UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
INGENIERÍA DE SISTEMAS – PLANEACIÓN

UN MÉTODO PARA LA GESTIÓN DE PROYECTOS DE SOFTWARE, UTILIZANDO EL MODELO ITERATIVO E INCREMENTAL: UN ENFOQUE DE SISTEMAS

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA

PRESENTA:
CÉSAR ALEXEI CARRIZALES MAYORGA

TUTOR PRINCIPAL:
DR. JAVIER SUÁREZ ROCHA,
FACULTAD DE INGENIERÍA

MÉXICO, D. F. NOVIEMBRE 2014

JURADO ASIGNADO:

Presidente:	Dr. Gabriel de las Nieves Sánchez Guerrero
Secretario:	Dr. José Jesús Acosta Flores
Vocal:	Dr. Javier Suárez Rocha
1er. Suplente:	M. I. Francisco José Álvarez y Cazo
2do. Suplente:	M. en E. Rosalba Rodríguez Chávez

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO, FACULTAD DE INGENIERÍA

TUTOR DE TESIS:

Dr. Javier Suárez Rocha

DEDICATORIA

A mi madre Rocío, gracias por tus siempre sabios consejos y por creer en mí.

A mi padre Adelaido, por motivarme para seguirme superando.

A mi hermana Mayda, por todo tu cariño y comprensión.

A mis amigos, por su apoyo en todo momento.

AGRADECIMIENTOS

Al Dr. Javier, por su asesoría en el desarrollo de esta tesis, y por su valiosa amistad.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT), por otorgarme la beca que me permitió cursar la Maestría en Ingeniería de Sistemas y obtener el grado académico.

A la Mtra. Artemisa, por escucharme, guiarme y sobre todo por la gran amistad que tenemos.

Contenido

Índice de Figuras

Índice de Tablas

Resumen

En esta investigación se concluyó que desarrollar software implica, creatividad, complejidad y cambio constante, por tanto, el modelo en cascada no es adecuado para el desarrollo de este tipo de proyectos, debido a su estructura lineal y rígida que evita la realización de cambios después de la recolección de requisitos.

Se detectó como alternativa al modelo iterativo e incremental, pero se determinó que aunque es útil para el desarrollo de software presenta una visión parcial y restringida, debido a su enfoque empírico.

Para obtener una visión holística sobre la gestión de proyectos de software se propuso aplicar el enfoque sistémico, de esta manera se conceptualizo a los proyectos como sistemas, se determinó que un proyecto está constituido por cuatro elementos: 1) entradas donde recibe los insumos que le proporciona el medio ambiente; 2) un proceso de transformación, para el procesamiento de los insumos; 3) salidas, productos únicos (software); y 4) retroalimentación, lo que le permite mantener la homeostasis.

Al analizar el proceso de trasformación del proyecto por medio del enfoque cibernético se determinó que contenía dos subsistemas, el de gestión y el conducido. Se analizó al subsistema de gestión para definir un proceso que permitiera planear, coordinar, dirigir y organizar el desarrollo de software.

Para el desarrollo del subsistema de gestión se decidió utilizar como base metodológica a la planeación como proceso de conducción e integrar en su estructura al modelo iterativo e incremental. De esta manera se diseñó un método, que propone una serie de pasos sistematizados para, ejecutar y gestionar un proyecto de manera eficaz y eficiente, y obtener productos que estén alineados con los objetivos que el cliente pretende alcanzar.

Palabras clave – Gestión de proyectos de software, desarrollo ágil, modelo iterativo e incremental, desarrollo de software, enfoque de sistemas, enfoque cibernético, planeación

Abstract

This research concludes that the waterfall model is not suitable for the development of software projects, due to its linear structure and avoiding change. Since software development involves creativity, complexity and constant change.

Was detected as an alternative to iterative and incremental model, but found that although it is useful for software development presents a partial and restricted vision due to its empirical approach.

For a holistic view of project management software was proposed to apply the systems approach, so I was conceptualized projects as systems, we determined that a project consists of four elements: inputs which receive inputs that gives the environment; a process of transformation, processing of inputs; outputs, unique products (software); and feedback, allowing you to maintain homeostasis.

When analyzing the transformation process of the project through the cybernetic approach was determined to contain two subsystems, the management and driven. We analyzed the management subsystem to set up a process to plan, coordinate, direct and organize the development of software.

