

Con la conformación de la Superintendencia de Telecomunicaciones (SUTEL), como ente regulador de las telecomunicaciones, y por la búsqueda del Instituto Costarricense de Electricidad (ICE) en una constante mejora de calidad en sus procesos, resulta un innegable desafío de la institución en acatar nuevas regulaciones y lineamientos, al exponerse a importantes sanciones por el incumplimiento y atrasos en los tiempos de resolución de averías, lo que puede provocar que no se le pueda cobrar al cliente el costo mensual del servicio o pagar costosas multas.

Contar con información de calidad, tanto de sus procesos operativos como del inventario y control de la infraestructura de la red de telecomunicaciones, es de vital importancia para la estrategia del negocio del Sector de Telecomunicaciones, considerando que la información es un elemento que sirve de apoyo para la toma decisiones; así como para la SUTEL, ente regulador que tiene la potestad de solicitar información al ICE, y en dado caso de suministrarla en forma incorrecta o no actualizada es motivo de sanciones por incumplimiento.

Entendiéndose información de calidad contemplando las tres dimensiones expuestas por Cohen & Asís (2009):

1. Dimensión de tiempo: la información debe estar disponible cuando se necesite, estar actualizada, proporcionarse con la periodicidad requerida y además representar el pasado, presente y futuro.
2. Dimensión de contenido: la información no debe contener errores, deber ser relevante en relación a lo que se analiza, ser completa, concisa, con un enfoque amplio o centrado y medir el desempeño.
3. Dimensión de forma: la información se debe integrar en una forma sencilla, sea detallada o, en resumen, debe estar ordenada de acuerdo a cierto criterio, además se puede presentar en diversos formatos, por ejemplo, tablas, gráficas, listas, entre otros; así como aparecer en diferentes medios: papel o medios digitales (pág. 3).

Además, la inminente entrada en competencia en el sector de las telecomunicaciones, supondrá mejorar significativamente el servicio para mantener los clientes; por tal razón, disponer de sistemas de información que gestionen en forma correcta, automatizada y actualizada el inventario y control de la infraestructura de la red de telecomunicaciones de la institución contribuirá a la prestación de servicios de calidad.

De no contar con el escenario idóneo en la institución; se corre el riesgo de sufrir estragos económicos por pérdida masiva de clientes y pagos significativos de multas a los clientes y a la SUTEL. La imagen institucional ante la opinión pública sería fuertemente cuestionada en comparación con la competencia que, agresivamente, ingresará en el sector, además, la credibilidad de los clientes ante el servicio que se les provee es fundamental para posicionarse fuertemente en el mercado nacional e internacional, y es determinante garantizar una mejora continua del servicio.

La decisión de atacar este problema se fundamenta, en el impacto directo que los sistemas de información representan en el accionar de la División Infraestructura, ya que, si bien es cierto, no son sistemas de cara a la venta de los servicios que ofrece el ICE, sí tienen un rol importante en lo que respecta la gestión y automatización de los diversos procesos operativos, así como la integración de todas las tecnologías que las plataformas proveen. Esto conlleva a un mejor rendimiento económico para la institución ya que le permitirá ofrecer servicios comerciales de forma expedita.

Desde la década de los ochenta hasta el presente, han surgido herramientas, metodologías y tecnologías que se presentaban como la solución definitiva al problema de la planificación, previsión de costos y aseguramiento de la calidad en el desarrollo de *software*, sin embargo, la dificultad propia de los nuevos sistemas, y su impacto en las organizaciones, ponen de manifiesto las ventajas, y en muchos casos la necesidad, de aplicar una metodología formal para llevar a cabo los proyectos. (Cendejas Valdes, 2014, pág. 86).

Aunado a lo anterior, aplicar una metodología formal permite obtener mejores rendimientos para el desarrollo de *software*, conforme a esto el área de Tecnologías de la Información de la División Infraestructura, ha incentivado identificar puntos de mejora para la metodología de desarrollo de *software* utilizada, con el propósito de optimizar su proceso de desarrollo de *software*.

Por lo tanto, se espera que la propuesta de mejora en la metodología de desarrollo de *software* permita a la División Infraestructura contar con mejores sistemas informáticos desarrollados en tiempos oportunos, conducentes a una mejor calidad, y con procesos de desarrollo controlados y normalizados. Esto también incide en una mayor alineación y efectividad con la estrategia del negocio establecida tanto por la División Infraestructura como por el Sector de Telecomunicaciones.

1.3. OBJETIVOS GENERAL Y ESPECÍFICOS.

A continuación, se definen los objetivos que se pretenden conseguir del proyecto.

1.3.1. Objetivo general.

- → Proponer una mejora a la metodología de desarrollo de *software*, que se adecue a las necesidades de la institución y solvente sus principales problemas, mediante una reestructuración de su proceso.

1.3.2. Objetivos específicos.

- → Diagnosticar la metodología de desarrollo de *software* utilizada actualmente, mediante la aplicación de instrumentos de recolección de información, para evaluar la eficiencia en el proceso de desarrollo de *software*.
- → Determinar puntos de mejora en el proceso de desarrollo de *software* mediante la identificación de las mejores prácticas usadas, que procedan de diversas fuentes como estándares y metodologías de desarrollo de *software*, para que facilite a los desarrolladores la creación y mantenimiento de *software*.

- → Desarrollar las mejoras en la metodología de desarrollo de *software*, basado en el análisis de la información recolectada para solventar los problemas existentes en el proceso.
- → Realizar un ejercicio mediante la ejecución del diseño propuesto en la metodología de desarrollo de *software*, para evaluar si se adecua a las necesidades de la institución.

1.4. MARCO DE REFERENCIA EMPRESARIAL Y CONTEXTUAL.

El Instituto Costarricense de Electricidad fue creado por el Decreto - Ley No.449 del 8 de abril de 1949. Su creación fue el resultado de una larga lucha de varias generaciones de costarricenses que procuraron solucionar, definitivamente, los problemas de la escasez de energía eléctrica presentada en los años 40 y en apego de la soberanía nacional, en el campo de la explotación de los recursos hidroeléctricos del país. El principal objetivo del ICE era desarrollar, de manera sostenible, las fuentes productoras de energía existentes en el país y prestar el servicio de electricidad. (Instituto Costarricence de Electricidad, 2017)

Posteriormente, en la década de los sesenta se le confirió al ICE un nuevo objetivo: el establecimiento, mejoramiento, extensión y operación de los servicios de comunicaciones telefónicas, radiotelegráficas y radiotelefónicas en el territorio nacional. De esta manera, las telecomunicaciones iniciaron su desarrollo.

Con el paso de los años, se conforma el Grupo ICE como una Corporación de empresas públicas dedicada a ofrecer servicios de electricidad e infocomunicaciones a los habitantes de Costa Rica.

Se caracteriza por ser un grupo con gran capacidad en infraestructura, desarrollo tecnológico, capital humano altamente calificado, así como responsabilidad social y ambiental que se refleja en todas las grandes obras que ha desarrollado a lo largo de los años. Posee autonomía administrativa y financiera. Además, La legislación le permite adaptarse y aplicar condiciones jurídicas, financieras y administrativas necesarias para desarrollar nuevos esquemas de financiamiento. Esto le permite constituir y capitalizar empresas, filiales y sucursales para aumentar el desarrollo y calidad de los servicios de infocomunicaciones y electricidad, dentro y fuera del territorio nacional.

A través del ICE (Sector Electricidad y Telecomunicaciones) y sus empresas: Radiográfica Costarricense S.A. (RACSA), Compañía Nacional de Fuerza y Luz S.A. (CNFL), y Cablevisión se aporta a la sociedad costarricense progreso con sentido económico y social. La naturaleza de sus actividades es fundamental para el desarrollo integral del país por lo que todos los proyectos de inversión, tienen como eje esencial la protección del medio ambiente, contribuyendo con ello a la calidad de vida y al desarrollo sostenible (Instituto Costarricence de Electricidad, 2017).

En materia de electricidad ha utilizado como fuente primaria los recursos limpios y renovables como la energía hidroeléctrica, geotérmica, eólica y solar, para brindar

soluciones integrales que incluyen la construcción, generación, transmisión y distribución del servicio eléctrico.

La robusta y confiable red de telecomunicaciones le ha permitido al país convertirse en un atractivo punto para que grandes empresas de tecnología, salud y otros campos desarrollen sus negocios.

Grupo ICE con el paso del tiempo ha desarrollado experiencia y conocimiento:

1. Investiga, planifica y desarrolla proyectos de electricidad y telecomunicaciones.
2. Construye, genera, opera, transporta, distribuye y comercializa electricidad.
3. Implementa una diversidad de modelos y esquemas de financiamiento para ofrecer servicios de electricidad y telecomunicaciones.
4. Desarrolla su gestión en apego a la protección del medio ambiente. (Instituto Costarricence de Electricidad, 2017).

1.4.1. Misión Grupo ICE.

Somos la Corporación propiedad de los costarricenses, que ofrece soluciones de electricidad y telecomunicaciones, contribuyendo con el desarrollo económico, social y ambiental del país. (Instituto Costarricence de Electricidad, 2017).

1.4.2. Visión Grupo ICE.

Ser una Corporación líder, innovadora en los negocios de electricidad y telecomunicaciones en convergencia, enfocada en el cliente, rentable, eficiente,

promotora del desarrollo y bienestar nacional, con presencia internacional (Instituto Costarricence de Electricidad, 2017).

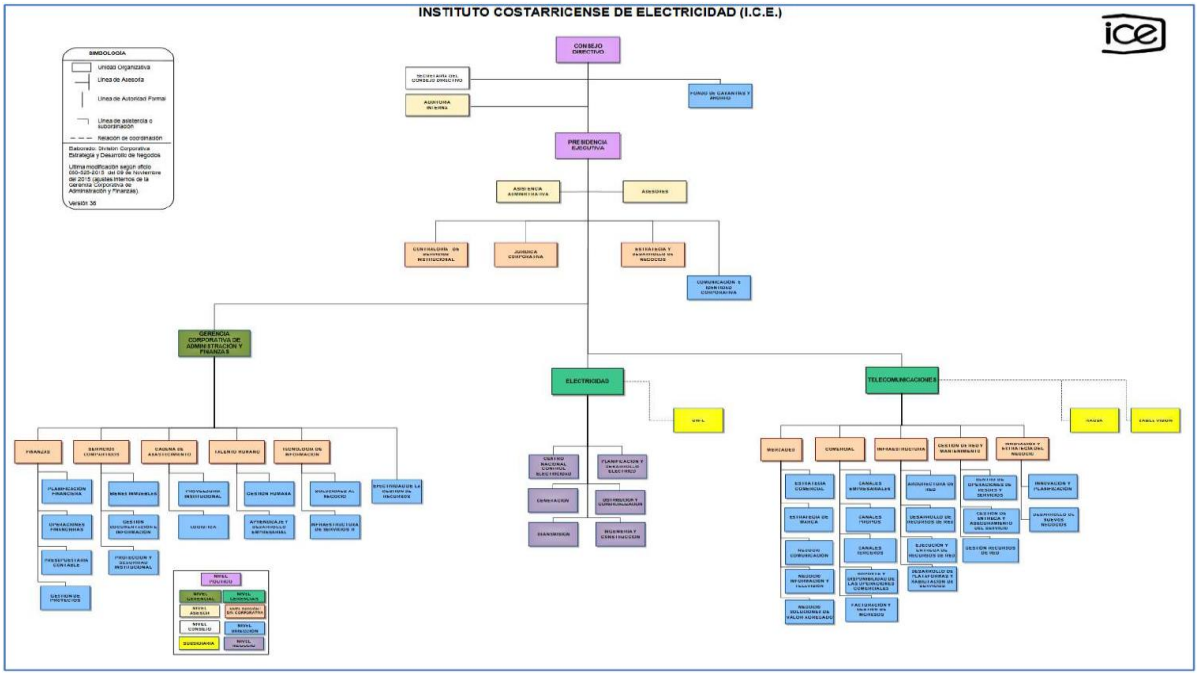


Figura 2 Organigrama Instituto Costarricense de Electricidad

Fuente Intranet Grupo ICE.

El ICE tiene al Sector de Telecomunicaciones como el área responsable de operar sus redes de transporte y de acceso, para brindar al mercado nacional e internacional soluciones de telecomunicaciones que soporte multiservicios de comunicación de nueva generación.

El ICE, en su misión de cubrir las necesidades y expectativas de los clientes y la sociedad costarricense, mediante el suministro oportuno de servicios y aplicaciones de

telecomunicaciones innovadoras, ve estrictamente necesario, proveer soluciones tecnológicas de vanguardia robustas y flexibles, que ayude a satisfacer las necesidades cambiantes de sus clientes. Esto mediante diversas plataformas, tecnologías y soluciones informáticas, que logren coexistir de forma integral sin provocar la afectación a los servicios de telecomunicaciones actuales, y garantice un crecimiento continuo de nuevos servicios de telecomunicaciones con miras a satisfacer la demanda comercial y los nuevos retos, como la actual competencia emergente en la sociedad costarricense.

El garantizar la operación adecuada de servicios de telecomunicaciones, no solo requiere de una adecuada interpretación de las necesidades comerciales de los clientes, sino, de crear sistemas informáticos que sirvan de apoyo a la estrategia del negocio, para así contar con los insumos necesarios para la toma de decisiones.

En el Sector de Telecomunicaciones, la División Infraestructura tiene el propósito de brindar soluciones de infocomunicaciones innovadoras, integrales, oportunas, y competitivas a nivel nacional; su fin es desarrollar e implementar una solución integral de la infraestructura de la red de telecomunicaciones, realizando una gestión eficiente en la infraestructura actual para optimizar e incorporar nuevos servicios y soluciones de telecomunicaciones.

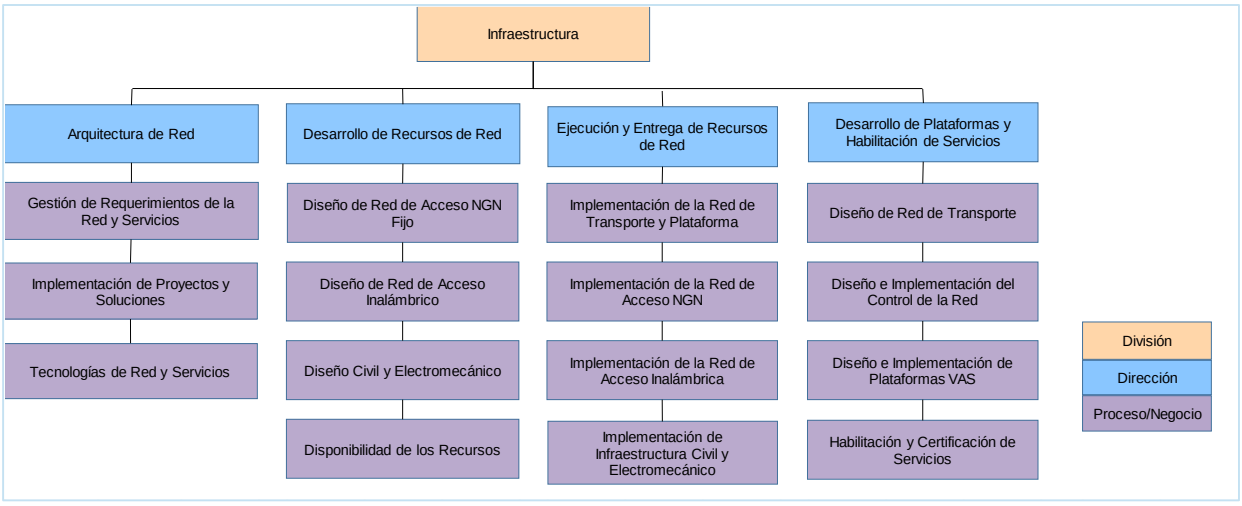


Figura 3. Organigrama División Infraestructura

Fuente: Elaboración propia.

El área Tecnologías de la información de la División Infraestructura, tiene como responsabilidad brindar las herramientas tecnológicas, la infraestructura de red y el soporte requeridos por los procesos adscritos a la División. Ver figura 3.

Por otra parte, esta área debe seguir el marco normativo de la División Corporativa de Tecnologías de la Información (DCTI), que contribuye, como parte de la gestión de las áreas de Tecnologías de la información, con la documentación referente a la implementación de las mejores prácticas, que proceden de diversas fuentes como estándares, mejores prácticas de la industria, tales como ITIL y COBIT y el conjunto de Normas Técnicas para la Gestión y el Control de las Tecnologías de Información, emitidas por la Contraloría General de la República.

Hoy día las tecnologías de la información y comunicación tienen gran impacto en el progreso, innovación y éxito de las empresas, específicamente su aporte en el desarrollo de *software*, el cual tendrá como propósito fundamental la correcta gestión de los datos, para que procesados se transformen en información de calidad y esta pueda servir de apoyo para la toma de decisiones.

Pressman (2010) expone lo siguiente:

> Hace 50 años, nadie hubiera podido predecir que el software se convertiría en una tecnología indispensable para los negocios, ciencias e ingeniería, ni que permitiría la creación de tecnologías nuevas (por ejemplo, ingeniería genética y nanotecnología), la ampliación de tecnologías ya existentes (telecomunicaciones) y el cambio radical de tecnologías antiguas (la industria de la impresión); tampoco que el software sería la fuerza que impulsaría la revolución de las computadoras personales, que productos de software empacados se comprarían en los supermercados, que el software evolucionaría poco a poco de un producto a un servicio cuando compañías de software “sobre pedido” proporcionaran funcionalidad justo a tiempo a través de un navegador web, que una compañía de software sería más grande y tendría más influencia que casi todas las empresas de la era industrial, que una vasta red llamada internet sería operada con software y evolucionaría y cambiaría todo, desde la investigación en bibliotecas y la compra de

> productos para el consumidor hasta el discurso político y los hábitos de encuentro de los adultos jóvenes (y no tan jóvenes). *(pág. 2)*.

Es imposible operar el mundo moderno sin *software*. El *software* está en todas partes, es un elemento que forma parte de la sociedad, ha facilitado muchas de las actividades cotidianas y ha incursionado en muchos campos; su auge sigue exponencialmente.

A pesar de lo anterior, muchas personas y empresas continúan desarrollando el *software* de manera empírica y no están conscientes de los métodos modernos. Como resultado, la calidad del *software* que producen es deficiente y esto contribuye a pérdidas económicas para las empresas ya que en muchas ocasiones se invierte este importante recurso para realizar ajustes incluso correcciones en el programa recién desarrollado. Asimismo, la fase de mantenimiento se torna compleja al carecer de estándares que faciliten este tipo de tareas.

El Consejo Nacional de Investigaciones Científicas y Técnicas de Argentina, en la celebración de la 39° Conferencia Mundial de Ingeniería de *Software*-ICSE 2017 menciona lo siguiente: “… un estudio de la Universidad de Cambridge estima que el 50% del tiempo del desarrollo de *software* se destina a localización y corrección de errores. Se calcula que esto impacta en la economía global en más de 300 mil millones de dólares anuales.”. (Consejo Nacional de Investigaciones Científicas y Técnicas de

Argentina., 2017). Considerar una metodología de desarrollo de *software* es indispensable para afianzar una disciplina que apunte a una ingeniería de *software* eficiente, permitiendo así desarrollar *software* sin errores, de mayor calidad, en un menor tiempo y lo más importante a un menor costo.

Para el autor Pressman (2010, pág. 3) algunas interrogantes que se plantean en torno a este tema de investigación son las siguientes:

- ¿Por qué se requiere tanto tiempo para terminar el *software*?
- ¿Por qué son tan altos los costos de desarrollo?
- ¿Por qué no podemos detectar todos los errores antes de entregar el *software* a nuestros clientes?
- ¿Por qué dedicamos tanto tiempo y esfuerzo a mantener los programas existentes?
- ¿Por qué seguimos con dificultades para medir el avance mientras se desarrolla y mantiene el *software*?

Las metodologías de desarrollo de *software* son las formas en las cuales se realiza la planeación para su desarrollo, básicamente con el propósito de mejorar, optimizar procesos y ofrecer una mejor calidad. Permiten enfocar esfuerzos en lo siguiente:

1. Reducir los costes del proceso de desarrollo del *software*.
2. Rápida adaptación del sistema a nuevas tecnologías.

3. Reducción del tiempo de desarrollo de nuevos sistemas informáticos.
4. Mejora de la calidad del *software*, permitiendo una fácil evolución del mismo.
5. Facilita las tareas de mantenimiento.

Actualmente los negocios operan en un entorno que cambia rápidamente. Tienen que responder a nuevas oportunidades y mercados, condiciones económicas cambiantes y la aparición de productos y servicios competidores. El *software* es parte de casi todas las operaciones de negocio, por lo que es importante que el nuevo *software* se desarrolle de manera más rápida para aprovechar las oportunidades y responder a la presión competitiva. De esto depende, en gran medida, el éxito de una empresa.

La naturaleza de evolución permanente de servicios de telecomunicaciones junto a las nuevas tecnologías, exige que se mantenga un constante desarrollo de sistemas informáticos, una metodología de desarrollo de *software* que proporcione soluciones informáticas alineadas a la estrategia del negocio y que además cumpla con las expectativas del cliente interno debe ser considerada por la Institución.

Según se define “Una metodología de desarrollo de *software* es un marco de trabajo que se usa para estructurar, planificar y controlar el proceso de desarrollo de sistemas de información.” (Instituto de Normas Técnicas de Costa Rica (INTECO), 2009, pág. 40). Además, una metodología de desarrollo de *software* no necesariamente es

adecuada para usar en todos los proyectos, cada una de las metodologías disponibles es más apropiada para tipos específicos de proyectos. La intensión de una metodología de desarrollo de *software* es realizar el proceso de la producción de *software* de manera eficiente, así como lograr alta calidad de una forma costeable, contribuyendo a la consecución de la estrategia del negocio.

El dinamismo del mercado de las telecomunicaciones ha traído nuevas tecnologías y la conformación de servicios más complejos, por lo cual el uso de una metodología de desarrollo de *software* que se adecue a las necesidades del negocio de la institución no solo garantiza la calidad en el *software*, también reducirá los tiempos de respuesta para implementar los aplicativos que sirven de soporte a la estrategia del negocio. Consecuentemente, la prestación de los servicios podrá comercializarse en forma más rápida y eficaz.

Lo anterior, conlleva al grupo de trabajo de Tecnologías de la Información, de la División Infraestructura, a emplear una metodología de desarrollo de *software* adecuada, enfocándose en las mejores prácticas, procedimientos, herramientas y técnicas existentes en la industria del *software*; así como el marco normativo definido por la DCTI.

1.5. ALCANCE Y LIMITACIONES.

El texto ofrece al área de Tecnologías de la Información, la elaboración de una propuesta de mejora para adecuar la metodología de desarrollo de *software* a las necesidades del negocio de la División Infraestructura, que actualmente se realiza en forma empírica; siendo esto un aporte valioso en la disminución de tiempos de implementación de soluciones informáticas, las cuales se encuentren alineadas a la estrategia de la División y consecuentemente al Sector Telecomunicaciones.

1.5.1. Alcances

1. El primer entregable es un diagnóstico de la situación actual de la metodología de desarrollo de *software* utilizada en Tecnologías de la Información de la División Infraestructura.
2. El segundo entregable del proyecto es la identificación de puntos de mejora para lograr aplicarlos a la nueva propuesta de metodología de desarrollo de *software*, mediante la investigación de las mejores prácticas, ampliamente usadas en la industria para analizar sus características y ser consideradas en el diseño de esta propuesta, las cuales brinden al área de Tecnologías de la Información las herramientas, técnicas, procedimientos y métodos adecuados para la consecución de los objetivos planteados, asimismo el alineamiento con la estrategia de la División Infraestructura y el Sector Telecomunicaciones.

3. El tercer entregable del proyecto es el diseño de las mejoras para la metodología de desarrollo de *software*, donde se apliquen los estándares, mejores prácticas y técnicas utilizadas ampliamente en la industria.
4. El cuarto entregable del proyecto consiste en aplicar todo el conocimiento adquirido durante la investigación en un ejercicio de desarrollo de *software*, para poner a prueba el diseño de mejora propuesto, y determinar si se adecua al negocio.

Exclusión:

i. La propuesta de mejora en la metodología de desarrollo de *software* tiene su ámbito de acción en el área de Tecnologías de la Información de la División Infraestructura, por lo que se excluye de esta propuesta un alcance organizacional completo.

1.5.2. Limitaciones.

Se consideran limitantes las siguientes:

- Marco normativo de la DCTI, considerando que contribuye con el conjunto general de normas, criterios, metodologías y lineamientos; así como con las herramientas de desarrollo.
- Por la facultad legal señalada en el artículo 35 de la ley No. 8660 Ley de Fortalecimiento y Modernización de las Entidades Públicas del Sector

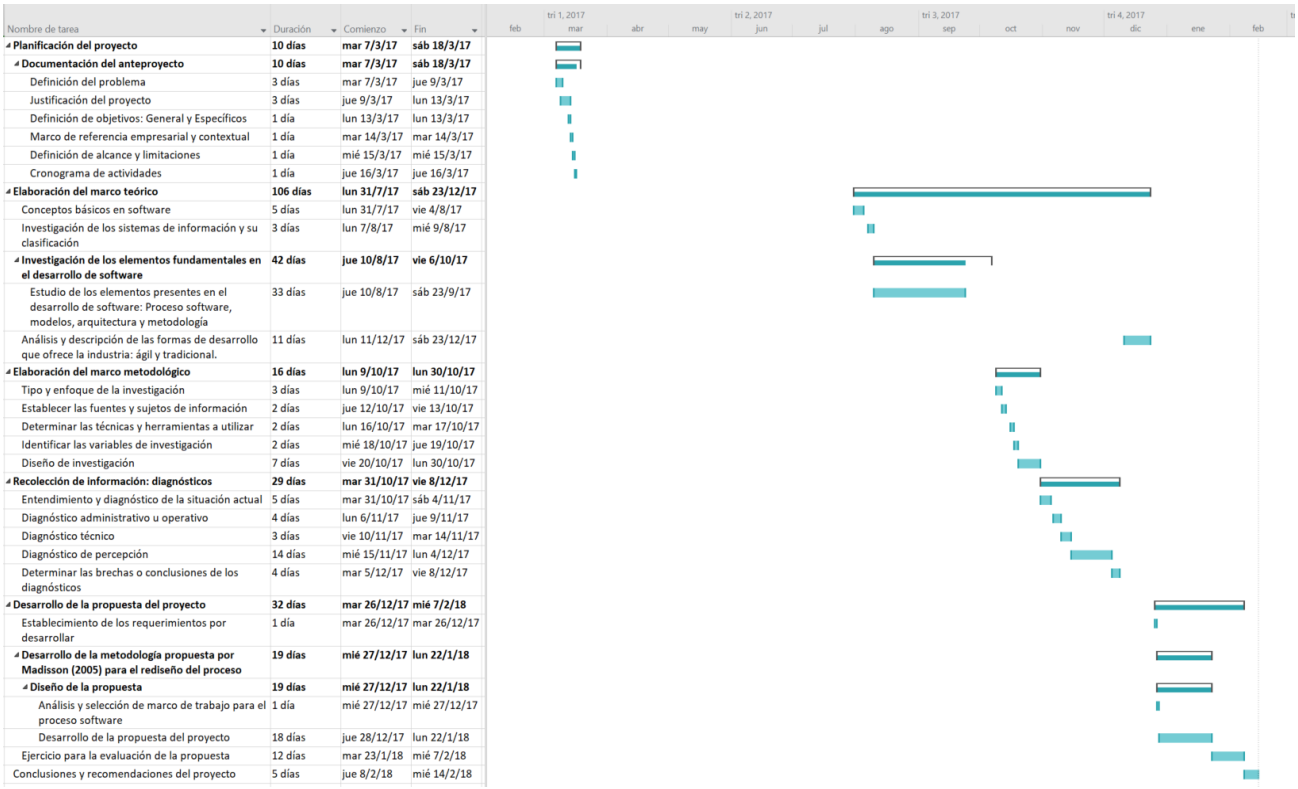
Telecomunicaciones sobre el manejo de la información confidencial, para la elaboración de este proyecto se utiliza información sensible y confidencial de la institución, la cual no podrá ser suministrada en la elaboración de este, más su análisis.

1.6. CRONOGRAMA DE ACTIVIDADES.

Esta sección consiste en la planificación de las actividades para llevar a cabo el proyecto. Con la finalidad de mantener un mejor control de las tareas que se realizarán en todas las etapas del proyecto, se elabora un cronograma de todas las actividades definidas en el alcance del proyecto.

El cronograma de actividades permitirá mantener tiempos establecidos para la elaboración y diseño de la propuesta del proyecto, para la revisión de cada fase y conseguir retroalimentación con el fin de mejorar las propuestas y soluciones.

A continuación se presenta el cronograma de actividades (ordenado por fecha), y cuenta con el tiempo de duración de cada actividad clasificado por etapas y acorde a cada objetivo específico, alcances del proyecto y entregables ya definidos en las anteriores secciones del proyecto.



Nombre de tarea	Duración	Comienzo	Fin
Planificación del proyecto	**10 días**	**mar 7/3/17**	**sáb 18/3/17**
Documentación del anteproyecto	**10 días**	**mar 7/3/17**	**sáb 18/3/17**
Definición del problema	3 días	mar 7/3/17	jue 9/3/17
Justificación del proyecto	3 días	jue 9/3/17	lun 13/3/17
Definición de objetivos: General y Específicos	1 día	lun 13/3/17	lun 13/3/17
Marco de referencia empresarial y contextual	1 día	mar 14/3/17	mar 14/3/17
Definición de alcance y limitaciones	1 día	mié 15/3/17	mié 15/3/17
Cronograma de actividades	1 día	jue 16/3/17	jue 16/3/17
Elaboración del marco teórico	**106 días**	**lun 31/7/17**	**sáb 23/12/17**
Conceptos básicos en software	5 días	lun 31/7/17	vie 4/8/17
Investigación de los sistemas de información y su clasificación	3 días	lun 7/8/17	mié 9/8/17
Investigación de los elementos fundamentales en el desarrollo de software	**42 días**	**jue 10/8/17**	**vie 6/10/17**
Estudio de los elementos presentes en el desarrollo de software: Proceso software, modelos, arquitectura y metodología	33 días	jue 10/8/17	sáb 23/9/17
Análisis y descripción de las formas de desarrollo que ofrece la industria: ágil y tradicional.	11 días	lun 11/12/17	sáb 23/12/17
Elaboración del marco metodológico	**16 días**	**lun 9/10/17**	**lun 30/10/17**
Tipo y enfoque de la investigación	3 días	lun 9/10/17	mié 11/10/17
Establecer las fuentes y sujetos de información	2 días	jue 12/10/17	vie 13/10/17
Determinar las técnicas y herramientas a utilizar	2 días	lun 16/10/17	mar 17/10/17
Identificar las variables de investigación	2 días	mié 18/10/17	jue 19/10/17
Diseño de investigación	7 días	vie 20/10/17	lun 30/10/17
Recolección de información: diagnósticos	**29 días**	**mar 31/10/17**	**vie 8/12/17**
Entendimiento y diagnóstico de la situación actual	5 días	mar 31/10/17	sáb 4/11/17
Diagnóstico administrativo u operativo	4 días	lun 6/11/17	jue 9/11/17
Diagnóstico técnico	3 días	vie 10/11/17	mar 14/11/17
Diagnóstico de percepción	14 días	mié 15/11/17	lun 4/12/17
Determinar las brechas o conclusiones de los diagnósticos	4 días	mar 5/12/17	vie 8/12/17
Desarrollo de la propuesta del proyecto	**32 días**	**mar 26/12/17**	**mié 7/2/18**
Establecimiento de los requerimientos por desarrollar	1 día	mar 26/12/17	mar 26/12/17
Desarrollo de la metodología propuesta por Madisson (2005) para el rediseño del proceso	**19 días**	**mié 27/12/17**	**lun 22/1/18**
Diseño de la propuesta	**19 días**	**mié 27/12/17**	**lun 22/1/18**
Análisis y selección de marco de trabajo para el proceso software	1 día	mié 27/12/17	mié 27/12/17
Desarrollo de la propuesta del proyecto	18 días	jue 28/12/17	lun 22/1/18
Ejercicio para la evaluación de la propuesta	12 días	mar 23/1/18	mié 7/2/18
Conclusiones y recomendaciones del proyecto	5 días	jue 8/2/18	mié 14/2/18

Figura 4. Cronograma del proyecto de tesis.

Fuente: Elaboración propia.

CAPITULO II: MARCO TEÓRICO

En este capítulo se explican los conceptos necesarios para el entendimiento y desarrollo de este proyecto. Además, se analizan los marcos de trabajo, metodologías y modelos de proceso *software* que contribuyen a una mejor comprensión del tema en estudio.

Por tanto, se definen los elementos básicos en *software*; la ingeniería de *software* como la disciplina de ingeniería que se interesa por todos los aspectos para la elaboración de sistemas.

Ahora bien, debido a la necesidad de contar con sistemas informáticos que satisfagan los diversos requerimientos de los usuarios finales, así como facilitar las actividades y automatizar los procesos para mejor los rendimientos de las empresas, el desarrollo de *software* ha aumentado significativamente en los últimos años. En efecto esta situación ha promovido que se clasifiquen los sistemas en dos tipos: *Enterprise Resourse Planning* (ERP) y sistemas hechos a la medida. Se expondrá cada uno.

Adentrando más en el tema de estudio, se describe el proceso *software*, el cual ha evolucionado para facilitar la elaboración de sistemas. Este punto se descompone en:

- Definición de proceso *software*, de esta manera se comprenderá que para la elaboración de un producto *software* es necesario contar con una secuencia de actividades.

- Descripción de las cuatro actividades básicas que componen el proceso *software*. Es importante conocerlas porque son la base para cualquier modelo de proceso *software* o metodología de desarrollo.
- Modelos de procesos *software*, la descripción simplificada de un proceso *software*. Esto es importante porque define cómo se resuelve la problemática del desarrollo del sistema. Se estudian los modelos: cascada, incremental, evolutivo, espiral y modelo de ingeniería de *software* orientada a la reutilización.
- Arquitectura para el desarrollo de *software*, se menciona el efecto en el desarrollo de sistemas al definir la estructura general, tanto su forma física como lógica en la que están construidos.
- Metodología de desarrollo de *software*, se explica su concepto, apuntando a la importancia de formalizar el estilo en que se realizan las actividades, adicionando rigurosidad y mejores prácticas que ofrece la industria del *software*.
- Desarrollo ágil de *software*, en contraste al desarrollo de *software* tradicional. Se analizan sus características y los marcos de trabajo Scrum y Programación extrema, consideradas como las más conocidas.
- Desarrollo de *software* tradicional, se investigan sus características, y la metodología más conocida RUP.

2.1. CONCEPTOS BÁSICOS EN *SOFTWARE*.

La ingeniería de *software* es una disciplina de ingeniería que se interesa por todos los aspectos de la producción de *software*, desde la etapa de la especificación del sistema hasta el mantenimiento del mismo, posterior a su puesta en operación. Es la aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación (funcionamiento) y mantenimiento de *software*; es decir, la aplicación de ingeniería de *software*. (IEEE Std 610.12-1990, 1990).

La ingeniería de *software* busca seleccionar el método más adecuado para un conjunto de circunstancias y, de esta manera, un acercamiento al desarrollo más creativo y formal, intentando la efectividad según ciertas situaciones.

Este enfoque sistemático es una tecnología con varias capas. Desde la perspectiva de Pressman (2010), cualquier enfoque de ingeniería, incluso la de *software*, debe basarse en un compromiso organizacional con calidad; este aspecto en conjunto con filosofías para la administración total de la calidad, como Six Sigma, Kaisen, entre otras; alimentan la cultura de mejora continua, y es esta cultura la que lleva en última instancia al desarrollo de enfoques más eficaces de la ingeniera de *software*. El fundamento en el cual se apoya esta ingeniería es el compromiso con la calidad. En la figura 5 se representa lo anterior.

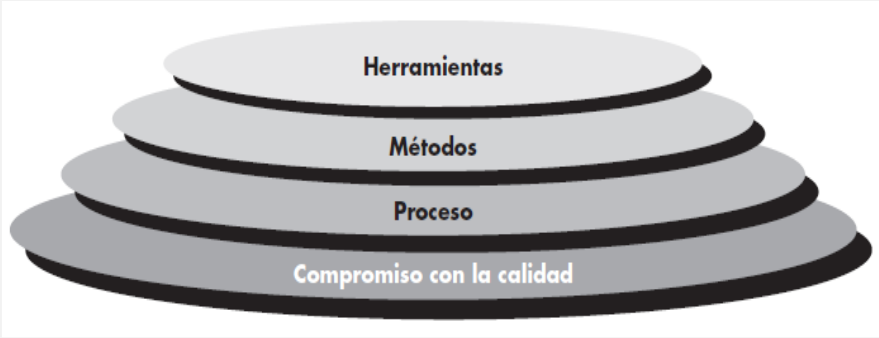


Figura 5, Capas de la ingeniería de *software*

Fuente: (S. Pressman, 2010, pág. 12).

Además Pressman (2010) explica que el proceso es el fundamento para la ingeniería de *software*, define una estructura que debe establecerse para la obtención eficaz de la tecnología de ingeniería de *software*, forma la base para el control de la administración de proyectos de *software*, y establece el contexto en el que se aplican métodos técnicos, se generan productos del trabajo (por ejemplo modelos, documentos, datos, reportes, formatos, entre otros.), igualmente se establecen puntos de referencia, se asegura la calidad y se administra el cambio de manera apropiada. Esta capa constituye el adhesivo para unir las capas de la tecnología y permite el desarrollo racional y oportuno del *software*.

Asimismo, este autor menciona que los métodos proporcionan la experiencia técnica para elaborar *software*, se basan en un conjunto de principios fundamentales que gobiernan cada área de la tecnología e incluyen actividades de modelación y otras técnicas descriptivas. Se consideran un conjunto amplio de tareas, como comunicación,

análisis de los requerimientos, modelación del diseño, construcción del programa, pruebas y apoyo.

Por su parte, las herramientas proporcionan un apoyo automatizado o semiautomatizado para el proceso y los métodos. Una adecuada integración de las herramientas da como resultado un sistema denominado ingeniería de *software*, este apoya el desarrollo de sistemas, de tal manera que la información que se produce en una capa pueda ser utilizada por otra.

Se define *software* como los programas informáticos, procedimientos y posiblemente la documentación y datos relacionados con el funcionamiento de un sistema informático (IEEE Std 610.12-1990, 1990). El enfoque de la ingeniería de *software* ofrece métodos y técnicas para desarrollar y mantener *software* de calidad que resuelva problemas de todo tipo. Esta disciplina abarca todas las fases del proceso de desarrollo de cualquier tipo de *software*, adaptable en áreas tales como los negocios, investigación científica, medicina, producción, logística, banca, entre otros; es decir, *software* de aplicación, diseñado para satisfacer los requerimientos específicos de un usuario.

La necesidad de contar con sistemas informáticos que satisfagan los diversos requerimientos de los usuarios finales, así como facilitar las actividades y automatizar los procesos para mejor los rendimientos de las empresas, ha generado un aumento significativo en el desarrollo de *software*. En específico, los sistemas de información han

venido a cambiar la forma en que operan las organizaciones actuales. A partir de su uso se obtienen importantes mejoras, como proveer la información que sirve de apoyo al proceso de toma de decisiones y, lo que es más importante, su implantación facilita el logro de ventajas competitivas.

Según (Cohen & Asís, 2009), en la actualidad podemos encontrarnos con dos tipos de programas o *software*, el primer tipo son todos los que comprenden un conjunto de subprogramas que permite interactuar con el sistema computacional, como ejemplos de esta clase de programas se encuentran los sistemas operativos como Windows, Linux o Solaris. Por otro lado, se encuentran los programas de aplicación, constituidos a su vez por subprogramas que resuelven problemas funcionales para todos sus usuarios, por ejemplo, Microsoft Office, o bien un sistema desarrollado para apoyar el proceso de toma de decisiones, también conocido como sistema transaccional.

2.2. SISTEMAS DE INFORMACIÓN (SI).

Partiendo de lo más básico, un sistema es un conjunto de elementos que interactúan entre sí para lograr un objetivo en común. Estamos rodeados de sistemas, por ejemplo, cualquier persona experimenta sensaciones físicas gracias a un complejo sistema nervioso central formado por el cerebro, los nervios, la médula espinal, entre otros componentes que funcionan en conjunto para transmitir y tener información del medio que nos rodea.

Ahora bien, los sistemas de información son un conjunto de componentes y procesos que interactúan entre sí, los cuales tienen la finalidad de procesar entradas (datos), almacenar archivos de datos relacionados con la organización, procesar, y producir información mediante reportes y otras salidas (Senn, 2001).