For the development process we decided to use as a methodological basis for planning and driving process and integrate in their structure to the iterative and incremental model. Thus you could design a method, which proposes a series of systematic steps, implement and manage a project effectively and efficiently, and get products that are aligned with client objectives pursued.

Keywords - Project management software, agile development, iterative and incremental model, software development, systems approach, cybernetic approach, planning

Introducción

En la actualidad el software es parte indispensable en la vida cotidiana de las personas, ya que se encuentra en muchas partes, por ejemplo: en las computadoras, en los celulares, en las tablets, en las computadoras de los autos, en los sistemas de monitoreo de los aviones, en los relojes digitales, en los televisores, en las consolas de videojuegos, en los electrodomésticos, en los cajeros automáticos, etc. Si uno revisa a su alrededor, por lo menos un dispositivo electrónico tiene instalado software para su funcionamiento.

En este contexto, parecería lógico que los equipos de trabajo utilicen métodos y técnicas sistematizadas para administrar proyectos de software y obtener como resultado productos que satisfacen las necesidades del cliente, pero la realidad no es así, en muchos casos el software obtenido es deficiente y de mala calidad.

Esto se presenta debido a que las empresas, utilizan modelos y procesos que no son adecuados para la gestión de sus proyectos. Al ser éstos deficientes para el desarrollo de software, las empresas optan por adaptar procesos basados en su propia experiencia (prueba y error). En otros casos buscan alternativas, que aunque a primera vista parecieran adecuadas, terminan generando productos solidos desde el punto de vista técnico, pero poco eficientes desde el punto de vista de su utilidad, ya que no cumplen con las expectativas de los clientes. Debido a que las alternativas presentan una visión parcial sobre la gestión de proyectos de software.

Por tanto, los encargados de guiar a los equipos de trabajo (administradores y líderes de proyecto), no tienen claro que elementos intervienen en el desarrollo de software, se centran solamente en recibir requisitos y codificar.

De los párrafos anteriores se desprende la necesidad de esta investigación, que tiene como objetivo, crear un método basado en la planeación como proceso de conducción y el modelo iterativo e incremental, para sistematizar los pasos que debe seguir un equipo de trabajo para planear, coordinar, dirigir y controlar un proyecto de desarrollo de software, de manera eficaz y eficiente, para obtener como resultado productos que estén alineados con los objetivos que el cliente desea alcanzar.

La tesis presenta el siguiente contenido:

Capítulo 1: Problemática en el desarrollo proyectos de software

- ✓ Identificación de las principales causas del fracaso en los proyectos de desarrollo de software.

- ✓ Descripción de las principales alternativas que se han presentado para resolver la problemática.
- ✓ Desarrollo la propuesta de solución así como de los objetivos; la hipótesis; y el alcance del trabajo de investigación.

Capítulo 2: El enfoque sistémico en la gestión de proyectos de software.

- ✓ Definición los conceptos básicos del enfoque sistémico y del enfoque cibernético.
- ✓ Análisis la estructura de los sistemas para determinas sus principales elementos.
- ✓ Aplicación del enfoque de sistemas en la gestión de proyectos de software.
- ✓ Definición de la estructura de un proyecto de software como un sistema e identificación de sus principales elementos.

Capítulo 3: El modelo iterativo incremental y la planeación en el proceso de conducción

- ✓ Se analiza el proceso de transformación de un proyecto de software para determinar sus subsistemas.
- ✓ Descripción del proceso que propone el modelo iterativo e incremental para el desarrollo de software.
- ✓ Análisis de la estructura de la planeación para el proceso de conducción.
- ✓ Integración del modelo iterativo e incremental en la estructura de la planeación como proceso de conducción.

Capítulo 4: *Método para la gestión de proyectos de software*

- ✓ Identificar los pasos a seguir para gestionar un proyecto de software tomando como base la planeación y al modelo iterativo e incremental como proceso de conducción.
- ✓ Conceptualizar los pasos como un método.
- ✓ Proponer técnicas que permitan aplicar el método propuesto.

En su parte final la tesis presenta las conclusiones generales, las líneas sugeridas para futuras investigaciones, así como las referencias correspondientes.

Capítulo 1. Problemática en la gestión de proyectos de software

Objetivos del capítulo:

- ✓ Identificar las principales causas del fracaso en los proyectos de desarrollo de software.
- ✓ Describir de las principales alternativas que se han presentado para resolver la problemática.
- ✓ Plantear y establecer los objetivos; la hipótesis; y el alcance del trabajo de investigación.