Los SI se pueden dividir en dos grandes categorías, uno de ellos es una solución creada por algún fabricante, estos reciben el nombre de *Enterprise Resourse Planning* (ERP), esta tiene como propósito responder a la mayor cantidad de requerimientos o procesos de una empresa para una tarea en específico, y se vende o entrega como producto ya terminado, quien compre la solución debe adaptar las funcionalidades que trae el SI a los procesos de la empresa.

Por otro lado, se encuentran los sistemas hechos a la medida, estos tardan más tiempo en su implementación ya que nacen a partir de una idea y lo que implica crearlos desde cero. Otro aspecto a considerar es que son más costosos debido al proceso de desarrollo que conllevan, sin embargo, su principal particularidad es que cumplen con todas las necesidades de la empresa, es decir, son personalizados y ajustados completamente. El tema del proyecto se enfocará en este tipo de SI.

2.2.1 *Enterprise Resource Planning (ERP).*

Existen muchas definiciones de distintos autores, no obstante, todas llegan al mismo punto. Se refiere a los sistemas y paquetes de *software* utilizados por las organizaciones para gestionar los procesos de negocio, como contabilidad, adquisiciones, gestión de proyectos y manufactura; es decir, es *software* con un comportamiento definido, que resuelven un problema determinado en una forma determinada. Estos sistemas ERP se integran y definen o modelan una variedad de procesos de negocio y permiten el flujo de datos entre ellos, además estos sistemas eliminan la duplicación de datos y proporcionan su integridad. (Oracle, 2016).

Los sistemas con esta característica ayudan a gestionar los procesos empresariales, automatizar procesos, además de coordinar la información en cada área del negocio, ya que utilizan una base de datos común y herramientas de gestión compartida.

Un ERP es un sistema ya creado por un fabricante, el cual cumple con una funcionalidad específica, algunos son flexibles y pueden cambiar un poco su comportamiento con base en las configuraciones que permita, pero en general se debe cumplir con procesos duros de gestión para que el *software* sea útil. De ser necesario, la empresa debe modificar su forma de trabajo (procesos de gestión del negocio) con el fin de adaptarse a los requisitos del *software* adquirido. Se puede citar como ejemplo, el paquete de Microsoft Office que contiene un procesador de texto (*Word*), hoja de cálculo (*Excel*), entre otros. También se puede citar Skype, este es un servicio de

mensajería y de video llamada que puede ser adquirido por las empresas para facilitar la comunicación de sus colaboradores.

La razón por la cual adquirir este tipo de sistemas es porque no es factible desarrollar un sistema personalizado para tareas específicas y para productos que ya existen, por ejemplo, las descritas anteriormente; además los costos inmersos en el proceso de desarrollo de *software* pueden resultar elevados en relación a adquirir un ERP.

2.2.2 Sistemas hechos a la medida.

Los sistemas hechos a la medida son aplicativos que cubren las necesidades específicas de la empresa, moldeándose a una ágil y eficiente operación. El *software* se adapta a los procesos de negocio que la empresa ya tiene consolidados (Thalú, 2013).

Para llevar a cabo el desarrollo de este tipo de sistemas se debe comprender muy bien los procesos del negocio, con el fin de identificar problemas o posibles afectaciones que se pueden presentar en el futuro y para que el producto *software* cumpla las expectativas del usuario final, así como obtener el máximo provecho y que este se refleje en el rendimiento de la empresa.

Una vez que se recolectan los requerimientos del sistema se continúa con las fases definidas en el proceso de *software*, para obtener el producto *software* deseado. Este proceso se explicará más adelante.

Siempre existirá el dilema ¿Qué es más conveniente, hacer el sistema o comprar algún *software* ya hecho? Pues la respuesta dependerá de las alternativas que tenga en su mercado y otros factores importantes, sin embargo, ninguna de las dos posibilidades debe descartarse, o dejarse como última alternativa, sin antes hacer un correcto análisis de lo requerido (MADO, 2012).

2.3. DESARROLLO DE *SOFTWARE*.

El desarrollo de *software* ha ido evolucionando con el paso de los años, procurando facilitar la metodología y manera de desarrollar para todos los involucrados en este proceso de desarrollo de *software*.

Dentro de los cambios más notables se encuentra en primer lugar los lenguajes de programación, los cuales se han ido simplificando, facilitando la tarea del desarrollador, donde inicialmente se encontraban lenguajes como Cobol o Ensamblador cuya programación era más extensa y compleja que la actual para realizar una simple tarea.

También se encuentra la interfaz que se muestra al usuario final, esta intenta ser más amigable y de fácil uso.

Por otro lado, las metodologías y proceso de *software* han evolucionado con el propósito de adaptarse en forma más ágil y rápida al entorno competitivo que afrontan las empresas, ajustándose a los procesos de negocio y al dinamismo del mercado y la tecnología, así como brindar comodidad al equipo de trabajo con el fin de obtener un producto *software* de calidad.

Asimismo, han surgido técnicas y buenas prácticas utilizadas en la industria del *software* para facilitar las tareas de desarrollo de *software*, potencializando aún más la eficiencia en el proceso de *software*.

A continuación, se detalla los temas relacionados con el desarrollo de *software*.

2.3.1. Proceso de *software*.

Las fases del ciclo de vida del desarrollo de *software* se contemplan en un enfoque sistemático que se conoce como proceso de *software*. “Un proceso de software es una secuencia de actividades que conducen a la elaboración de un producto de software.” (Sommerville, 2011, pág. 9). Este proceso es complejo, no hay uno ideal, la mayoría de

las organizaciones han diseñado sus propios procesos de desarrollo de *software*, los cuales se adecuan al entorno y necesidades del negocio.

Un producto *software* se define como el conjunto de los programas informáticos, procedimientos y posiblemente la documentación y datos asociados, que fueron diseñados para ser entregados al usuario final. (IEEE Std 610.12-1990, 1990).

El Instituto de Normas Técnicas de Costa Rica (INTECO) (2009) define un producto *software* como el conjunto de tres componentes:

- → Programas: Son conjuntos de instrucciones que proporcionan la funcionalidad deseada cuando son ejecutadas por el ordenador. Están escritos usando lenguajes específicos que los ordenadores pueden leer y ejecutar, por ejemplo, lenguaje ensamblador, Basic, FORTRAN, COBOL, C, Java, entre otros.
- → Datos: los programas proporcionan la funcionalidad requerida manipulando datos. Usan datos para ejercer el control apropiado en lo que hacen. El mantenimiento y las pruebas de los programas también necesitan datos. El diseño del programa asume la disponibilidad de las estructuras de datos tales como bases de datos y archivos que contienen datos.
- → Documentos: además de los programas y los datos, los usuarios necesitan también una explicación de cómo utilizar el programa. Estos pueden ser manuales de usuario final y de operación, son necesarios para permitir a los usuarios operar con el sistema; asimismo requeridos por las personas

encargadas de mantener el *software* para entender el interior del *software* y modificarlo, en el caso en que sea necesario.

El proceso de *software* corresponde a la ejecución de una serie de pasos predecibles, en forma de analogía, es un mapa de carretera que ayuda a orientar y brindar el camino adecuado para alcanzar el punto de destino deseado. En todos los procesos de *software* existen cuatro actividades básicas, Sommerville (2011) las define de la siguiente manera:

- ✓ Especificación del *software*: se define el *software* que se producirá y sus restricciones de operación.
- ✓ Desarrollo del *software*: contempla el diseño y la programación del *software*.
- ✓ Validación del *software*: se validan los requerimientos indicados en la especificación del *software*, para asegurar que sea lo que el cliente requiere.
- ✓ Evolución del *software*: corresponde al mantenimiento del sistema para mejorar sus funcionalidades.

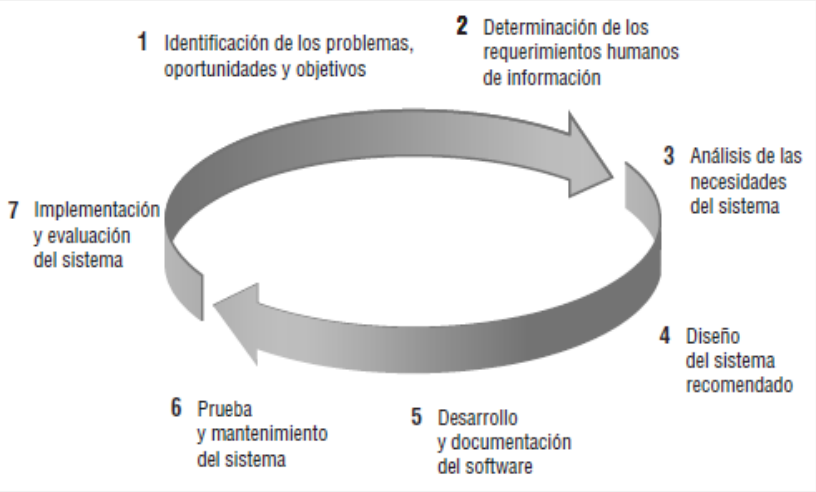


Figura 6. Representación Proceso de *Software*.

Fuente: (Kendall & Kendall, 2011, pág. 8).

En relación con la figura 6, el proceso de *software* convierte las necesidades del usuario en requerimientos de *software*, estos requerimientos son transformados en diseño mediante el respectivo análisis de necesidades del sistema; posteriormente el diseño es implementado en código (programación), y el código es probado, documentado y certificado para su uso operativo. Concretamente define quién está haciendo qué, cuándo hacerlo y cómo alcanzar un cierto objetivo. Asimismo, este ciclo requiere un conjunto de conceptos, una metodología y un lenguaje de programación propio (Kendall & Kendall, 2011).

En este sentido, Sommerville (2011) señala lo siguiente:

> Los procesos de software pueden mejorarse con la estandarización de los procesos, donde se reduce la diversidad en los procesos de software en una organización. Esto conduce a mejorar la comunicación, a reducir el

> tiempo de capacitación, y a que el soporte de los procesos automatizados sea más económico. La estandarización también representa un primer paso importante tanto en la introducción de nuevos métodos y técnicas de ingeniería de software, como en sus buenas prácticas *(pág. 29)*.

Cada organización posee su proceso de *software*, quizás incluyan técnicas obsoletas o tal vez no aprovechen las mejores prácticas de la industria de la ingeniería de *software*, por lo que no sacan provecho de los métodos que esta ingeniería ofrece; ocasionando, posiblemente, deficiencias en el producto *software*, incremento de costos, insatisfacción del cliente en relación al producto, alta duración para la implementación del sistema de información, entre otros.

2.3.2. Modelos de proceso de *software*.

Un proceso define quién hace qué, cuándo y cómo lo hace para alcanzar cierto objetivo, en el caso de *software* se requieren procesos especializados que abarquen desde la creación hasta la administración de un sistema de *software*. El desarrollo de *software* puede llegar a ser extremadamente complejo y para administrar la complejidad es necesario contar con modelos de procesos de *software* apropiados.

Un modelo de proceso de *software* define cómo resolver la problemática del desarrollo de *software*. Para desarrollar *software* se requiere resolver ciertas fases de un proceso que se conocen en su conjunto como el ciclo de vida del desarrollo de

software. Sommerville (2011) se refiere de la siguiente manera: "…un modelo de proceso de software es una representación simplificada de este proceso. Cada modelo del proceso representa a otro desde una particular perspectiva y, por lo tanto, ofrece solo información parcial acerca de dicho proceso." (pág. 29).

De esta manera un modelo de proceso de *software* es una descripción simplificada de un proceso del *software* que presenta una visión de este proceso, el propósito es explicar los diferentes enfoques del desarrollo del *software*. A modo de ejemplo, un modelo de flujo de datos, representa la entrada, transformación y salida de los datos en el proceso; pero quizá sin presentar los roles de las personas que intervienen en dicho proceso.

Los principales modelos de proceso de *software* son los siguientes:

1. Modelo cascada (*Waterfall*): Este modelo "…sugiere un enfoque sistemático y secuencial para el desarrollo del software". (S. Pressman, 2010, pág. 34). Toma las actividades fundamentales del proceso de especificación, desarrollo, validación y evolución; luego, los representa como fases separadas del proceso, tales como: especificación de requerimientos, diseño de *software*, desarrollo y pruebas, implementación y por último operación y mantenimiento. La estrategia principal es seguir el progreso del desarrollo de *software* hacia puntos de revisión bien definidos (llamados *checkpoints*), mediante la entrega programadas con fechas precisas.

Su nombre, modelo en cascada, se debe a que “La siguiente fase no debe comenzar sino hasta que termine la fase previa. En la práctica, dichas etapas se traslapan y se nutren mutuamente de información” (Sommerville, 2011, pág. 31). Por lo que es posible que en determinado momento del proceso se presenten estados de bloqueo, en los que los miembros del equipo del proyecto deben esperar a otros a fin de terminar las tareas interdependientes.

Se recomienda su uso cuando los requerimientos se entiendan bien y sea improbable el cambio radical durante el desarrollo del sistema ya que por la naturaleza del proceso la implementación del sistema se realiza hasta completar sus fases. Esto indudablemente implicaría un retroceso a la etapa inicial, repercutiendo en tiempo y costo; además puede significar que el sistema no hará lo que el usuario desea, esta situación también podría conducir a sistemas mal estructurados conforme los problemas de diseño se evaden con la implementación de trucos. (Sommerville, 2011).

Este proceso tiene como característica que no se muestra una etapa explícita de documentación dado que esta se lleva a cabo en el transcurso de todo el desarrollo.

Este modelo también es llamado ciclo de vida clásico ya que es el paradigma más antiguo de la ingeniería de *software*, fue desarrollado entre las décadas de los años 60 y 70.

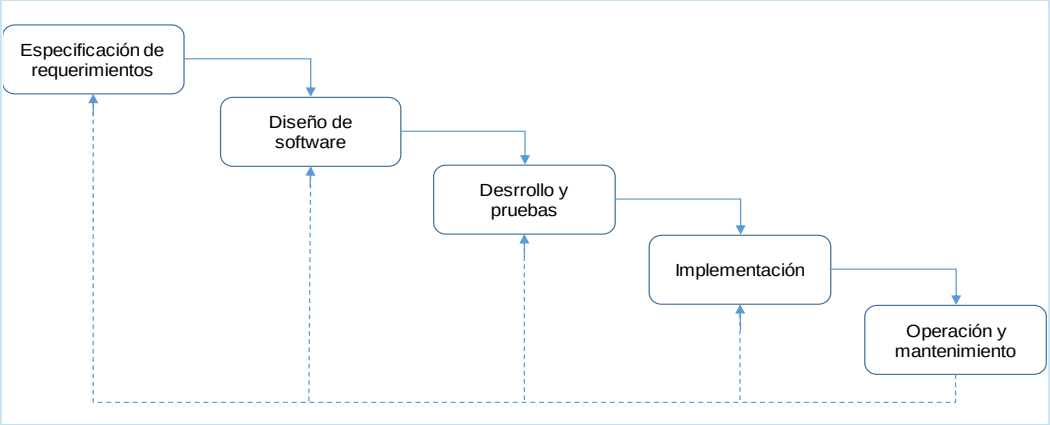


Figura 7. Representación del modelo en cascada.

Fuente: Elaboración propia.

2. Modelo desarrollo incremental: Se basa en la idea de diseñar una implementación inicial, para mostrarla al usuario y recibir retroalimentación sobre el trabajo de desarrollo que se realizó, la siguiente implementación o incremento incluye la modificación del producto fundamental para cumplir con las necesidades del cliente, así como la entrega de características adicionales y más funcionalidad. El proceso se repite después de entregar cada incremento, hasta terminar el producto final (Sommerville, 2011).

Por lo general, los primeros incrementos del sistema incluyen la función más esencial o la más urgente (Ortiz, 2012). Esto significa que el cliente puede evaluar el desarrollo del sistema en una etapa relativamente temprana, para constatar si se entrega lo que se requiere. En caso contrario, solo el incremento actual debe cambiarse y, posiblemente, definir una nueva función para incrementos posteriores.

Por otra parte, las actividades de especificación, desarrollo y validación (pruebas) están ligadas en vez de separadas, y se presenta una rápida retroalimentación a través de las actividades, este aspecto representa un aporte significativo para los sistemas de información utilizados en empresas, ya que no se trabaja en una solución completa del problema, más bien se avanza en una serie de pasos hacia una solución y se retrocede cuando se detecta que se cometieron errores, de esta manera resulta más barato y fácil realizar cambios en el *software* conforme este se diseña.

En la siguiente imagen se muestra como las actividades concurrentes: especificación, desarrollo y validación (pruebas) se realizan durante el desarrollo de las versiones hasta llegar al producto final.

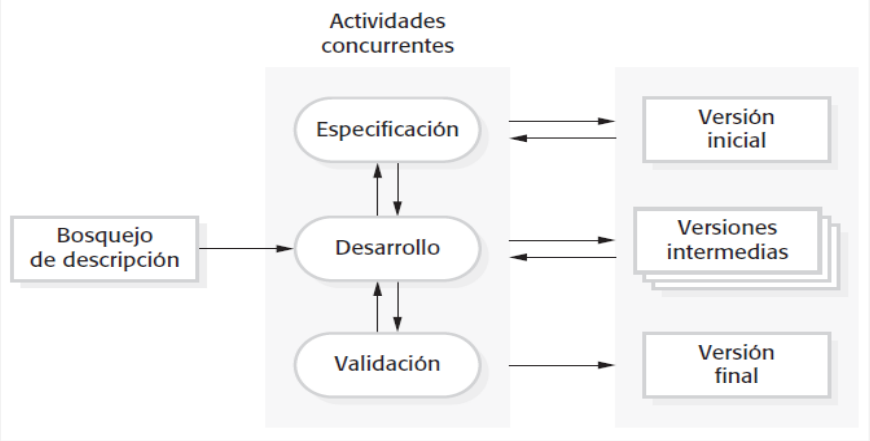


Figura 8. Representación del modelo desarrollo incremental.

Fuente: (Sommerville, 2011, pág. 33).

3. Modelo evolutivo: El *software*, como todos los sistemas complejos, evoluciona en el tiempo. Es muy común que los requerimientos del negocio y del producto cambien conforme se lleva a cabo el desarrollo, lo que hace que no sea realista trazar una trayectoria rectilínea hacia el producto final, a su vez los tiempos cortos del mercado, aunado a la posible competencia, hacen que sea imposible la terminación de un *software* perfecto, lo que conlleva al lanzamiento de una versión limitada con el propósito de aliviar la presión ejercida por la competencia o el entorno del negocio.

En estas situaciones donde se comprende bien el conjunto de requerimientos o el producto básico, pero los detalles del producto o extensiones del sistema aún están por definirse se necesita de un modelo de proceso diseñado explícitamente para adaptarse a un producto que evoluciona con el tiempo. Los modelos evolutivos son iterativos, se caracterizan por la manera en la que permiten desarrollar versiones cada vez más completas del *software*. Se presentan dos modelos comunes en este proceso evolutivo:

a) Prototipos: Facilita la comprensión de lo que hay que elaborar cuando los requerimientos no están claros. Por lo general el cliente define un conjunto de objetivos generales para el *software*, pero no identifica los requerimientos detallados para las funciones y características; en otros casos, el analista no está seguro de haber comprendido los requerimientos o de alguna particularidad del sistema; en estas situaciones, y muchas otras, optar por prototipos tal vez ofrezca el mejor enfoque. (Castellanos, Chacón , & Carvajal, 2016).

El proceso consiste en identificar los objetivos generales del *software*, así como definir los requerimientos imprescindibles. Seguidamente se planea rápidamente una iteración para hacer el prototipo y se lleva a cabo el modelado (“diseño rápido”). Este se enfoca en la presentación de aquellos aspectos del *software* que serán visibles para los usuarios finales (se muestran aspectos de interfaz humana, formatos de la pantalla de salida, entre otros.). Este se entrega y es evaluado por los participantes (clientes) para que brinden la retroalimentación que permitirá mejorar los requerimientos.

La iteración ocurre a medida que el prototipo es afinado para satisfacer las necesidades de los distintos participantes, y al mismo tiempo le permite al analista entender mejor lo que se necesita hacer. La intención es que el prototipo sirva como mecanismo para identificar los requerimientos del *software*. (S. Pressman, 2010).

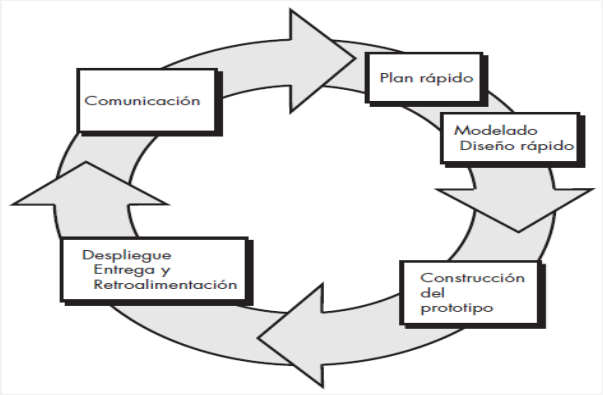


Figura 9. Representación modelo por prototipo.

Fuente: (S. Pressman, 2010, pág. 37).

b) Espiral: Su principal característica es hacer un desarrollo rápido de versiones cada vez más completas. Además, se acopla con la naturaleza iterativa de hacer prototipos con los aspectos controlados y sistemáticos del modelo de cascada, es decir, el *software* se desarrolla en una serie de entregas evolutivas. (Grijalva, 2012).

Se basa en una estrategia para reducir el riesgo del proyecto en áreas de incertidumbre. Durante las primeras iteraciones, lo que se entrega puede ser un modelo o prototipo, en las iteraciones posteriores se producen versiones cada vez más completas del sistema. Estas actividades se realizan en forma estructural. (S. Pressman, 2010).

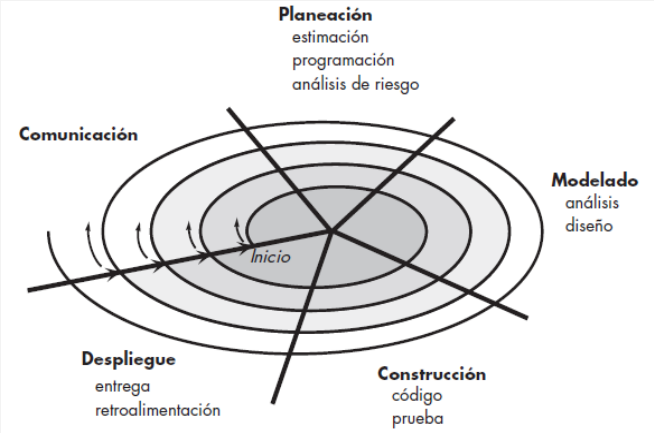


Figura 10. Representación modelo evolutivo.

Fuente: (S. Pressman, 2010, pág. 39).

El proceso consiste en que el equipo de desarrollo de *software* realiza actividades implícitas en un circuito alrededor de la espiral en el sentido de las manecillas del reloj, partiendo del centro. El riesgo se considera conforme se desarrolla cada revolución, esto permite que el desarrollador y cliente comprendan y reaccionen mejor ante los riesgos; y lo más importante, aplicar el enfoque de hacer prototipos en cualquier etapa de la evolución del producto *software*, así como el enfoque de escalón sistemático sugerido en el ciclo de vida clásico, pero lo incorpora en una estructura iterativa que refleja al mundo real en una forma más realista. En cada paso evolutivo se marcan puntos de referencia puntuales: combinación de productos del trabajo y condiciones que se encuentran a lo largo de la trayectoria de la espiral.

El primer circuito alrededor de la espiral da como resultado el desarrollo de una especificación del producto, las vueltas sucesivas se usan para desarrollar un prototipo y luego, versiones cada vez más sofisticadas del *software*. Cada paso por la región de planeación da como resultado ajustes en el plan del proyecto. El costo y la programación de actividades se ajustan en relación a la retroalimentación obtenida del cliente después de la entrega. Conforme se avanza el director del proyecto ajusta el número planeado de iteraciones que se requieren para finalizar el producto *software*.

4. Modelo ingeniería de *software* orientada a la reutilización: Es la reutilización de diseños o códigos que son similares a lo que se requiere; una vez identificados los componentes, se modifican según se necesite y se integran al sistema, en vez de desarrollarlo desde cero (Sommerville, 2011).

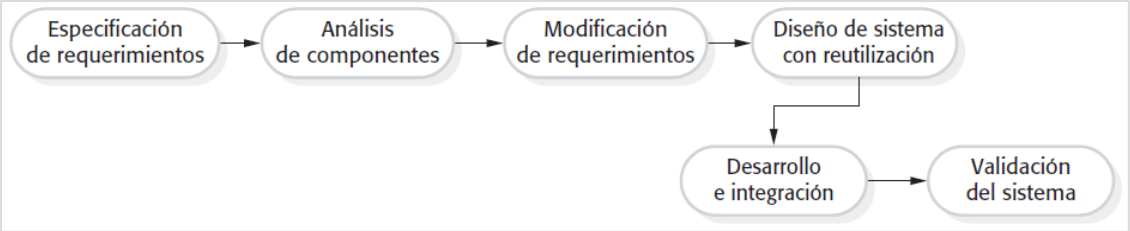


Figura 11. Representación modelo ingeniería de software orientada a la reutilización.

Fuente: (Sommerville, 2011, pág. 35).

El proceso consiste en que una vez dadas las especificaciones de requerimientos, se realiza una búsqueda de componentes para implementar dicha especificación. Es muy probable que no exista coincidencia exacta y los componentes proporcionen solo una parte de la funcionalidad requerida. Seguidamente, los componentes descubiertos son analizados y se modifican para reflejar los componentes disponibles. En los casos donde las especificaciones son imposibles, puede regresarse a la actividad de análisis de componentes para buscar soluciones alternativas.

En la etapa de diseño de sistema con reutilización se diseña el marco conceptual del sistema o se reutiliza un marco conceptual existente. Los creadores toman en cuenta los componentes que se reutilizan y organizan el

marco de referencia para atenderlo. Es posible que deba diseñarse algo de *software* nuevo, si no están disponibles los componentes reutilizables. Por último, se integran los componentes para crear el nuevo sistema. La integración del sistema puede ser parte del proceso de desarrollo, en vez de una actividad por aparte.

Según Sommerville (2011), "…la reutilización tiene la clara ventaja de reducir la cantidad de software a desarrollar y, por lo tanto, la de disminuir costos y riesgos; por lo general, también conduce a entregas más rápidas del software" (pág. 36). También menciona que "…son inevitables los compromisos de requerimientos y esto conduciría hacia un sistema que no cubra las necesidades reales de los usuarios" (pág. 36). Además, señala la posibilidad de perder algo de control sobre la evolución del sistema, considerando que las nuevas versiones de los componentes reutilizables no estén bajo el control de la organización que los usa.

2.3.3. Arquitectura para el desarrollo de *software*.

Según Weitzenfeld & Guardati (2008) la arquitectura de *software* define la estructura general de un sistema de información. Las arquitecturas varían de acuerdo con el tipo de sistema a desarrollarse. Estas arquitecturas pueden basarse en elementos sencillos o en componentes prefabricados de mayor tamaño. Además, elegir una arquitectura afecta aspectos como la extensibilidad del sistema, es decir, qué tan fácil es extenderlo

en el futuro para incorporar más funcionalidad o mayor capacidad. Por lo tanto, la arquitectura debe ser escogida de manera que minimice los efectos de los cambios que pueda hacer en el futuro en el sistema.

Por su parte, Sánchez (2015) expone que la arquitectura de *software* se refiere básicamente a la forma, tanto física, como lógica en la que están creadas los sistemas; señala además la importancia y las repercusiones que tiene una arquitectura de *software* bien definida en el éxito o fracaso de los proyectos.

En otras palabras, una arquitectura de desarrollo de sistemas pretende facilitar el desarrollo de *software*, ya que define la forma de trabajar de un sistema, cómo se deben manejar los datos y los componentes, cómo es la interacción del sistema y posibles soluciones para problemas similares. Estos aspectos agilizan el trabajo de los desarrolladores ya que se cuenta con un marco de trabajo definido, además en caso de ser necesario realizar algún mantenimiento al sistema, resulta más sencillo para cualquier persona dar una solución eficiente considerando que todo se maneja sobre un mismo estándar, facilitando así la comprensión de lo que se ejecutó en su momento y de esta manera corregirlo.

2.3.4. Metodología de desarrollo de *software*.

La constante innovación tecnológica hace que cada vez más sea necesaria la aplicación de nuevas metodologías adaptadas a los tiempos modernos y requerimientos

de los usuarios, es ineludible seguir ciertas pautas predefinidas en el desarrollo del *software*, es decir, llevar un comportamiento metódico: seguir una metodología. Se puede decir que la ausencia de una metodología en el desarrollo de un proyecto de *software* garantiza con seguridad la ausencia de calidad. (Vic, 2015).

El proceso de *software* se compone de una serie de tareas para obtener un producto de *software*, esas tareas pueden ser resueltas de diversas maneras, mediante distintas herramientas, métodos y técnicas; a su vez definiendo quién debe realizarla, cuándo debe ser concluida, qué actividades preceden o continúan, qué documentación se utilizará para llevar a cabo determinada tarea.

Estos aspectos pueden ser considerados detalles organizativos o bien un estilo de realizar las cosas, pero si este simple estilo se formaliza adicionando algo de rigurosidad y normas empleadas en la industria del *software* se obtiene una metodología. Así, los integrantes de un equipo de desarrollo de *software* deben seguir un criterio común al ejecutar las actividades o tareas del proceso de *software*, por lo que ese criterio, esa manera común, es una metodología de desarrollo de *software*.

Existe una serie de metodologías de desarrollo de *software* y ninguna es claramente superior a las demás, todas estas son, en esencia, bien intencionadas. Según las circunstancias que se enfrente es aplicable una o la otra, evidentemente las más modernas responden a problemas y necesidades más actuales.

2.3.5. Desarrollo ágil de *software*.

Conforme la tecnología digital se convertía en el nuevo epicentro de la economía mundial, en los años 80 se fue desarrollando y afianzando una metodología muy estructurada para el desarrollo de software, denominada modelo en cascada. Esta forma de gestionar los proyectos pronto se encontró con el deseo de los desarrolladores al momento de trabajar de manera menos lenta y burocrática.

Lo anterior propició lo que hoy se conoce como desarrollo ágil de *software*, una tendencia en la que el desarrollo iterativo e incremental se impone a las gestiones habituales en la industria de *software*. De manera más puntual, consiste en una metodología en la que el desarrollador va adaptando sus soluciones a unos requisitos también cambiantes a lo largo del tiempo. (Iglesias Fraga, 2016).

En relación con el modelo en cascada, el cual consiste en cinco fases (Especificación de requerimientos, Diseño de software, Desarrollo y pruebas, Implementación y por último Operación y mantenimiento) el desarrollo ágil de software se basa en seis fases: Planificación, Especificación de requisitos, Diseño, Desarrollo, Pruebas y Documentación. Si bien es cierto consta una fase más, sin embargo, su principal característica es la iteración, lo que permite realizar entregables de aquellas funcionalidades que aportan valor para el usuario, evitando la espera para la entrega

del producto software hasta finalizar el desarrollo de todos los requerimientos del sistema.

Se puede definir el desarrollo ágil de software como el conjunto de técnicas dirigidas a la gestión de proyectos, que han surgido como contraposición a los métodos clásicos de gestión. Según Pérez Alabarce (2016) todas las metodologías que se consideran ágiles cumplen con el Manifiesto Ágil, que no es más que una serie de principios agrupados en cuatro valores:

1) Los individuos y su interacción por encima de los procesos y herramientas.
2) El software funcional, frente a la documentación exhaustiva.
3) La colaboración con el cliente sobre la negociación contractual.
4) Respuesta ante el cambio por encima de seguir un plan.

Además, este autor acota que en definitiva se trata de eliminar todos los pasos o tareas innecesarias, también de impulsar una mayor eficiencia de todo el equipo involucrado en el desarrollo de *software*.

Como se indica anteriormente, los valores se sustentan de una serie de principios, los cuales se describen en el Manifiesto Ágil, a continuación el detalle según la organización Agile Alliance (Manifesto for Agile Software Development, s.f.):

1) Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de *software* que aporte valor.
2) Son bienvenidos los requisitos cambiantes, incluso cuando su llegada es tardía en el proceso de desarrollo. Los procesos ágiles aprovechan el cambio para obtener ventajas competitivas para el cliente.
3) Entregar con frecuencia *software* que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia a periodos de tiempo más corto posible.
4) Las personas de negocios y los desarrolladores deben trabajar juntos todos los días durante el proyecto.
5) Construir proyectos en torno a individuos motivados. Hay que proveer el entorno adecuado y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6) El método más eficiente y eficaz de transmitir información, en dos vías (ida y vuelta), dentro de un equipo de desarrollo es mediante la conversación cara a cara.
7) El *Software* que funcione es la principal medida de progreso.
8) Los procesos ágiles promueven el desarrollo sostenible. Los patrocinadores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma indefinida.
9) La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.
10) Simplicidad, o el arte de maximizar la cantidad de trabajo no realizado. Es esencial.
11) Las mejores arquitecturas, requisitos y diseños emergen de equipos autoorganizados.
12) A intervalos regulares, el equipo reflexiona sobre cómo ser más efectivo, para ajustar y perfeccionar consecuentemente su comportamiento.

La figura 12 muestra algunos ejemplos de metodologías ágiles.

Figura 12. Ejemplos de metodologías de desarrollo ágil de software.

Fuente: (Pérez Alabarce, 2016).

A continuación, los beneficios de aplicar desarrollo ágil de software:

- Mayor calidad del producto: En el desarrollo ágil, las pruebas se integran durante el ciclo, lo que significa que hay revisiones regulares para ver si el producto está funcionando durante el desarrollo. Esto permite al propietario del producto hacer cambios si es necesario y hace que el equipo esté al tanto de cualquier problema.
- Mayor satisfacción del cliente: El propietario del producto siempre está involucrado, el progreso del desarrollo tiene una gran visibilidad y la flexibilidad para cambiar es muy importante. Esto implica compromiso y satisfacción del cliente.
- Aumenta la productividad: al procesar tareas lo más simple y rápidamente posible, así como liberar el talento de cada miembro del equipo de trabajo.

- Simplicidad: Los equipos que trabajan sobre normas, lineamientos y otro tipo de regulación han de validar su trabajo constantemente, lo cual representa un doble sentido de trabajo. Las metodologías por iteración simplifican el proceso de entrega versus validación, este aspecto también permite adoptar cambios sobre la marcha del alcance del proyecto.
- Reducción de riesgos: Las técnicas ágiles prácticamente eliminan las posibilidades de una falla absoluta del proyecto. Este método de desarrollo significa tener siempre un producto en funcionamiento, comenzando con la primera iteración, para que ningún proyecto falle por completo. También da libertad cuando se deben implementar nuevos cambios. Se pueden implementar a muy bajo costo debido a la frecuencia de los nuevos incrementos que se producen.
- Lanzamientos más rápidos: El hecho de que el desarrollo ágil sea iterativo significa que las características o funcionalidades se entregan de manera incremental. Por lo tanto, los beneficios se obtienen anticipadamente mientras el producto se encuentra en el proceso de desarrollo.

Los largos ciclos de entrega a menudo son un problema para las empresas, especialmente en los mercados en rápido movimiento. El desarrollo bajo metodologías ágiles, significa lanzamientos rápidos de productos y la capacidad de medir la reacción del cliente y modificarlos en consecuencia, manteniéndolo por delante de la competencia.

2.3.5.1 Scrum.

Scrum es un marco de trabajo para el desarrollo y el mantenimiento de productos complejos, no es un proceso o una técnica para construir productos; en lugar de eso, es un marco de trabajo dentro del cual se pueden emplear varios procesos y técnicas. Scrum muestra la eficacia relativa de las prácticas de gestión de producto y las prácticas de desarrollo de modo que podamos mejorar. (Schwaber & Sutherland, 2016, pág. 3).

Según la definición anterior, este proceso de gestión y control reduce la complejidad en el desarrollo de productos para satisfacer las necesidades de los clientes; es simple y promueve la colaboración en los equipos para lograr desarrollar productos complejos.

Como se menciona, Scrum es simple, no es una colección de partes y componentes obligatorios definidos de manera prescriptiva. Scrum no es una metodología; está basado en un modelo de proceso empírico, con respecto a las personas y basado en la auto-organización de los equipos para lidiar con lo imprevisible y resolver problemas complejos inspeccionando y adaptando continuamente. (Francia, 2017).

Por otra parte según La Guía de Scrum (2016, pág. 4), Scrum se basa en la teoría de control de procesos empírica o empirismo. El empirismo asegura que el conocimiento procede de la experiencia y de tomar decisiones basándose en lo que se

conoce. Scrum emplea un enfoque iterativo e incremental para optimizar la predictibilidad y el control del riesgo.

Tres pilares soportan toda la implementación del control de procesos empírico:

- Transparencia: Los aspectos significativos del proceso deben ser visibles para aquellos que son responsables del resultado. La transparencia requiere que dichos aspectos sean definidos por un estándar común, de tal modo que los observadores compartan un entendimiento común de lo que se están viendo; esto quiere decir que todos los participantes manejen un lenguaje común para referirse al proceso.
- Inspección: Los usuarios de Scrum deben inspeccionar frecuentemente los artefactos de Scrum y el progreso hacia un objetivo para detectar variaciones indeseadas. Estas inspecciones no deben ser tan frecuentes para evitar interferencias en el trabajo. Las inspecciones son más enriquecedoras cuando se realiza por inspectores expertos en el mismo lugar de trabajo.
- Adaptación: En caso que un inspector detecte que uno o más aspectos de un proceso se desvían de límites aceptables y que el producto resultante será inaceptable, el proceso o material que está siendo procesado deben ajustarse. Este ajuste debe hacerse cuanto antes para minimizar desviaciones mayores.

Scrum prescribe cuatro eventos formales, contenidos dentro del *Sprint*, para la inspección y adaptación: Planificación del *Sprint* (*Sprint Planning*), Scrum Diario (*Daily*

Scrum), Revisión del *Sprint* (*Sprint Review*) y Retrospectiva del *Sprint* (*Sprint Retrospective*).

2.3.5.1.1 Valores.

Cuando el equipo Scrum incorpora y vive los valores de compromiso, coraje, foco, apertura y respeto, los pilares de Scrum de transparencia, inspección y adaptación se materializan y fomentan la confianza en todas las personas. Los miembros del equipo Scrum aprenden y exploran estos valores a medida que trabajan en los eventos, roles y artefactos Scrum (La Guía de Scrum, 2016, pág. 4).

El marco de trabajo Scrum consiste en los roles, eventos, artefactos y reglas asociadas. Cada componente dentro del marco de trabajo sirve a un propósito específico y es esencial para el éxito de Scrum y para su uso. Las reglas de Scrum relacionan los eventos, roles y artefactos, gobernando las relaciones e interacciones entre ellos.

2.3.5.1.2 Roles.

A continuación, se detallan los roles utilizados en este marco de trabajo.

2.3.5.1.2.1 El equipo Scrum (Scrum *Team*).

El equipo Scrum está conformado por Dueño del Producto (Product Owner), el Equipo de Desarrollo (Development Team) y un Scrum Master. Los equipos Scrum son:

- → Autoorganizados: porque eligen la mejor forma de llevar a cabo su trabajo y no son dirigidos por personas externas al equipo.
- → Multifuncionales: tienen todas las competencias necesarias para llevar a cabo el trabajo sin depender de otras personas que no son parte del equipo.

Los equipos Scrum entregan productos de forma iterativa e incremental, maximizando las oportunidades de obtener retroalimentación. Estas entregas de producto terminado aseguran que siempre estará disponible una versión potencialmente útil y funcional del producto.

2.3.5.1.2.2 El dueño de producto (*Product Owner*).

El dueño de producto es el responsable de maximizar el valor del producto y el trabajo del equipo de desarrollo (Schwaber & Sutherland, 2016, pág. 5), además es la única persona responsable de gestionar la lista del producto (*Product Backlog*). Para simplificar la comunicación y toma de decisiones el dueño de producto es una única persona, no un comité; sin embargo, este podría representar los deseos de un comité

en la lista del producto, pero aquellos que quieran cambiar la prioridad de un elemento de la lista deben hacerlo a través del dueño del producto.

Para que el dueño de producto logre hacer bien su trabajo es fundamental que toda la organización respete sus decisiones, y que además el equipo de desarrollo actúe exclusivamente en el contenido y en la priorización de la lista del producto definida por el dueño de producto. La coordinación con el dueño de producto es indispensable.

2.3.5.1.2.3 El Equipo de desarrollo (*Development Team*).

Grupo de personas que de manera conjunta desarrollan el producto del proyecto. Tienen un objetivo común, comparten la responsabilidad del trabajo que realizan (así como de su calidad) en cada iteración y en el proyecto.