1.1 Planteamiento del problema

En 1968, en la primera conferencia organizada por la OTAN sobre el desarrollo de software[1], se mencionó por primera vez el término *crisis del software*. Esta crisis era el resultado de la introducción de las nuevas computadoras basadas en circuitos integrados. El software que se requería para este tipo de dispositivos era mucho más complejo que los sistemas previos (Sommerville, 2005), por ejemplo, la primera computadora electrónica (ENIAC) contaba con 18, 000 tubos de vacío y sólo requerían algunas decenas o centenas de tarjetas perforadas para programarla, el resto de su funcionamiento dependía del hardware[2]. Mientras tanto en los años 70´s con la llegada de los microprocesadores, el hardware seguía siendo esencial; pero para utilizarlo se requerían de cientos de líneas de código, lo que ocasiono una dependencia total del software para el funcionamiento de las computadoras.

En esos momentos el desarrollo de software se basaba principalmente en el modelo denominado *codificar y corregir* (**FIGURA 1**), el cual consistía en (Dooley, 2011):

1. *Codificar*: desarrollar el código según los requisitos solicitados.
2. *Corregir*: evaluar y corregir los errores encontrados en el programa conforme se presentaban.

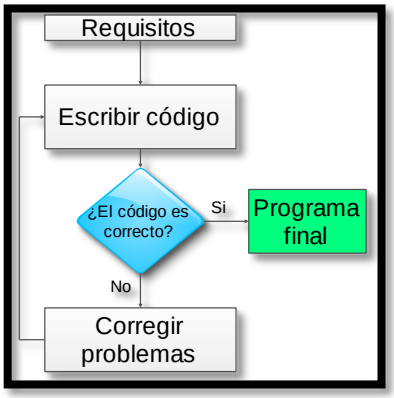


Figura 1. Modelo codificar y corregir.

[1] El software se define como el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación (IEEE, 1990). *Más adelante se dará una explicación más extensa sobre el término.*

[2] Conjunto de elementos físicos o materiales que constituyen una computadora o un sistema informático. Los componentes pueden ser: eléctricos, electrónicos, electromecánicos y mecánicos. El término es propio del idioma inglés (literalmente traducido: partes duras), su traducción al español no tiene un significado por tanto se adopta tal cual.

Los ingenieros recibían cierto número de requisitos y automáticamente comenzaban a programar, no tenían un proceso sistematizado para crear el producto. La experiencia demostró que este enfoque sólo se podia utilizar cuando el software a desarrollar era sencillo y donde intervenian uno o dos programadores, pero no era adecuado para proyectos grandes, pues a menudo éstos llegaban a tener años de retraso, costaban mucho más de lo presupuestado, eran difíciles de mantener y su desempeño era pobre.

Como respuesta a esta crisis, en la citada conferencia de la OTAN, surgió el término *ingeniería de software.* El cual, según el IEEE (Institute of Electrical and Electronics Engineers) lo define como "*la aplicación de un enfoque sistémico (estructurado), disciplinado y cuantificable para el desarrollo, operación y mantenimiento de software. Esto es, la aplicación de la ingeniería en el área del software*".

Para evitar que los proyectos siguieran fracasando, algunos ingenieros comenzaron a publicar artículos para brindar herramientas que permitieran desarrollar software de manera sistémica, el más popular y que se sigue ocupando hasta la fecha, fue propuesto por Winston W. Royce. Quien en 1970 publicó el artículo "Managing the development of large software systems" (Royce, 1970), en donde propuso una serie de etapas para la gestión de sistemas de software de gran envergadura (***Figura 2***):

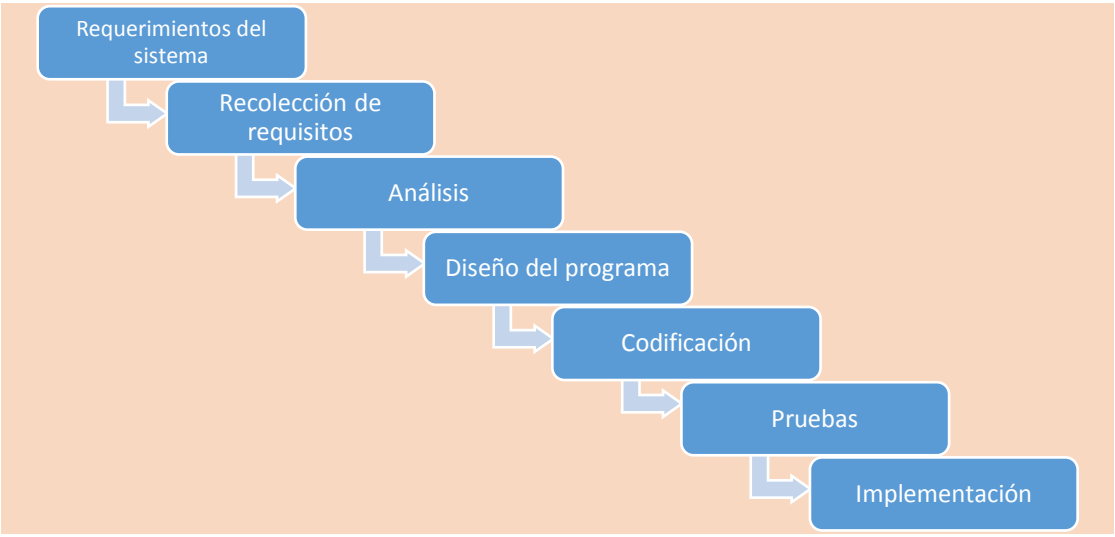


Figura 2. Etapas propuestas por Winston para el desarrollo de software.

Royce indicó claramente en su artículo que, "*la implementación de este enfoque es riesgoso e invita al fracaso*", para tratar de mitigar los defectos del proceso, realizó tres valiosas recomendaciones:

1. Incluir una fase preliminar de diseño (***Figura 3***), en donde se recorrieran todos las etapas propuestos en el proceso, con la finalidad de crear un prototipo para realizar una simulación, la cual mostraría cómo funcionaría el producto de manera general y posteriormente volver a recorrer todas las fases para obtener el producto deseado.

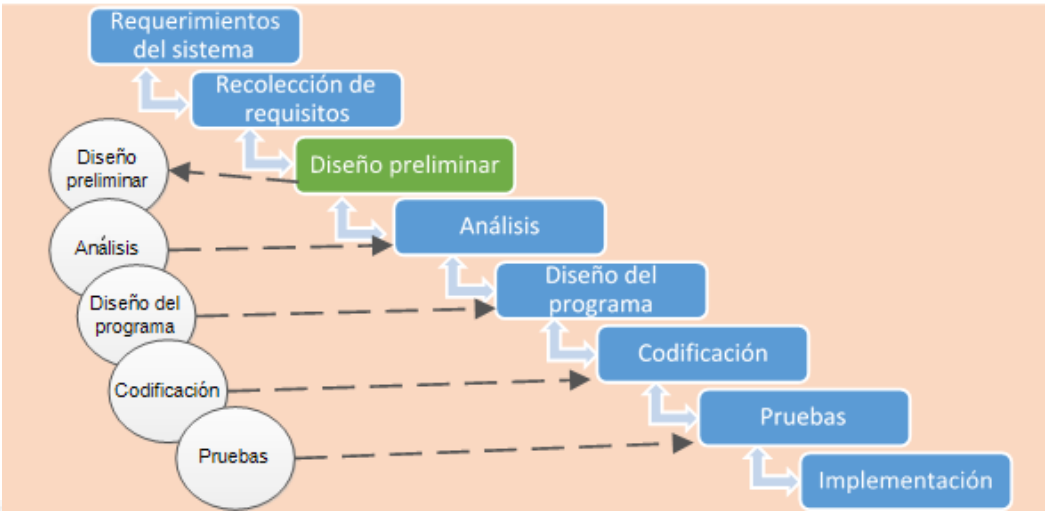


Figura 3. Pasos para el desarrollo de software de Royce con la fase preliminar de diseño

2. La segunda recomendación fue: si se encuentra en alguna etapa y se detecta algún cambio, puede apoyarse en las fases contiguas para realizar las modificaciones necesarias, ya que el proceso es iterativo.
3. La tercera recomendación fue, mantener al cliente presente durante todo el proyecto, no solamente en la fase de recolección de datos y en la fase de pruebas.

A esta serie de etapas, posteriormente se les denominó *modelo en cascada*. La popularidad del modelo se debió a que fue adoptado y difundido por el Departamento de Defensa de los Estados Unidos, en el documento (DOD-STD-2767, 1985), donde se describe a las etapas del modelo (***Figura 3***), como fases del ciclo de vida para el desarrollo de software.