El tamaño óptimo del equipo de desarrollo es lo suficientemente pequeño como para permanecer ágil y lo suficientemente grande como para completar una cantidad de trabajo significativa. Por debajo de 5 personas cualquier imprevisto o interrupción sobre un miembro del equipo compromete seriamente el compromiso que han adquirido y, por tanto, el resultado que se va a entregar al cliente al finalizar la iteración. Por encima de nueve personas, la comunicación y colaboración entre todos los miembros se vuelve

más difícil y se forma subgrupos; es decir, se requiere demasiada coordinación. (proyectosagiles.org, s.f.).

2.3.5.1.2.4 El Scrum Master.

Según Pérez (2017) el Scrum Master es el responsable de asegurar que se cumplan las buenas prácticas, teorías, valores, reglas descritos en el modelo Scrum. Tiene una figura similar a la de un *coach*/mentor que acompañará al equipo durante todo el desarrollo del proyecto y asegurará que se cumplan las buenas prácticas, actuando como un facilitador y solucionador de problemas (esto no significa que siempre los resuelva él, pero sí de ocuparse para que se resuelvan).

Además, el Scrum Master elimina los obstáculos que dificultan al equipo para lograr el objetivo del sprint. El Scrum Master potencia la productividad.

Entre otras funciones el Scrum Master debe:

- Asesorar y formar a los diferentes miembros para trabajar en forma auto organizada, multifuncional y con responsabilidad de equipo.
- Moderar las reuniones.

- Resolver impedimentos que en el Sprint pueden entorpecer la ejecución de las tareas.
- Encargarse de la configuración, diseño y mejora continua de las prácticas de Scrum en la organización.

2.3.5.1.3 Eventos.

En Scrum existen eventos predefinidos con el propósito de crear regularidad y minimizar la necesidad de reuniones no definidas en Scrum. Todos los eventos son bloques de tiempo (*time-boxes*), de tal modo que todos tiene una duración máxima y se detallan a continuación.

2.3.5.1.3.1 *Sprint*.

Según Bara (2015) el corazón de Scrum es el *Sprint*, es un periodo de tiempo definido durante el cual se crea un incremento de producto hecho o terminado utilizable, potencialmente entregable.

Además, señala que cada *sprint* se puede considerar un mini-proyecto de no más de un mes. Al igual que los proyectos, los *sprint* se utilizan para lograr algo. Cada *sprint*

cuenta con una definición de lo que se va a construir, un diseño y un plan flexible que guiará la construcción del plan, el trabajo del equipo y el producto resultante.

Cada *sprint* inicia con una reunión de planificación, en la cual se llega a un acuerdo sobre cuáles tareas se van a llevar a cabo durante el *sprint*.

De acuerdo a La Guía de Scrum (2016, pág. 8) durante el *sprint*:

- No se realizan cambios que puedan afectar al Objetivo del *Sprint* (*Sprint goal*).
- Los objetivos de calidad no disminuyen.
- El alcance puede clarificarse y renegociarse entre el Dueño de Producto y el Equipo de Desarrollo a medida que se va aprendiendo más.

En esta guía también se especifica que un *Sprint* puede cancelarse antes de que el bloque de tiempo llegue a su fin. Solo el Dueño de Producto tiene la autoridad para cancelar el *Sprint*, aunque puede hacerlo bajo la influencia de los interesados, el Equipo de Desarrollo o del Scrum Master. Un *Sprint* se cancelaría si el Objetivo del *Sprint* llega a quedar obsoleto, es decir, debería cancelarse si no tuviese sentido seguir con él dadas las circunstancias. En caso de ser cancelado, se revisan todos los elementos de la Lista de Producto que se hayan terminado, si una parte del trabajo es potencialmente entregable el Dueño de Producto normalmente lo acepta. Los elementos de la Lista de

Producto no terminados se vuelven a estimar y se vuelven a introducir en la Lista de Producto (2016, pág. 9).

Los *Sprint* contienen y consisten en la Planificación del *Sprint* (*Sprint Planning*), los Scrums Diarios (*Daily* Scrums), la Revisión del *Sprint* (*Sprint Review*) y la Retrospectiva del *Sprint* (*Sprint Retrospective*).

2.3.5.1.3.2 Planificación de *Sprint* (*Sprint Planning*).

Sandoval (2016) lo define como una actividad acotada en el tiempo (*time-boxes*), en donde el Equipo Scrum colabora para seleccionar y comprender el trabajo que será realizado en el *Sprint* que está por comenzar.

La planificación de *Sprint* tiene un máximo de duración de ocho horas para un *Sprint* de un mes. Para *Sprint* más cortos el evento es usualmente más corto. Por ejemplo, para un *sprint* de dos semanas, las reuniones de planificación son de cuatro horas de duración.

Básicamente, el equipo completo participa de la reunión de Planificación de *Sprint*. Consiste en trabajar a partir del *Product Backlog* ordenado, el dueño de producto y los miembros del equipo de desarrollo discuten cada ítem y llegan a un acuerdo compartido respecto al mismo y al trabajo necesario para completarlo en forma consistente.

La reunión de Planificación de *Sprint* tradicionalmente consta de dos partes, cada una de la mitad de tiempo de duración de la reunión de Planificación de *Sprint*; respondiendo a las siguientes dos preguntas:

- → ¿Qué trabajo será realizado en el *Sprint*?
- → ¿Cómo se realizará el trabajo seleccionado?

En esta reunión de Planificación de *Sprint* se crea el Objetivo del *Sprint* (*Sprint Goal*), el cual es una meta establecida para el *Sprint* que puede lograrse mediante la implementación de la Lista de Producto. Además, proporciona una guía al Equipo de Desarrollo acerca de por qué está construyendo el incremento. (Schwaber & Sutherland, 2016, pág. 11).

2.3.5.1.3.3 Scrum Diario (*Daily* Scrum).

Casanova (2015) expone que la reunión diaria, como su nombre lo indica, consiste en reunirse todos los días el Equipo Scrum completo, para que cada miembro secuencialmente hable durante dos o tres minutos y responda a estas tres preguntas:

1) ¿Qué hiciste ayer?
2) ¿Qué harás hoy?
3) ¿Hay algún impedimento?

Los objetivos que se buscan al responder a estas tres preguntas son los siguientes:

- Compartir con el equipo el compromiso para avanzar hacia el objetivo del Sprint.
- Tomar decisiones coordinadas entre todos para eliminar obstáculos que impidan llegar a ese objetivo.

Asimismo, Francia (2017) indica que el Scrum Diario se realiza a la misma hora y en el mismo lugar todos los días para reducir complejidad. La duración de la reunión es corta, quince minutos y sirve como un alineamiento diario para que los miembros del Equipo de Desarrollo crezcan y mejoren un entendimiento compartido de las cosas más importantes que deberían ser realizadas a continuación, en las siguientes 24 horas para lograr el mejor progreso posible hacia el Objetivo del *Sprint*.

En la siguiente figura se muestra un resumen de lo antes descrito.

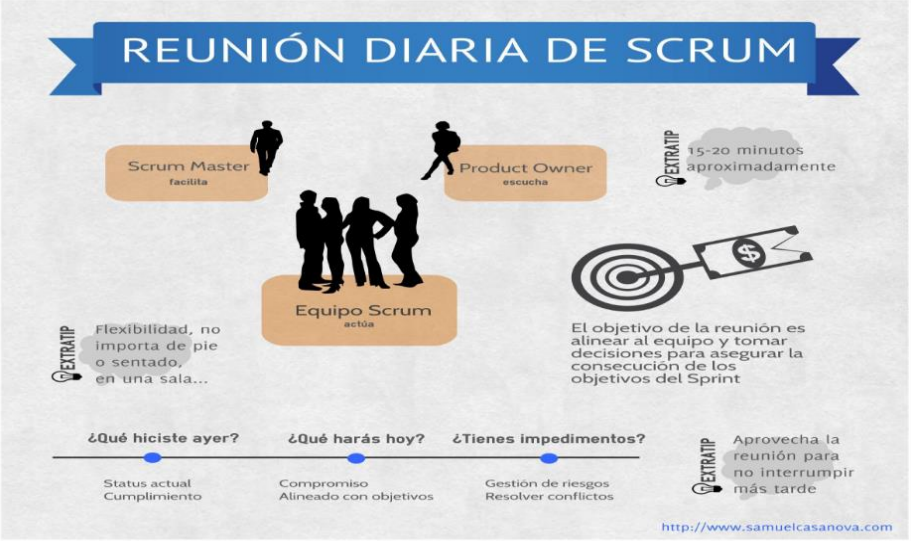


Figura 13. Reunión diaria de Scrum

Fuente: (Casanova, 2015)

2.3.5.1.3.4 Revisión de *Sprint* (*Sprint Review*).

Según Barba (2015) al final del *Sprint* se lleva a cabo una Revisión de *Sprint* para inspeccionar el incremento y adaptar la Lista de Producto si fuese necesario, es decir, se inspeccionan los elementos de la Lista de Producto incluidos en el *Sprint* para valorar si cumplen su definición de terminado. Durante esta revisión, el Equipo Scrum y los interesados comparten sobre lo que se hizo en el periodo de tiempo del *Sprint*. Basándose en esto y en cualquier cambio a la Lista de Producto durante el *Sprint*, los asistentes colaboran para determinar las siguientes cosas que podrían hacerse para optimizar el valor.

Además, expone que el Dueño del Producto es el encargado de decidir si un ítem cumple o no con la definición de terminado y puede requerir la ayuda del equipo de desarrollo para valorar cuestiones técnicas como, por ejemplo, el nivel de cobertura de pruebas unitarias del proyecto.

Los elementos de la Lista de Producto pueden estar terminados o no, pero en ningún caso podrán estar parcialmente terminados. Aquellos elementos que no hayan sido completamente terminados, o que presenten deficiencias, volverán a la Lista de Producto para ser estimados y priorizados de nuevo.

En la práctica suele ocurrir que la mayoría de elementos no terminados en el *Sprint* que termina sean puestos en la cima de la pila del *Sprint* que empieza, pues suelen ser tareas bien definidas y a las que en muchas ocasiones solo les restan unas pocas horas de dedicación; no obstante, esto no se cumple siempre o no nos interesa siempre, por ello es conveniente que vuelvan a la Lista de Producto y sean replanteados. (Barba Prieto, 2015).

La reunión para la Revisión del *Sprint* es un evento máximo de cuatro horas para *Sprints* de un mes y proporcionalmente menor en *Sprints* más cortos. Por otra parte, consiste en una reunión informal, no de seguimiento. El propósito de la presentación del

incremento es facilitar la retroalimentación de información y fomentar la colaboración. (La Guía de Scrum, 2016, pág. 12).

2.3.5.1.3.5 Retrospectiva de *Sprint* (*Sprint Retrospective*).

La reunión de retrospectiva tiene lugar el último día del *sprint*, tras la reunión de revisión del *Sprint*. Se trata de una reunión restringida a un bloque de tiempo de tres horas para *Sprint* de un mes, para *Sprints* más cortos se reserva un tiempo usualmente más corto. Según Bara (2015) es una oportunidad para el Equipo Scrum de inspeccionarse a sí mismo y crear un plan de mejoras para ejecutar durante el siguiente Sprint. El propósito de la Retrospectiva de *Sprint* es:

- ✓ Revisar cómo fue el último *Sprint* en lo que respecta a las personas, relaciones, procesos y herramientas.
- ✓ Identificar y ordenar los temas principales que salieron bien y las potenciales mejoras.
- ✓ Crear un plan para la implementación de mejoras con respecto a cómo el Equipo Scrum hace su trabajo.

Esta reunión y las mejoras resultantes son fundamentales para lograr una organización ágil y autónoma.

Si el equipo no completó todos los casos de usuario asignados al *Sprint*, en la reunión retrospectiva se abordarán los motivos. El equipo determinará si puede adaptar sus procesos de modo que la probabilidad de que surjan esos problemas sea menor. También debe abordarse los problemas que afectaron la eficacia global del equipo, la productividad, la calidad y el grado de satisfacción del grupo con respecto al proyecto (Microsoft, s.f.).

2.3.5.1.4 Artefactos.

Los artefactos de Scrum representan trabajo o valor en diversas formas que son útiles para proporcionar transparencia y oportunidades para la inspección y adaptación. Están diseñados específicamente para maximizar la transparencia de la información clave, necesaria para asegurar que todos tengan el mismo entendimiento del artefacto. Una transparencia imprescindible para maximizar el valor aportado en la próxima entrega y minimizar el riesgo durante el periodo que dure el *Sprint*. Si los artefactos no son suficientemente transparentes, las decisiones pueden ser erróneas, con lo que el valor puede disminuir y el riesgo aumentar (Ramos Vega, 2017).

2.3.5.1.4.1 Lista de Producto (*Product Backlog*).

Según Ramos (2017) la Lista de Producto es una lista que recoge todo lo que necesita el producto para satisfacer las necesidades de clientes potenciales. Además, explica lo siguiente:

- Se trata de una lista única por producto, de la que el Dueño de Producto es el único responsable de su contenido, disponibilidad y ordenación.
- Se trata de un artefacto vivo. Mientras el producto exista y sea utilizado, su lista cambiará en base a la retroalimentación recibida del mercado. La evolución del producto también está sujeta a que cambien los requisitos del negocio, la tendencia del mercado o la tecnología. En este caso la Lista de Producto también cambiará.
- Los atributos de los elementos de la Lista de Producto son: la descripción, el orden, la estimación y el valor. Los atributos se pueden utilizar para agrupar y priorizar elementos de la Lista de Producto.
- Varios Equipos Scrum pueden trabajar para un mismo producto. En este caso, los equipos compartirán la Lista de Producto. Lo que no compartirán nunca serán los elementos de la Lista de Producto.
- El proceso de añadir mayor detalle a los elementos de la Lista de Producto se conoce como Refinamiento de la Lista de Producto. Normalmente el proceso de refinamiento no consume mucho más del 10% de la capacidad del equipo. El equipo Scrum decide cómo y cuándo se hace el refinamiento.

- El Equipo de Desarrollo es el responsable de la estimación de los elementos de la Lista de Producto, considerando que es quien realmente va a realizar el trabajo y quien se va a comprometer a cumplir el Objetivo del *Sprint*. La estimación será más precisa cuanto más claro y detallado esté el elemento.
- Los elementos de la Lista de Producto seleccionados para el *Sprint* se descomponen de forma que cualquier elemento pueda ser terminado en el *time-box* acordado para el *Sprint*.

2.3.5.1.4.2 Lista de Pendientes del *Sprint* (*Sprint Backlog*).

La lista de Pendientes del *Sprint* es un conjunto de elementos de la Lista de Producto seleccionados para abordar en el *Sprint*, más un plan para entregarlos como incremento del producto y conseguir el Objetivo del *Sprint*. Estos elementos normalmente se componen de tareas técnicas más pequeñas que permiten conseguir un incremento de *software* terminado.

De acuerdo a Roche (2017), la Lista de Pendientes del *Sprint* es una predicción realizada por el Equipo de Desarrollo acerca de qué funcionalidad formará parte del próximo incremento y del trabajo necesario para entregar esa funcionalidad en un

incremento terminado; por lo tanto, la Lista de Pendientes de *Sprint* pertenece al Equipo de Desarrollo, y este es el único que podría alterarlo.

A su vez explica que este artefacto es un elemento para visualizar el trabajo a realizar durante cada *Sprint* y está gestionado por el Equipo de Desarrollo. Su propósito es mantener la transparencia dentro del desarrollo, actualizándolo durante toda la iteración especialmente a través de los Scrum Diario.

La Lista de Pendientes del *Sprint* permite visualizar, durante cada *Sprint*, aquellos elementos que aún no han empezado a desarrollarse, aquellos que sí y quiénes están trabajando en los mismos, así como aquellos que están esperando a desplegarse o están completamente terminados.

Este artefacto permite entender cuál es la evolución del trabajo durante el *Sprint*, así como hacer un análisis de riesgos. Dado que cada *Sprint* tiene una meta específica y existen elementos seleccionados de la Lista de Producto que tienen más o menos valor, la Lista de Pendientes de *Sprint* permite analizar hasta donde se ha cumplido el objetivo y qué se podría eliminar. De esta forma, maximizamos el retorno de la inversión en desarrollo. (Roche, 2017).

2.3.5.1.4.3 Incremento.

Se trata del resultado del *Sprint*, de su entregable. Un entregable terminado, utilizable y potencialmente desplegable. El incremento debe estar terminado, es decir, listo para ser utilizado y cumplir la definición de terminado del Equipo Scrum.

El incremento consiste en una versión de producto totalmente utilizable. Una versión que se liberará o no, dependiendo del criterio y decisión del Dueño de Producto. Esta versión es la suma de los elementos de la Lista de Producto completados en un *Sprint* más el valor de los incrementos de los *Sprints* anteriores (Ramos Vega, 2017).

En la figura 14 se muestra lo anteriormente descrito.

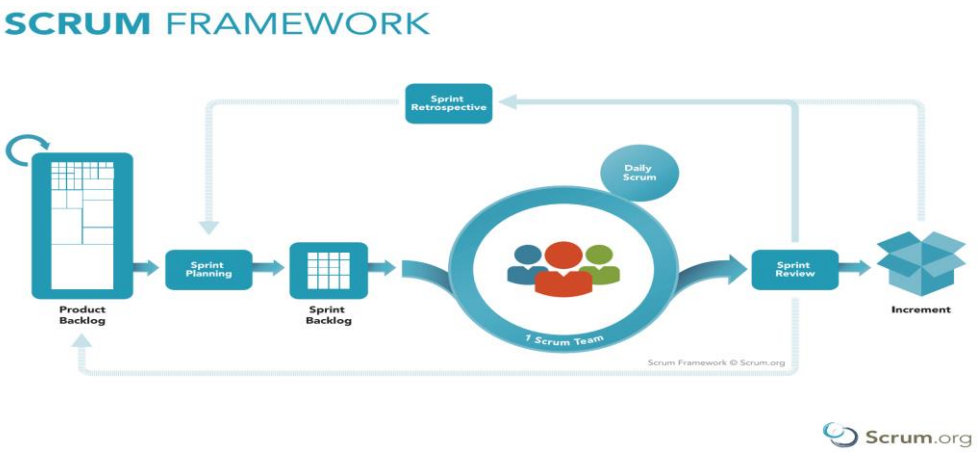


Figura 14. Marco de Trabajo Scrum

Fuente: (Scrum.org, s.f.)

2.3.5.2 Programación Extrema *(eXtreme Programming)*.

Extreme Programming (XP) es un marco de trabajo para el desarrollo ágil de software que tiene como propósito producir software de mayor calidad, y una mejor calidad de vida para el equipo de desarrollo. *Extreme Programming* es el marco de trabajo ágil más específico relacionado a las prácticas de ingeniería afines en el desarrollo de *software* (Agile Alliance, s.f.).

La programación extrema lleva las mejores prácticas de las metodologías tradicionales de ingeniería de *software* a niveles "extremos", de ahí el nombre. Este marco de trabajo se diferencia de los métodos tradicionales porque se enfoca en la adaptabilidad en vez de la previsibilidad. Además, se basa en la filosofía en la que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de *software*. Se considera que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto permite una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

Lo que caracteriza la Programación Extrema, al igual que al resto de los procesos ágiles, es un proceso de *software* dinámico, mediante ciclos de desarrollo cortos (llamados iteraciones), al fin de los cuales se generan unos entregables funcionales.

Aunado a lo anterior, en cada iteración se realiza un proceso de *software* completo: análisis, diseño, desarrollo y pruebas, pero utilizando un conjunto de reglas y prácticas específicas de Programación Extrema (Vila Grau, 2017).

La Programación Extrema es exitosa porque enfatiza la satisfacción del cliente. En lugar de ofrecerle todo lo que pueda desear en una fecha futura, este proceso ofrece las funcionalidades del sistema que necesita el cliente a medida que lo requiere. La Programación Extrema le permite a sus desarrolladores responder con confianza a los requisitos cambiantes de los clientes, incluso a fines del ciclo de vida.

La programación extrema enfatiza el trabajo en equipo. Los gerentes, clientes y desarrolladores son socios iguales en un equipo colaborativo. También implementa un entorno simple pero eficaz que permite a los equipos ser altamente productivos. El equipo se autoorganiza alrededor del problema para resolverlo de la manera más eficiente posible (Extreme Programming, 2009).

Por otra parte, Mota (2017) explica que la Programación Extrema se enfoca en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de *software*, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores y propiciando un buen clima de trabajo.

Las características generales donde la Programación Extrema es recomendable aplicar, según la organización Agile Alliance (s.f.), son:

- ✓ Cambios dinámicos en los requerimientos del *software*.
- ✓ Altos riesgos asociados al proyecto, por ejemplo, proyectos donde el cliente establece fecha específica, nuevos desafíos para el grupo de trabajo o para la industria del *software*; en estos se pretende mitigar el riesgo y aumentar el éxito.
- ✓ Equipo de desarrollo pequeño y compartido.
- ✓ La tecnología que se está utilizando permite pruebas unitarias y funcionales automatizadas.

El marco de trabajo de Programación Extrema consiste en roles, valores, prácticas y reglas simples asociadas. Cada componente dentro del marco de trabajo es como una pieza de un rompecabezas, individualmente no tienen sentido, pero cuando se combinan juntas se puede ver una imagen completa. Cada componente dentro del marco de trabajo sirve a un propósito específico y es esencial para el éxito de Programación Extrema y para su uso.

2.3.5.2.1 Valores.

La Programación Extrema se fundamenta en valores. Estos valores asociados con las reglas son la extensión natural y la consecuencia de maximizar los valores establecidos en este marco de trabajo.

La Programación Extrema no es un conjunto de reglas, más bien lo que pretende es una forma de trabajar en armonía con sus valores personales y corporativos. Una buena práctica es agregar valores al marco de trabajo y que estos se reflejen en los cambios que realizan en las reglas.

Los cinco valores de la Programación Extrema son la simplicidad, la retroalimentación, la comunicación, el respeto y el coraje o valentía; tomando como referencia la enciclopedia en línea EcuRed (Programación Extrema (XP), s.f.) y la organización *Agile Alliance* (Extreme Programming, s.f.) estos valores se describen con más detalle a continuación.

1) Simplicidad: Es la base de la Programación Extrema. Se refiere a sencillez. Se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento. Un diseño complejo del código junto a sucesivas modificaciones por parte de diferentes desarrolladores hace que la complejidad aumente exponencialmente. El propósito es evitar el desperdicio y realizar solo las cosas absolutamente necesarias y conocidas. Enfocar esfuerzos para llevar a cabo las tareas de manera simplificada y comprensible para todos. Evitar la complejidad.

   La manera de mantener el código simple a medida que crece es mediante la refactorización de código.

La simplicidad no solo aplica en el código, también aplica en la documentación. El código debe comentarse en su justa medida, intentando que esté autodocumentado.

2) Retroalimentación: Mecanismo para la mejora continua. Comentarios constantes con el fin de identificar áreas de mejora y revisar sus prácticas para ajustar el producto *software* o proceso en el futuro.

   Al estar el cliente integrado en el proyecto, su opinión acerca del estado del proyecto se conoce en tiempo real.

   Por otra parte, al realizar desarrollo incremental y mostrar los resultados, se minimizan los retrocesos y se aumenta la precisión de los requerimientos del sistema. Los desarrolladores se enfocan en lo que realmente es importante.

   Aunado al párrafo anterior, la retroalimentación del cliente en periodos de tiempo cortos (iteraciones) evita que meses de trabajo se tiren a la borda debido a cambios en los criterios del cliente o malentendidos por parte del equipo de desarrollo.

   Otra fuente de retroalimentación son las pruebas unitarias, ya que informan sobre la condición del código. Ejecutar pruebas unitarias con frecuencia permite descubrir fallos debido a modificaciones en el código.

3) Comunicación: La comunicación a diario asegura la transferencia de conocimiento de un miembro del equipo a todos los demás. La Programación Extrema enfatiza la importancia de la conversación cara a cara.

   Una forma de comunicación es el código programado con el desarrollador, este se comunica mejor cuanto más simple sea. Si el código es complejo, es necesario un mayor esfuerzo para entenderlo. Por tal razón, el código

autocomentando es más fiable que los comentarios, los cuales quedan imprecisos a medida que el código es modificado. Se recomienda comentar solo aquello que no va a variar, por ejemplo, el objetivo de una clase o la funcionalidad de un método.

Las pruebas unitarias son otra forma de comunicación ya que describen el diseño de las clases y los métodos al mostrar la forma concreta de cómo utilizar su funcionalidad.

Los programadores se comunican constantemente para realizar la programación en parejas. La comunicación con el cliente es fluida ya que el cliente es parte del equipo. El cliente decide cuáles características tienen prioridad y siempre debe estar disponible para aclarar dudas.

4) Respeto: Todos dan y sienten el respeto que merecen como un valioso miembro del equipo. Todos aportan valor, incluso si es simplemente entusiasmo. El respeto fomenta una mejor autoestima en el equipo, consecuentemente este aspecto aumenta el ritmo de producción. Proporcionar y aceptar comentarios que respeten la relación, y trabajar juntos para identificar diseños y soluciones simples.
5) Coraje o valentía: Enfrentar el miedo en forma efectiva. Consiste en un esfuerzo para realizar las cosas de la mejor manera. Proactividad en los miembros del equipo.

Se refiere a adaptar los cambios cuando sucedan. Coraje es dejar de hacer algo que no funciona y probar otra cosa. Coraje es aceptar y actuar sobre los comentarios, incluso cuando son difíciles de aceptar.

La valentía permite a los desarrolladores que se sientan cómodos al reconstruir su código cuando sea necesario. Esto significa revisar el sistema existente y modificarlo si con ello los cambios futuros se implementan más fácilmente.

Otra forma de representar la valentía es saber cuándo desechar un código: valentía para quitar código fuente obsoleto, sin importar cuánto esfuerzo y tiempo se invirtió en crear ese código.

Además, valentía significa persistencia: un desarrollador puede permanecer sin avanzar en un problema complejo por un día entero, y luego lo resolverá rápidamente al día siguiente, solo si es persistente.

2.3.5.2.2 Roles.

Aunque el marco de trabajo de la Programación Extrema detalla prácticas particulares que su equipo debe seguir, en realidad no establece roles específicos para los miembros de su equipo.

De acuerdo a la fuente de información, no hay orientación, o hay una descripción de cómo los roles típicamente encontrados en proyectos más tradicionales se comportan en proyectos de Programación Extrema. En seguida se describen los roles más comunes asociados a este marco de trabajo, según OBS Business School (Consejos

para aplicar una Programación Extrema, 2014), la organización *Agile Alliance* (Extreme Programming, s.f.) y Meléndez Valladarez, Gaitán, & Pérez Reyes (2016):

2.3.5.2.2.1 Desarrollador.

Se considera el rol primordial. Se encarga de traducir y plasmar las historias de usuario en código fuente. Además, estima el tiempo de desarrollo de cada historia de usuario para que el cliente pueda asignarle prioridad dentro de la iteración, por lo tanto, cada iteración incorpora nueva funcionalidad de acuerdo a las prioridades establecidas por el cliente.

El desarrollador también es responsable de diseñar y ejecutar las pruebas de unidad del código fuente que ha implementado o modificado.

Debido a que diferentes proyectos requieren una combinación diferente de habilidades, y debido a que la Programación Extrema se basa en un equipo multifuncional que proporciona la combinación adecuada de habilidades, los creadores de este marco de trabajo no sintieron la necesidad de una definición de rol adicional.

2.3.5.2.2.2 Cliente.

El rol de cliente determina la funcionalidad que se pretende en cada iteración, es decir, toma todas las decisiones de negocio relacionadas con el proyecto. Asimismo, define las prioridades de implementación, según el valor que aporta al negocio cada historia de usuario. También es responsable de diseñar y ejecutar las pruebas de aceptación.

Aunado a lo anterior, se cuestiona:

- ¿Qué debería hacer el sistema (qué características están incluidas y qué logran)?
- ¿Cómo sabemos cuándo se termina el sistema (cuáles son nuestros criterios de aceptación)?
- ¿Cuánto tenemos que gastar (cuál es el financiamiento disponible, cuál es el caso comercial)?
- ¿Qué deberíamos hacer a continuación (en qué orden entregamos estas características)?

El escenario ideal es la participación activa del cliente en el proyecto, que sea parte del equipo. Además, el cliente tiene la capacidad de introducir variantes en los requerimientos del sistema, según las necesidades del entorno.

Se asume que el Cliente es una sola persona, sin embargo, la experiencia ha demostrado que una persona no puede proporcionar de manera adecuada toda la información de negocio relacionada con un proyecto. Su equipo necesita asegurarse de tener una idea completa de la perspectiva de negocio, pero tiene algunos medios para tratar los conflictos en esa información, de modo que pueda obtener una dirección clara.

2.3.5.2.2.3 Encargado de pruebas (*Tester*).

Este rol ayuda al cliente a escribir las pruebas funcionales. Con regularidad ejecuta pruebas y difunde los resultados en el equipo; además es el responsable de las herramientas de soporte para las pruebas unitarias.

2.3.5.2.2.4 Encargado de seguimiento (*Tracker*).

Algunos equipos pueden tener un encargado de seguimiento en su seno. Este es a menudo uno de los desarrolladores que pasa parte de su tiempo cada semana ocupando este rol adicional. El objetivo principal de este rol es realizar un seguimiento de las métricas relevantes que el equipo considera necesarias para seguir su progreso e identificar áreas de mejora. Las métricas clave que su equipo puede rastrear incluyen la velocidad, las razones de los cambios en la velocidad, la cantidad de horas extra

trabajadas y las pruebas de aprobación y falla. De esta forma, puede aportar información estadística en lo que refiere a la calidad de las estimaciones para que puedan ser mejoradas.

Este no es un rol obligatorio, y generalmente solo se establece si el equipo determina una verdadera necesidad de realizar un seguimiento de varias métricas.

2.3.5.2.2.5 *Coach* o entrenador.

Es el responsable general del proyecto. Se encarga de iniciar y de guiar a los miembros del equipo para poner en marcha cada una de las prácticas del marco de trabajo de la Programación Extrema. Además, puede promover la motivación y autodisciplina a los miembros del grupo para la consecución de los objetivos.

2.3.5.2.2.6 Consultor.

Se trata de un miembro externo al grupo de trabajo con un conocimiento específico en algún tema necesario para el proyecto. Su función es guiar al equipo para resolver un problema específico.

2.3.5.2.2.7 Administrador (*Big boss*).

Su función es la de mantener el vínculo entre el grupo de trabajo y los clientes. Experto en tecnología y labores de gestión. Construye el plan de trabajo, obtiene los recursos necesarios y maneja los problemas que se generan. Administra a su vez las reuniones (plan de entregas, agenda de compromisos, entre otros). Su labor fundamental es de coordinación.

2.3.5.2.3 Prácticas.

Según la organización *Alige Alliance* (Extreme Programming, s.f.) el núcleo de la Programación Extrema es el conjunto interconectado de prácticas de desarrollo de *software* que se enumeran a continuación. Si bien es posible llevar a cabo estas prácticas de manera aislada, muchos equipos han encontrado que algunas refuerzan a las demás y deben hacerse en conjunto para eliminar por completo o disminuir los riesgos que a menudo se enfrentan en el desarrollo de *software*.

Las prácticas de la Programación Extrema han evolucionado un poco desde que se introdujeron inicialmente. A continuación, se describen las mencionadas en la segunda edición de *Extreme Programming Explication Embrace Change*. Estas descripciones

incluyen mejoras basadas en la experiencia de muchos en este marco de trabajo y reflejan un conjunto de buenas prácticas en la industria del *software*.

1) Sentarse juntos (*Sit Together*): Como la comunicación es uno de los cinco valores de la Programación Extrema, y la mayoría de las personas acepta que la conversación cara a cara es la mejor forma de comunicación, esta práctica consiste en sentar al equipo en el mismo espacio, sin barreras de comunicación, como por ejemplo evitar las paredes de los cubículos.

2) Todo el equipo (*Whole Team*): Un grupo funcional cruzado con las funciones necesarias para elaborar un producto conforman un solo equipo. Esto significa que las personas con necesidades (cliente), así como todas las personas que desempeñan algún papel en la satisfacción de esa necesidad, trabajan juntas todos los días para lograr un resultado específico.

3) Espacio de trabajo informativo (*Informative Workspace*): Acondicionar el espacio de trabajo para facilitar la comunicación cara a cara en el equipo de trabajo, no obstante, las personas deben tener algo de privacidad cuando lo necesiten. Un entorno de transparencia y agradable para el equipo de trabajo es lo ideal; con las partes interesadas (cliente) fuera del equipo. Utilizar medios de información para comunicar activamente la información actualizada.

4) Enfoque en el trabajo (*Energized Work*): Esta práctica señala que la efectividad en el desarrollo de *software* mejora cuando se está concentrado y libre de distracciones. Esto aplica en forma general a todo el equipo de trabajo, aumentando el conocimiento, motivación, y creatividad en el entorno del proyecto; y significa tomar medidas para asegurar que el equipo de trabajo pueda física y mentalmente entrar en un estado enfocado, es decir, concentración en las tareas.

5) Programación en parejas (*Pair Programming*): significa que todo el *software* es desarrollado por dos personas sentadas en la misma estación o lugar de trabajo. La idea detrás de esta práctica es que dos cerebros y cuatro ojos son mejores que un cerebro y dos ojos. De hecho, obtiene una revisión continua del código y una respuesta más rápida a los molestos problemas que pueden detener a una persona abruptamente.

   Esta práctica mejora la calidad y en realidad no lleva el doble de tiempo porque pueden resolver los problemas más rápidamente y se concentran más en la tarea en cuestión, creando así menos código para lograr lo mismo.

6) Historias de usuario (*Stories*): Consiste en la descripción de las funcionalidades del sistema, debe ser en lenguaje simple, en términos significativos para clientes y usuarios. Estas historias pretenden ser descripciones breves de las cosas que los usuarios quieren y necesitan del producto. Estas historias de usuario pueden ser utilizadas para planificar y servir a futuro para ampliar más detalles cuando

los desarrolladores están construyendo (desarrollando el código fuente) de la historia en particular.

7) Ciclo semanal (*Weekly Cycle*): El ciclo semanal es sinónimo de una iteración. En la Programación Extrema, el equipo se reúne el primer día de la semana para reflexionar sobre el progreso hasta la fecha, el cliente elige las historias de usuario que quiere se entreguen en esa semana, y el equipo determina cómo abordarán esas historias. El objetivo es que al final de la semana las historias seleccionadas se conviertan en funcionalidades del sistema y que estas se encuentren probadas (con retroalimentación del usuario) y puedan liberarse.

8) Ciclo trimestral (*Quarterly Cycle*): El ciclo trimestral es sinónimo de lanzamiento. El propósito es mantener el trabajo detallado de cada ciclo semanal en el contexto del proyecto general. El cliente establece el plan general para el equipo en términos de características deseadas dentro de un trimestre en particular. Esto ayuda al cliente a trabajar con otras partes interesadas quienes necesitan tener noción de cuándo las características estarán disponibles, y a su vez minimiza las sorpresas.

9) Equilibrio (*Slack*): La idea en esta práctica es agregar algunas tareas o historias de usuario de baja prioridad en sus ciclos semanales y trimestrales, las cuales puedan descartarse en caso de algún imprevisto o atraso en las tareas o historias de mayor prioridad. Dicho de otra manera, considerar la variabilidad

inherente en las estimaciones para asegurar la oportunidad de cumplir sus pronósticos.

10) *Ten-Minute Build*: El objetivo es ejecutar todas las pruebas a lo construido (desarrollado en código fuente) en diez minutos. La Programación Extrema sugiere un periodo de tiempo de diez minutos, una mayor duración es menos probable que se ejecute con frecuencia, lo que supone un mayor tiempo entre los errores.

Se recomienda esta práctica en conjunto con integración continua y *Test First Development*.

11) Integración continua: es una práctica donde los cambios de código fuente se prueban inmediatamente, previo a cargar la versión actual al repositorio de código fuente. Se recomienda que se realice en forma constante, en pocas horas, o cuando se presenta un salto significativo. El beneficio de esta práctica es que puede detectar y solucionar problemas de integración más rápido.

La mayoría de los equipos teme el paso de integración del código fuente debido al descubrimiento inherente de conflictos y problemas que se presentan. La mayoría de los equipos adoptan el enfoque "Si duele, evítelo el mayor tiempo posible"; no obstante, los seguidores de la Programación Extrema sugieren que “si duele, hágalo más a menudo".

El razonamiento detrás de ese enfoque es que si tiene problemas cada vez que integra código y tarda un poco en encontrar dónde están los problemas, quizás deba integrarse más a menudo para que, si hay problemas, sean mucho

más fáciles de encontrar porque existen menos cambios incorporados en la construcción (desarrollo de código fuente).

Esta práctica requiere un poco de disciplina adicional y es altamente dependiente de *Ten Minute Build* y *Test First Development*.

12) *Test-First Programming:* Se basa en cambiar el estilo tradicional que consiste en: desarrollar código -> escribir pruebas -> ejecutar pruebas.
En su lugar seguir el estilo: Diseño de casos de prueba -> ejecutar casos de pruebas -> desarrollar código fuente en función de los casos de pruebas -> ejecutar pruebas unitarias -> repetir.

Al igual que la integración continua, *Test-First Programming* reduce el ciclo de retroalimentación para que los desarrolladores identifiquen y resuelvan problemas, disminuyendo así la cantidad de errores que se comente en el proceso de desarrollo del código fuente.

13) Diseño Incremental: La práctica del diseño incremental sugiere la comprensión del diseño del sistema desde una perspectiva macro, y posteriormente profundizar en los detalles de los aspectos particulares del diseño que se entregarán como funciones específicas. Este enfoque reduce el costo de los cambios y permite tomar decisiones de diseño cuando sea necesario en función de la información más reciente disponible.

La práctica de refactorización inicialmente fue incluida entre las doce prácticas, pero en la segunda edición de *Extreme Programming Explication Embrace Change* se incorporó dentro de la práctica del diseño incremental. La

refactorización es una práctica excelente para mantener el diseño simple, y uno de los usos de refactorización más recomendados es eliminar la duplicación de procesos.

2.3.5.2.4 Reglas.

Según el sitio web oficial de Programación Extrema (2009) existen cinco reglas en este marco de trabajo, las cuales se describen a continuación.

2.3.5.2.4.1 Planificación.

La planificación consiste en un diálogo continuo entre las partes involucradas en el proyecto, incluyendo al cliente, desarrolladores y a los coordinadores. El proyecto comienza recopilando las historias de usuarios, las cuales serán evaluadas rápidamente por los desarrolladores para estimar el tiempo de programación (codificación) de cada una. Se compone de:

- → Las historias de usuario: corresponde a las descripciones de las funcionalidades del sistema, son escritas por los usuarios en forma breve y en su propio lenguaje, es decir, sin tecno-sintaxis. Se usan en lugar de un gran documento de requerimientos. Además, tienen el mismo propósito que los casos de uso, pero

no son los mismo. Se utilizan para crear las estimaciones de tiempo para el plan de entregas.

Las historias de usuario difieren de las especificaciones de requerimientos tradicionales. La mayor diferencia radica en el nivel de detalle. Estas solo deben proporcionar detalles suficientes para realizar un cálculo razonablemente bajo del tiempo que demorará la historia en implementarse. Cuando llegue el momento de implementar la historia, los clientes darán una descripción detallada de los requisitos a los desarrolladores.

Otra diferencia entre historias y un documento de requisitos es un enfoque en las necesidades del usuario. Se debe tratar de evitar los detalles de la tecnología específica, el diseño de la base de datos y los algoritmos. Es fundamental mantener las historias centradas en las necesidades y los beneficios de los usuarios, en lugar de especificar los diseños de la interfaz gráfica de usuario (GUI).

→ Plan de entregas y cronograma de lanzamientos: tiene como finalidad definir los planes de iteración. Establece que las historias de usuario serán agrupadas para conformar una entrega, así como su priorización. Este cronograma será el resultado de una reunión entre todos los actores del proyecto.

Estas historias seleccionadas se convierten en tareas de desarrollo (construir código fuente) que se implementarán durante la iteración para completar las historias, las cuales se prueban durante la iteración. Estas pruebas permiten verificar el correcto funcionando y determinar si las historias están terminadas.

Los desarrolladores estiman cuánto tiempo les llevará la implementación, cada historia recibirá una estimación de una, dos o tres semanas en tiempo de

desarrollo ideal; comprende el tiempo que llevaría construir el código fuente de esa historia sin distracciones, sin ninguna otra tarea y conociendo exactamente qué hacer. Posteriormente, el cliente decide cuál es la historia más importante o tiene la más alta prioridad para completarse.

La dinámica en esta fase consiste en que las historias de usuario estén impresas o escritas en tarjetas. Los desarrolladores y clientes mueven las tarjetas en una mesa para crear un conjunto de historias que se implementarán como el primer (o el próximo) entregable.

La filosofía básica del plan de entrega es que un proyecto puede cuantificarse mediante cuatro variables: alcance, recursos, tiempo y calidad. A saber, el alcance define cuánto se debe hacer. Los recursos se refieren a cuántas personas están disponibles. El tiempo a cuándo se realizará el proyecto o entregable. Y la calidad especifica qué tan bueno es el *software* y lo bien probado que será.

No se puede controlar las cuatro variables, cuando cambia uno, inadvertidamente se cambia otro en respuesta.

→ Realizar pequeños y frecuentes entregables: El equipo de desarrollo libera versiones del sistema en forma iterativa o visto de otra manera, realiza con frecuencia entregables de ciertas funcionalidades del producto a los clientes. Estos pequeños entregables aportan valor tanto al cliente como al negocio.

Al final de cada iteración, tendrá un *software* probado, funcionando correctamente y listo para el ambiente de producción para ser entregado al cliente. Él decidirá el momento para ponerlo en producción.

Cuanto más espere para presentar una función importante a los usuarios del sistema, menos tiempo tendrá para solucionarlo, en caso que lo requiera.

→ El proyecto está dividido en iteraciones: El desarrollo iterativo agrega agilidad al proceso de desarrollo. Es importante mantener duración de la iteración constante a lo largo del proyecto. Este es el latido del corazón del proyecto. Esta es la constante que hace que medir el progreso y la planificación sea simple y confiable en este marco de trabajo.

Está en contra de las reglas mirar hacia adelante e intentar implementar cualquier cosa que no esté programada para esta iteración. Habrá mucho tiempo para implementar esa funcionalidad cuando se convierta en la historia más importante en el plan de entrega.

Es vital dar seguimiento al progreso de cada iteración. Si parece que no se terminará lo previsto, se debe convocar otra reunión para volver a estimar y eliminar algunas de las tareas.

Concentrar el esfuerzo para completar las tareas más importantes elegidas por el cliente, en lugar de tener varias tareas sin terminar elegidas por los desarrolladores.

→ Planeación de iteraciones: consiste en una reunión para planificar la iteración, esto se realiza al inicio de cada iteración. La finalidad es establecer el plan de tareas de desarrollo (construir código fuente) que abordará esa iteración. Cada iteración tiene una duración de una a tres semanas.

El cliente selecciona las historias de usuario para la iteración que está por comenzar, tomando en cuenta la prioridad, es decir, la que aporta mayor valor

será la primera por desarrollar. Pueden existir más de una historia de usuario por iteración.

Por su parte, los desarrolladores estiman el tiempo que demorarán realizando las tareas. Se recomienda estimar la duración de las tareas en uno, dos o tres días, según la complejidad. Se denomina días ideales de desarrollo al tiempo efectivo para desarrollar una tarea (construir código fuente), es decir, que no se presenten distracciones. Las tareas que son cortas y se pueden realizar en menos de un día, se pueden agrupar con otras.

También se recomienda no agregar funcionalidades que no estén contempladas en la programación. Solo ejecutar lo que necesita para el día a día. No cambiar las tareas ni estimaciones de las historias. Este proceso de planificación se basa en estimaciones consistentes.

Dar seguimiento a la velocidad del proyecto es importante, permite controlar y tomar decisiones precisas, evitando que los problemas se acumulen al final. Es normal volver a estimar las historias de usuario y renegociar cada cierta cantidad de iteraciones. Lo importante es implementar las historias de mayor importancia de primero, es lo que satisface al cliente. Esto agrega agilidad al proceso de desarrollo.

2.3.5.2.4.2 Gestión.

Hace referencia a las actividades afines a la administración del proceso, promoviendo un ambiente de trabajo confortable, ritmo de trabajo sostenible, cargas de trabajo equilibradas, reuniones diarias para obtener retroalimentación del avance del proyecto, y retrospectiva del proceso como mecanismo de mejora continua.

- → Ofrece al equipo un espacio de trabajo dedicado y abierto: en la Programación Extrema la comunicación es fundamental para el equipo de trabajo, por lo que es importante evitar aquellas barreras que dividen a las personas. Contar con un entorno de trabajo dedicado para que las personas puedan concentrarse en sus actividades, sin desconectarse del resto del equipo.

  Es recomendable disponer de una mesa de conferencias o sala de reuniones para las discusiones grupales que ocurren espontáneamente a lo largo del día. También contar con pizarras para elaborar bocetos de diseño o realizar anotaciones importantes agrega más canales para la comunicación.
- → Establece un ritmo sostenible: Para establecer el ritmo, debe tomarse en serio la iteración. El ideal es disponer del *software* más completo, probado, integrado y listo para el ambiente de producción al final de cada iteración. El *software* incompleto o defectuoso representa una cantidad desconocida de esfuerzo a futuro, por lo que no puede medirlo.

  Si todo apunta a que no podrá terminar todo al finalizar la iteración, convoca a una reunión de planificación de iteración y vuelve a enfocar la iteración para

maximizar la velocidad de su proyecto. Incluso si solo queda un día en la iteración, es mejor hacer que todo el equipo se vuelva a enfocar en una sola tarea completada que en muchas incompletas.

Trabajar horas extras absorbe el espíritu y la motivación del equipo. Cuando el equipo se cansa y desmoraliza, realizará menos trabajo. En lugar de presionar a las personas para que hagan más de lo humanamente posible, se puede convocar a una reunión para cambiar el alcance o el cronograma del proyecto definido en la planificación.

Agregar más personas también es una mala idea cuando un proyecto ya está retrasado. La contribución de muchas personas nuevas suele ser negativa.

Un ritmo sostenible ayuda a planificar los lanzamientos e iteraciones y le impide entrar en una marcha de la muerte. Se debe analizar los recursos para encontrar la velocidad perfecta del equipo que se mantendrá constante durante todo el proyecto. Cualquiera que sea la velocidad del equipo es simplemente aceptarlo, protegerlo y usarlo para hacer planes realistas.

→ Reunión *stand-up* diaria: consiste en una reunión cada mañana para comunicar problemas, soluciones y promover el enfoque del equipo. Todos se ponen de pie en círculo para evitar largas discusiones. Es más eficiente tener una reunión breve en la que asistan todos los del equipo, que muchas reuniones con algunos miembros del equipo en cada una.

La mayoría de las reuniones pueden tener lugar espontáneamente frente a una computadora, donde se puede buscar el código fuente y probar las ideas.

Durante una reunión *stand-ud*, los desarrolladores informan al menos tres cosas:

- Lo que se logró ayer.
- Lo que se intentará hoy.
- Los problemas que causan demoras.

→ Velocidad del proyecto: es una medida para conocer cuánto trabajo se está realizando en el proyecto. Para medir la velocidad del proyecto, simplemente se suman las estimaciones de las historias de los usuarios que se completaron durante la iteración. Es así de simple. También se puede calcular sumando las estimaciones de las tareas terminadas durante la iteración. Ambas mediciones se utilizan para la planificación de iteraciones.

Durante la reunión de planeación de iteración, los clientes pueden elegir el mismo número de historias de usuarios igual a la velocidad del proyecto medida en la iteración anterior. Esas historias se dividen en tareas técnicas (las tareas que realiza el desarrollador) y el equipo puede aceptar o comprometerse con la misma cantidad de tareas que sea igual a la velocidad del proyecto de la iteración anterior.

→ Mover a las personas alrededor del conocimiento: se recomienda mover a las personas para evitar la pérdida de conocimiento. El entrenamiento cruzado es un aspecto importante para evitar las islas de conocimiento.

Un equipo es mucho más flexible si todos saben lo suficiente sobre cada parte del sistema para trabajar en él. En lugar de tener a algunas personas sobrecargadas de trabajo mientras que otros miembros del equipo tienen poco que hacer, todo el equipo puede ser productivo. Se puede asignar cualquier número de desarrolladores a la parte más caliente del sistema.

- → Reuniones de retrospectiva: el propósito es conversar sobre lo que funciona y lo que no funciona y pensar algunas maneras de mejorar el proceso. La mejora continua debe ser parte del proceso.

2.3.5.2.4.3 Diseño.

El diseño es la guía para la implementación del sistema, por lo tanto, debe ser claro, y para poder ser claro necesita de simplicidad, ya que no solo será entendido por el desarrollador, sino que también en muchas ocasiones por el usuario. Se compone de:

- → Simplicidad: Un diseño simple siempre toma menos tiempo para terminar que uno complejo. Reemplazar lo complejo por algo simple.

  Muchas personas intentan medir la simplicidad. La medición simple desafía porque es una cualidad muy subjetiva. El simple de una persona es el complejo de otra persona; por lo tanto, el equipo decide qué es simple. Juntos juzgan el código subjetivamente. Se sugiere cuatro cualidades subjetivas; Testeable, comprensible, navegable y explicable.

  - Testable: diseño de pruebas unitarias y pruebas de aceptación para verificar automáticamente si hay problemas.
  - Navegable: capacidad para poder encontrar lo que se quiere cuando lo requiera. Por ejemplo, los nombres o identificadores utilizados correctamente ayudan a encontrar cosas. Usar polimorfismo, delegación y herencia correctamente son otro ejemplo de esta cualidad.

- Comprensible: Fácil de entender.
- Explicable: Facilidad para mostrar a las personas nuevas cómo funciona todo, sin tener que recurrir a descargar grandes documentos.

Mantener todo lo más simple posible, el mayor tiempo posible, sin agregar funcionalidad antes de programarlo. Sin embargo, no se puede dejar a un lado que el mantener un diseño simple es un trabajo duro.

→ Sistema metafórico: un sistema metafórico es una metáfora en sí misma para un diseño simple con ciertas cualidades. La cualidad más importante es ser capaz de explicar el diseño del sistema a personas nuevas. Un diseño debe tener una estructura que ayude a las personas a construir rápidamente. La segunda cualidad es un diseño que permita nombrar tipos y métodos constantes.

La identificación de los objetos es muy importante para entender el diseño general del sistema y código fuente. Se ahorra mucho tiempo si se identifican estos adecuadamente de tal forma que sea comprensible a simple vista. Por tal razón, se recomienda elegir un sistema de nombres de los objetos que todas las personas puedan relacionar sin mucha dificultad para adquirir conocimiento sobre el sistema.

→ Tarjetas Clase Responsabilidad Colaborador (CRC): funcionan para diseñar el sistema. Este instrumento permite al equipo de todo el proyecto contribuir con el diseño. Entre más personas colaboren al diseño del sistema mayor será el número de buenas ideas.

Estas tarjetas se utilizan para representar objetos. La Clase se escribe en la parte superior de la tarjeta, las Responsabilidades enumeradas abajo del lado izquierdo, las Colaboraciones aparecerán a la derecha de cada responsabilidad.

→ *Spike solution* para disminuir el riesgo: consiste en concentrar fuerzas en un determinado problema e ignorar las demás preocupaciones. El objetivo es reducir el riesgo de un problema técnico o aumentar la confiabilidad de la historia de usuario.

Cuando una dificultad técnica amenaza con retrasar el desarrollo del sistema es factible disponer de un par de trabajadores en el problema por una o dos semanas, de esta forma se reducirá el potencial de riesgo.

→ No agregar funcionalidades adicionales: Mantener el sistema ordenado con lo que realmente se requiere. La funcionalidad adicional siempre retrasará y desperdiciará los recursos disponibles.

Mantener el código fuente listo para cambios inesperados es parte del diseño simple. Agregar más de lo que se necesita hará un diseño más complejo, y es lo que se pretende evitar.

→ Refactorizar siempre que sea posible: cuando se elimina la redundancia, cuando se eliminan las funciones no utilizadas y se mejoran los diseños obsoletos; se está refactorizando. La refactorización en todo el ciclo de vida de todo el proyecto ahorra tiempo y aumenta la calidad.

Refactorizar en todo momento para mantener el diseño simple, evita la complejidad y el desorden innecesario. Mantener el código fuente limpio y conciso para que sea más fácil de entender, modificar y ampliar. Evitar

redundancias. Esto contribuye a disminuir el tiempo en la construcción del código fuente y producirá un sistema ordenado y simple.

2.3.5.2.4.4 Desarrollo.

Después de que las historias de usuario han sido elaboradas y de que se ha hecho el trabajo de diseño preliminar, el equipo no inicia las tareas de desarrollo (construcción de código fuente), sino que desarrolla una serie de casos de pruebas a cada una de las historias que se van a llevar a cabo durante la iteración.

→ El cliente siempre está disponible: corresponde a uno de los pocos requisitos de la Programación Extrema y lo es la disponibilidad del cliente, pero no debe ser cualquier persona, debe ser un experto en el tema, ya que no solo es para ayudar al equipo de desarrollo, si no para que sea también parte del equipo.

Las historias de usuario están escritas y priorizadas por el cliente, posteriormente junto a los desarrolladores se llevan a cabo las estimaciones de tiempo. Es responsabilidad de los clientes asegurar que la mayoría de la funcionalidad del sistema deseada esté integrada en las historias.

El cliente selecciona las historias que deben ser entregadas al finalizar la iteración, también decidirá en qué momento se liberará o se hará uso de esta funcionalidad. Los clientes deben tomar las decisiones que afectan sus objetivos

de negocio. También al formar parte del equipo y estar disponibles permitirá probar el sistema previamente y retroalimentar a los desarrolladores más pronto.

- → Estandarizar el código fuente: Es indispensable estandarizar o normalizar el código fuente, esto mantendrá la consistencia del código fuente y facilitará al equipo completo la lectura y refactorización.
- → Realizar los casos de pruebas: antes de realizar las tareas de desarrollo del sistema (construir el código fuente), se elaboran los casos de pruebas, esta tarea facilitará posteriormente el desarrollo del mismo.

  Crear los casos de pruebas ayuda al desarrollador a considerar lo que realmente necesita realizar. Esto también facilita refinar el requerimiento y determinar cuándo se ha terminado de construir la funcionalidad necesaria.

  Asimismo, esta tarea previa contribuye a construir código fuente más simple, ya que solo se tendrá que pensar en cómo superar esa prueba.
- → Programación en parejas: el código fuente es construido por dos personas, las cuales trabajan juntas en un equipo. Dos personas trabajando en una computadora agregan tanta funcionalidad como dos por separado. Esto da un mecanismo para la solución de problemas en tiempo real (es frecuente que dos cabezas piensen más que una) y para el aseguramiento de la calidad también en tiempo real (el código se revisa conforme se crea).
- → Integración secuencial: al desarrollar el sistema en forma incremental, por cada iteración se añade una funcionalidad o característica, lo que provoca nuevas versiones del sistema. Integrar el nuevo código fuente es una tarea que amerita orden y control. Sin control de integración los desarrolladores integran y prueban su código fuente y piensan que todo está bien, sin embargo, existe la posibilidad

de la integración paralela, lo que provocaría una combinación de código fuente que no ha sido probado en conjunto. Es en este punto donde los problemas de integración pasan sin ser detectados. La integración secuencial resuelve este problema y evita trabajar con versiones obsoletas.

El desarrollo (construir el código fuente) se realiza en paralelo mientras que la integración es secuencial. Muchos desarrolladores, por experiencia, consideran que la integración de código fuente consume mucho tiempo por lo que debe realizarse pocas veces; no obstante, se ha comprobado que integrar más seguido resulta una tarea rápida y fácil.

→ Integración frecuente: los desarrolladores deben integrar el código fuente y realizarlo cada vez que sea posible, en periodos de tiempo cortos. Evitar integrar los cambios por más de un día.

La integración continua evita fragmentaciones de esfuerzos de desarrollo, donde los desarrolladores no estén comunicándose entre sí sobre lo que puede ser reutilizado, o lo que podría ser compartido. Es necesario que el equipo trabaje con la última versión. Realizar cambios al código fuente obsoleto causa dolores de cabeza en la integración.

La integración continua evita o detecta problemas de compatibilidad.

→ Propiedad colectiva: alienta a todos a contribuir con nuevas ideas para todos los segmentos del proyecto. Cualquier desarrollador puede cambiar cualquier línea de código fuente para añadir funcionalidad, corregir errores, mejorar los diseños o refactorizar. No hay una sola persona que se convierta en un cuello de botella para los cambios.

En la práctica la propiedad colectiva es más confiable que poner a una sola

persona a cargo (lo que se conoce en la industria como Arquitecto de *software*). Especialmente porque esa persona puede abandonar el proyecto en cualquier momento.

2.3.5.2.4.5 Pruebas.

Como se menciona en la sección anterior, la creación de casos de pruebas antes de que comience las tareas de desarrollo (construcción de código fuente) es un elemento clave del enfoque de la Programación Extrema. Las pruebas son la medida de comprobación de la funcionalidad de cada uno de los módulos o componentes del sistema, pueden ser ejecutadas a diario y brindan información del avance que tiene el proyecto. Básicamente se compone de:

- → Casos de Prueba: la buena práctica es crear los casos de pruebas antes de construir el código fuente. Esto ayudará al desarrollador a elaborar el código fuente para pasar esa prueba. Por lo tanto, le permite determinar qué es lo que realmente necesita realizar.

  Son especificaciones de las entradas a la prueba y la salida esperada del sistema (los resultados de la prueba), además de información sobre lo que se pone a prueba. El resultado refleja el uso previsto de la funcionalidad o requerimiento (Sommerville, 2011, pág. 209).
- → Pruebas unitarias: Es el proceso de probar componentes del programa, como métodos o clases de objetos. Es poner a prueba el código fuente, el cual sin

pruebas no puede ser liberado. Si una prueba de unidad no aparece debe ser creada en el momento.

Otro aspecto a tomar en cuenta: es posible que agregar una nueva funcionalidad requerirá el cambio de las pruebas de unidad para reflejarla.

→ Pruebas de aceptación: Son realizadas por el usuario final o el cliente. Consiste en verificar el correcto funcionamiento de la característica o requerimiento.

Las pruebas garantizan un *software* sin errores -se debe realizar pruebas a diario y por separado- lo que ayuda a entregar al cliente un producto *software* confiable. Las pruebas no solo las realiza el equipo de desarrollo, también interviene el cliente, él tiene la última palabra, y en caso de tener nuevos requerimientos puede darlos en el momento de las pruebas (Murillo Montesdeoca, 2015).

2.3.6. Desarrollo de *software* tradicional.

Esta forma de desarrollo de *software* se caracteriza porque el proyecto está dirigido por un plan, es decir, el proceso de *software* se planea y organiza en etapas que se ejecutan secuencialmente, hasta que una etapa no finalice con éxito no se pasa a la siguiente. Por ejemplo, hasta finalizar la etapa diseño del sistema no se inicia con el desarrollo (construcción del código fuente).

Lo anterior conlleva a la ejecución de las etapas una sola vez. En caso de presentarse algún inconveniente o fallo durante la ejecución de una etapa implica todo un retroceso en las etapas.

Otra de las características importantes dentro de este enfoque son los altos costos al implementar un cambio y al no ofrecer una buena solución para proyectos donde el entorno es volátil.

El desarrollo de *software* tradicional (metodologías tradicionales) se focalizan en documentación, planificación y procesos; por ejemplo plantillas, técnicas de administración, revisiones, entre otros.

Entre la principal metodología tradicional se encuentra la conocida RUP (Proceso Unificado Racional), que centra su atención en llevar una documentación exhaustiva de todo el proyecto y se focaliza en cumplir con un plan (de proyecto) definido en la fase inicial del desarrollo del proyecto.

Por lo tanto, la modalidad de desarrollo de *software* tradicional se recomienda en proyectos donde los requerimientos sean estables, que no varíen o evolucionen en el tiempo. Aquellos casos donde los requerimientos evolucionen o sean cambiantes producto del dinamismo de la organización se recomienda el desarrollo ágil de *software*.

2.3.6.1 El Proceso Unificado Racional (RUP).

El Proceso Unificado Racional (RUP por sus siglas en inglés *Rational Unified Process*) es un ejemplo de un modelo de proceso moderno que se derivó del trabajo sobre el UML (*Unified Modeling Language*) y el proceso asociado de desarrollo de *software* unificado. Se considera el más comúnmente utilizado para el desarrollo de *software*.

Este proceso proporciona un enfoque disciplinado para asignar tareas y responsabilidades en un equipo de desarrollo de *software*. Su objetivo es garantizar la producción de software de alta calidad que satisfaga las necesidades de los usuarios finales (IBM, 2017).

Asimismo, es considerado un modelo de proceso híbrido, ya que conjunta elementos de todos los modelos de proceso de *software* (cascada, desarrollo incremental, evolutivo, entre otros), ilustra la buena práctica en especificación y diseño, y apoya la creación de prototipos y entrega incremental.

Por su parte, Vincze (2016) señala que RUP posee un enfoque orientado a objetos que se caracteriza por ser iterativo e incremental, manejando una serie de entregas ejecutables e integrando constantemente la arquitectura para una evolución continua del mismo, produciendo versiones mejoradas. De este modo, se centra en la

arquitectura para conceptualizar, construir y gestionar el sistema que se está desarrollando. Otras de sus características son:

- → Es conceptualmente amplio y diverso.
- → Busca implementar las mejores prácticas en Ingeniería de *Software*.
- → Maneja una forma disciplinada de asignar tareas y responsabilidades.
- → Permite mediciones, tales como: estimación de costos y tiempo, nivel de avance, entre otras.
- → Está dirigido por casos de uso, permitiendo establecer el comportamiento deseado del sistema.

Sommerville (2011, pág. 50) expone que RUP reconoce que los modelos de proceso convencionales presentan una sola visión del proceso. En contraste, el RUP por lo general se describe desde tres perspectivas:

a) Una dinámica que muestra las fases del modelo a través del tiempo.
b) Una estática que presenta las actividades del proceso establecidas.
c) Una práctica que sugiere buenos procedimientos a usar durante el proceso.

Mucha literatura referente a RUP describe el modelo combinando las perspectivas estática y dinámica en un solo diagrama; esto sin duda hace que este resulte más difícil de entender, por lo que a continuación se usarán descripciones separadas de las mismas.

El enfoque dinámico describe el modelo en cuatro fases discretas en el proceso *software*. Sin embargo, a diferencia del modelo en cascada, donde las fases se igualan con actividades del proceso, en el RUP están más estrechamente vinculadas con la organización que con las preocupaciones técnicas. Estas son:

1. Concepción: La meta de la fase de concepción es establecer un caso empresarial para el sistema. Deben identificarse todas las entidades externas (personas y sistemas) que interactuarán con este y definirán dichas interacciones. Luego se usa esta información para valorar la aportación del sistema a la empresa. Si esta aportación es menor, entonces el proyecto puede cancelarse después de esta fase.
2. Elaboración: Esta fase consiste en desarrollar la comprensión del problema de dominio, establecer un marco conceptual arquitectónico para el sistema, diseñar el plan del proyecto e identificar los riesgos clave.

    Al completar esta fase, debe tenerse un modelo de requerimientos para el sistema, que podría ser una serie de casos de uso del UML, una descripción arquitectónica y un plan de desarrollo para el *software*.

3. Construcción: La fase de construcción incluye diseño, programación y pruebas del sistema. Partes del mismo se desarrollan en paralelo y se integran durante esta fase.

    Al completar esta, debe tenerse un sistema de *software* funcionando y la documentación relacionada y lista para entregarse al usuario.
4. Transición: La fase final del RUP se interesa por el cambio del sistema desde el ambiente de desarrollo hacia el de producción, es decir, ponerlo a funcionar en

un ambiente real. Esto es algo ignorado en la mayoría de los modelos de proceso de *software*, aunque, en efecto, es una actividad costosa y en ocasiones problemática. En el complemento de esta fase se debe tener un sistema documentado que funcione correctamente en su entorno operacional.

La iteración con el RUP se apoya en dos formas. Cada fase puede presentarse en una forma iterativa, con los resultados desarrollados incrementalmente. Además, todo el conjunto de fases puede expresarse de manera incremental, como se muestra en la flecha en curva desde transición hasta concepción en la figura 15.

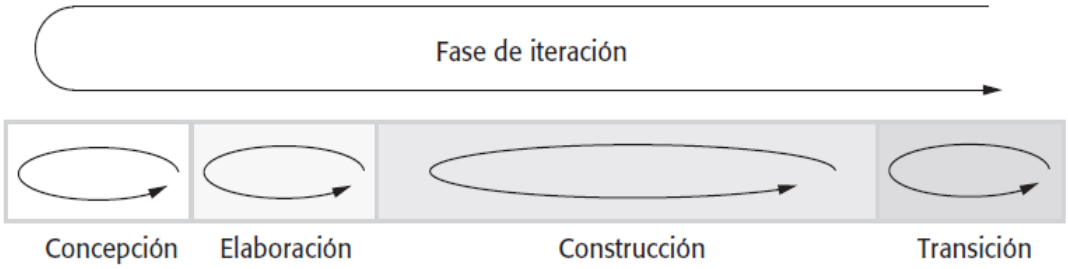


Figura 15. Fases en el Proceso Unificado Racional

Fuente: (Sommerville, 2011, pág. 51).

La visión estática del RUP se enfoca en las actividades que tienen lugar durante el proceso de desarrollo. Se les llama flujos de trabajo en la descripción RUP. En este se identifican seis flujos de trabajo centrales y tres flujos de trabajo de apoyo. El RUP se diseñó en conjunto con el UML, de manera que la descripción del flujo de trabajo se orienta sobre modelos UML asociados, como modelos de secuencia, modelos de

objeto, entre otros. En la tabla 1 se describen los flujos de trabajo de proceso centrales y los flujos de apoyo.

Tabla 1. Flujos de trabajo estáticos en el Proceso Unificado Racional

Flujo de trabajo	**Descripción**
Modelado del Negocio	Se modelan los procesos de negocios utilizando casos de uso de la empresa.
Requerimientos	Se identifican los actores que interactúan con el sistema y se desarrollan casos de uso para modelar los requerimientos del sistema.
Análisis y diseño	Se crea y documenta un modelo de diseño utilizando modelos arquitectónicos, de componentes, de objetos y de secuencias.
Implementación	Se implementan y estructuran los componentes del sistema en subsistemas de implementación. La generación automática de código a partir de modelos de diseño ayuda a acelerar este proceso.
Pruebas	Las pruebas son un proceso iterativo que se realiza en conjunto con la

	implementación. Las pruebas del sistema siguen al completar la información.
Despliegue	Se crea la liberación de un producto, se distribuye a los usuarios y se instala en su lugar de trabajo.
Administración de la configuración y del cambio	Este flujo de trabajo de apoyo gestiona los cambios al sistema.
Administración del proyecto	Este flujo de trabajo de apoyo gestiona el desarrollo del sistema.
Entorno	Este flujo de trabajo de apoyo pone a disposición del equipo de desarrollo de software, las herramientas adecuadas de este.

Fuente: (Sommerville, 2011, pág. 52)

Según el autor Sommerville (2011) la ventaja en la presentación de las visiones dinámica y estática radica en que las fases del proceso de desarrollo no están asociadas con flujos de trabajo específicos. En principio, al menos, todos los flujos de trabajo RUP pueden estar activos en la totalidad de las etapas del proceso. En las fases iniciales, es probable que se use mayor esfuerzo en los flujos de trabajo como modelado del negocio y requerimientos y, en fases posteriores, en las pruebas y el despliegue.

Por último, el enfoque práctico del RUP describe las buenas prácticas de ingeniería de *software* que se recomiendan para su uso en el desarrollo de sistemas. Las seis mejores prácticas fundamentales que se recomiendan son:

1) Desarrollo de *software* de manera iterativa. Incrementar el plan del sistema de acuerdo con las prioridades del cliente, y desarrollar oportunamente las características del sistema de mayor prioridad en el proceso de desarrollo.
2) Gestión de requerimientos. Documentar de manera explícita los requerimientos del cliente y seguir la huella de los cambios a dichos requerimientos. Analizar el efecto de los cambios sobre el sistema antes de aceptarlos.
3) Usar arquitecturas basadas en componentes. Estructurar la arquitectura del sistema en componentes.
4) *Software* modelado visualmente. Usar modelos UML gráficos para elaborar representaciones de *software* estáticas y dinámicas.
5) Verificar la calidad del *software*. Garantizar que el sistema cumpla con los estándares de calidad de la organización.
6) Controlar los cambios al *software*. Gestionar los cambios al sistema con un sistema de administración del cambio, así como con procedimientos y herramientas de administración de la configuración.

En síntesis, las innovaciones más importantes en el RUP son la separación de fases y flujos de trabajo, y el reconocimiento de que el despliegue del *software* en un entorno del usuario forma parte del proceso. Las fases son dinámicas y tienen metas. Los flujos

de trabajo son estáticos y son actividades técnicas que no se asocian con una sola fase, sino que pueden usarse a lo largo del desarrollo para lograr las metas.

CAPITULO III: MARCO METODOLÓGICO

3.1. TIPO Y ENFOQUE DE LA INVESTIGACIÓN.

Para efecto de este trabajo de investigación se ha optado por una finalidad aplicada, ya que mediante esta metodología logran identificarse las causas y efectos que contribuyen al problema, y permite aplicar los conocimientos de ingeniería para mejorar aspectos que minimicen los tiempos de implementación de un nuevo sistema, así como adecuar el proceso de *software* a las necesidades de la División Infraestructura y del Sector Telecomunicaciones del ICE, procurando simplificar las actividades del proceso.

A su vez, esta investigación es de naturaleza cualitativa, ya que buscará interpretar lo que se realiza en el proceso de desarrollo de *software*, a través de la observación del comportamiento natural del proceso; no consta de un descubrimiento sino de una construcción de conocimiento, obtenido del contexto natural entre las personas implicadas y toda su conducta observable.

3.2. FUENTES Y SUJETOS DE INFORMACIÓN.

A continuación, se detallan las fuentes de información de donde se obtiene el sustento teórico y práctico para fundamentar la propuesta a desarrollar en el proyecto,

así como los sujetos de información a quienes se contacta para la obtención de información valiosa para el proyecto.

3.2.1. Fuentes primarias.

Las fuentes de esta tesis serán principalmente las siguientes:

- Entrevista aplicada al equipo de desarrollo de *software* del área Tecnologías de la Información de la División Infraestructura, Sector Telecomunicaciones del ICE.
- Metodología para el rediseño de procesos propuesta por Madison (2005).
- Marco de trabajo Scrum.
- Marco de trabajo Programación Extrema.
- Libros digitales afines con temas de ingeniería de *software*.

3.2.2. Fuentes secundarias.

Se tomará las siguientes fuentes:

- Revistas científicas y tecnológicas.
- Tesis.
- Información obtenida de internet.

3.2.3. Sujetos de información.

En la tabla 2 se muestran los sujetos de información.

Tabla 2. Sujetos de información.

Puesto laboral o descripción general	**Profesión u oficio**	**Experiencia**	**Relación con el tema**
Coordinadora	Ingeniera en Sistemas con énfasis en Administración en Recursos Informáticos.	12 años	Directora de proyectos en el área Tecnologías de la Información, División Infraestructura, ICE.
Desarrollador	Ingeniero en Sistemas	15 años	Desarrollador de proyectos el área Tecnologías de la Información, División Infraestructura, ICE.
Desarrollador	Ingeniero en Sistemas	20 años	Desarrollador de proyectos el área Tecnologías de la Información, División Infraestructura, ICE.
Analista	Ingeniero en Sistemas	5 años	Analista de sistemas en los proyectos de desarrollo, área Tecnologías de la

			Información, División Infraestructura, ICE.

Fuente: Elaboración propia.

3.3. TECNICAS Y HERRAMIENTAS.

Para cumplir con los objetivos planteados en la investigación, se requiere la aplicación de instrumentos de recolección de la información que serán utilizados para el análisis de la situación por estudiar y el planteamiento de soluciones.

Según Ulate y Vargas (2014) se define la observación como "… el procedimiento para obtener datos de la realidad mediante la percepción intencionada y selectiva de un objeto o fenómeno determinado" (pág. 76). Esta técnica permitirá obtener información acerca del proceso de desarrollo de *software* que se lleva a cabo, así como la interacción que tienen los involucrados en este.

Por otra parte, se define encuesta como "… una serie de preguntas que están dirigidas a una porción representativa de una población, y tiene como finalidad averiguar estados de opinión, actitudes o comportamientos de las personas ante asuntos específicos." (Significados, 2014).

Se conoce como entrevista, la conversación que sostienen dos o más personas que se encuentran en el rol de entrevistador y entrevistado, con la finalidad del primero de obtener determinada información sobre un tema que pueda proporcionarle el segundo. Consiste en plantear una serie de preguntas al entrevistado con el objetivo de que se exponga, explique o argumente su opinión, su punto de vista o simplemente brinde información sobre determinado tema (Editorial MD, 2017).

Brainwriting es una dinámica de grupo similar a *brainstorming* (lluvia de ideas) pero con la particularidad de que los participantes escriben en un papel sus ideas antes de ponerlas en común con los demás. Esta técnica busca generar ideas de manera espontánea, rápida y creativa, así como aumentar la participación de las personas y que, además sea de forma uniforme (Duro Lima, 2016).

Para esta investigación se aplicarán las siguientes herramientas:

a) Una encuesta estructurada con 19 preguntas distribuidas de la siguiente manera:
    i. 12 preguntas para conocer la percepción del grupo de trabajo de desarrollo respecto a la metodología de desarrollo de *software* utilizada. (Ver apéndice 1).
    ii. 7 preguntas a los clientes para identificar puntos de mejora en el servicio brindado, así como para conocer la satisfacción con el producto *software*. (Ver apéndice 2).

b) Una entrevista conformada por 14 preguntas para conocer la percepción de la coordinadora referente a la metodología de desarrollo de *software* utilizada, así como otros aspectos de la gestión de TI. (Ver apéndice 3).
c) Investigación bibliográfica y *benchmarking* (Ver apéndice 4) con el fin de ver ejemplos de cómo llevan a cabo el proceso de *software* otras personas, con el propósito de tomar ideas que se puedan aplicar al proyecto.
d) Reuniones con los compañeros de trabajo con el fin de mostrar los avances en el diseño de la metodología para considerar la retroalimentación de cada uno.

3.4. VARIABLES DE INVESTIGACIÓN.

Tabla 3. Variables de investigación

Objetivos específicos	**Variables asociadas**	**Descripción**
Diagnosticar la metodología de desarrollo de *software* actual para evaluar su proceso *software*.	Metodologías de desarrollo de *software*. Características de metodologías de desarrollo. Características de proceso *software*.	Generar un diagnóstico que permita identificar puntos de mejora en el proceso *software*.
Identificar las mejores prácticas usadas en la	Estándares de desarrollo de *software*.	Crear un documento donde se explique los

industria, que procedan de diversas fuentes como estándares y metodologías de desarrollo.	Metodologías de desarrollo.	estándares y metodologías de desarrollo identificadas.
Diseñar la metodología de desarrollo de *software* basado en estándares y las mejores prácticas, ampliamente utilizadas, en la industria.	Estándares de desarrollo de *software*. Mejores prácticas de la industria del *software*.	Crear un documento donde se explique la propuesta de estándares y las mejores prácticas empleadas en la metodología de desarrollo.
Realizar un ejercicio para evaluar si el diseño propuesto en la metodología de desarrollo de *software* se adecúa a las necesidades de la institución.	Metodología propuesta.	Desarrollar un módulo de un proyecto haciendo uso de la metodología de desarrollo de *software* propuesta.

Fuente: Elaboración propia.

3.5. DISEÑO DE INVESTIGACIÓN.

Para perfeccionar los procesos de negocio en una organización es importante hacer uso de una metodología, ya que esto propicia la mejora estratégica en los procesos que lo requieran, con el fin de conseguir una mayor efectividad, eficiencia y flexibilidad y afrontar así los cambios constantes que ocurren en el entorno. El proceso metodológico colabora para dicha mejora ejecutando un análisis del mismo congruente con la estrategia establecida, definiendo métricas y controles, detectando debilidades e implementando soluciones para lograr la mejora continua afín a los objetivos empresariales.

Para el diseño de esta investigación se tomó como referencia la metodología para el rediseño de procesos propuesta por Madison (2005), la cual define este proceso de rediseño como un conjunto de 10 pasos divididos en cuatro grandes fases, a continuación la explicación de cada fase.

Es importante mencionar que para efectos y alcance de este proyecto, se ejecutará hasta el paso 6 de la fase 3.

3.5.1. Fase 1.

Esta fase consta de 2 pasos:

Paso 1 - Introducción al proceso de rediseño: El primer paso es la introducción al proceso de mejora y consta de 2 partes. La primera consiste en una serie de reuniones y actividades para elegir el proceso que será rediseñado. La segunda constituye en crear un diagrama del proceso que se rediseñará, con el fin de definir el alcance, las principales actividades y los límites del mismo; así como los objetivos de mejora.

Paso 2 – Formación del equipo de proceso: Consiste en crear el equipo de trabajo que llevará a cabo el rediseño. Junto con los miembros del equipo hay un director de proyecto, quien debe haber liberado un esfuerzo de cambio organizacional y tener experiencia manejando problemas que ocurren durante el rediseño.

También es importante considerar en el equipo de trabajo a un facilitador con experiencia en gestión de calidad (*six sigma*, principios de diseño, *benchmarking*, mejores prácticas, entrenamiento, entre otras); y por último un informático experto en temas de desarrollo de *software*.

3.5.2. Fase 2.

Esta fase consiste en el diagnóstico de la metodología de desarrollo de *software* actual mediante la creación del diagrama *AS-IS*, y las entrevistas a los clientes inmersos en el proceso. El análisis del diagnóstico da como resultado la identificación de los puntos de mejora. Asimismo, a través de técnicas como la investigación y *benchmarking* se descubrirán maneras para solventar las deficiencias del proceso.

Paso 3 – Creando el diagrama de proceso "*AS-IS*": El objetivo de este paso es crear un diagrama del proceso (funcional-actividad) antes de ser sometido al rediseño. Esta representación gráfica facilita el análisis y la descomposición de los procesos en actividades; obteniendo así un diagnóstico preliminar, el cual en conjunto con el siguiente paso afianza el diagnóstico.

Paso 4 – Entrevista al cliente: se realiza con el objetivo de averiguar lo que el cliente necesita, quiere, desea y requiere. Además, se deben formular las preguntas que se generaron a partir del estado en que se encuentra el diagrama realizado en el paso tres.

Paso 5 – *Benchmarking* y mejores prácticas: Se concentra en la evaluación comparativa y la investigación de mejores prácticas. La idea es descubrir nuevas formas de atacar los problemas.

3.5.3. Fase 3.

En la fase 3 se crea el diagrama de procesos según los objetivos de mejora planteados, por ejemplo, mejoras en tiempo, la calidad o el costo. Esta propuesta es revisada por el equipo de trabajo con el propósito de recibir opiniones y retroalimentación; posteriormente, y en caso de ser necesario, hacer una mayor investigación sobre ciertos puntos donde se encuentren deficiencias, para realizar el diseño final y su respectiva implementación.

Paso 6 – Rediseño desde cero: Cada miembro del equipo escribe una historia del proceso ideal. Esta es presentada al equipo, y los miembros del equipo enlistan las ideas de su agrado. La idea final es que el equipo sea capaz de llegar a un nuevo proceso consensuado, con base en las nuevas ideas, y así lograr que todos estén de acuerdo. En los casos en que el equipo no puede ponerse de acuerdo en un único diseño nuevo, se puede determinar el mejor a través de la ejecución de pruebas (simulación, práctica y juego de roles) que se produce en el siguiente paso. Por lo tanto, no es necesario que el equipo acuerde un solo diseño inicialmente.

Paso 7 – Presentar el rediseño a la alta dirección: se presenta el diseño elaborado en el paso seis a los miembros de la alta gerencia, con el fin de evitar sorpresas en el futuro. Debe incorporarse los temas de implementación y gestión de riesgos.

Paso 8 – Compartir el rediseño con el personal y clientes: Después de que la alta gerencia lo haya aprobado, este debe ser compartido con el personal y los clientes, el propósito es recibir retroalimentación. Deben documentarse las reacciones y aportes del nuevo proceso.

Paso 9 – Implementación del rediseño: este paso implica implementar el rediseño. Se puede iniciar con ejecutar una práctica, seguido de una prueba piloto, y finalmente realizar una introducción progresiva.

Las estrategias de implementación varían de proceso a proceso. Cada diseño tiene sus propios factores de riesgo y de ejecución, por lo que la elección de opciones de implementación serán diferentes para cada tiempo. El paso nueve termina cuando el diseño nuevo está completamente desplegado e implementado.

3.5.4. Fase 4.

La última fase consiste en establecer un sistema de mejora continua, mediante la implantación de mecanismos de control e indicadores que permitan la retroalimentación acerca de los resultados del rediseño del proceso.

Paso 10 – Instalación de métricas y mejora continua: Consiste en establecer un sistema de mejora continua. Se definen e instalan mecanismos de medición y retroalimentación en el nuevo proceso. Un asesor o consultor supervisa las métricas de los problemas presentados. Se eligen empleados para que trabajen en el ensamblaje y para actividades de mejora continua que son facilitadas por el asesor de proceso.