A la forma de trabajo del modelo en cascada se le denomina prescriptivo, ya que todas las tareas deben ser planeadas y detalladas de manera exhaustiva, antes de comenzar el desarrollo del producto, requiere:

1. Tener todos los detalles sobre las características del producto.
2. Documentar los requisitos de manera exhaustiva.
3. Predecir el tiempo, el costo y los recursos con la mayor precisión posible.

Para posteriormente analizar los requisitos solicitados, diseñarlos, programarlos probarlos y liberarlos.

En la actualidad este modelo es el que tradicionalmente se sigue utilizando para el desarrollo de software, también es conocido como *ciclo de vida*[3] *clásico* (Pressman, 2010), en la literatura es comúnmente citado, pero en la mayor parte de los libros se ignoran las recomendaciones que hizo

[3] El ciclo de vida del software, se define como “un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, explotación y mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de requisitos hasta que se deja de utilizar” (ISO, 2008).

Royce y se basan en el primer modelo (***FIGURA 2***), esto da como resultado que el modelo tenga las siguientes características e inconvenientes (***TABLA 1***):

Característica e inconvenientes
No puede comenzar una fase antes de terminar la anterior y no se puede volver atrás. ⚠ Inconvenientes: Esta característica lo vuelve un modelo lineal, y en la realidad esto no es así, las etapas se superponen y son iterativas **(Falgueras, 2002)**. ⚠
El equipo debe documentar de manera exhaustiva todos los requisitos en la primera o segunda fase del modelo, para estimar el tiempo, el costo y los recursos que se utilizarán en el proyecto, esto implica una alta inversión en la recolección de requisitos. ⚠ Inconvenientes: Como se requiere tener todos los requisitos al inicio del proyecto por lo general el alcance se "infla" debido a que los clientes y los interesados piden: ○ Todo lo que necesitan. ○ Todo lo que ellos piensan que pueden necesitar. ○ Todo lo que quieren. ○ Todo lo que se les ocurre.
Todo el proyecto debe ser documentado de manera exhaustiva. ⚠ Inconvenientes: Este es otro grave problema, se genera documentación que no es de utilidad para el proyecto, se invierte dinero, tiempo y recursos, para generar manuales de cientos de páginas, los cuales solamente son consultados por su autor y por las personas que realizan correcciones al documento.
Los interesados pueden o no estar disponibles para responder preguntas, ya que se hace la suposición de que todos los requisitos fueron entregados en la primera fase. ⚠ Inconvenientes: Cuando los interesados no están disponibles o no se solicita su apoyo durante todas las fases del proyecto, se comienzan a realizar suposiciones, lo que provoca que el equipo de desarrollo del proyecto interprete los requisitos, lo que genera como resultado un producto de baja calidad, debido a que no se cumplen las expectativas del cliente.
Se tienen que documentar los cambios, lo que añade más tiempo y presupuesto. ⚠ Inconvenientes: Como los requisitos se congelan (no se pueden modificar) en la primera o segunda fase del modelo, los cambios son difícilmente asumibles en el resto de las fases, ya que una modificación implica regresar a los primeros pasos del modelo.
La prueba final se realiza como última fase del proyecto, por tanto los comentarios de los clientes se escuchan hasta el final del proyecto. ⚠ Inconvenientes: Los interesados pueden ver avances generales del proyecto, pero pocas veces observan la funcionalidad del producto durante las fases intermedias del modelo, la funcionalidad real la observan hasta el final del proyecto en la fase de pruebas y aquí es donde los cambios hacen que los costos y el tiempo se dupliquen o se tripliquen.
El financiamiento está en curso durante todo el proyecto. ⚠ Inconvenientes: El valor del producto aparece al final del proyecto y si el financiamiento se acaba, por lo general el proyecto ofrece un valor cero a la organización.

Tabla 1. Características e inconvenientes del modelo en cascada.

En el siguiente ejemplo se observa cómo se maneja un proyecto de software con base en el modelo en cascada (***FIGURA 4***).