En la figura 16 se presenta un resumen de cada paso:

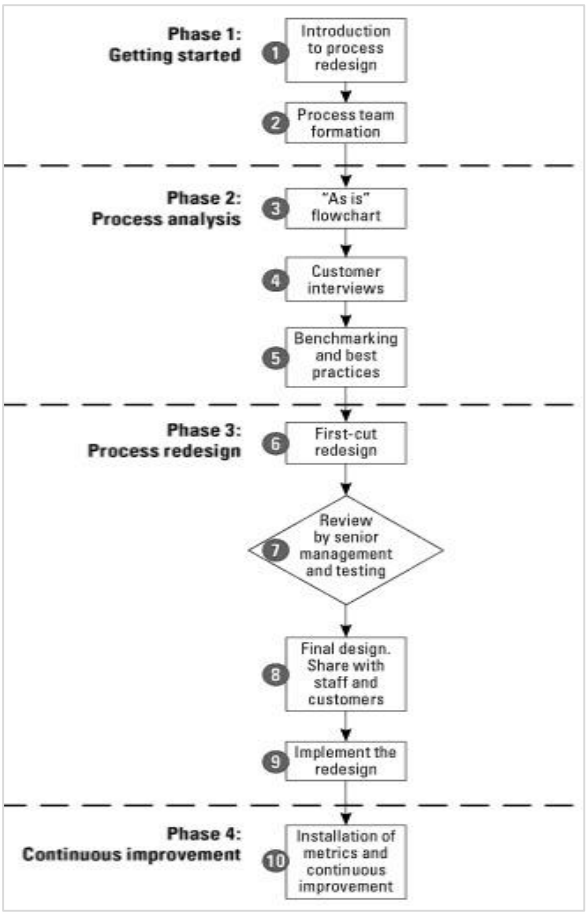


Figura 16. Metodología de rediseño de procesos en diez pasos.

Fuente: (Madison, 2005, pág. 66).

CAPITULO IV: DIAGNÓSTICO

4.1. DIAGNÓSTICO DE LA SITUACION ACTUAL.

En esta sección se explica a modo muy general la situación actual del área Tecnologías de la Información, División Infraestructura, Sector Telecomunicaciones del ICE, referente al desarrollo de sistemas, con el propósito de identificar posibles puntos de mejora e incorporarlos en la propuesta.

Actualmente el esquema de trabajo actual consta de: directora de proyectos, analista de sistema y dos desarrolladores. Además, el equipo de desarrollo dispone de una arquitectura de herramientas establecida para el desarrollo de sistemas, en el cual se define el manejo de estándares, estilos y otros aspectos que permiten implementar funcionalidades básicas tales como: reportes de sistema, modo de autenticación, entre otros. Asimismo, cuenta con los siguientes ambientes: desarrollo, *staging* (pruebas) y producción, los cuales son idénticos entre sí con el objetivo de aumentar la validez de las pruebas.

Cuando algún proceso o área requiere el desarrollo de un sistema informático designa a un intermediario, este puede ser un director de proyecto técnico, Director de Dirección, jefe de la División, el cual solicita a la directora de proyectos (coordinadora de TI, División Infraestructura) una sesión de trabajo para exponer la necesidad del sistema. La directora de proyectos designa a un desarrollador para asistir a la sesión de trabajo en conjunto con el analista de sistema. Además, de ser necesario también participa un usuario experto interesado en el sistema.

La primera sesión consiste en conocer la necesidad y los requerimientos preliminares del sistema, se determina la factibilidad de su desarrollo o bien; se analiza alternativas para brindar alguna solución con otra herramienta, quizás, ya desarrollada. En caso de proceder con el desarrollo, se realiza el siguiente procedimiento para la fase de levantamiento de requerimientos:

- El solicitante (intermediario) debe recolectar, depurar y priorizar los requerimientos del sistema, así como completar los documentos: Perfil del proyecto (este contiene la información general del proyecto, objetivos, descripción del proyecto, problema/necesidad/oportunidad a resolver, entre otros) y el Formulario para el levantamiento de requerimientos (básicamente especifica los requerimientos funcionales y no funcionales, datos de entrada y salida, entre otros). (Ver Anexo 1 y Anexo 2, respectivamente).

El analista en conjunto con el desarrollador designado para el proyecto realiza una sesión de trabajo con el intermediario para explicar cómo se debe completar la documentación antes mencionada.

Una vez que se completan los documentos, los envía mediante correo electrónico a la directora de proyecto, seguidamente se pasa al desarrollador asignado y al analista para que proceda con el estudio y comprensión de los requerimientos; en caso de dudas se coordina una o más sesiones para aclararlas y garantizar la comprensión del requerimiento.

En esta fase de levantamiento de requerimientos, se considera la complejidad y naturaleza técnica de la División por lo que el alcance de los requerimientos varía, es decir, no puede ser tan estricto, para así asegurar la correcta definición de la funcionalidad del sistema. Por lo tanto, en algunas ocasiones se utiliza el modelo de proceso de *software* evolutivo por prototipo.

Cuando los requerimientos se encuentran bien definidos, el desarrollador y directora de proyecto llevan a cabo la estimación de tiempo para el desarrollo del *software*, y se contempla: la experiencia obtenida de otros proyectos y la codificación (programación) que con lleva el nuevo sistema. Estas métricas no se encuentran asociadas a estándares utilizados en la industria.

Posteriormente se elabora el cronograma del proyecto, en esta fase se consideran:

- Las actividades actualmente asignadas al desarrollador.
- La priorización del nuevo sistema.

Con lo anterior, se establece la fecha de inicio del desarrollo.

La fase de diseño consiste en elaborar a mano alzada el esquema del sistema, con sus principales funcionalidades, esto le facilita al desarrollador el entendimiento e interacción de los componentes del sistema, y le permite determinar qué es lo que necesita para poder llevar a cabo la programación. No existe documentación, el diseño se realiza en papel y lápiz, y posteriormente se desecha.

La fase de desarrollo inicia con la creación de la base de datos requerida, la cual almacenará los datos que gestionará el sistema informático. Posteriormente, se realiza la programación del sistema informático, mediante la codificación de las funcionalidades por módulos, esto por aspectos de simplicidad; sin embargo, se realiza el entregable del producto *software* hasta finalizar la totalidad del desarrollo del sistema.

En esta fase, la documentación es mínima. Además, es posible que el desarrollador consulte al intermediario o usuario experto para despejar dudas que surgen en el proceso de programación o por alguna definición de requerimiento, repitiendo este ciclo hasta terminar el proyecto.

Durante el proceso de desarrollo de *software*, la directora de proyecto consulta el estado de las actividades asignadas al desarrollador con el fin de brindar seguimiento al avance, y verificar el estado del cronograma definido. Se utiliza Microsoft Project como herramienta para el seguimiento del proyecto.

Una vez finalizado el desarrollo del sistema, el desarrollador realiza un manual muy básico para el usuario final.

El proceso de *software* se caracteriza por ser un ambiente de desarrollo atípico, por las siguientes razones:

→ Cambios repentinos en las prioridades de las actividades.

- → Salidas abruptas de las actividades programadas, por ejemplo, interrupciones en la ejecución de las tareas asignadas para asistir a reuniones, atender incidentes de sistemas informáticos que se encuentra en producción, administración de bases de datos, entre otros.

Lo anterior ocasiona desenfoque en el proyecto, y consecuentemente impacta en la fluidez de las ideas, creatividad, y motivación en el equipo de desarrollo.

4.2. DIAGNÓSTICO ADMINISTRATIVO U OPERATIVO.

El proceso de *software* actual no posee documentación, sin embargo, sí es bien conocido por todos los trabajadores y se encuentra bastante definido.

Los sistemas informáticos que se implementan por el área de Tecnologías de la Información son desarrollos de aplicación interna en la División Infraestructura, y se ponen a disposición del Sector Telecomunicaciones de la institución.

No se cuenta con un portafolio de proyectos definida en forma anual, se atienden las solicitudes conforme surgen las necesidades de sistemas por los diferentes procesos/negocios de la División Infraestructura, incluso algunos casos obedecen a solicitudes inmediatas por directivos del Sector Telecomunicaciones. Además, se priorizan según la necesidad, y estrategia del negocio de la División (por ejemplo, algún requerimiento por parte de la SUTEL, ventaja competitiva, entre otros).

Existen excepciones en algunos sistemas informáticos, los cuales no son desarrollados a lo interno de la organización considerando que el mercado los elaboró acorde con las mejores prácticas y estándares particulares para el diseño de la red, los cuales son adquiridos e implementados por el área Tecnologías de la Información, son casos muy particulares y corresponde a *software* muy específico y especializado de uso en la División Infraestructura, en estos casos se opta por adquirir un *ERP*, considerando que no es factible su desarrollo y resulta más costoso su elaboración que la compra; por ejemplo, *iBwave Design*: *software* para diseñar redes inalámbricas grandes y complejas en edificios, permitiendo soporte multi-tecnología, multi-construcción, modelado 3D avanzado, simulaciones de cobertura, cálculos de presupuesto, comprobación de errores en diseño, entre otras.

Finalizada la fase de programación del *software* por parte del desarrollador, se procede a hacer el pase al ambiente *staging* (pruebas), con el propósito de testear el sistema. Se lleva a cabo el siguiente procedimiento:

a) Publicación (*deploy*) de la aplicación Web en servidor de aplicaciones *Glassfish* en ambiente *staging*.
b) Import de la base de datos Oracle desde el ambiente de desarrollo al servidor *staging*.
c) Creación de usuarios y roles en ambiente *staging*.
d) Comunicación a los usuarios para que realicen las pruebas.
e) Recepción y realización de eventuales ajustes que se requieran para continuar las pruebas.
f) Recepción de documento resultado de las pruebas.

g) Realización de eventuales ajustes.
h) Volver al punto a) sucesivamente, hasta que el sistema quede certificado por el usuario.

Cuando el sistema se encuentra en el ambiente de pruebas, tanto los usuarios expertos como usuarios finales designados por el intermediario se encargan de realizar las pruebas respectivas, con el propósito de validar las funcionalidades requeridas y dar el visto bueno, o dar al desarrollador la retroalimentación para mejorar el requerimiento, con el fin de obtener lo solicitado; y por último hacer el pase al ambiente de producción para que el sistema esté disponible al proceso/negocio interesado.

Por último, se realiza una breve inducción al intermediario y a algunos usuarios expertos con el producto *software* en su versión final, con el propósito de que estos realicen la transferencia de conocimiento a los usuarios finales que harán uso del sistema informático.

4.3. DIAGNÓSTICO TÉCNICO.

La División Infraestructura cuenta con 35 servidores virtuales, de los cuales 6 son para el ambiente de desarrollo de sistemas (desarrollo, *staging* y producción). A continuación, la distribución:

Funcionalidad/Descripción	Cantidad de servidores
Reservorios de información de la División Infraestructura	26
Licenciamiento software: AutoCAD, Atoll	
Ambiente de desarrollo de sistemas: Desarrollo	2
Ambiente de desarrollo de sistemas: Staging	2
Ambiente de desarrollo de sistemas: Producción	2

Figura 17. Distribución de servidores en la División Infraestructura

Fuente: Elaboración propia.

En la siguiente figura se muestra el detalle de los servidores para el ambiente de desarrollo de sistemas.

Funcionalidad/Descripción	Rol	Especificaciones Técnicas	Software
Ambiente de desarrollo de sistemas: Desarrollo	Base de datos	Procesador: AMD Opteron Procesador 6176, 2.30 GHz (8 procesadores) RAM: 16 GB DD: 750 GB	Oracle 11g R2 Windows Server 2008 R2 Enterprise Service Pack 1
	Aplicaciones	Procesador: AMD Opteron Procesador 6176, 2.30 GHz (8 procesadores) RAM: 16 GB DD: 250 GB	Glassfish 3.1 Windows Server 2008 R2 Enterprise Service Pack 1
Ambiente de desarrollo de sistemas: Staging	Base de datos	Procesador: Intel Xeon 5160, 2.99 GHz (2 procesadores) RAM: 8 GB DD: 400 GB	Oracle 11g R2 Windows Server 2008 R2 Enterprise Service Pack 1
	Aplicaciones	Procesador: AMD Opteron Procesador 6176, 2.30 GHz (8 procesadores) RAM: 16 GB DD: 750 GB	Glassfish 3.1 Windows Server 2008 R2 Enterprise Service Pack 1
Ambiente de desarrollo de sistemas: Producción	Base de datos	Procesador: AMD Opteron Procesador 6176, 2.30 GHz (8 procesadores) RAM: 16 GB DD: 250 GB	Oracle 11g R2 Windows Server 2008 R2 Enterprise Service Pack 1
	Aplicaciones	Procesador: AMD Opteron Procesador 6176, 2.30 GHz (8 procesadores) RAM: 16 GB DD: 250 GB	Glassfish 3.1 Windows Server 2008 R2 Enterprise Service Pack 1

Figura 18. Detalle de servidores para el ambiente de desarrollo de sistemas.

Fuente: Elaboración propia.

Estos servidores se encuentran acondicionados para todos los sistemas que se desarrollan en el lenguaje de programación Java, además como una buena práctica poseen las mismas especificaciones técnicas para garantizar su rendimiento óptimo y aumentar la validez de las pruebas.

4.4. DIAGNÓSTICO DE PERCEPCIÓN.

Se realizó una encuesta para conocer la apreciación sobre la metodología de desarrollo de *software*, con la participación de los tres miembros del grupo de desarrollo de *software*, coordinadora de TI, y clientes (incluyendo jefaturas); los resultados obtenidos se muestran a continuación.

4.4.1 Resultado de la encuesta realizada al grupo de desarrollo de software.

Este grupo está conformado por tres personas. Los resultados fueron los siguientes:

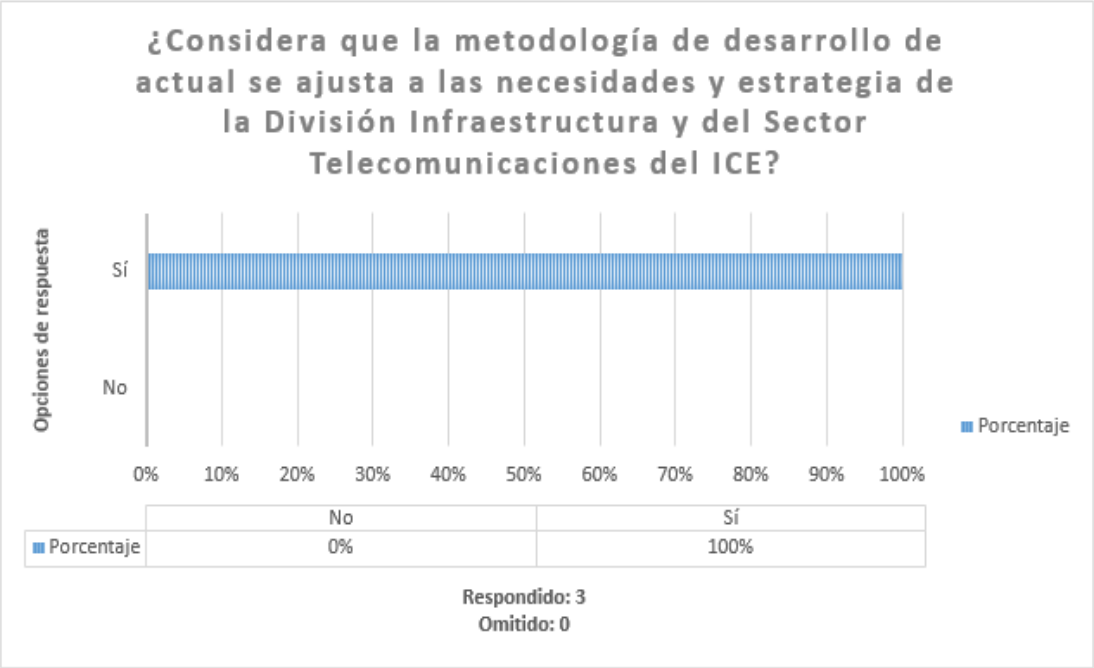


Figura 19. Percepción metodología de desarrollo actual.

Fuente: Elaboración propia.

De acuerdo con la figura 19, todos los encuestados consideran que la metodología de desarrollo actual se ajusta a las necesidades y estrategia de la División Infraestructura y del Sector Telecomunicaciones del ICE.

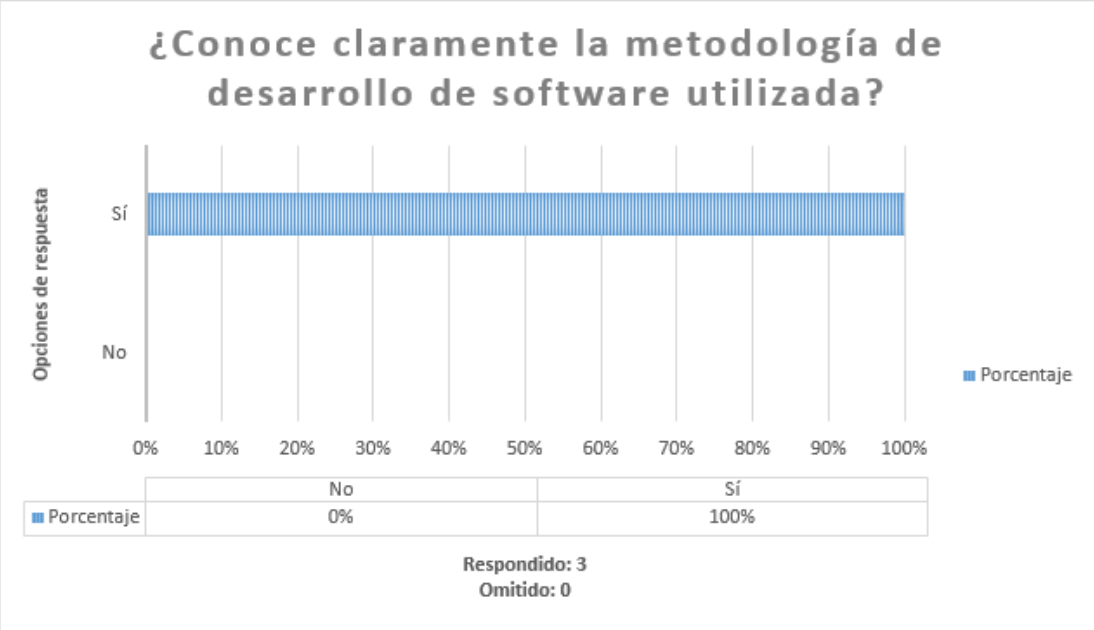


Figura 20. Conocimiento de la metodología de desarrollo de software utilizada.

Fuente: Elaboración propia.

Según la figura 20, todos los encuestados conocen claramente la metodología de desarrollo de *software* utilizada.

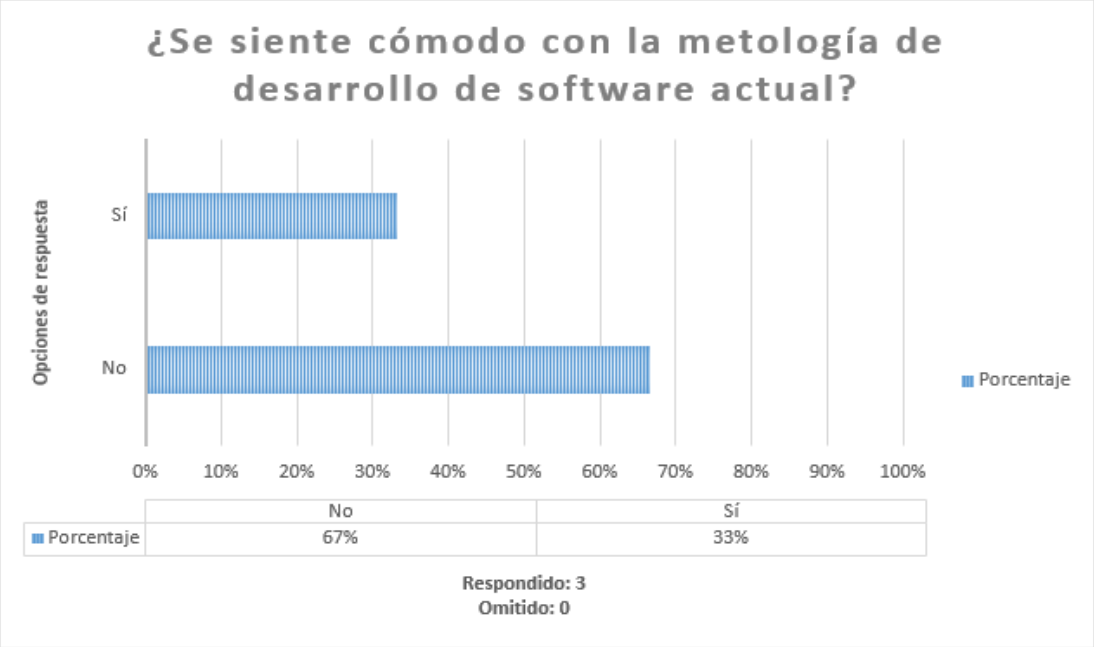


Figura 21. Percepción de comodidad con la metodología de desarrollo de software.

Fuente: Elaboración propia.

En la figura 21, la mayoría de los encuestados no se sienten cómodos con la metodología de desarrollo de *software* actualmente utilizada, esto podría incidir con el rendimiento y productividad del grupo de desarrollo de *software*.

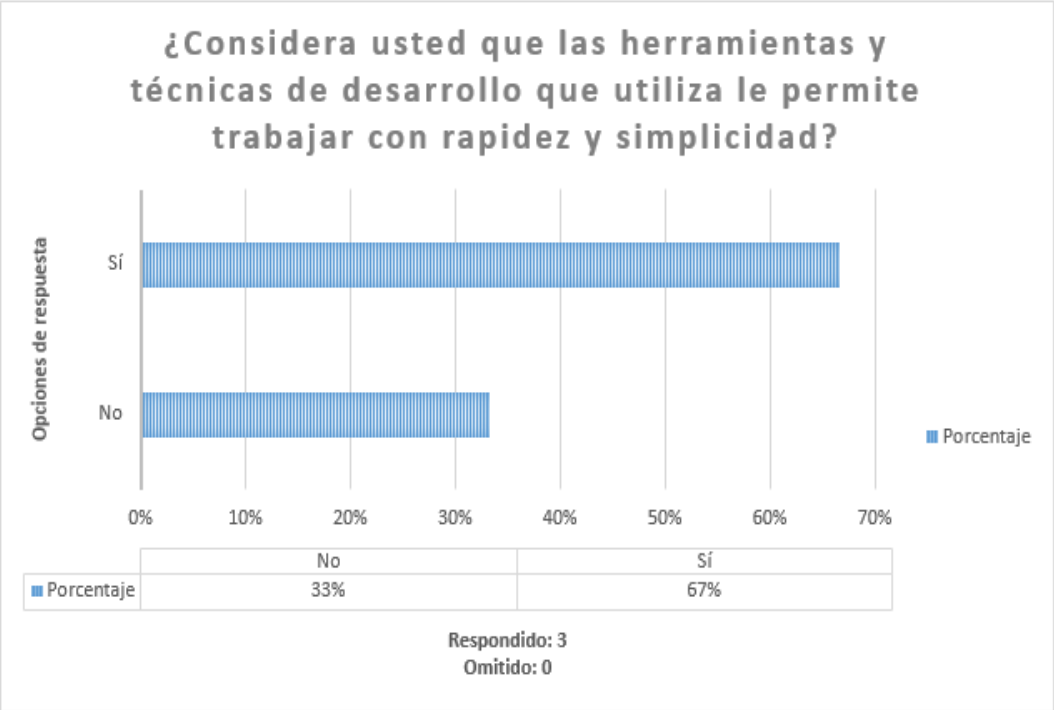


Figura 22. Percepción de las herramientas y técnicas utilizadas.

Fuente: Elaboración propia.

De conformidad con la figura 22, se observa que la mayor parte de los encuestados consideran que las herramientas y técnicas de desarrollo que utilizan les permite trabajar con rapidez y simplicidad.

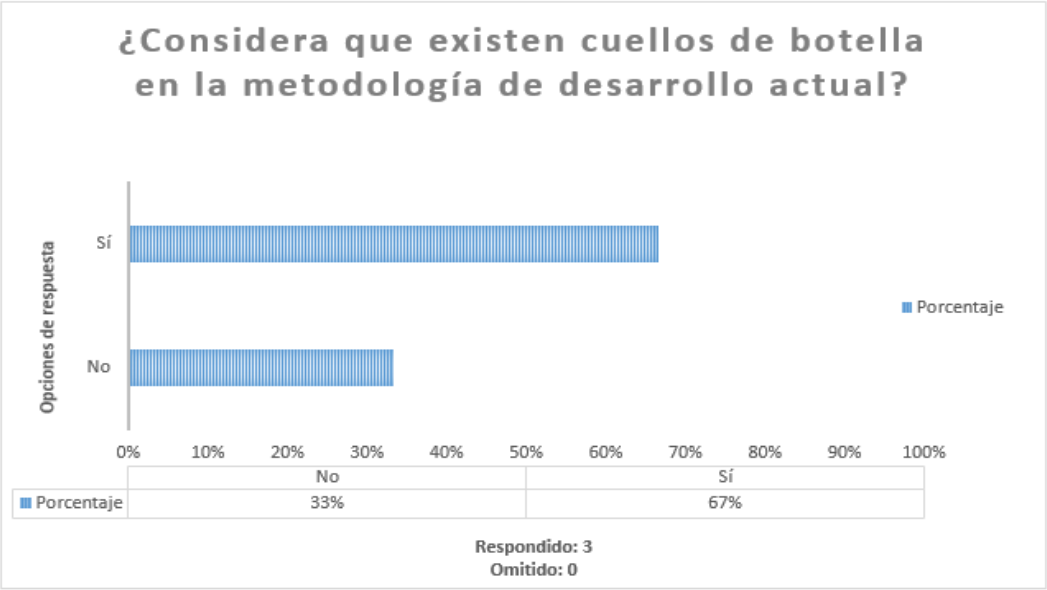


Figura 23. Cuellos de botella en la metodología de desarrollo de *software*.

Fuente: Elaboración propia.

En la figura 23, la mayoría opinan que existen cuellos de botella en la metodología de desarrollo de *software*, uno de estos factores se debe a la complejidad de la documentación.

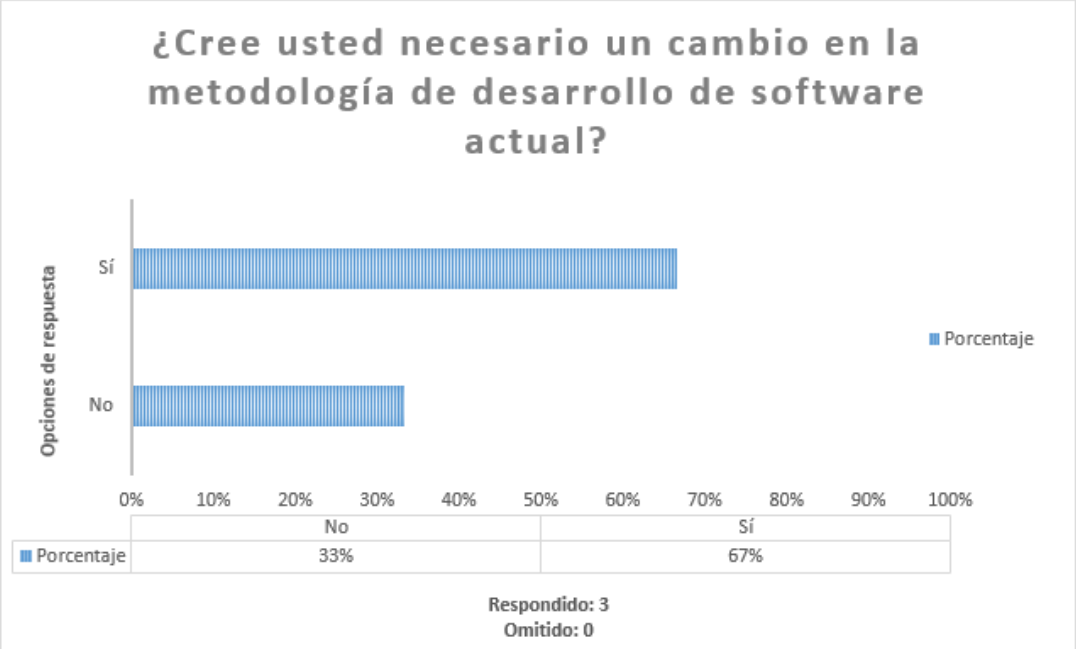


Figura 24. Opinión sobre cambio en la metodología de desarrollo de software.

Fuente: Elaboración propia.

De acuerdo a la figura 24, solo una persona considera que no es necesario un cambio en la metodología de desarrollo de *software* actualmente utilizada.

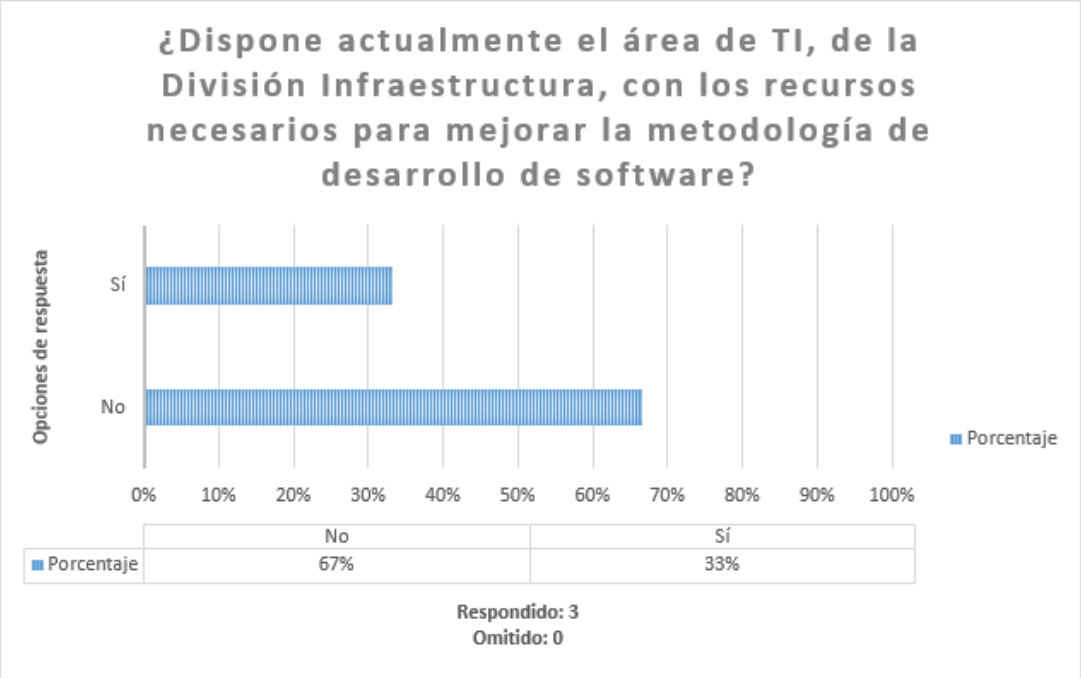


Figura 25. Disponibilidad de recursos para mejorar la metodología de desarrollo de software.

Fuente: Elaboración propia.

Según la figura 25, la mayor parte de los encuestados opinan que el área de Tecnologías de la Información no dispone de los recursos necesarios para mejorar la metodología de desarrollo de *software*, señalan el poco recurso humano.

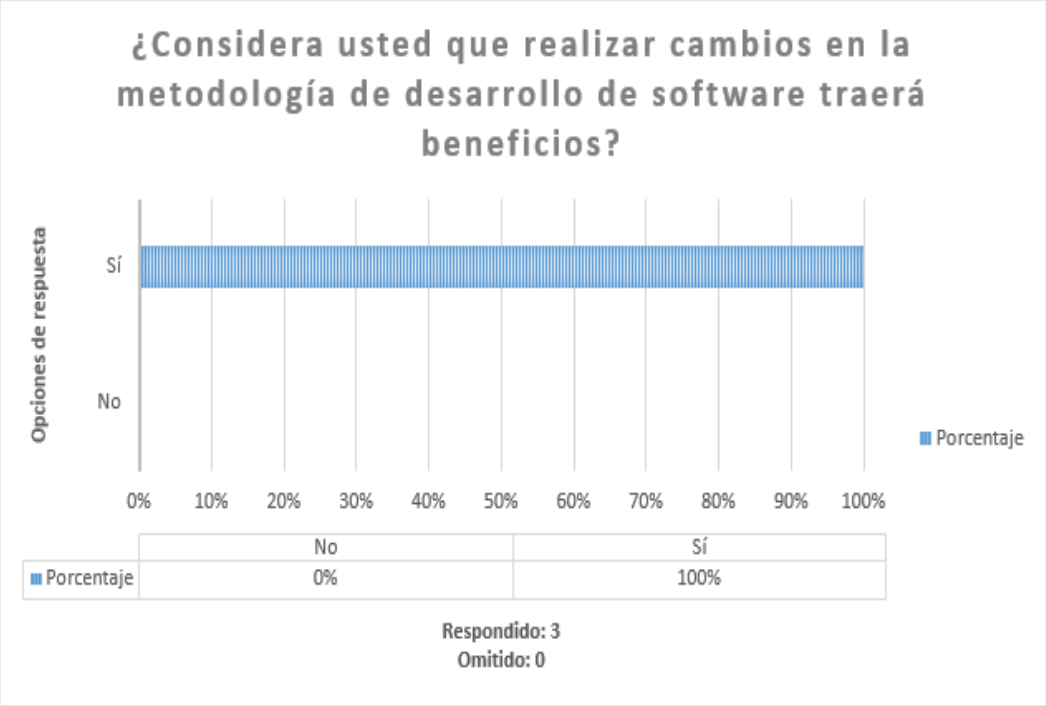


Figura 26. Apreciación de cambios en metodología de desarrollo de software.

Fuente: Elaboración propia.

En la figura 26, todos los encuestados consideran provechoso realizar cambios en la metodología de desarrollo de *software*, esta posición facilitaría la implementación de mejoras en la metodología actualmente utilizada.

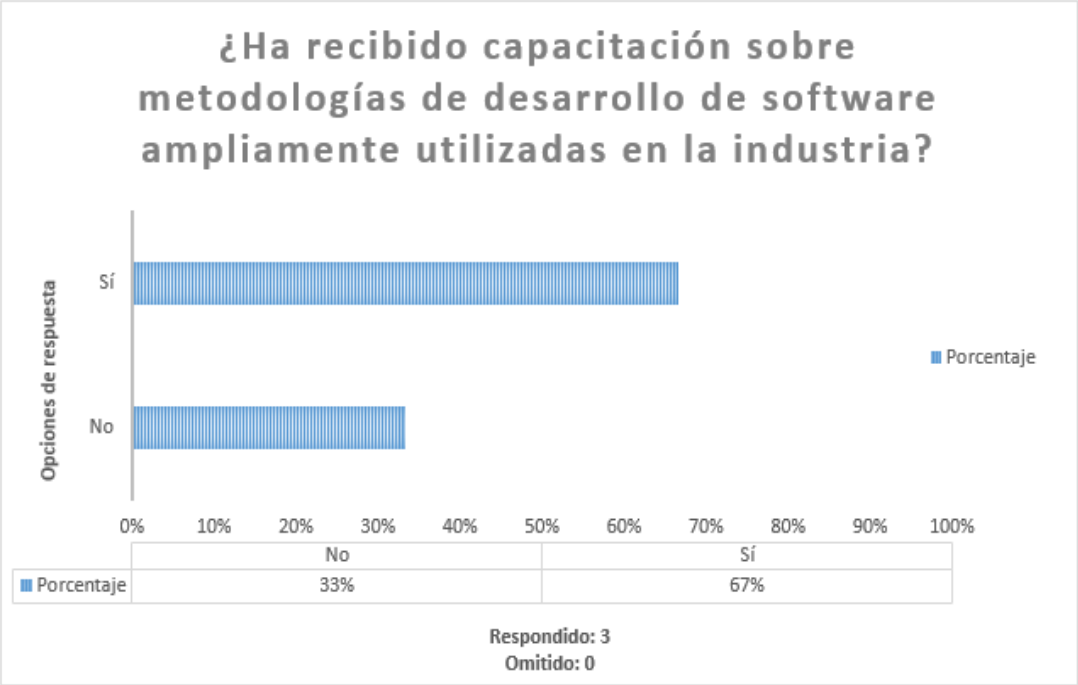


Figura 27. Capacitación en metodologías de desarrollo de software.

Fuente: Elaboración propia.

En la figura 27, solamente una persona no ha recibido capacitación referente a metodologías de desarrollo de *software* utilizadas en la industria, siendo esta la que reciba una afectación con el cambio.

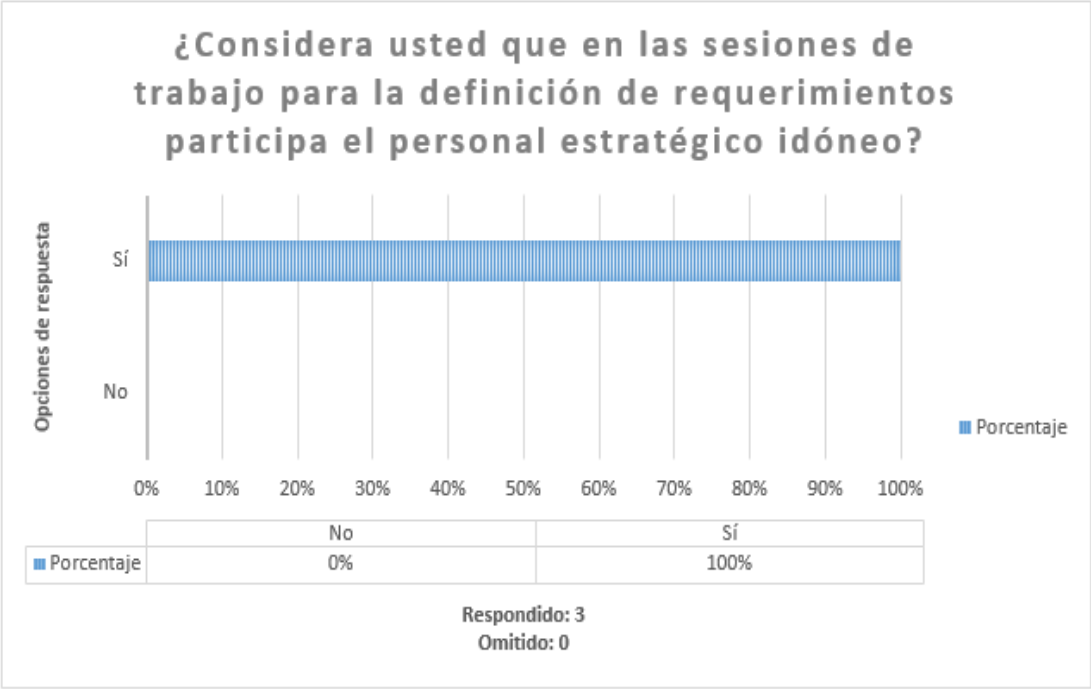


Figura 28. Participación de personal estratégico en sesiones de trabajo.

Fuente: Elaboración propia.

Según la figura 28, todos los encuestados coinciden en que sí participa el personal estratégico idóneo en las sesiones de trabajo para el levantamiento de requerimientos, por lo que la definición de requerimientos puede resultar más precisa.