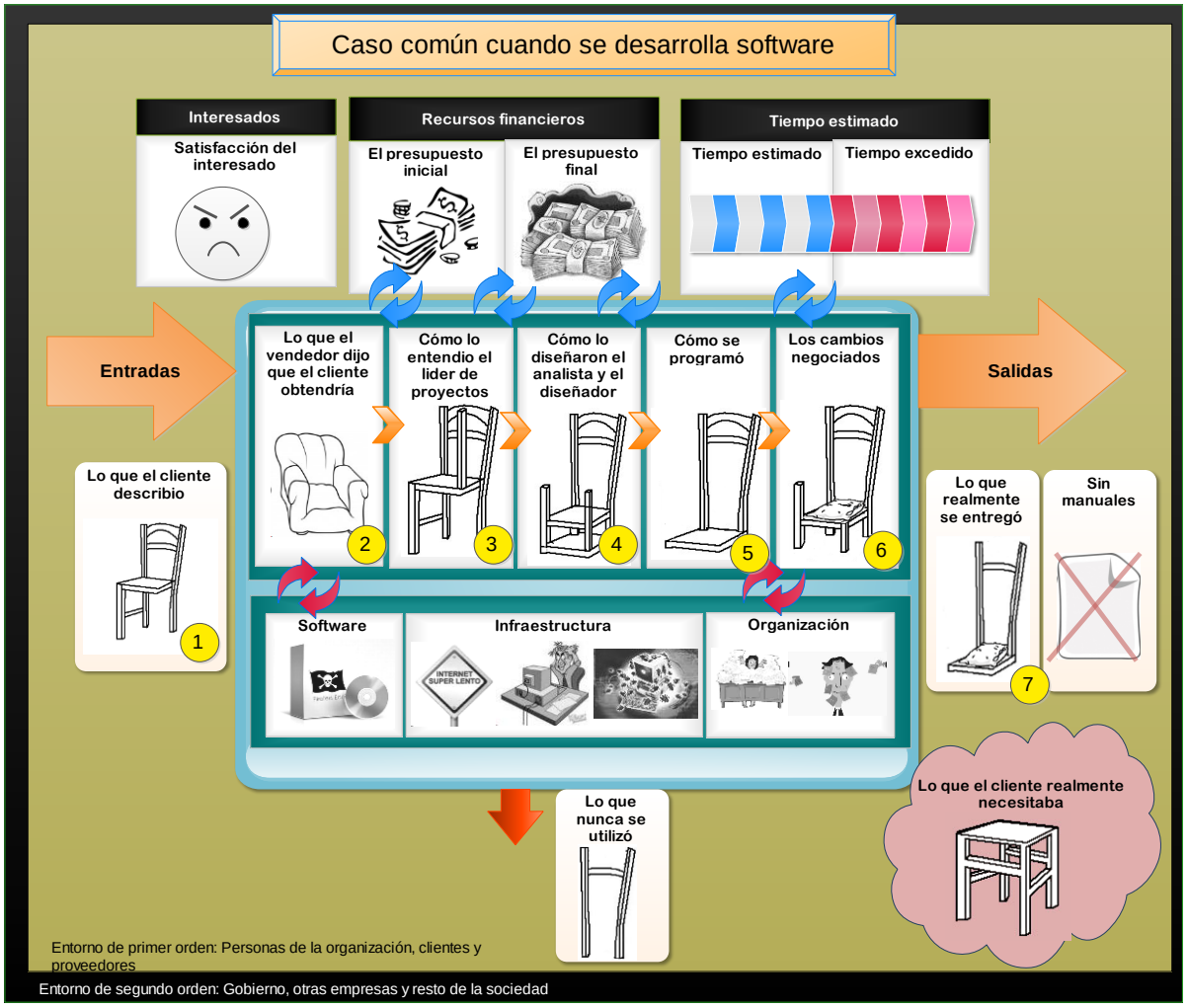


Figura 4. Caso común al desarrollar el modelo en cascada. Elaboración propia.

1. El cliente describe el producto que desea obtener.
2. El vendedor documenta de manera exhaustiva los requisitos y describe al cliente lo que obtendrá como resultado.
3. El analista realiza un análisis e interpretación de los requisitos para describir a los diseñadores la solución que él propone desarrollar.
4. Los diseñadores crean un diseño detallado del producto.
5. Los programadores codifican el producto.
6. Se realizan las pruebas y se corrigen los errores detectados, es posible que el cliente solicite algunas modificaciones en este punto por tanto se negocian los cambios y se vuelven a recorrer todas las fases para generar los cambios solicitados.
7. Se entrega el producto algunas veces con manuales que no tienen utilidad.