Adicionalmente, se realizaron dos preguntas abiertas para conocer la apreciación con respecto a las deficiencias o problemas que presenta la metodología de desarrollo actual, así como las medidas que sugieren para mejorar los tiempos en las entregas del producto software. Se muestran los resultados:

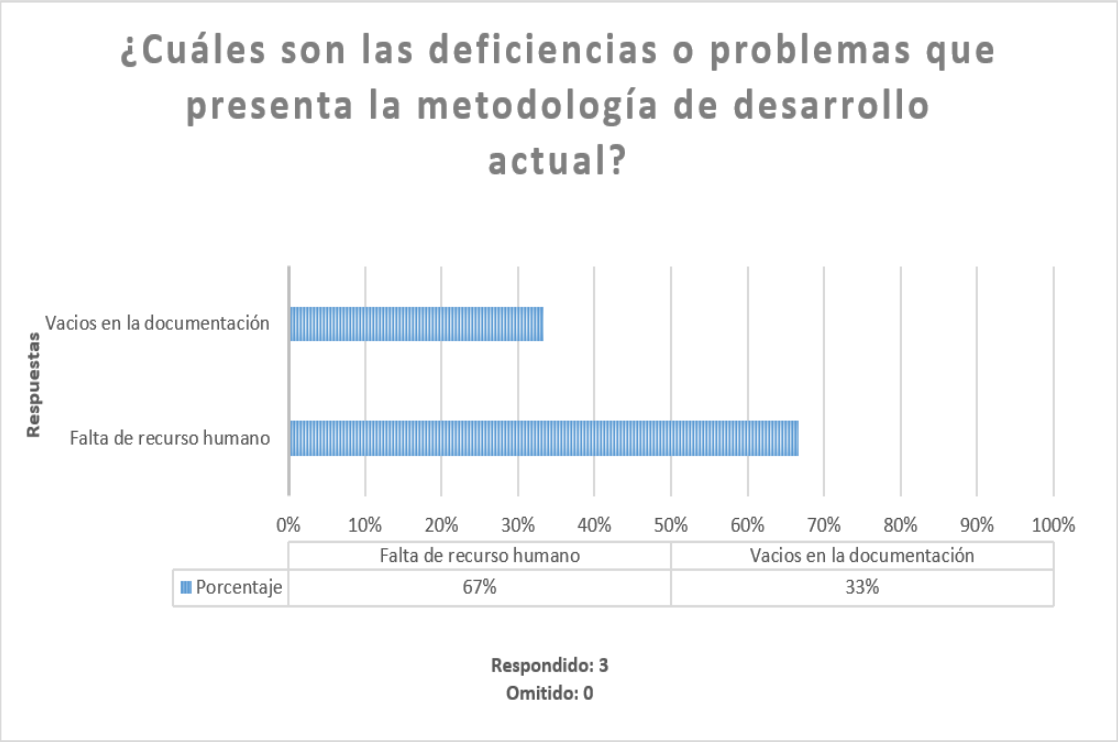


Figura 29. Deficiencias de la metodología de desarrollo de software actual.

Fuente: Elaboración propia.

En la figura 29, se aprecia que los encuestados coinciden en que una de las deficiencias en la metodología de desarrollo actual son vacíos en la documentación, por ejemplo, no se tiene claro qué documentos se debe completar acorde con particularidades del desarrollo del sistema (tamaño, complejidad, prioridad, entre otros), aunado a este aspecto, la entrega del producto *software* se realiza hasta que la documentación del sistema se encuentra completamente finalizada. Estos vacíos en la documentación provocan reprocesos.

Otro problema que incide en la aplicación eficiente de la metodología es la falta de recurso humano, lo cual ocasiona que las cargas de trabajo sean altas, cambios constantes de prioridades, reasignación de actividades, entre otros.

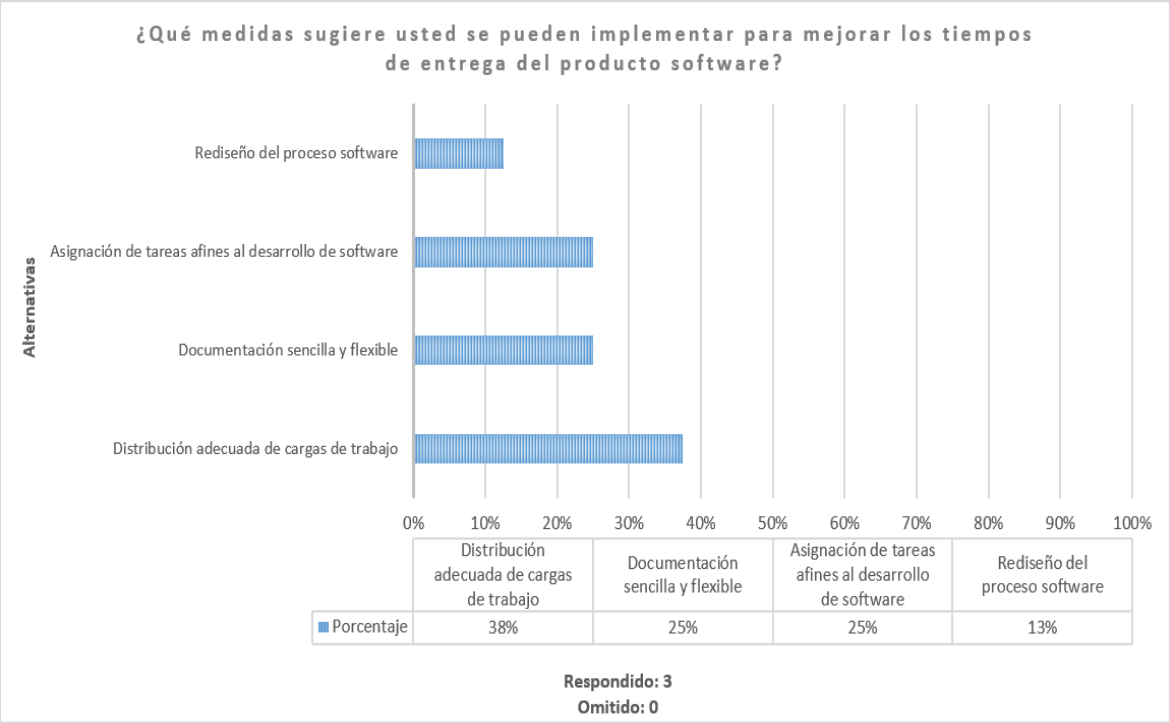


Figura 30. Alternativas para mejorar los tiempos de entrega del producto software.

Fuente: Elaboración propia.

Según la figura 30, los encuestados sugieren las siguientes alternativas para mejorar los tiempos de entrega del producto *software*:

- Rediseño del proceso software: sesiones de trabajo bisemanales para el levantamiento de requerimientos, considerando la complejidad de los sistemas que se desarrollan. Además, incentivar el uso de plataformas virtuales para estas

sesiones de levantamiento de requerimientos y de inducciones, disminuyendo de esta manera tiempos.

- Asignación de tareas afines al desarrollo de software: promover un ambiente de desarrollo de *software* típico, donde las tareas que se asignen sean afines al proceso, evitando otras actividades como: reuniones, administración de bases de datos, atención de incidencias relacionadas a sistemas, entre otras.
- Documentación sencilla y flexible: actualmente las plantillas para el levantamiento de requerimientos y control de cambios son complejas, lo cual provoca atrasos en el proceso de *software*.
- Distribución adecuada de las cargas de trabajo: asignar cargas de trabajo conforme se avance en los desarrollos, fomentando así un ambiente laboral confortable, potencializando las habilidades de los colaboradores; evitando lo que ellos describen como código espagueti, es decir, mezcla de ideas y código producto de salidas abruptas en los procesos o actividades que están ejecutando por atender cambios en prioridades o reasignaciones de tareas.

4.4.2 Resultado de la encuesta realizada a clientes de la División Infraestructura.

A continuación, los resultados de la encuesta realizada a los clientes que han solicitado desarrollo de *software*, y que, por las experiencias pueden brindar sus puntos de vista.

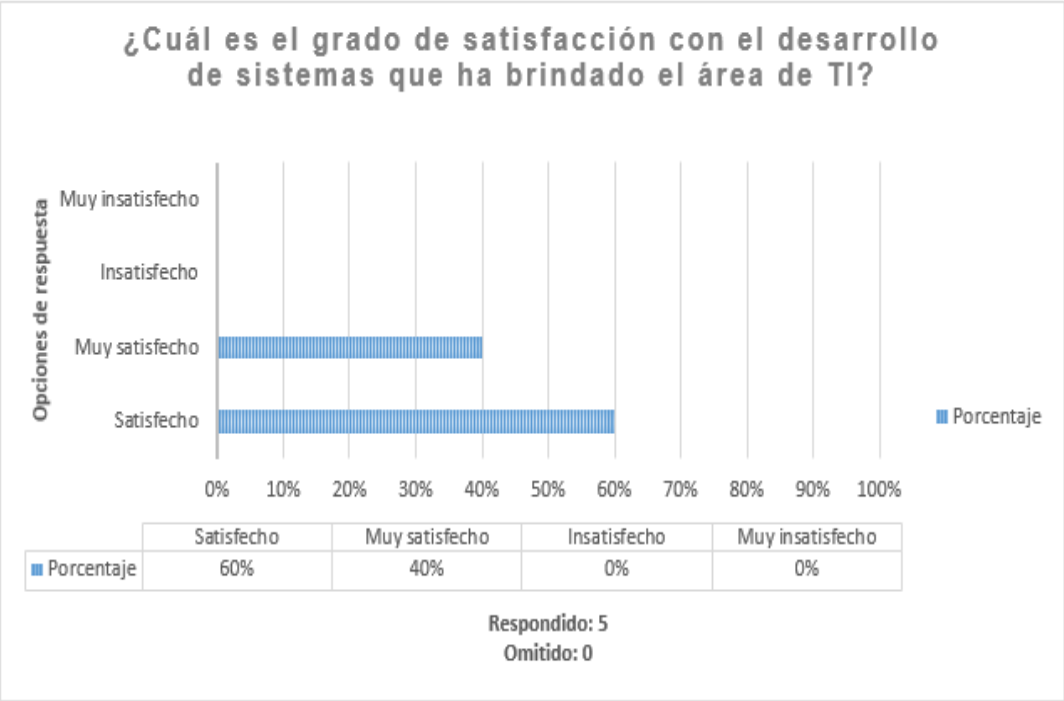


Figura 31. Satisfacción de clientes con el desarrollo de sistemas.

Fuente: Elaboración propia.

De acuerdo a la figura 31, se aprecia que existe un grado importante de satisfacción en el desarrollo de sistemas que ha brindado el área de Tecnologías de la información.

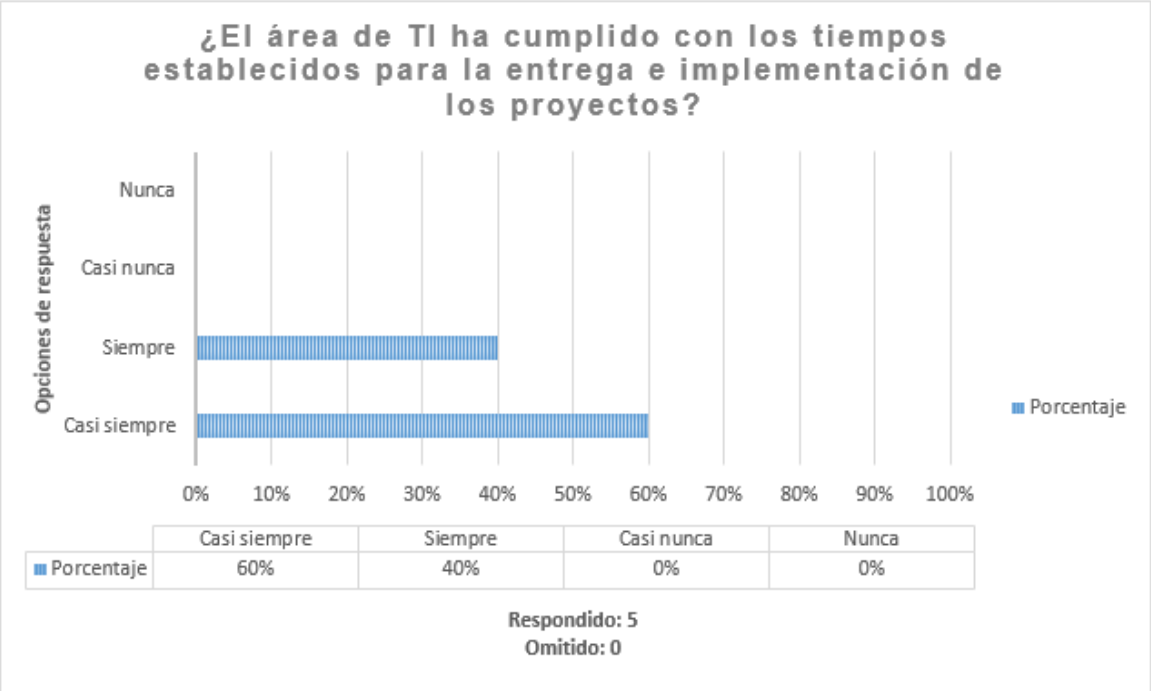


Figura 32. Cumplimiento de tiempos en la implementación de los proyectos.

Fuente: Elaboración propia.

De conformidad con la figura 32, la mayoría de los encuestados coinciden en que casi siempre se cumple con los tiempos establecidos para la entrega e implementación de los proyectos.

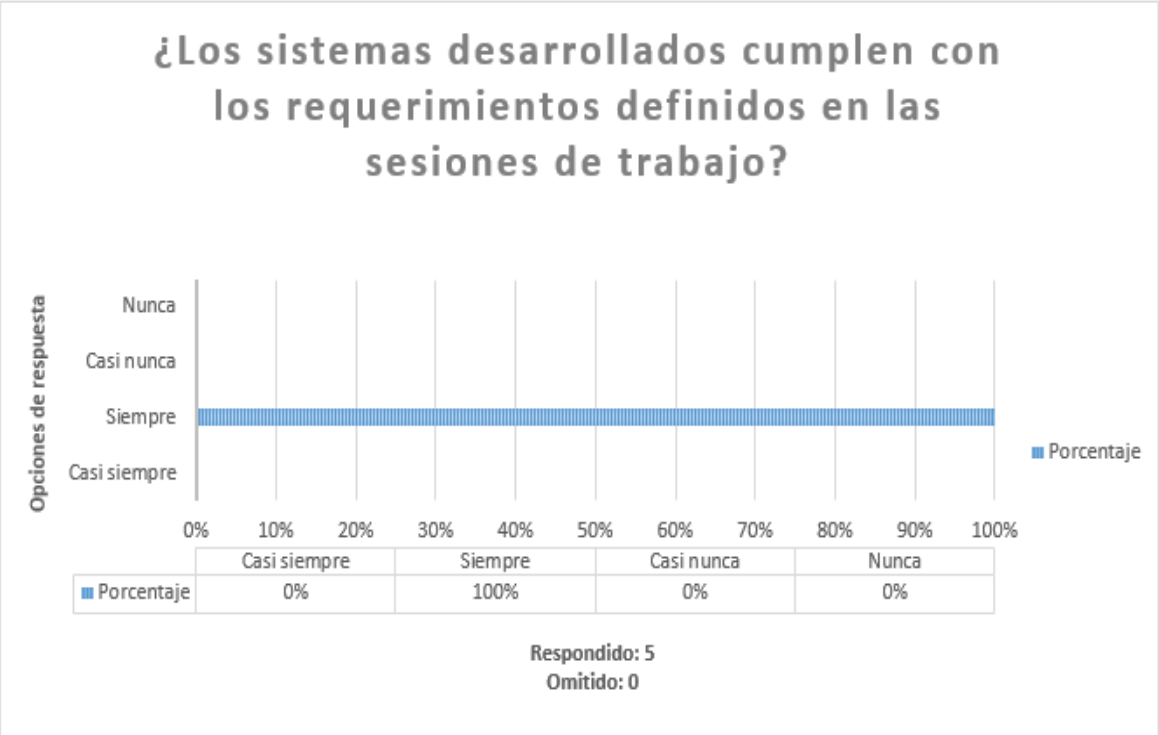


Figura 33. Apreciación de cumplimiento de requerimientos.

Fuente: Elaboración propia.

En la figura 33, se observa que todos los entrevistados opinan que los sistemas desarrollados siempre cumplen con los requerimientos definidos en las sesiones de trabajo, por lo que el *software* dispone de las funcionalidades solicitadas.

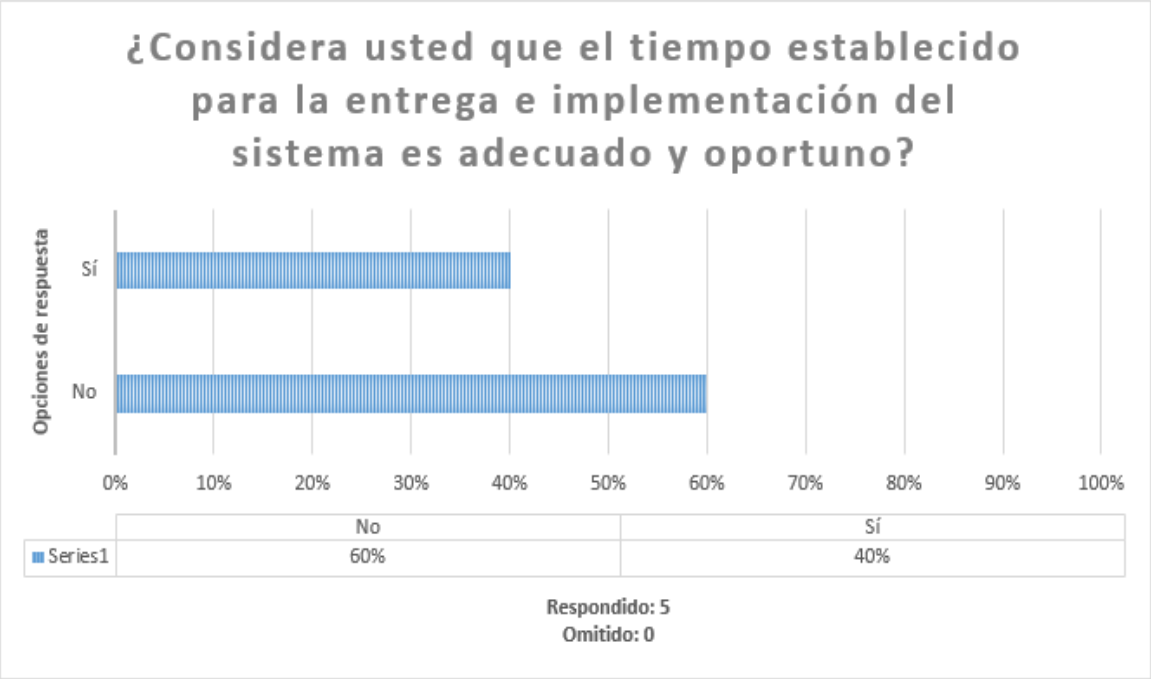


Figura 34. Tiempos en la implementación de proyectos.

Fuente: Elaboración propia.

De acuerdo a la figura 34, la mayoría de los entrevistados consideran que el tiempo establecido para la entrega e implementación del sistema no es adecuado ni oportuno.

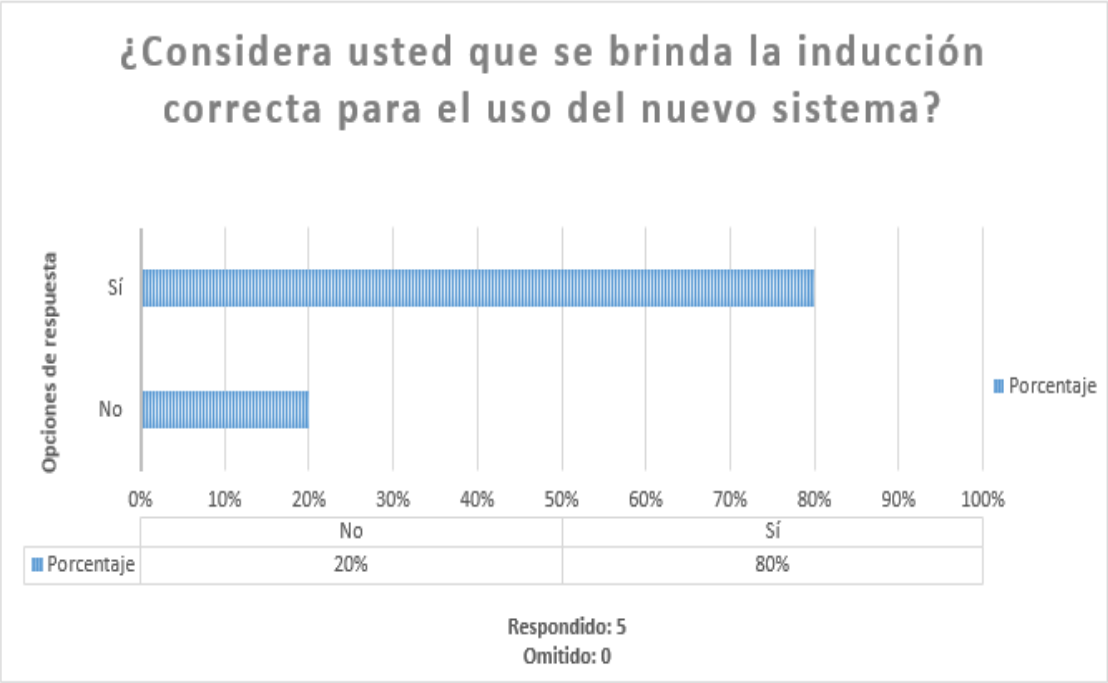


Figura 35. Apreciación inducción del nuevo sistema.

Fuente: Elaboración propia.

Según la figura 35, se observa que los entrevistados concuerdan en que sí se brinda la inducción correcta para el uso del nuevo sistema.

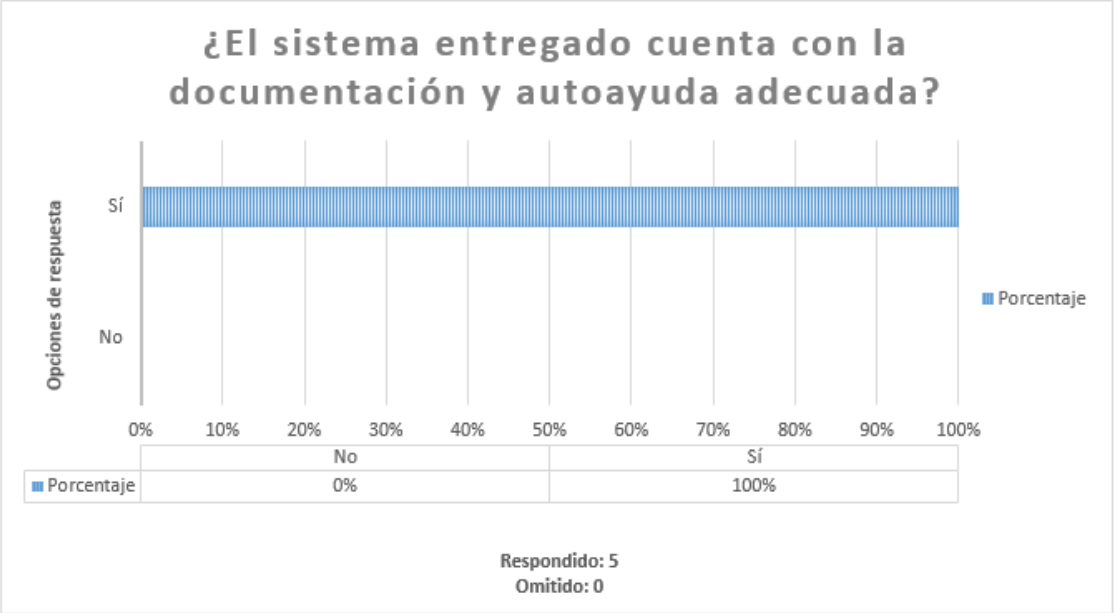


Figura 36. Documentación y autoayuda en los sistemas.

Fuente: Elaboración propia.

En la figura 36, todos los entrevistados opinan que el sistema entregado cuenta con la documentación y autoayuda adecuada.

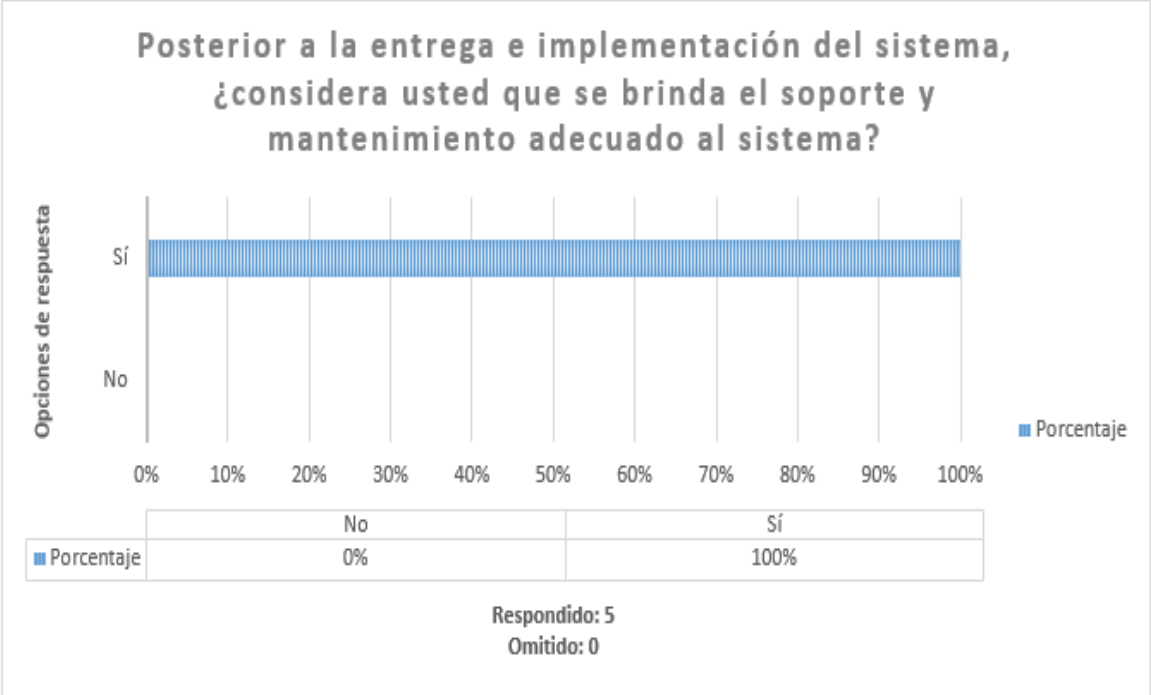


Figura 37. Soporte y mantenimiento a los sistemas.

Fuente: Elaboración propia.

De acuerdo a la figura 37, se aprecia que todos los entrevistados concuerdan en que se brinda el soporte y mantenimiento adecuado al sistema posterior a su implementación.

4.4.3 Resultado de la entrevista realizada a la coordinadora del área Tecnologías de la Información, de la División Infraestructura.

Se realizó una entrevista a la coordinadora de TI con el propósito de conocer la apreciación que tiene sobre la metodología de desarrollo de *software*, y ella considera que esta se ajusta a las necesidades y estrategia de la División Infraestructura y del Sector Telecomunicaciones, la cual ha sido utilizada por aproximadamente un año. Además, indica que basándose en la experiencia del producto *software* entregado a los clientes, no es necesario un cambio en la metodología de desarrollo de *software* actual.

Con respecto a la pregunta sobre las deficiencia o problemas que presenta la metodología de desarrollo actual, indicó que los problemas no se encuentran ligados a la metodología, sino que radica en la cantidad de requerimientos y el poco personal disponible para la atención de las solicitudes. Asimismo, recalcó la respuesta de la pregunta tres, referente a la existencia de cuellos de botella en la metodología de desarrollo; para lo que indica que no se presentan cuellos de botella en el proceso de *software*.

Por otra parte, señala que los constantes cambios en las prioridades de los desarrollos de *software* impactan en el cumplimiento de los tiempos establecidos para la entrega e implementación de los proyectos que se desarrollan, no obstante, casi siempre se cumplen según lo acordado.

También indica que utiliza una herramienta para la gestión de proyectos, la cual se adecua a sus necesidades y las del área. esta le sirve de apoyo para brindar seguimiento a los proyectos de desarrollo de *software*; este seguimiento lo realiza siempre, y le permite tomar decisiones que contribuyan a mejorar los tiempos de respuesta.

A pesar de no considerar necesario un cambio en la metodología de desarrollo de *software* actual, está anuente a identificar puntos de mejora, tomando en cuenta que el área de TI dispone de los recursos necesarios para mejorar el proceso de *software*. En relación con esto, un aspecto importante es que ha recibido capacitaciones en metodologías de desarrollo de *software*, aspecto que podría facilitar la implementación de las mejoras. Ver el registro de la minuta en el Apéndice 5.

4.5. BECHAS O CONCLUSIONES DEL DIAGNÓSTICO.

En conclusión, se puede mejorar la metodología de desarrollo de *software* actual, estructurándola de una manera diferente, con el propósito de gestionar los recursos disponibles, para así aumentar la eficiencia y productividad del proceso *software*.

Lo anterior, considerando la complejidad de los sistemas que se desarrollan en la División Infraestructura, apuntando a aliviar la presión ejercida por la competencia o el entorno del negocio.

Tabla 4. Brechas o conclusiones del diagnóstico.

Situación actual	**Brecha**	**Situación deseada**
El proceso de *software* no cuenta con una metodología de desarrollo de *software* asociada, la cual sea reconocida en la industria.	Se requiere conocer sobre las metodologías de desarrollo de *software* modernas para sacar provecho de sus características.	Conocimiento de las características de las metodologías de desarrollo de *software* modernas, las cuales sean aplicables en el área de TI, considerando sus recursos.
No existe un ambiente de desarrollo de *software* típico.	Se requiere establecer roles para los funcionarios del grupo de desarrollo de *software*.	Definición y asignación de roles a los funcionarios del grupo de desarrollo de *software*, para la designación de actividades afines al proceso de *software*.
No existen mecanismos de control para el manejo de nuevas solicitudes de desarrollo de *software*, o cambios de prioridades planificadas.	Se requiere crear un portafolio de proyectos de desarrollo de *software*, que permita el manejo adecuado de todas las solicitudes referentes a	Contar con un portafolio de proyectos de desarrollo de *software* que se encuentren en proceso y en espera.

	nuevos desarrollos de *software* o bien cambios en las prioridades planificadas.	
No se cuenta con métricas de desarrollo de *software* asociadas a estándares utilizados en la industria.	Se requiere crear estándares útiles para ser usados al momento de estimar tiempos.	Estándares en métricas para la estimación de tiempos.
No se cuenta con técnicas, herramientas o mejores prácticas de la industria para realizar el diseño del nuevo *software*.	Se requiere definir alguna técnica, herramienta o mejor práctica de la industria para realizar el diseño de *software*.	Contar con una técnica, herramienta o mejor práctica de la industria para realizar el diseño del *software* por desarrollar.
No se cuenta con técnicas, herramientas o mejores prácticas para documentar las fases de diseño y desarrollo de *software*.	Se requiere establecer alguna técnica, herramienta o mejor práctica de la industria para documentar las fases de diseño y desarrollo de *software*, la cual sea sencilla de usar.	Contar con una técnica, herramienta o mejor práctica de la industria para realizar la documentación de las fases de diseño y desarrollo, las cuales sean aplicables al área de TI, considerando los recursos; y que sean fáciles de usar.

No se tiene claridad de la documentación que se debe completar, según las particularidades del proyecto.	Se requiere crear un estándar para ser usado en la fase de análisis y levantamiento de requerimientos.	Estándar para documentar la fase de análisis y levantamiento de requerimientos.
La documentación del proceso *software* es compleja.	Se requiere modificar las plantillas actuales (levantamiento de requerimientos, perfiles de proyecto).	Contar con plantillas (documentación del proceso *software*) más sencillas y prácticas.
Entrega del producto *software* hasta finalizar la totalidad del desarrollo y contar con su respectiva documentación.	Se requiere realizar entregables de las principales funcionalidades del sistemas, y posteriormente desarrollar los requerimientos menos prioritarios.	Realizar entregables de las principales funcionalidades del sistema y posteriormente realizar el desarrollo de sus extensiones, las cuales no sean tan prioritarias; con el propósito de: a) Aliviar la presión ejercida por la competencia, o el entorno del negocio.

		b) Satisfacer a los clientes brindándoles la solución informática en tiempos adecuados y oportunos.

Fuente: Elaboración propia.

CAPITULO V: PROPUESTA DEL PROYECTO

El siguiente capítulo contiene la propuesta de mejora en la metodología de desarrollo de *software* que utiliza actualmente el área de Tecnologías de la Información de la División Infraestructura, Sector de Telecomunicaciones del ICE; brindando beneficios a la organización para aliviar la presión ejercida por el entorno del negocio, o por la competencia. Asimismo, para ofrecer a los clientes internos soluciones informáticas que faciliten sus actividades diarias y convertir los datos en información valiosa para la toma de decisiones.

El proyecto consiste en el rediseño del proceso *software*, contemplando técnicas, herramientas y mejores prácticas de la industria del *software* para conformar las mejoras en la metodología de desarrollo de *software*.

La propuesta se fundamenta en la metodología para el rediseño de procesos propuesta por Madison (2005). La explicación de sus fases y sus respectivos pasos se realizó en la sección 3.5 del presente documento, y se llevan a cabo para el desarrollo de los objetivos específicos.

La siguiente tabla muestra la relación de los objetivos específicos con las fases de la metodología Madison (2005), esto para facilitar al lector la comprensión referente al cumplimiento de los objetivos planteados en el proyecto.

Tabla 5. Relación objetivos específicos con fase y pasos Metodología Madison.

Objetivo específico	**Fase y paso donde se cumple el objetivo**	**Resultado**
Diagnosticar la metodología de desarrollo de *software* utilizada actualmente, mediante la aplicación de instrumentos de recolección de información, para evaluar la eficiencia en el proceso de desarrollo de *software*.	Fase 2, paso 3 y 4.	Brechas o conclusiones del diagnóstico descritas en la sección 4.5 del presente documento.
Determinar puntos de mejora en el proceso de desarrollo de *software* mediante la identificación de las mejores prácticas usadas, que procedan de diversas fuentes como estándares y metodologías de desarrollo de *software*, para que facilite a los desarrolladores la creación y mantenimiento de *software*.	Fase 2, paso 5.	El análisis y resultado de los diagnósticos permitió la identificación de puntos de mejora que contribuyan al proceso *software* la elaboración de sistemas. Esto mediante metodologías de desarrollo de *software* y mejores prácticas utilizadas en la industria del *software*, expuestas en la sección

		2.3.5 y 2.3.6 del presente documento.
Desarrollar las mejoras en la metodología de desarrollo de *software*, basado en el análisis de la información recolectada para solventar los problemas existentes en el proceso.	Fase 3, paso 6.	Diseño propuesto en la metodología de desarrollo de *software*, descrita en la sección 5.2.3.1 de este capítulo.
Realizar un ejercicio mediante la ejecución del diseño propuesto en la metodología de desarrollo de *software*, para evaluar si se adecua a las necesidades de la institución.	Fase 3, paso 6.	Conclusiones del ejercicio expuestas en la sección 5.2.3.1.3.2 de este capítulo.

Fuente: Elaboración propia.

5.1 REQUERIMIENTOS POR DESARROLLAR.

En esta sección se mencionan los requerimientos que serán desarrollados para estructurar el proceso de *software* actual, que conduzcan a conformar una metodología de desarrollo de *software* ajustada a los recursos del área de Tecnologías de la Información, y a las necesidades de la División Infraestructura; tomando en cuenta los lineamientos de la DCTI en esta materia.

- Determinar puntos de mejora en el proceso de *software* actual.
- Estructurar el proceso de *software* con el propósito de organizar un conjunto coherente de actividades para la elaboración de *software*.
- Aplicar marcos de trabajo relacionados al proceso de *software*, que contribuyan a formar un proceso de *software* típico.
- Crear un estándar para contar con documentación simple que facilite y agilice las actividades del proceso *software*.
- Realizar un ejercicio para validar la funcionalidad de la propuesta y comparar el tiempo de duración con respecto a otros proyectos previamente desarrollados.

5.2 METODOLOGÍA PARA EL REDISEÑO DEL PROCESO *SOFTWARE*.

La metodología para el rediseño de procesos propuesta por Madison (2005) guía el desarrollo de la propuesta de mejora en la metodología de desarrollo de *software* utilizada en la División Infraestructura. A continuación el desarrollo de cada fase.

5.2.1 Fase 1: Comienzo.

Esta fase consta de dos pasos y establece las bases para la ejecución del proyecto.

Paso 1 – Introducción al proceso de rediseño: el principal objetivo del proyecto consiste en identificar puntos de mejora en la metodología de desarrollo de *software* actual, se pretende lograr un proceso de *software* más flexible acorde al recurso disponible en el área de TI, alineado a las mejores prácticas, técnicas y herramientas de la industria del *software*, así como a las políticas establecidas por la DCTI.

Por otra parte, se intenta que el personal se aleje de estados negativos como la frustración o desagrado, por ejemplo, por la presión ejercida para cumplir metas desafiantes. Estar lejos de estos estados aumentan la motivación y consecuentemente se reflejará en un mayor rendimiento y productividad en sus actividades.

El proyecto se enfatiza en mejorar el proceso de *software*, de acuerdo al diagnóstico de la situación actual y el diagnóstico operativo, correspondiente a la sección 4.1 y 4.2 respectivamente. Asimismo, considerando la información obtenida del diagnóstico de percepción, ubicado en la sección 4.4, ambas secciones del presente documento.

Adicionalmente, se contempla las secciones 1.1 definición del problema, 1.3 objetivos específico y generales, y por último el 1.5 alcances y limitaciones.

Paso 2 – Formación del equipo de proceso: Para que el equipo trabaje en conjunto de manera efectiva se recomiendan las siguientes reglas:

1) Las decisiones serán tomadas en consenso. Si se produce un desacuerdo debe ser debatido hasta llegar al consenso.

   El consenso significa que todos aceptarán, pero no que a todos les debe gustar.
2) Como equipo, se invertirá el tiempo para atender los problemas de proceso. En caso de existir problema entre personas estos deben ser tratados fuera del grupo con la coordinadora de TI.
3) Tratar a todos con respecto.
4) Todos en el equipo son iguales, es decir, no hay distinción de jerarquía.
5) Todas las preguntas son válidas y no existen las preguntas tontas.
6) Actitud positiva ante el proceso de mejora, en caso contrario, excluir del grupo para evitar ruidos en el proceso.

El equipo de trabajo para el proceso de rediseño está conformado por el grupo de desarrollo actual:

a) Coordinadora del área Tecnologías de la Información.
b) Los desarrolladores.
c) Analista.

d) Ejecutor del proyecto.

5.2.2 Fase 2: Análisis de proceso.

En esta fase se desarrolla el primer objetivo específico del proyecto. Consiste en diagnosticar y posteriormente analizar los resultados que conducen a la identificación de los puntos de mejora del proceso. Contempla tres pasos, los cuales se desarrollan a continuación.

Paso 3 – Creando el diagrama de proceso *AS-IS*: el diagrama muestra el proceso tal como está, este representa en forma gráfica lo que está sucediendo realmente y no lo que se pretende alcanzar tras el proceso de rediseño. Según Madison (2005) es importante porque:

- → Permite observar dónde se realiza la mayor parte del trabajo.
- → Puede determinar quién realiza los pasos de valor agregado y quién no.
- → Puede ver dónde ocurren los problemas en las transferencias.
- → Puede detectar revisiones innecesarias.
- → Puede determinar si tiene más sentido que alguien más haga el trabajo.

El siguiente diagrama se basa en el diagnóstico de la situación actual y el administrativo u operativo, sección 4.1 y 4.2 respectivamente.

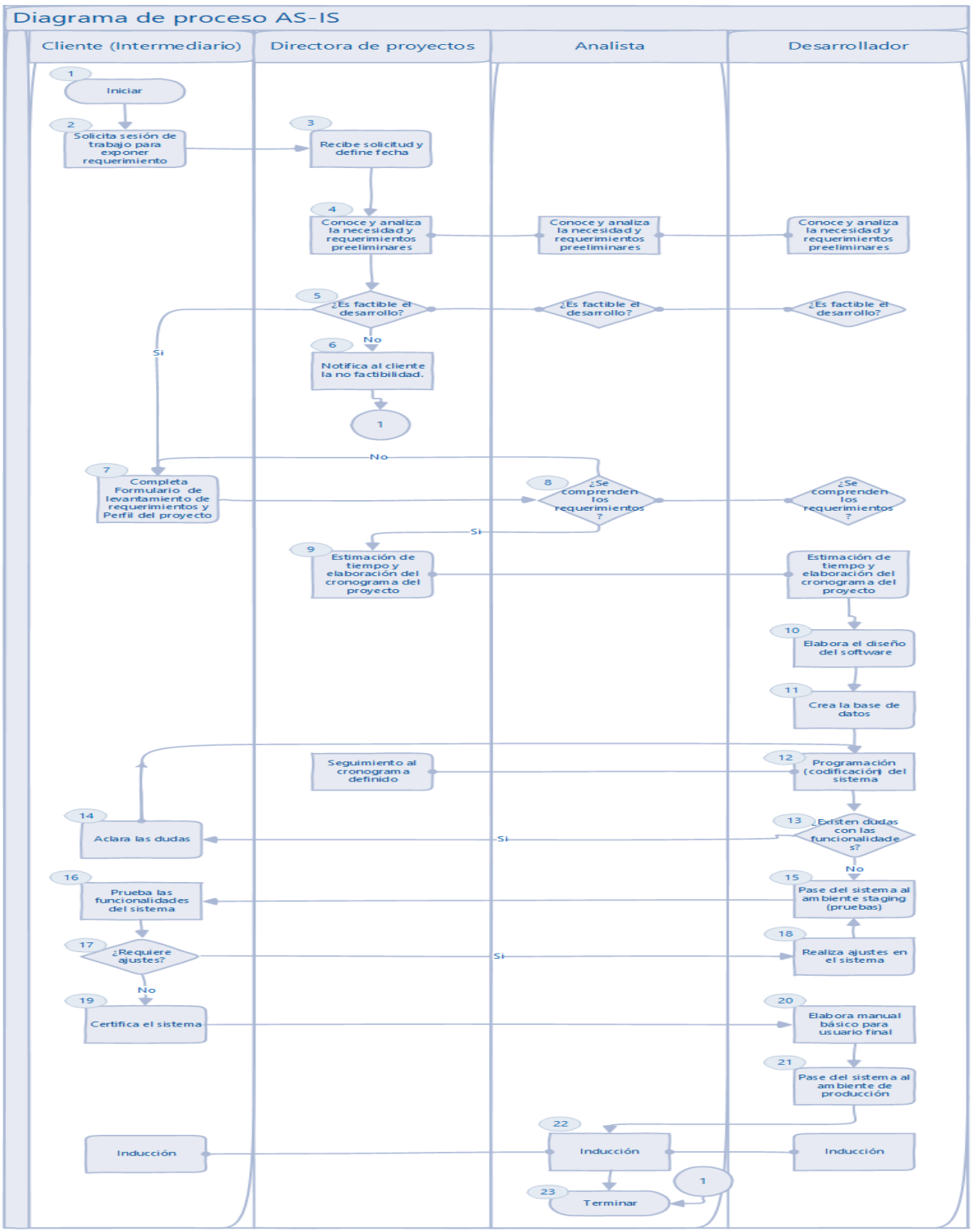


Figura 38. Diagrama de proceso actividad/función, proceso de *software*.

Fuente: Elaboración propia.

En términos muy generales, se observa en la figura 38 que la mayor parte del trabajo en el proceso *software* lo ejecuta el equipo de desarrollo, son la pieza clave, son quienes construyen el código fuente y, por lo tanto, convierte los requerimientos en un producto concreto. Precisamente es ahí donde existe un mayor grado de frustración e incomodidad, según el resultado del diagnóstico de percepción de la sección 4.4.1 de este documento. Por otra parte, se identifican oportunidades de mejora, las cuales aportan una distribución más equitativa en las cargas de trabajo.

Paso 4 – Entrevista al cliente: para cumplir con este paso y con el propósito de conocer la percepción del cliente en relación al proceso de *software* se llevó a cabo una encuesta, los resultados se encuentran en la sección 4.4.2 del presente documento.

Adicionalmente, se consultó a los clientes lo que necesitan, quieren, desean y requieren del proceso. Con base en la información obtenida se realizó la siguiente tabla, la cual resume la información recopilada. Estos criterios se considerarán para el rediseño del proceso, y se enfocarán esfuerzos de mejora para satisfacer a los clientes internos.

Tabla 6. Resumen calificaciones del cliente.

Criterio	Clasificación	Cómo alcanzar un 3
Tiempos de entrega e implementación del sistema.	2	Tiempos adecuados y oportunos de entrega e implementación del sistema.
Retroalimentación a nivel ejecutivo	2	Brindar información periódica, preferiblemente bisemanal o al menos una vez al mes, en forma resumida.
Comunicación	1	Disponer de algún medio para canalizar consultas, para evitar la dependencia a una única persona que dé respuestas.

Rango de clasificación: 1: Malo, 2: Regular, 3: Bueno

Fuente: Elaboración propia.

Además, se realizó un ejercicio con la coordinadora del área Tecnologías de la Información, el cual consistió en que adivinara los criterios de los clientes y su respectiva clasificación, el resultado fue que de los tres criterios concuerda con uno: Tiempo de entrega e implementación del sistema, según tabla 6. Resumen clasificaciones del cliente. Este resultado es importante, ya que significa que se conocen las necesidades del cliente, sin embargo, acota que para alcanzar un 3 requiere contar con más recursos.

Paso 5 – *Benchmarking* y mejores prácticas: Según Espinoza (2017), el *benchmarking* es un proceso continuo que consiste en tomar como referencia los productos, servicios o procesos de trabajo de empresas líderes, para compararlos con los de nuestra propia empresa, y posteriormente realizar y aplicar mejoras.

Por otra parte, se entiende por mejores prácticas como un conjunto coherente de acciones que han incrementado el rendimiento en un determinado contexto y que se espera que en contextos similares, rindan similares resultados (Jiménez, 2014).

Para efectos del proyecto, se realizó únicamente *benchmarking* de tipo interno, es decir, se lleva a cabo dentro de la misma empresa. Se identificó el área Tecnologías de la información, Negocio Ingeniería y Construcción, del Sector Electricidad, para obtener puntos de referencia del proceso de *software.* A continuación, el resultado de los principales temas tratados:

La coordinadora del área considera que la metodología de desarrollo de *software* actual se ajusta a las necesidades del área, la cual ha sido utilizada por 5 años, por lo que el proceso *software* se encuentra en un estado de madurez importante.

Como metodología de desarrollo de *software* utilizan Scrum, adicionalmente emplean Gestión de Proyectos, es decir, es un híbrido que les permite ajustar técnicas, herramientas y mejores prácticas de la industria con las necesidades del área.

Menciona que algunos de los beneficios de la metodología de desarrollo de *software* son los siguientes:

- ✓ Mejores tiempos de respuesta.
- ✓ Control y seguimiento real de los entregables.
- ✓ Involucramiento con el cliente en toda la gestión del proyecto.
- ✓ El cliente rápidamente puede contar con funcionalidades.

En relación con las mejores prácticas de la industria del *software*, fueron cubiertas en el capítulo II, sección 2.3.5 y 2.3.6 del presente documento; además la aplicación de mejores prácticas referente a los principios para rediseño de proceso desarrollados en la siguiente fase (paso 6).

5.2.3 Fase 3: Rediseño del proceso.

Mediante el paso 6 se diseña la propuesta de las mejoras en la metodología de desarrollo de *software*.

Paso 6 – Rediseño desde cero: El propósito de este paso es crear un nuevo proceso que sea significativamente mejor que el anterior. Asimismo, una meta y desafío es generar satisfacción en el cliente.

Un aspecto importante es que la estructura organizacional y los mecanismos de control deberían adaptarse a este proceso de rediseño. Ambos deberían apoyar y habilitar cualquier nuevo diseño de proceso. El escenario ideal es que el proceso de diseño conduzca a la estructura y los controles para que se maximice el éxito, caso contrario aumenta el riesgo de fracaso.

En síntesis, los dos objetivos de un nuevo diseño en limpio son:

- → Satisfacer a los clientes inmersos en el proceso.
- → Reducir los pasos o el tiempo del proceso significativamente, apuntando a una reducción del cincuenta por ciento o más.

Adicionalmente, se toma en cuenta los principios de diseño para el rediseño de procesos expuestos en la metodología Madison (2005, pág. 151), los cuales son considerados mejores prácticas obtenidas de organizaciones de clase mundial. Estos treinta y ocho principios de diseño representan los principales conceptos involucrados en la mayoría de los proyectos de rediseño de procesos. A modo de resumen se agrupan de acuerdo a su uso:

- Principios 1-16: uso para la estructura del trabajo.
- Principios 17-19: uso para el flujo de la información.
- Principios 20-30: uso para guías de diseño.
- Principios 31-37: uso para organizar personal.
- Principio 38: uso para la orientación general.

Lo primero para dar inicio a este paso fue realizar una sesión de trabajo con el equipo de trabajo conformado, el propósito era conocer la historia del proceso ideal de cada miembro. Ver registro de la sesión de trabajo en Apéndice 6. Se consideraron las actividades del proceso por encima de quién las realizará o las limitantes actuales, con la finalidad de animar la imaginación y despreocuparse de la realidad actual.

Se aplicó la técnica *Brainwriting* (escritura de ideas), y se logró obtener ideas valiosas las cuales se reflejarán en el proceso de rediseño.

5.2.3.1. Metodología de desarrollo propuesta.

Hoy día las empresas operan en un entorno global que cambia rápidamente. En ese sentido, deben responder frente a nuevas oportunidades y mercados, al cambio en las condiciones económicas, así como al surgimiento de productos y servicios competitivos. El *software* es parte de casi todas las operaciones industriales, de modo que el nuevo *software* se desarrolla rápidamente para aprovechar las actuales oportunidades, con la

finalidad de responder ante la amenaza competitiva. En consecuencia, en la actualidad la entrega y el desarrollo rápidos son por lo general el requerimiento fundamental de los sistemas. De hecho, muchas empresas están dispuestas a negociar la calidad del sistema y el compromiso con los requerimientos, para lograr con mayor celeridad la implementación que necesitan del sistema informático.

Debido al dinamismo del entorno, a menudo es prácticamente imposible derivar un conjunto completo de requerimientos de *software* estable. Esos requerimientos iniciales cambian de modo inevitable, ya que a los clientes les resulta casi imposible predecir cómo un sistema afectará sus prácticas operacionales, cómo interactuará con otros sistemas y sobre todo cuáles operaciones de usuario se automatizarán. Es muy posible que sea solo hasta después de entregar un sistema, y que los usuarios adquieran experiencia con este, cuando se aclaren los requerimientos reales. Incluso, es probable que debido a factores externos, los requerimientos cambien rápida e impredeciblemente. En tal caso, el sistema podría ser obsoleto al momento de entregarse (S. Pressman, 2010, pág. 56).

En algunos tipos de sistemas, como los sistemas de control críticos para la seguridad, donde es esencial un análisis completo del sistema, resulta oportuno un enfoque basado en un plan. Sin embargo, en un ambiente empresarial de rápido movimiento, esto llega a causar verdaderos problemas. Al momento en que el sistema esté disponible para su uso, la razón original para su adquisición quizás haya variado

tan radicalmente que el sistema sería inútil a todas luces. Por lo tanto, para sistemas empresariales, son esenciales en particular los procesos de diseño que se enfocan en el desarrollo y la entrega de *software* rápidos (Sommerville, 2011, pág. 57).

Por lo anterior, estructurar el proceso de *software* con la finalidad de organizar en forma coherente las actividades para el producto *software*, es de vital importancia para el área de Tecnologías de la Información de la División Infraestructura; permitiéndole contar con un proceso que se adecue al entorno y dinamismo de la División, así como los recursos del área.

Promover el desarrollo incremental donde, por lo general, se crean nuevas liberaciones del sistema, y cada cierto periodo de tiempo se ponen a disposición de los clientes, será la propuesta del proyecto. Además, involucrar a los clientes en el proceso de *software* para conseguir una rápida retroalimentación sobre los requerimientos cambiantes; y minimizar la cantidad de documentación con el uso de sesiones informarles, en vez de reuniones formales con documentos escritos también será parte de la propuesta.

Asimismo, la propuesta pretende fomentar un ambiente de desarrollo de *software* típico, en el cual se distribuyan en forma adecuada las cargas de trabajo, y se asignen tareas afines al proceso de desarrollo de *software*. Contar con un ambiente de trabajo

cómodo con lleva a un estado sano físico y mental para el equipo de trabajo; con lo que se pretende incrementar la producción.

5.2.3.1.1 Diagrama de proceso.

A través de la siguiente representación, se muestra al lector una visión del proceso de *software*, la finalidad es facilitar la comprensión del proceso propuesto.

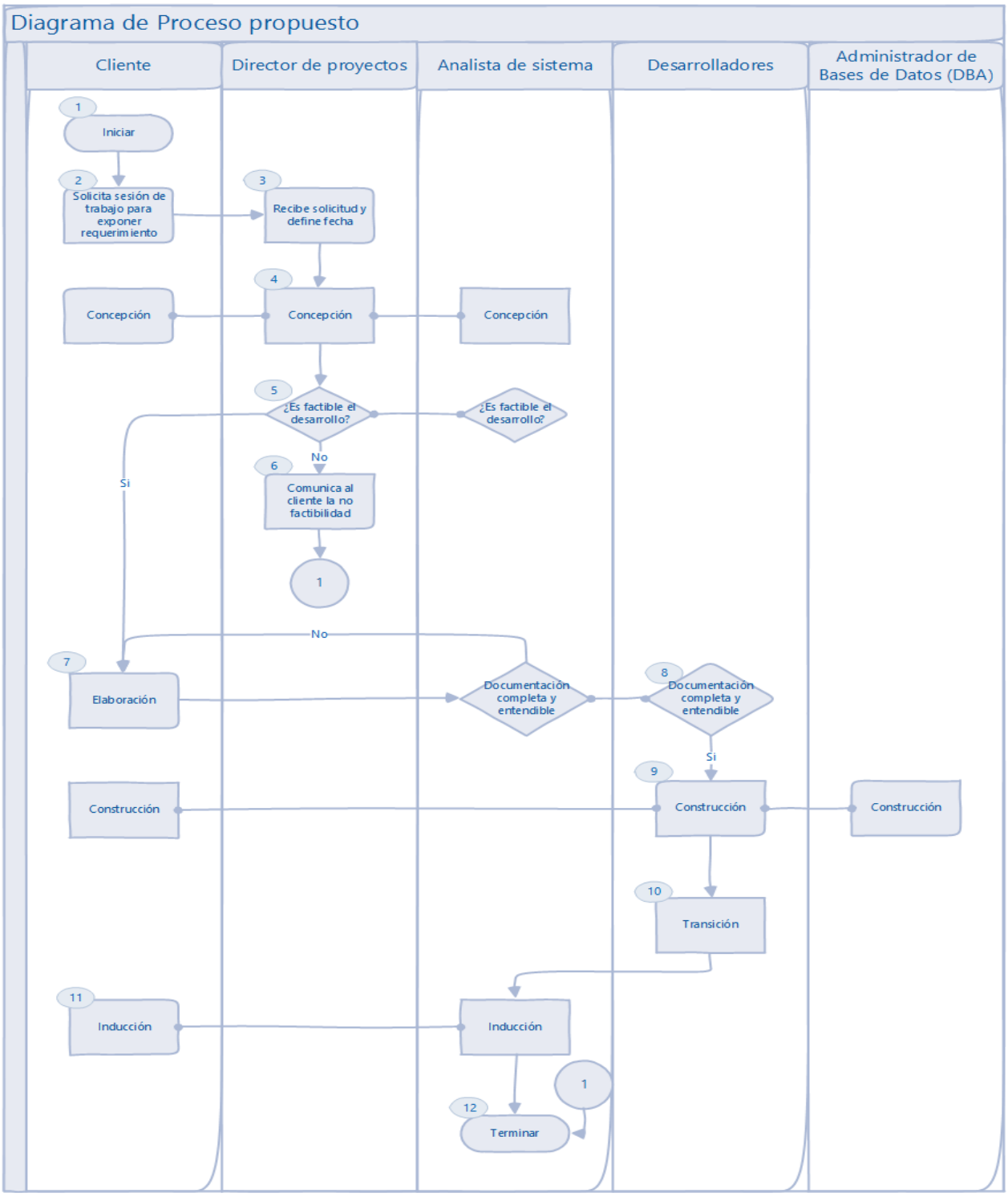


Figura 39. Propuesta de diagrama de proceso.

Fuente: Elaboración propia.

En relación con la figura 39, se detalla el flujo del proceso.

Paso Uno. Inicia el proceso.

Paso Dos. Cuando algún proceso/negocio requiere el desarrollo de un sistema informático designa a una persona experta en el tema a resolver, denominado cliente. El cliente solicita al director de proyectos una sesión de trabajo para exponer el requerimiento.

Paso Tres. Director de proyectos recibe la solicitud y define una fecha para atender al cliente.

Paso Cuatro. Da inicio la primera fase del modelo: Concepción.

Paso Cinco. Consiste en el análisis del requerimiento. ¿Cuál es la razón de ser del sistema?

Paso Seis. En caso de no ser factible el desarrollo del sistema el director de proyecto comunica al cliente la no factibilidad, o bien, informa si ya existe alguna herramienta informática que solvente la necesidad. Con esto finaliza el flujo.

Paso Siete. En caso que sea factible el desarrollo del sistema, el director de proyectos y analista de sistema brindan el Formulario Tarjeta de Historias de usuario (apéndice 7) y Perfil del Proyecto (anexo 1) al cliente, para que sea completado. También se brinda una breve inducción de cómo llenar la documentación requerida. Este paso da inicio a la fase dos del modelo: Elaboración. En esta fase se inicia la aplicación de las reglas de la Programación Extrema, que consiste en la planeación.

Paso Ocho. Una vez completada la documentación, el cliente la remite al analista de sistema y desarrollador asignado para su verificación. Si existen dudas al respecto son aclaradas por el cliente, de lo contrario se estaría finalizando la fase dos del modelo: Elaboración.

Paso Nueve. La fase tres del modelo: Construcción. Dará inicio cuando el director de proyecto lo decida, considerando las actividades actualmente asignadas al desarrollador y la priorización del nuevo sistema. Esta fase incluye el diseño, desarrollo y pruebas del sistema. En este punto se aplican las reglas de la Programación Extrema: diseño, desarrollo y pruebas.

Paso Diez. Al culminar la fase de construcción, se ejecuta la cuarta y última fase del modelo: Transición. La cual consiste en liberar la nueva versión del sistema para que el cliente decida si la implementa en ambiente de producción; es decir, ponerlo a funcionar en un ambiente real. El sistema debe funcionar correctamente en su entorno operacional.

Paso Once. Consiste en una inducción realizada por el cliente a los usuarios finales del sistema, considerando que el cliente fue parte del proceso y realizó las pruebas de aceptación, tiene amplio conocimiento del sistema informático y podrá realizar la transferencia de conocimiento.

Paso Doce. Finaliza el proceso.

A lo largo del proceso de *software*, se aplica la regla de la Programación Extrema denominada: gestión. Esta tiene relación con las actividades afines a la administración del proceso.

Es importante señalar y para un mejor entendimiento del lector, es en el paso nueve y diez donde se ponen en juego los eventos seleccionados de Scrum. Estos pasos se repiten hasta finalizar por completo todas las funcionalidades requeridas del sistema. Es ahí donde se aprecia el enfoque de desarrollo incremental e iterativo del proceso.

Como se menciona en el párrafo anterior, el marco de trabajo propuesto posee un enfoque de desarrollo incremental e iterativo; mediante las iteraciones, o bien periodos de tiempo en donde se desarrollan ciertas funcionalidades del sistema, se producen los incrementos lo que permite que el sistema evolucione en forma continua, y cada vez más se liberen versiones mejoradas del mismo. Los eventos de Scrum fueron ajustados para regular los periodos de tiempo que son parte del proceso de *software* descrito.

5.2.3.1.2 Marco de trabajo.

A partir de los marcos de trabajo definidos en la sección 2.3.5 y 2.3.6, bajo las mejores prácticas para el desarrollo de *software* la propuesta incluye dentro de la definición del proceso *software* elementos como fases, reglas, prácticas, eventos y artefactos.

Para iniciar el rediseño del proceso, se toma como referencia el diagrama de la sección 5.2.2, paso 3, figura 38. Asimismo, y como se mencionó anteriormente para la

elaboración de esta propuesta se consideraron los principios para el rediseño de proceso expuestos por Madison (2005, pág. 151), que en síntesis fueron:

- Evitar actividades secuenciales, en contraste realizar actividades en paralelo o simultáneas. La finalidad del principio es acortar el tiempo de duración entre las tareas.
- Enfocar el valor al cliente. Intentar cubrir sus deseos, necesidades y preferencias.
- Simplificación y unificación de pasos.
- Compartir la información relevante del proceso con todos los actores, esto evitará retrocesos en las actividades del proceso.
- Garantizar la calidad desde el inicio. Invertir tiempo al empezar para asegurar la calidad y así evitar corregir problemas a futuro, o bien, prevenir revisiones y volver a trabajar más a futuro.
- Retroalimentación como mecanismo para la mejora continua.

El Proceso Unificado Racional (RUP) compone la estructura base para el proceso de *software*, proporciona la columna vertebral del proceso mediante sus cuatro fases, a saber: la concepción, la elaboración, la construcción y la transición.

Además, los marcos de trabajo Scrum y Programación Extrema brindarán elementos que consolidarán la estructura de la propuesta de la metodología de desarrollo de

software, siendo los marcos de trabajo más ampliamente usados en la industria de *software*. La siguiente imagen representa las tres perspectivas de lo antes descrito.

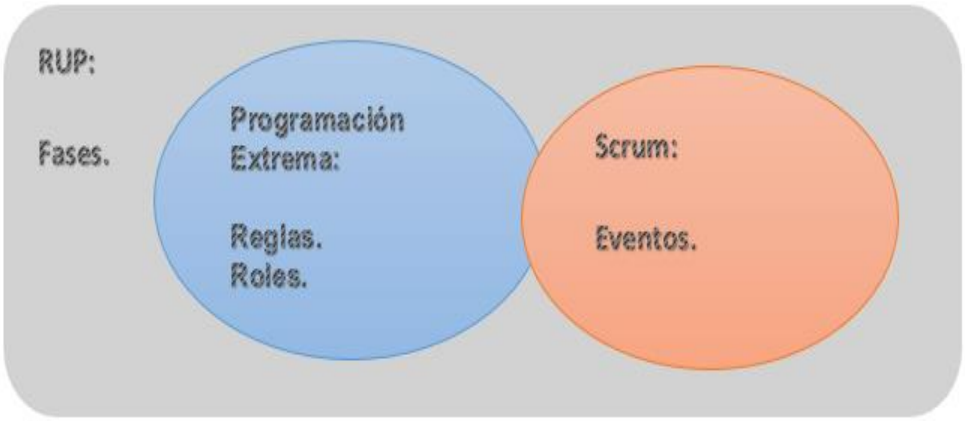


Figura 40. Marco de trabajo propuesto.

Fuente: Elaboración propia.

Para simplificar la representación del proceso, se realizarán las descripciones separadas de cada una de estas perspectivas.

5.2.3.1.2.1 Fases.

Las fases de RUP marcan el estado del proceso *software*. Se detalla la descripción de cada una.

a) Concepción: En esta fase, para cada solicitud de desarrollo de *software* se establece un caso empresarial para el sistema; en donde se identifican aquellas entidades externas (personas y sistemas) que interactuarán con el sistema y

definirán dichas interacciones. Con base en esta información se valora la aportación del sistema hacia el Proceso/Negocio, a la División Infraestructura y al Sector Telecomunicaciones. En caso que la aportación sea menor o poco significativa entonces el proyecto puede cancelarse después de esta fase.

b) Elaboración: la meta de esta fase es desarrollar la compresión del problema de dominio, establecer un marco conceptual para el sistema, diseñar el plan del proyecto (lo que se conoce como planeación en la Programación Extrema), identificar los riesgos clave del proyecto y los factores críticos de éxito.

   Al completarse esta fase, debe tenerse un modelo de los requerimientos para el sistema, el cual corresponde a las Tarjetas historias de usuario. Ver Apéndice 7. Este formulario contiene la descripción simple de los requerimientos, lo que se conoce como historias de usuario en el marco de trabajo de Programación Extrema. Adicionalmente, y en cumplimiento con los lineamientos de la DCTI, el cliente debe completar el documento Perfil de Proyecto. Ver Anexo 1.

c) Construcción: esta fase incluye el diseño, el desarrollo y las pruebas del sistema. Mediante el desarrollo incremental, se realizarán entregables de producto, considerando aquellas historias de usuario que el cliente prioriza y que a criterio del desarrollador pueden ser terminados durante la iteración (conocido en Scrum como *Sprint*).

Al completar esta fase, debe tenerse el entregable del producto probado y funcionando, así como la documentación relacionada y lista para entregarse al usuario.

d) Transición: en esta fase se llevará a cabo el pase del sistema desde el ambiente de desarrollo hacia el ambiente de producción, es decir, ponerlo a funcionar en un ambiente real. El sistema debe funcionar correctamente en su entorno operacional.

A través de RUP y su enfoque que se caracteriza por ser iterativo e incremental, se realizará con frecuencia entregables de ciertas funcionalidades del producto a los clientes. Estos pequeños entregables aportan valor tanto al cliente como al negocio.

5.2.3.1.2.2 Reglas.

Las reglas de la Programación Extrema se enfocan en las actividades que tienen lugar durante el proceso de *software*. Las reglas se entrelazan con las fases RUP, y los eventos de Scrum; definen un entorno que promueve la colaboración y el empoderamiento del equipo.

5.2.3.1.2.2.1 Planeación.

Consiste en la recopilación de las historias de usuario, las cuales serán evaluadas rápidamente por los desarrolladores para estimar el tiempo de desarrollo (construcción del código fuente) de cada una. De esta manera se elabora el cronograma general del proyecto.

Las historias de usuario corresponden a las descripciones de las funcionalidades y/o características del sistema, son escritas por el cliente en forma breve y en su propio lenguaje, es decir, sin tecno-sintaxis. Se recolectan en la Tarjeta de historia de usuario. Ver Apéndice 7. Cuando llegue el momento de implementar la historia, el cliente dará una descripción detallada de los requisitos a los desarrolladores. La tarjeta historias de usuarios, que representa los requerimientos, se descomponen en tareas de desarrollo (construcción de código fuente), y estas son la principal unidad de implementación.

Cuando surjan preguntas que no pueden responderse fácilmente y se requiere trabajo adicional para explorar posibles soluciones. El equipo puede elaborar algún prototipo o tratar de desarrollarlo para entender el problema y la solución. El propósito es esclarecer la especificación del requerimiento.

Mediante el plan de entregas y cronograma de lanzamientos se define los planes del *sprint*. Se establecen las historias de usuario que serán agrupadas para conformar una entrega y su respectiva prioridad.

Las historias de usuario seleccionadas se convierten en tareas de desarrollo (construir el código fuente) que se implementará durante el *sprint*. Los desarrolladores estiman cuánto tiempo llevarán las historias para implementar durante el *sprint*.

Al finalizar cada *sprint*, se tendrá ciertas funcionalidades del producto probadas, funcionando correctamente y listo para el ambiente de producción para ser entregado al cliente.

5.2.3.1.2.2.2 Gestión.

Esta regla tiene relación con las actividades afines a la administración del proceso *software*, promoviendo un ambiente de trabajo confortable, ritmo de trabajo sostenible o constante y cargas de trabajo equilibradas.

El equipo de trabajo actualmente posee un espacio de trabajo dedicado y abierto, permitiendo la comunicación y privacidad para lograr la concentración en sus tareas.

También cuenta con medios para realizar esquemas de diseño o anotaciones, lo que agrega más canales para la comunicación.

Para lograr un ritmo sostenible el plan de entregas y cronograma de lanzamientos (conocido en Scrum como planeación del *sprint*) será realizada por el desarrollador, y se basa en la teoría de control de procesos empírica o empirismo. El empirismo asegura que el conocimiento procede de la experiencia y de tomar decisiones basándose en lo que se conoce. De esta forma el desarrollador será quien decida las cargas de trabajo durante el *sprint*. Esta teoría es aplicada en el marco de trabajo Scrum.

5.2.3.1.2.2.3 Diseño.

La filosofía es la simplicidad. El diseño es la guía para el desarrollo (construcción de código fuente) del sistema; debe ser claro, conciso y simple. El objetivo del diseño es crear representaciones que describan lo que necesita el cliente. El modelado cruza la brecha entre la representación del sistema, que describe el sistema en su conjunto, y la funcionalidad del negocio.

Se utilizarán las Tarjetas Clase-Responsabilidad-Colaborador, ya que proporcionan una manera sencilla de identificación y organización de las clases que son relevantes para los requerimientos del sistema o del producto. El objetivo es desarrollar una

representación organizada de las clases. Las responsabilidades son los atributos y operaciones relevantes para las clases. En pocas palabras, una responsabilidad es cualquier cosa que la clase sepa o haga. Los colaboradores son aquellas clases que se requieren para dar a una clase la información necesaria a fin de completar la responsabilidad. En general, una colaboración implica una solicitud de información o de cierta acción (S. Pressman, 2010, pág. 149). Ver Apéndice 8.

Refactorizar en todo momento será una buena práctica para mantener el diseño simple, evita la complejidad y el desorden innecesario. Evitar redundancias. Esto contribuye a disminuir el tiempo en la construcción del código fuente y producirá un sistema ordenado y simple.

5.2.3.1.2.2.4 Desarrollo.

Una vez que las historias de usuario han sido elaboradas y de que se ha hecho el trabajo de diseño preliminar, el equipo no inicia las tareas de desarrollo (construcción de código fuente), si no que desarrolla una serie de casos de prueba a cada una de las historias que se van a llevar a cabo durante el *sprint*. Esto significa que la prueba puede realizarse conforme se escribe el código fuente y descubrir problemas durante el desarrollo.

Por otra parte, cada historia de usuario se descompone en tareas de desarrollo, cada una de estas tareas genera una o más pruebas de unidad, que verifican la implementación descrita en dicha tarea.

Realizar pruebas implícitamente define tanto una interfaz como una especificación del comportamiento para la funcionalidad a desarrollar. Con esto se reducen los problemas de la mala interpretación de los requerimientos y la interfaz. Crear los casos de prueba ayuda al desarrollador a considerar lo que realmente necesita realizar. A su vez, facilita determinar cuándo se ha terminado de construir la funcionalidad necesaria.

Asimismo, esta tarea previa contribuye a construir código fuente más simple, ya que solo se tendrá que pensar en cómo superar esa prueba.

Para tal fin, se utilizará la tarjeta casos de prueba propuesta. Ver Apéndice 9. En estas tarjetas se describen un caso de prueba que permite validar una funcionalidad del sistema.

Además, en esta parte del proceso el cliente estará disponible, considerado experto en el tema, para formar parte del equipo y aclarar las dudas que surjan durante el desarrollo del sistema (construcción de código fuente).

Una buena práctica será la integración del código fuente en forma frecuente, en periodos de tiempo cortos. Esto conlleva a evitar la fragmentación de esfuerzos, por ejemplo, se podría compartir elementos o reutilizar el código fuente elaborado por otro desarrollador. Aunado a este aspecto, se propicia que el equipo trabaje con la última versión, evitando así la integración con el código fuente obsoleto y los problemas de compatibilidad.

Asociado al párrafo anterior, se realizará la integración en forma secuencial, considerando que al realizar el desarrollo (construcción del código fuente) del sistema de manera incremental se añadirá funcionalidad o características, lo que provoca nuevas versiones del sistema; y es posible que exista la integración paralela, lo que provocaría una combinación de código fuente que no ha sido probado en conjunto. Es en este punto donde los problemas de integración pasan sin ser detectados. La integración secuencial resuelve este problema y evita trabajar con versiones obsoletas.

Otra característica importante será la propiedad colectiva, esto quiere decir que cualquier desarrollador puede cambiar cualquier línea de código fuente para añadir funcionalidad, corregir errores, mejorar los diseños o refactorizar. No hay una sola persona que se convierta en un cuello de botella para los cambios. Todos contribuyen con nuevas ideas para todo el proyecto.

5.2.3.1.2.2.5 Pruebas.

El propósito de las pruebas es validar las funcionalidades y características del sistema, es un elemento clave para garantizar la calidad del mismo. Este paso se compone de:

a. Casos de prueba: realizar casos de prueba antes de construir el código fuente, es considerada una buena práctica. Le ayudará al desarrollador a elaborar el código fuente necesario para pasar esa prueba; le permite determinar qué es lo que realmente necesita realizar.
b. Las pruebas unitarias: comprueban la funcionalidad de cada uno de los módulos o componentes del sistema, pueden ser ejecutadas a diario y podrían brindar información del avance que tiene el proyecto. Es poner a prueba el código fuente.
c. Las pruebas de aceptación: también llamadas pruebas del cliente (S. Pressman, 2010, pág. 65), son especificadas y llevadas a cabo por el cliente. Se centran en las características y funcionalidad generales del sistema que son visibles y revisables por parte del cliente. El sistema se pone a prueba usando datos para verificar que se cubren las necesidades reales del cliente.

   Las pruebas de aceptación se derivan de las historias de los usuarios que se han implementado como parte de la liberación del sistema.

   Estas pruebas se realizan durante el *sprint*, con el propósito de tener un tiempo adecuado para posibles ajustes.

5.2.3.1.2.3 Eventos.

RUP tiene como característica el desarrollo incremental e iterativo; mediante las iteraciones, o bien periodos de tiempo en donde se desarrollan ciertas funcionalidades del sistema, se producen los incrementos lo que permite que el sistema evolucione en forma continua, y cada vez más se liberen versiones mejoradas del mismo.

Para controlar estos periodos de tiempo se consideran los eventos del marco de trabajo Scrum. Estas se ajustaron según las necesidades del área y para poder ser entrelazadas con las reglas de Programación Extrema.

En Scrum existen eventos predefinidos con el propósito de crear regularidad y minimizar la necesidad de reuniones no definidas. Todos los eventos son bloques de tiempo (*time-boxes*), de tal modo que todos tiene una duración máxima y se detallan a continuación.

5.2.3.1.2.3.1 Sprint.

Es un periodo de tiempo definido durante el cual se crea un incremento de producto terminado utilizable, potencialmente entregable. Este bloque de tiempo tendrá una duración de 2 semanas y estará conformado por la siguiente estructura:

5.2.3.1.2.3.2 Plan de entregas y cronograma de lanzamientos.

Conocido como planificación del *sprint* en Scrum. Es una reunión de planificación con una duración de cuatro horas, en la que se acuerda las tareas a realizar durante el *sprint*. Con esto inicia el *sprint*.

El equipo de trabajo colabora para seleccionar y comprender el trabajo que será realizado en el *sprint* que está por comenzar.

A partir de la Tarjeta historia de usuario ordenada, el cliente y desarrolladores discuten cada ítem y llegan a un acuerdo compartido respecto al mismo y al trabajo necesario para completarlo en forma consistente.

El resultado de la reunión es lograr responder estas preguntas:

- → ¿Qué trabajo será realizado en el *Sprint*?
- → ¿Cómo se realizará el trabajo seleccionado?

5.2.3.1.2.3.3 Reunión diaria.

Conocida en Scrum como Scrum diario (*Daily* Scrum). Es una reunión todos los días, a las 7:15 am con una duración de quince minutos, y tiene lugar en la cercanía de los

cubículos donde se ubica los desarrolladores. Participan en la reunión los desarrolladores y el analista de sistema. Los participantes se encuentran de pie en círculo para evitar largas discusiones.

Se puntualiza en las siguientes preguntas:

1) ¿Qué hiciste ayer?
2) ¿Qué harás hoy?
3) ¿Hay algún impedimento?

El objetivo es comunicar problemas, soluciones y promover el enfoque del equipo, además, seguir el progreso de las tareas fijadas para el *sprint* en curso y la retroalimentación, para tomar decisiones que garanticen el entregable del producto.

5.2.3.1.2.3.4 Revisión y retrospectiva del sprint.

En el marco de trabajo Scrum son dos eventos separados, pero para evitar sesiones de trabajo consecutivas se realiza un solo evento.

Esta sesión de trabajo tiene una duración de 2 horas y tiene una connotación informal. En la primera hora el equipo de trabajo verifica el resultado del *sprint*. El

cliente determina si los elementos seleccionados y acordados en la planificación del *sprint* fueron terminados.

Aquellos elementos que no fueron terminados completamente y presenten deficiencias volverán al Formulario Historias de usuario para ser estimados y priorizados de nuevo, y se enlistarán para el siguiente *sprint*.

En la segunda hora participan de la sesión el Director de proyecto, el Analista de sistema y Desarrollador. Se trata de una retroalimentación sobre lo que se hizo en el periodo de tiempo del *sprint* como un mecanismo de mejora continua.

- → ¿Qué se hizo bien?
- → ¿Qué se hizo mal?
- → ¿Qué se debería empezar a hacer?
- → ¿Qué se debería dejar de hacer?
- → ¿Qué se debería seguir haciendo?

Se pretende ir moldeando a las personas para lograr una mayor agilidad y cohesión en el proceso.

5.2.3.1.2.4 Equipo de trabajo.

Las personas inmersas en este proceso deberán cumplir con ciertos roles, para tal efecto, se consideran los descritos en el marco de trabajo de Programación Extrema, pero se realiza cierto ajuste para homologarlos con los puestos identificados en la institución. La finalidad es puntualizar las tareas afines al proceso para cada persona.

Este equipo tendrá las siguientes dos características:

- ✓ Autoorganizados: porque eligen la mejor forma de llevar a cabo su trabajo y no son dirigidos por personas externas al equipo.
- ✓ Multifuncionales: tienen todas las competencias necesarias para llevar a cabo el trabajo sin depender de otras personas que no son parte del equipo.

El equipo de trabajo entrega producto de forma iterativa e incremental, maximizando las oportunidades de obtener retroalimentación. Estas entregas de producto terminado aseguran que siempre estará disponible una versión potencialmente útil y funcional del producto.

El equipo de trabajo estará conformado por cuatro roles, los cuales se detallan a continuación.

5.2.3.1.2.4.1 Cliente.

Es el responsable de maximizar el valor del producto y el trabajo del equipo de desarrollo. Es la persona que tiene el conocimiento de cómo debe funcionar el sistema determinado, cubriendo todas las necesidades del proceso/negocio, o personas interesadas.

Es responsabilidad de los clientes asegurar que la mayoría de la funcionalidad del sistema deseada esté integrada en las historias de usuario. Además, el cliente determina la funcionalidad que se pretende en cada *sprint*, y define las prioridades de la implementación, según el valor que aporta al negocio cada historia de usuario. También decidirá en qué momento se liberará o se hará uso de esta funcionalidad.

Los clientes deben tomar las decisiones que afectan sus objetivos de negocio. Al formar parte del equipo y estar disponibles permitirá probar el sistema previamente y retroalimentar a los desarrolladores más pronto. Como miembro del equipo de trabajo no solo probará el sistema, tendrá participación activa durante el proceso de *software* para aclarar dudas.

Es el encargado de ejecutar las pruebas de aceptación para dar un entregable por terminado.

5.2.3.1.2.4.2 Desarrollador.

Grupo de personas que de manera conjunta desarrollan el producto del proyecto. Se encargan de traducir y plasmar las historias de usuario en código fuente. Tienen un objetivo en común. Además, son quienes estiman el tiempo de desarrollo de cada historia de usuario, para que de esta manera el cliente pueda asignarle prioridad dentro del sprint; por lo tanto, cada sprint incorpora nueva funcionalidad de acuerdo a las prioridades establecidas por el cliente.

También es el encargado de ejecutar los casos de prueba y las pruebas unitarias que ha implementado o modificado.

5.2.3.1.2.4.3 Analista de sistema.

En Programación Extrema se conoce como Entrenador/Mentor. Es la persona responsable general del proyecto, acompaña al equipo durante todo el desarrollo del proyecto y asegura que se cumplan las buenas prácticas en el marco de trabajo establecido, actuando como un facilitador, eliminando aquellos obstáculos que dificultan al equipo lograr el objetivo establecido en la iteración. Esta persona puede fomentar la motivación y la autodisciplina al equipo de trabajo para la consecución de los objetivos.

5.2.3.1.2.4.4 Director de proyecto.

Conocido en Programación Extrema como Administrador (*Big boss*). Tiene como función mantener el vínculo entre el equipo de trabajo y el cliente. Realiza labores de gestión. Administra las sesiones de trabajo, construye el plan de trabajo, obtiene los recursos necesarios y maneja los problemas que se generan. Su labor fundamental es de coordinación.

5.2.3.1.3 Ejercicio para la evaluación de la propuesta.

Con el propósito de evaluar si la propuesta en la metodología de desarrollo de *software* se adecua a las necesidades de la institución, específicamente del área Tecnologías de la Información, División Infraestructura; se realizó un ejercicio.

En este se desarrollaron algunas funcionalidades del Sistema de Inventario del Espectro Radioeléctrico del ICE (SIERICE); el cual tiene como finalidad almacenar, administrar, consultar, visualizar y obtener una trazabilidad de todos los documentos con información relevante sobre el espectro radioeléctrico asignado al ICE.

Este sistema web manejará información relevante y sensible de la institución, en cuanto al espectro radioeléctrico asignado, por lo que por políticas de confidencialidad del ICE se maneja y se muestra cierta información para los fines del proyecto de tesis. Este aspecto es considerado una limitante para efectos de este ejercicio, no obstante, es posible la aplicación del ejercicio.

Este entregable consistió en una historia de usuario que se descompone en cuatro tareas de desarrollo, las cuales son descritas en forma general:

1) Facilidad para gestionar documentos de diversos tópicos a WindChill[1].
2) Visualización de documentos listados en directorios de WindChill.
3) Búsqueda de documentos.
4) Visualización de documentos.

En este proyecto, utilizando la metodología tradicional que se venía manejando, duraría aproximadamente 12 meses en desarrollar la totalidad de sus funcionalidades y características. Se desarrolla en el primer *sprint* (dos semanas) algunas funcionalidades del módulo: Documentos.

[1] WindChill es una herramienta informática utilizada en la División Infraestructura, que consiste en un reservorio de archivos.

El módulo de Documentos gestiona toda la documentación relacionada al espectro radioeléctrico y consta de los siguientes requerimientos:

→ En el menú principal de Documentos, y para cada una de las opciones:

- ❖ “Acuerdos ejecutivos” (Concesionario y Anualizado).
- ❖ “Canon de Espectro” (Anualizado).
- ❖ “Permisos” (Concesionario y Anualizado).
- ❖ “PNAF y Modificaciones” (Anualizado) Subopción/subcarpeta para Modificaciones.
- ❖ “Resoluciones Técnicas” (Concesionario y Anualizado).
- ❖ “Audiencias” (Concesionario y Anualizado).
- ❖ “Concesiones” (Concesionario y Anualizado).
- ❖ “Renuncias de Frecuencias / Canales” (Concesionario y Anualizado).
- ❖ “Solicitudes de Frecuencias / Canales” (Concesionario y Anualizado).
- ❖ “Interferencias” (Anualizado).
- ❖ “Prórrogas de uso de frecuencias” (Concesionario y Anualizado).

▪ Se debe mostrar y gestionar (eliminar, exportar, reportes, referencia, entre otras) los documentos cargados en los directorios.

→ Cargar documentos: esta opción debe desplegar una ventana emergente donde permite al usuario final buscar el archivo (en su máquina), que desea cargar al sistema, e indicar los lugares donde se debe almacenar (directorios del servidor).

→ Búsqueda de documentos: esta opción debe permitir localizar un archivo por alguno de los siguientes campos:

- Descripción (Nombre del documento).

- Tipo de documento.
- Código (Nemónico + Número + Año).
- Concesionario.
- Año.
- Fecha.

Los campos de búsqueda deberán ser mostrados al usuario para su elección.

5.2.3.1.3.1 Desarrollo del ejercicio.

Se llevó a cabo una sesión de trabajo con el cliente, la Directora de proyectos y el Analista de sistema, para conocer el requerimiento del cliente y determinar el valor que aportaría al proceso/negocio y a la División Infraestructura el desarrollo del sistema informático.

Se determinó la factibilidad del proyecto, el cliente completó y entregó la documentación requerida; el Analista de sistema revisó la documentación y las historias de usuario brindadas por el cliente fueron analizadas por el desarrollador asignado al proyecto, estimó los tiempos y realizó la planeación. El cliente en conjunto con el desarrollador estableció la historia a entregar al finalizar el *sprint* (dos semanas)

El desarrollador en conjunto con el Administrador de base de datos, realizaron el diseño preliminar, y posteriormente este último creó la base de datos; el desarrollador realizó las pruebas unitarias y construcción el código fuente. Durante el *sprint* el cliente fue contactado para esclarecer algunas especificaciones de requerimiento y antes de finalizar el *sprint* el cliente realizó las pruebas de aceptación, las cuales se superaron satisfactoriamente.

Con lo anterior, se obtuvo el módulo entregable, probado y funcional en dos semanas, el ejercicio se realizó del 23 de enero al 06 de febrero del 2018. Ver Apéndice 11.

5.2.3.1.3.2 Resultado del ejercicio.

Con este ejercicio se constató que el cliente puede hacer uso de las funcionalidades y características, que aportan valor al proceso/negocio y a la División Infraestructura, en un tiempo corto, oportuno y adecuado; sin tener que esperar hasta que finalice el desarrollo de la totalidad del sistema.

La documentación del proceso *software* mejoró significativamente, pasó de documentación compleja, incompleta o nula (particularmente de diseño) a contar con documentación (instrumentos) simples, que a su vez mejoran la calidad del sistema.

Por otra parte, se tuvo la experiencia de un ambiente de desarrollo de *software* típico, donde se distribuyó de una mejor manera las cargas de trabajo, y se asignaron tareas con mayor afinidad al desarrollo de *software*. Esto se convirtió en bienestar y satisfacción en el equipo de desarrollo. Además, al ser el desarrollador quien estimó los tiempos y realizó la planeación basado en su experiencia contribuyó a un ambiente confortable, es decir, sin tanta presión para cumplir sus tareas.

Asimismo, mejoró la comunicación en el equipo de trabajo, especialmente se evitó la fragmentación de esfuerzos, ya que se compartió código fuente e ideas entre el equipo, esto consecuentemente conduce a la propiedad colectiva. Además, el cliente al ser parte del equipo conocía el estado del progreso.

CAPITULO VI: CONCLUSIONES Y RECOMENDACIONES DEL PROYECTO.

6.1. CONCLUSIONES.

Inicialmente, se concluye que se cumplieron los objetivos establecidos en el proyecto, el trabajo establece una propuesta para mejorar la metodología de desarrollo de *software* que contiene los elementos necesarios de acuerdo a las mejores prácticas, modelos y marcos de trabajo de la industria del *software*. El área cuenta con el recurso humano competente, lo que aumentaría el éxito de la implementación.

- Se concluye que la metodología de desarrollo de *software* utilizada es funcional para el proceso *software*, sin embargo, no se encuentra alineada en su totalidad con las mejores prácticas de la industria del *software*, por lo que formalizar este estilo de trabajo adicionando rigurosidad y mejores prácticas de la industria del *software* asegura mayor calidad en el producto *software*.
- Se determina que el proceso *software* es complejo y dirigido por un plan, lo que ocasiona que al realizar la entrega del sistema los requerimientos iniciales del usuario no se ajusten a las necesidades reales debido al tiempo transcurrido que incide con el dinamismo de la organización, convirtiéndose en un sistema obsoleto.
- Se determina que la documentación es un elemento importante que no solo garantiza la calidad del *software*, también facilita el mantenimiento del mismo. Proporciona un instrumento para que otros desarrolladores comprendan la arquitectura del sistema ante posibles cambios por mejoras o la inclusión de nuevas funcionalidades y características.

- El proceso *software* se caracteriza por ser atípico, debido a la asignación de tareas con poca afinidad al desarrollo de sistemas, o bien, con salidas abruptas en la ejecución de lo planeado; esto causa frustración en el equipo de desarrollo, desenfoque en las tareas de desarrollo (construcción de código fuente) y un ambiente de cierta tensión. Estos aspectos generan disminución en la productividad, ralentización en el proceso *software* y empobrece la creatividad.
- Se obtiene que la comunicación es un valor fundamental durante el proceso *software*. Este principio, directamente, con lleva a la satisfacción del cliente, y fomenta la transparencia en el equipo de trabajo, permitiendo que todos estén enterados del proceso y actividades, promoviendo así la toma de decisiones más acertadas dejando a un lado los “supuestos” que provocan retrocesos, y el fraccionamiento de esfuerzos en los miembros del equipo de desarrolladores.
- Se concluye que la aplicación de una metodología ampliamente utilizada en la industria de *software* es indispensable para afianzar una disciplina que apunte a una ingeniería de *software* eficiente, permitiendo así desarrollar sistemas sin errores, de mayor calidad, en un menor tiempo y lo más importante a un menor costo. Además, es utilizada para estructurar, planificar y controlar el proceso de desarrollo de sistemas de información.
- Se determina que una metodología de desarrollo de *software* no necesariamente es adecuada para usar en todos los proyectos, cada una de las metodologías disponibles es más apropiada para tipos específicos de proyectos. La intensión de una metodología de desarrollo de *software* es realizar el proceso de la producción del sistema en forma eficiente, contribuyendo a la calidad de una

forma costeable, asimismo facilitando la consecución de la estrategia del negocio.

- El desarrollo ágil de *software* propicia un mayor acercamiento con el cliente, lo que asegura la especificación de requerimientos del sistema; por otra parte, realizar entregables de aquellas funcionalidades que aportan valor al proceso/negocio, y consecuentemente a la institución, aumenta el éxito del sistema, debido a que le cliente prioriza los requerimientos y toma las decisiones de negocio.
- Las características del desarrollo ágil de *software* se ajustan al dinamismo de la institución, específicamente de la División Infraestructura, a través del enfoque iterativo e incremental se satisfacen las necesidades de los clientes mediante la liberación de versiones del sistema cada vez mejores en periodos cortos de tiempo, siendo estos entregables adecuados y oportunos. Asimismo, favorece para aliviar la presión ejercida por la competencia o el entorno del negocio.
- Se concluye que la propuesta en la mejora de la metodología de desarrollo de *software* satisface las necesidades de los clientes, ya que se entregaron funcionalidades probadas y listas para su operación en dos semanas. Además, se brindó instrumentos para documentar el sistema.
- Se obtiene que mediante la propuesta el equipo de trabajo, especialmente desarrolladores, experimentaron un ambiente de desarrollo de *software* típico, donde las cargas de trabajo fueron equilibradas en todo el equipo de trabajo, y se asignaron tareas con mayor afinidad al proceso *software*, produciendo un

ambiente confortable, aspecto que incide en la potencialización de sus habilidades.

- La propuesta permitió estandarizar el proceso *software*, aprovechando las mejores prácticas de la industria ajustándolas a los recursos del área.
- Se concluye que no existen metodologías o procesos ágiles, solo hay equipos de trabajo ágiles. Los procesos estudiados en este proyecto y considerados como ágiles cuentan con entornos que contribuyen para que este aprenda y ajuste lo necesario para ser ágil. La forma en que un equipo trabaja en conjunto es mucho más importante que cualquier proceso.

6.2. RECOMENDACIONES.

- Se recomienda implementar la propuesta en la mejora de la metodología de desarrollo de *software*, con la finalidad de aprovechar las mejores prácticas de la industria del *software*.
- Aunado a lo anterior, y con el propósito de asegurar el éxito de la implementación se debe continuar con los pasos siete, ocho, nueve y diez de la metodología para el rediseño de procesos propuesta por Madison (2005), expuestos en la sección 3.5, Diseño de investigación, del presente documento. Uno de los pasos más importantes es presentar el rediseño del proceso a las jefaturas (directores y jefe de la División Infraestructura) y clientes para promover

la transformación del ADN organizacional, referente al proceso para el desarrollo de sistemas.

- Se recomienda tomar en consideración las mejores prácticas de la industria del *software* para la automatización de las pruebas, con el propósito de agilizar más el proceso, y asegurar que los desarrolladores las ejecuten.
- Considerar la implementación de un portafolio de proyectos que permita el manejo adecuado de las solicitudes relacionados a desarrollos de *software*, o bien, cambios en las prioridades planificadas; brindando así un panorama del estado de los proyectos.
- Para mejorar la comunicación con los clientes, especialmente con las jefaturas y directores de la División Infraestructura, se sugiere crear un buzón de correo electrónico administrado por el director de proyectos y analista de sistema, con el fin de brindar respuestas ante las consultas de seguimiento de los proyectos y retroalimentación en forma ejecutiva. Este aspecto satisface dos de los deseos, necesidades de los clientes, según la tabla 6. Resumen calificaciones del cliente, sección 5.2.2.
- Se recomienda establecer un repositorio de código que permita una integración de código fuente en forma frecuente y secuencial, evitando problemas por el acoplamiento de código fuente en versiones obsoletas.
- Se recomienda establecer métricas en forma consensuada, y basadas en la filosofía del empirismo de los desarrolladores, que permitan regular el cálculo de las estimaciones para la construcción de código fuente, evitando así discrepancias significativas en las estimaciones realizadas por cada desarrollador.

CAPITULO VII: APÉNDICES Y ANEXOS

7.1. APÉNDICES.

En esta sección se muestran los medios utilizados para recolectar la información.

7.1.1 Apéndice 1. Encuesta al equipo de desarrollo de software.

Encuesta al equipo de desarrollo de software

1. ¿Considera que la metodología de desarrollo actual se ajusta a las necesidades y estrategia de la División Infraestructura y del Sector Telecomunicaciones del ICE?

   ☐ Si.
   ☐ No.

2. ¿Conoce claramente la metodología de desarrollo de software utilizada?

   ☐ Si.
   ☐ No.

3. ¿Se siente cómodo con la metodología de desarrollo de software actual?

   ☐ Si.
   ☐ No.

4. ¿Considera usted que las herramientas y técnicas de desarrollo que utiliza le permite trabajar con rapidez y simplicidad?

   ☐ Si.
   ☐ No.

5. ¿Considera que existen cuellos de botella en la metodología de desarrollo actual?

   ☐ Si.
   ☐ No.

6. ¿Cree usted necesario un cambio en la metodología de desarrollo de software actual?

   ☐ Si.
   ☐ No.

7. Desde su punto de vista, ¿cuáles son las deficiencias o problemas que presenta la metodología de desarrollo actual?

   _____
   _____
   _____
   _____
   _____
   _____

8. ¿Dispone actualmente el área de TI, de la División Infraestructura, con los recursos necesarios para mejorar la metodología de desarrollo de software?

   ☐ Si.
   ☐ No.

9. ¿Considera usted que realizar cambios en la metodología de desarrollo de software traerá beneficios?

   ☐ Si.
   ☐ No.

10. ¿Ha recibido capacitación sobre metodologías de desarrollo de software ampliamente utilizadas en la industria?

    ☐ Si.
    ☐ No.

11. ¿Considera usted que en las sesiones de trabajo para la definición de requerimientos participa el personal estratégico idóneo?

    ☐ Si.
    ☐ No.

12. ¿Qué medidas sugiere usted se pueden implementar para mejorar los tiempos de entrega del producto software?

    _____
    _____
    _____
    _____
    _____
    _____

7.1.2 Apéndice 2. Encuesta a clientes de la División Infraestructura.

Encuesta a clientes de la División Infraestructura.

1. ¿Cuál es el grado de satisfacción con los desarrollos de sistemas que ha brindado el área de TI?

   ☐ Satisfecho.
   ☐ Muy satisfecho.
   ☐ Insatisfecho.
   ☐ Muy insatisfecho.

2. ¿El área de TI ha cumplido con los tiempos establecidos para la entrega e implementación de los proyectos?

   ☐ Casi siempre.
   ☐ Siempre.
   ☐ Casi nunca.
   ☐ Nunca.

3. ¿Los sistemas desarrollados cumplen con los requerimientos definidos en las sesiones de trabajo?

   ☐ Casi siempre.
   ☐ Siempre.
   ☐ Casi nunca.
   ☐ Nunca.

4. ¿Considera usted que el tiempo establecido para la entrega e implementación del sistema es adecuado y oportuno?

   ☐ Si.
   ☐ No.

5. ¿Considera usted que se brinda la inducción correcta para el uso del nuevo sistema?

   ☐ Si.
   ☐ No.

6. ¿El sistema entregado cuenta con la documentación y autoayuda adecuada?

   ☐ Si.
   ☐ No.

7. Posterior a la entrega e implementación del sistema, ¿considera usted que se brinda el soporte y mantenimiento adecuado al sistema?

   ☐ Si.
   ☐ No.

7.1.3 Apéndice 3. Entrevista a la coordinadora de TI, División Infraestructura.

Entrevista a la coordinadora de TI, División Infraestructura.

1. ¿Considera que la metodología de desarrollo actual se ajusta a las necesidades y estrategia de la División Infraestructura y del Sector Telecomunicaciones del ICE?

   ☐ Si.
   ☐ No.

2. ¿Por cuánto tiempo han utilizado la metodología de desarrollo de software?

   _____

3. ¿Considera que existen cuellos de botella en la metodología de desarrollo actual?

   ☐ Si.
   ☐ No.

4. ¿Cree usted necesario un cambio en la metodología de desarrollo de software actual?

   ☐ Si.
   ☐ No.

5. ¿Dispone actualmente el área de TI, de la División Infraestructura, con los recursos necesarios para mejorar la metodología de desarrollo de software?

   ☐ Si.

   ☐ No.

6. Desde su punto de vista, ¿cuáles son las deficiencias o problemas que presenta la metodología de desarrollo actual?

   _____
   _____
   _____
   _____
   _____
   _____

7. ¿Ha recibido capacitación sobre metodologías de desarrollo de software ampliamente utilizadas en la industria?

   ☐ Si.
   ☐ No.

8. ¿Con qué frecuencia se solicitan cambios en las prioridades de los desarrollos que se ejecutan?
   - ☐ Casi siempre.
   - ☐ Siempre.
   - ☐ Casi nunca.
   - ☐ Nunca.

9. ¿Con qué frecuencia se cumplen los tiempos de implementación de los proyectos que se desarrollan?
   - ☐ Casi siempre.
   - ☐ Siempre.
   - ☐ Casi nunca.
   - ☐ Nunca.

10. ¿En su área, se cuenta con alguna herramienta para la gestión de proyectos de TI?
    - ☐ Si.
    - ☐ No.

11. ¿Emplea algún programa de mejora continua en su grupo de trabajo?
    - ☐ Si.
    - ☐ No.

12. ¿Considera que la herramienta utilizada para el seguimiento de los proyectos de desarrollo de software se adecua a las necesidades del área?
    - ☐ Si.
    - ☐ No.

13. ¿Con qué frecuencia brinda seguimiento a los proyectos de desarrollo de software?

    ☐ Casi siempre.
    ☐ Siempre.
    ☐ Casi nunca.
    ☐ Nunca.

14. ¿Dispone de los recursos necesarios para hacer frente a la cantidad de proyectos de desarrollo de software que solicita la División Infraestructura?

    ☐ Si.
    ☐ No.

7.1.4 Apéndice 4. Benchmarking al área Tecnologías de la información, Negocio Ingeniería y Construcción, del Sector Electricidad.

Benchmarking al área Tecnologías de la información, Negocio Ingeniería y Construcción, del Sector Electricidad.

1. ¿Considera que la metodología de desarrollo de software actual se ajusta a las necesidades de su área?

   ☐ Si.
   ☐ No.

2. ¿Por cuánto tiempo ha utilizado esta metodología?

   _____

3. ¿Cuál metodología de desarrollo de software utiliza?

   _____

4. ¿Cuáles herramientas, técnicas o mejores prácticas de la industria del software utilizan para complementar la metodología de desarrollo de software utilizada?

   _____
   _____
   _____
   _____
   _____
   _____

5. ¿Cuáles son los beneficios de la metodología de desarrollo de software utilizada?

   _____
   _____
   _____
   _____
   _____
   _____

6. ¿Existen barreras o factores que dificultan el uso de la metodología de desarrollo de software utilizada? En caso de existir, mencione cuáles.

   ☐ Si.
   ☐ No.

   _____
   _____

7.1.5 Apéndice 5. Minuta sesión de trabajo con la coordinadora del área Tecnologías de la Información.

Minuta de Sesión de trabajo

Asunto: Entrevista para conocer la percepción de la coordinadora de TI en relación a la metodología de desarrollo de software actual.	**Fecha:** 15-12-17 **Hora:** 3:00 pm **Convoca:** Oscar Rojas Salas
Temas tratados:	✓ Respuesta al cuestionario. ✓ Retroalimentación acerca de la metodología de desarrollo de software actual.
Asistentes	Ausentes
Leidy Hernández Rodríguez Oscar Rojas Salas	----------
Próxima reunión	
Fecha: Por definir Hora: Lugar:	

7.1.6 Apéndice 6. Minuta sesión de trabajo con el equipo de trabajo.

Minuta de Sesión de trabajo

Asunto: Sesión de trabajo para rediseño de proceso software, proyecto de graduación	**Fecha:** 09-01-18 **Hora:** 1:00 pm **Convoca:** Oscar Rojas Salas	
Temas tratados:	✓ Revisión del diagrama de proceso AS-IS. ✓ Sugerencias para el rediseño de proceso.	
Tareas identificadas	Acuerdo	Responsable
Modificar el diagrama de proceso AS-IS, según la retroalimentación recibida.	Editar el diagrama de proceso.	.Oscar Rojas Salas
Realizar el rediseño del proceso de acuerdo a las sugerencias brindadas.	Llevar a cabo la propuesta de mejora	Oscar Rojas Salas
Tarea Pendiente	Responsable	
Mostrar la propuesta de rediseño de proceso software con el propósito de recibir retroalimentación.	Oscar Rojas Salas.	
Asistentes	Ausentes	
Walter Pérez Morales José González Arguedas Gustavo Mata Mata	Leidy Hernández Rodríguez	
Próxima reunión		
Fecha: Por definir Hora: Lugar:		

7.1.7 Apéndice 7. Tarjeta de historias de usuario.

Tarjeta Historias de Usuario
Desarrollos Informáticos
División Infraestructura - Tecnologías de Información

Según la IEEE un requerimiento es:
(1) Una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo. (2) Una condición o capacidad que debe estar presente en un sistema o componentes de sistema para satisfacer un contrato, estándar, especificación u otro documento formal. (3) Una representación documentada de una condición o capacidad como en (1) o (2).

Dirección Técnica:	
Proceso/Negocio:	
Jefatura:	
Cliente:	
Fecha:	

Objetivo del proyecto
Descripción macro de las actividades
Problemática actual y oportunidades de mejora

Tarjeta Historias de usuario

Número de historia	Descripción	Detalle	Consultas de TI	Respuesta del solicitante

Observaciones:
* El número y el código de requerimiento deben ser únicos.
* El número de historia tiene relación a la prioridad que el cliente asignó a la historia.

7.1.8 Apéndice 8. Tarjeta CRC (Clase-Responsabilidad-Colaborador).

Clase:	
Descripción:	
Responsabilidad:	Colaborador:

7.1.9 Apéndice 9. Tarjeta casos de prueba.

Prueba #: Descripción/Finalidad:
Entrada:
Pruebas:
Salida:

7.1.10 Apéndice 10. Minuta sesión de trabajo con el equipo de trabajo.

Minuta de Sesión de trabajo

Asunto: Sesión de trabajo para rediseño de proceso software, proyecto de graduación	**Fecha:** 18-01-18 **Hora:** 1:00 pm **Convoca:** Oscar Rojas Salas	
Temas tratados:	✓ Presentación de la propuesta.	
Tareas identificadas	Acuerdo	Responsable
Considerar la retroalimentación brindada con respecto a la propuesta.	---------	Oscar Rojas Salas
Tarea Pendiente	Responsable	
----------	---------	
Asistentes	Ausentes	
Leidy Hernández Rodriguez Walter Pérez Morales José González Arguedas Gustavo Mata Mata	--------	

7.1.11 Apéndice 11. Carta de comprobación de la ejecución del ejercicio.

09 de febrero del 2018

Universidad Hispanoamericana

Asunto: Ejercicio para evaluar propuesta de la metodología de desarrollo.

En relación al proyecto de graduación "Propuesta de Mejora en la Metodología de Desarrollo de Software, en la División Infraestructura del Sector Telecomunicaciones del Instituto Costarricense de Electricidad, en el periodo 2017-2018", elaborado por Oscar Rojas Salas, cédula 113850640, se llevó a cabo el ejercicio para evaluar la propuesta, la cual contribuye a fortalecer la metodología de desarrollo de software actual.

En el ejercicio se desarrollaron funcionalidades del módulo "Documentos" del Sistema de Inventario del Espectro Radioeléctrico del ICE, durante 2 semanas.

Atentamente,

Leidy Hernández Rodríguez

José González Arguedas

Walter Pérez Morales

Gustavo Mata Mata

7.2. ANEXOS.

En esta sección se encuentra la información recolectada de diversas fuentes.

7.2.1 Anexo 1. Perfil del proyecto.

ice	INSTITUTO COSTARRICENSE DE ELECTRICIDAD GERENCIA GENERAL		Código F01.20.00.001.2005
	PERFIL DEL PROYECTO		Versión 2 Página 1 de 2
Solicitud de Cambio No: 1	Elaborado por: Comité de Proyectos	Aprobado por: Gerencia General	Rige a partir de: 2009/11/05

	Registro No. F01: <Número de versión del documento>
INFORMACIÓN GENERAL DEL PROYECTO	
NOMBRE DEL PROYECTO <Nombre con que se conoce el proyecto>	**CODIGO DEL PROYECTO** <Digite el código del proyecto>
DIRECTOR DEL PROYECTO <Nombre del funcionario que será responsable por alcanzar los objetivos del proyecto>	**PATROCINADOR** <Persona o grupo que ofrece recursos financieros, monetarios o en especie para el proyecto>

PERFIL DEL PROYECTO	
ELABORADO POR <Nombre de persona que elabora este documento>	**FECHA DE ELABORACIÓN** <AAAA-MM-DD> Fecha en que se elabora el documento
NOMBRE DEL SOLICITANTE <Nombre de la jefatura que solicita el proyecto>	**DEPENDENCIA SOLICITANTE** <Dependencia que solicita el proyecto>
ENFOQUE DEL PROYECTO- VISIÓN EJECUTIVA	
DESCRIPCIÓN DEL PROYECTO <Breve descripción del proyecto y por qué es importante desarrollarlo>	
PROBLEMA/NECESIDAD/OPORTUNIDAD DE NEGOCIOS U ORGANIZACIONAL A RESOLVER <Breve descripción de la necesidad,problemática u oportunidad de negocio que se resuelve con el proyecto>	
OBJETIVOS ESTRATÉGICOS 1. < Listar los objetivos estratégicos del negocio a los cuales contribuirá el proyecto>	
OBJETIVO DEL PROYECTO <Criterios enfocados a la entrega del proyecto (Acción del verbo en infinitivo + entrega principal del proyecto + marco de tiempo (para el / antes del dd/mm/aaaa) + costo (horas o colones)>	
ALCANCE DEL PROYECTO	
PRODUCTOS ENTREGABLES 1. <Lista de todos los entregables que se esperan obtener a través del proyecto>	
MÉTRICAS 1. <Definir y listar los indicadores que vienen a ser las normas o calificadores que se aplican a las principales entregables como criterios de aceptación>	
EXCLUSIONES 1. <Definir y listar los elementos que no se entregarán con el proyecto>	
RESTRICCIONES 1. <Definir y listar las limitantes externas o internas al proyecto que afectará su rendimiento y no pueden ser cambiadas>	
SUPUESTOS 1. <Definir y listar los factores considerados reales o ciertos para la planificación del proyecto>	
RIESGOS	

1. <Listar posibles riesgos visibles durante la confección del perfil del proyecto>
FACTORES CRÍTICOS DE ÉXITO 1. <Definir y listar los aspectos más importantes que deben ocurrir para conseguir el objetivo del proyecto y cuyo cumplimiento es absolutamente necesario>
OTROS PROYECTOS RELACIONADOS
PROYECTOS RELACIONADOS: 1. <Listar los proyectos con los que este se relaciona>

FIRMAS DE PARTICIPANTES ELABORARON EL PERFIL	
PARTICIPANTE <Nombre completo y firma de la persona que participó en la elaboración del documento>	**FECHA** <AAAA-MM-DD> Fecha en que el participante firma el documento
PARTICIPANTE <Nombre completo y firma de la persona que participó en la elaboración del documento>	**FECHA** <AAAA-MM-DD> Fecha en que el participante firma el documento
PARTICIPANTE <Nombre completo y firma de la persona que participó en la elaboración del documento>	**FECHA** <AAAA-MM-DD> Fecha en que el participante firma el documento
AUTORIZACIÓN PARA EL PROYECTO	
PATROCINADOR <Nombre completo y firma del patrocinador del proyecto>	**FECHA** <AAAA-MM-DD> Fecha en que el patrocinador firma el documento
DIRECTOR DEL PROYECTO <Nombre completo y firma del director del proyecto>	**FECHA** <AAAA-MM-DD> Fecha en que el director del proyecto firma el documento>

UBICACIÓN ELECTRÓNICA DEL DOCUMENTO
DIRECCIÓN : <Hipervínculo al sitio donde se encuentra la información del proyecto>
RESPONSABLE: <Nombre completo, teléfono, email, responsable de la administración de los documentos del proyecto en el sitio>

7.2.2 Anexo 2. Formulario para el levantamiento de requerimientos.

Formulario para el Levantamiento de Requerimientos
Desarrollos Informáticos
División Infraestructura - Tecnologías de Información

Según la IEEE un requerimiento es:
(1) Una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo. (2) Una condición o capacidad que debe estar presente en un sistema o componentes de sistema para satisfacer un contrato, estándar, especificación u otro documento formal. (3) Una representación documentada de una condición o capacidad como en (1) o (2).

División:	
Dirección Técnica:	
Área/Proceso:	
Jefatura:	
Solicitante:	
Número de Requerimiento:	
Nombre del Requerimiento:	
Fecha de solicitud:	

Objetivo
Descripción de las Actividades
Problemática Actual y Oportunidades de Mejora
Datos de Entrada (Fuentes)
Datos de Salida (Indicar destinos)
Observaciones y Comentarios

LEVANTAMIENTO DE REQUERIMIENTOS

Requerimientos Funcionales

Versión:	
Sistema:	
Módulo :	

Observaciones:

* El número y el código de requerimiento deben ser únicos.
* En caso se selección de opciones, dejar bien claro cuando la opción es única o múltiple. Normalmente se utilizan las cajitas de check para selección múltiple y los puntos para selección única.

Número Requerimient	Código requerimiento	Descripción	Detalle	Consultas de TI	Respuesta del solicitante

LEVANTAMIENTO DE REQUERIMIENTOS

Requerimientos No Funcionales

Versión:	
Sistema:	
Módulo :	

Observaciones:

* El número y el código de requerimiento deben ser únicos.
* En caso se selección de opciones, dejar bien claro cuando la opción es única o múltiple. Normalmente se utilizan las cajitas de check para selección múltiple y los puntos para selección única.

Número Requerimiento	Código requerimiento	Descripción	Detalle	Consultas de TI	Respuesta del solicitante

REFERENCIAS BIBLIOGRÁFICAS.

Agile Alliance. (s.f.). *Agile Alliance*. Recuperado el 13 de Enero de 2018, de https://www.agilealliance.org/agile101/the-agile-manifesto/

Agile Alliance. (s.f.). *AgileAlliance.org*. Recuperado el 28 de Diciembre de 2017, de https://www.agilealliance.org/glossary/xp/#q=~(filters~(postType~(~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'xp))~searchTerm~'~sort~false~sortDirection~'asc~page~1)

Bara, M. (13 de Abril de 2015). *OBS Business School*. Recuperado el 2017 de Diciembre de 27, de OBS Business School: https://www.obs-edu.com/int/blog-investigacion/project-management/las-5-etapas-en-los-sprints-de-un-desarrollo-scrum

Barba Prieto, A. (27 de Julio de 2015). *_becoming an Agile architect*. Recuperado el 27 de Diciembre de 2017, de http://www.becominganagilearchitect.com/scrum-eventos-revision-de-sprint

Casanova, S. (3 de Marzo de 2015). *Samuel Casanova Efectividad en equipos de desarrollo*. Recuperado el 3 de Diciembre de 2017, de https://samuelcasanova.com/2015/03/reunion-diaria-de-scrum-mas-alla-de-las-3-preguntas/

Castellanos, J., Chacón , F., & Carvajal, J. (2016). *emaze*. Recuperado el 24 de Agosto de 2017, de https://www.emaze.com/@AZZTOZQZ

Cendejas Valdes, J. L. (Mayo de 2014). Implementación del modelo integral colaborativo (MDSIC) como fuente de innovación para el desarrollo ágil de software en las empresas de la zona centro - occidente en México. Puebla, México.

Cohen, D., & Asís, E. (2009). *Tecnologías de la información en los negocios* (Quinta ed.). McGRAW-HILL/INTERAMERICANA EDITORES, S.A. DE C.V.

Consejo Nacional de Investigaciones Científicas y Técnicas de Argentina. (2017). *Consejo Nacional de Investigaciones Científicas y Técnicas de Argentina.* Recuperado el 15 de Mayo de 2017, de http://www.conicet.gov.ar/conferencia-mundial-de-ingenieria-de-software-en-buenos-aires/

Duro Lima, S. (28 de Diciembre de 2016). *semrush*. Recuperado el 02 de Diciembre de 2017, de https://es.semrush.com/blog/brainwriting-evolucion-brainstorming/

EcuRed. (s.f.). *EcuRed Conocimiento con todos y para todos*. Recuperado el 28 de Diciembre de 2017, de https://www.ecured.cu/Programaci%C3%B3n_Extrema_(XP)

Editorial MD. (1 de Febrero de 2017). *Editorial MD*. Recuperado el 04 de Noviembre de 2017, de https://www.editorialmd.com/ver/que-es-una-entrevista

Espinoza, R. (13 de mayo de 2017). *RobertoEspinoza.* Recuperado el 29 de diciembre de 2017, de http://robertoespinosa.es/2017/05/13/benchmarking-que-es-tipos-ejemplos/

Extreme Programming. (2009). *Extreme Programming*. Obtenido de http://www.extremeprogramming.org/

Francia, J. (25 de Setiembre de 2017). *Scrum.org*. Recuperado el 27 de Diciembre de 2017, de https://www.scrum.org/resources/blog/que-es-scrum

Grijalva, N. (15 de octubre de 2012). *Ingenieria del Software I*. Recuperado el 26 de Agosto de 2017, de http://sofware1nathalygrijalva.blogspot.com/2012/10/modelo-espiral.html

Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, M. (2014). *Metodología de la investigación.* McGRAW-HILL. Obtenido de http://www.eduteka.org/TaxonomiaBloomCuadro.php3

IBM. (02 de Abril de 2017). *IBM Knowledge Center*. Recuperado el 02 de Enero de 2018, de https://www.ibm.com/support/knowledgecenter/es/SS4JE2_7.5.5/com.ibm.xtools.tutorial.rupguidance/topics/conceptrationalunifiedprocessguidance.html

Iglesias Fraga, A. (11 de Diciembre de 2016). *TICbeat*. Recuperado el 27 de Diciembre de 2017, de http://www.ticbeat.com/tecnologias/que-es-el-desarrollo-agil-y-como-esta-transformando-la-industria-del-software/

Instituto Costarricence de Electricidad. (2017). *intranet.ice*. Recuperado el 02 de Marzo de 2017, de http://intranet.ice/Paginas/default.aspx

Instituto de Normas Técnicas de Costa Rica (INTECO). (2009). *issuu.* Recuperado el 16 de Marzo de 2017, de https://issuu.com/editzacabrera/docs/guia_de_ingenieria_del_software

International Organization for Standardization. (2014). *International Organization for Standardization (ISO).* Obtenido de https://www.iso.org/files/live/sites/isoorg/files/archive/pdf/en/10goodthings.pdf

Jiménez, D. (19 de Septiembre de 2014). *Pymes y Calidad 2.0*. Recuperado el 03 de Enero de 2018, de https://www.pymesycalidad20.com/mejores-practicas-diferencia.html

Kendall, K., & Kendall, J. (2011). *Análisis y Diseño de Sistemas* (Octava ed.). México: Pearson Educación de México, S.A. de C.V.

Madison, D. (2005). *Process Mapping, Process Improvement, and Process Management: A Practical Guide for Enhancing Work and Information Flow.* Paton Professional.

MADO. (7 de Octubre de 2012). *ecuaportales.com*. Recuperado el 01 de Setiembre de 2017, de http://www.ecuaportales.com/mado/index.php/noticias-publicidad/189-sistemas-hechos-a-la-medida

Meléndez Valladarez, S., Gaitan, M. E., & Pérez Reyes, N. N. (28 de Enero de 2016). *UNAM.* Recuperado el 28 de Diciembre de 2017, de http://repositorio.unan.edu.ni/1365/1/62161.pdf

Microsoft. (s.f.). *Microsoft*. Recuperado el 27 de Diciembre de 2017, de https://msdn.microsoft.com/es-es/library/ee191586

Morales, R. (1 de Setiembre de 2014). *Corporación Colombia Digital*. Recuperado el 06 de Setiembre de 2017, de https://colombiadigital.net/actualidad/articulos-informativos/item/7669-lenguajes-de-programacion-que-son-y-para-que-sirven.html

Mota, E. (18 de Enero de 2017). *Azul Web*. Recuperado el 27 de Diciembre de 2017, de https://www.azulweb.net/desarrollo-agil-esta-cambiando-la-industria-del-software/

Murillo Montesdeoca, R. (15 de Mayo de 2015). *Ingeniería de Software*. Recuperado el 29 de Diciembre de 2017, de http://jraquelm2.wixsite.com/ingenieriadesoftware/single-post/2015/05/15/-TEMA-5-PROGRAMACI%C3%93N-EXTREMA-XP

OBS Business School. (15 de Diciembre de 2014). *OBS Business School*. Recuperado el 28 de Diciembre de 2017, de https://www.obs-edu.com/int/blog-project-management/temas-actuales-de-project-management/consejos-para-aplicar-una-programacion-extrema

Oracle. (28 de Setiembre de 2016). *Oracle*. Recuperado el 1 de Setiembre de 2017, de https://www.oracle.com/applications/erp/what-is-erp.html

Ortiz, M. (Setiembre de 2012). *Ingeniería de Software, Modelos de desarrollo de software.* Recuperado el 30 de Agosto de 2017, de http://isw-udistrital.blogspot.com/2012/09/ingenieria-de-software-i.html

Pérez Alabarce, J. (10 de Junio de 2016). *Bravent, IT Consulting Company*. Recuperado el 26 de Diciembre de 2017, de http://info.bravent.net/blog/scrum-metodologias-agiles-beneficios-aportan-al-desarrollo-del-software

Pérez, A. (8 de Mayo de 2017). *CEOLEVEL*. Recuperado el 27 de Diciembre de 2017, de http://www.ceolevel.com/scrum-master-que-es-y-que-no-es

Project Management Institute. (2013). *Guía de los Fundamentos para la Dirección de Proyectos (PMBOK)* (Quinta ed.). Pensilvania, Estado Unidos: Project Management Institute, Inc.

proyectosagiles.org. (s.f.). *Proyectos Agiles.org*. Recuperado el 27 de Diciembre de 2017, de https://proyectosagiles.org/equipo-team/

Ramos Vega, C. (20 de Febrero de 2017). *cristinaramos.com*. Recuperado el 28 de Diciembre de 2017, de https://cristinaramosvega.com/z-los-artefactos-scrum/

Roche, J. (06 de Diciembre de 2017). *Deloitte.* Recuperado el 28 de Diciembre de 2017, de https://www2.deloitte.com/es/es/pages/technology/articles/artefactos-scrum.html

S. Pressman, R. (2010). *Ingeniería del Software: Un enfoque práctico.* (Séptima ed.). New York, Estado Unidos: McGraw-Hill Companies, Inc.

Sánchez, J. (27 de Marzo de 2015). *freelancer.es*. Recuperado el 05 de Setiembre de 2017, de https://www.freelancer.es/community/articles/arquitectura-del-software-desarrollo-aplicaciones

Sandoval N., J. (15 de Noviembre de 2016). *Medium*. Recuperado el 27 de Diciembre de 2017, de https://medium.com/@jlsandovaln/sprint-plannings-planificaci%C3%B3n-de-la-iteraci%C3%B3n-ddb78ac60536

Schwaber, K., & Sutherland, J. (2016). *La Guía de Scrum.*

Scrum.org. (s.f.). *Scrum.org*. Recuperado el 29 de Diciembre de 2017, de https://www.scrum.org/resources/what-is-scrum

Senn, J. (2001). *Análisis y Diseño de Sistemas de Información* (Segunda ed.). México: MCGRAW-HILL.

Significados. (20 de Diciembre de 2014). *Significados.com*. Recuperado el 15 de Setiembre de 2017, de https://www.significados.com/encuesta/

Sommerville, I. (2011). *Ingeniería de Software* (Novena ed.). Naucalpan de Juárez, Estado de México, México: Pearson Educación de México, S.A. de C.V.

Thalú. (9 de Abril de 2013). *thalu.net*. Recuperado el 1 de Setiembre de 2017, de http://www.thalu.net/software-enlatado-vs-software-a-medida/

The Institute of Electrical and Electronics Engineers. (1990). *IEEE Std 610.12-1990, IEEE Software Engineering Standard: Glossary of Software Engineering Terminology.* New York, Estados Unidos: The Institute of Electrical and Electronics Engineers.

The Institute of Electrical and Electronics Engineers, Inc. (1998). *IEEE Std 830-1998: Especificación de Requisitos Software.* New York, Estados Unidos.

Ulate Soto, I., & Vargas Morúa, E. (2014). *Metodología para elaborar una tesis.* San José, Costa Rica: EUNED.

Vic. (29 de Julio de 2015). *latecladeescape.com*. Recuperado el 2017 de Setiembre de 05, de http://latecladeescape.com/h/2015/07/metodologias-de-desarrollo-del-software

Vila Grau, J. (23 de Noviembre de 2017). *Management Plaza THE MANAGEMENT CERTIFICATION COMPANY*. Recuperado el 28 de Diciembre de 2017, de http://managementplaza.es/blog/sabes-como-funciona-xp/

Vincze, J. (07 de Junio de 2016). *DTyOC De Tecnología y Otras Cosas*. Recuperado el 02 de Enero de 2018, de https://dtyoc.com/2016/06/07/modelo-rup-ibm/

Weitzenfeld, A., & Guardati, S. (2008). *Introducción a la computación.* México: Cengage Learning.