Los resultados son los siguientes:

- Los interesados, quedan insatisfechos y enojados porque los requisitos se mal interpretaron y sus expectativas no se cumplieron.
- El presupuesto se duplica o triplica por los cambios realizados.
- El tiempo de entrega se extiende debido a los cambios y a la corrección de errores realizados en la etapa de pruebas.
- Como las características del producto no son las deseadas, se ocupa solamente parte del producto y el resto se desecha, en algunas ocasiones el producto se desecha por completo.

El principal problema que se detecta es que el modelo en cascada, es lineal y no es flexible cuando se deben realizar cambios, funciona para generar productos en los cuales se conoce exactamente sus características de manera inicial. Pero en el caso del software no es funcional, ya que por lo general, es un producto nuevo que requiere de creatividad, innovación, personalización, por tanto, durante todo su proceso de producción hay cambios. Según Parnas sucede lo siguiente cuando un cliente desea adquirir software (Parnas, 1986):

- Los usuarios casi nunca saben lo que quieren, no tienen claro cuál será el diseño final del producto, ni la claridad de lo que se quiere obtener.
- Incluso si se pudieran indicar todos los requisitos, hay muchos detalles que sólo se podrán descubrir cuando se esté realizando la ejecución.
- Si se tuvieran todos los detalles, como seres humanos, no se puede dominar tanta complejidad.
- Incluso si se dominará esa complejidad, las fuerzas externas conducen a cambios en los requisitos.

Tomando como referencia los puntos anteriores se determina que los interesados sólo llegan a tener una *visión* (una imagen abstracta e incompleta) de lo que se quiere llegar a obtener (***Figura 5***). Esta visión se debe ir construyendo conforme se va conociendo el producto, por eso es que el modelo en cascada no funciona para la generación de software, no es

Figura 5. El software como una visión en su primera etapa Elaboración propia.

conveniente seguir un modelo lineal.

El uso de modelos predictivos, no ha sido el enfoque más adecuado, ya que la mayoría de los proyectos relacionados con el desarrollo de software siguen fracasando, según el estudio denominado *Manifiesto CHAOS 2013,* realizado por la consultoría Standish Group (Standish, 2013), indica que el 18% de los proyectos son cancelados (son suspendidos antes de ser finalizados o el software nunca fue utilizado), mientras que el 43% son modificados (se entregaron fuera de tiempo, se salieron del presupuesto y/o no se cumplió con las expectativas del cliente), y sólo el 39% fueron exitosos (se entregaron en tiempo, dentro del presupuesto y se cumplió con las expectativas del cliente). El mismo estudio muestra que el 74% de los proyectos se excedieron en el tiempo establecido, 59% excedieron el presupuesto y en promedio sólo se cumplió con el 69% de las características solicitadas.

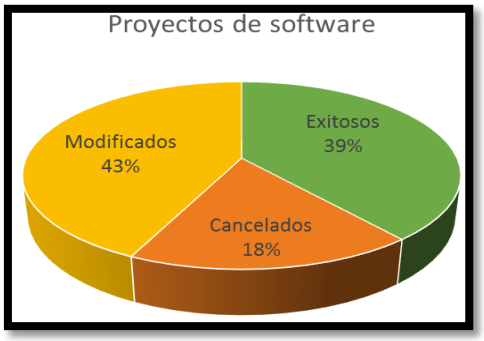


Figura 6. Estadísticas sobre los proyectos de software

Para determinar los problemas que se presentan cuando se desarrolla software, (Nizam & Sahibuddin, 2011) realizaron un análisis, donde concluyeron que existen 26 problemas frecuentes, los cuales se agrupan en tres categorías:

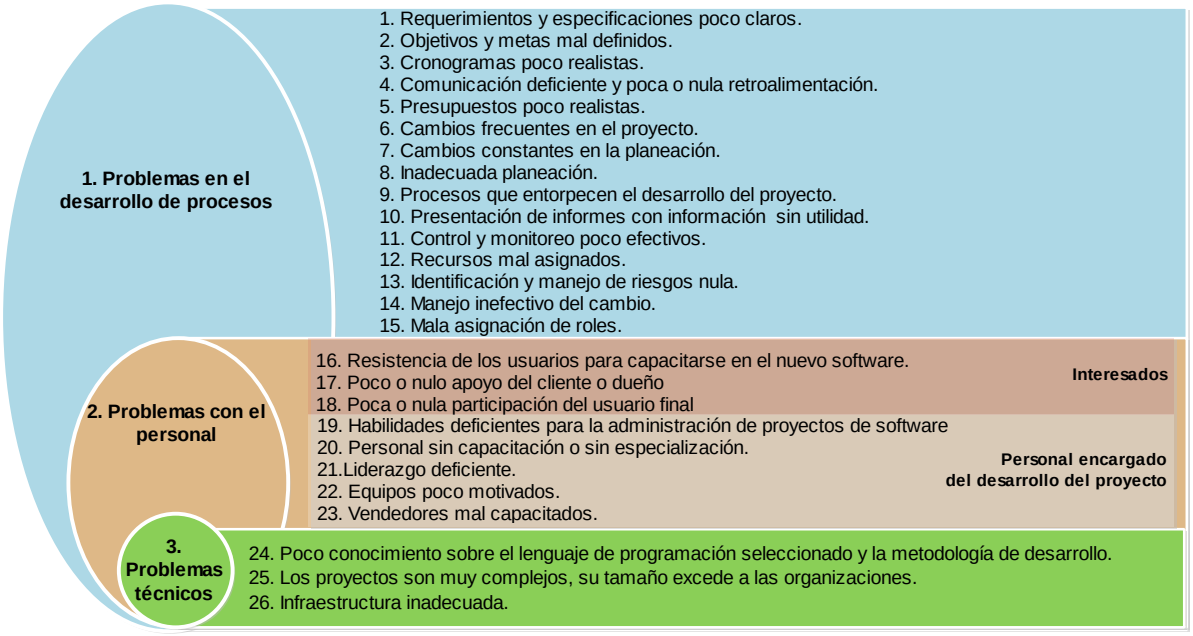


Figura 7. Principales problemas en el desarrollo de proyectos de software. Elaboración propia.

1. ***Problemas relacionados con la planeación y el desarrollo de procesos***: si no existen procesos adecuados y la planeación no está sistematizada se obtiene como resultado:
    a. Insatisfacción de los clientes con el producto final, debido a que el software entregado no cumple con las expectativas y no cumple las funciones para las cuales se pretendía utilizar.

b. Pérdida de recursos financieros, los productos mal elaborados pueden generar, retraso en su entrega y productos de mala calidad, lo cual impacta en las finanzas, ya que se eleva el costo del producto o se pierde el dinero totalmente debido a la cancelación del proyecto.

c. Entradas basura, salidas basura, si los procesos son inadecuados cuando se realiza la recolección y el análisis de requisitos, el proyecto recibe como entradas información que no es útil y se generan dos tipos de salida (***FIGURA 8***):

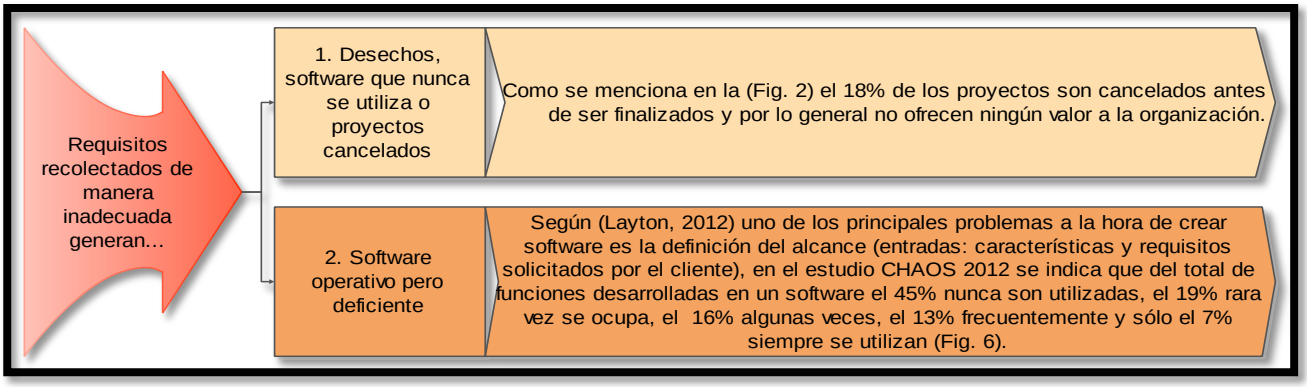


Figura 8. Entradas basura salidas basura. Elaboración propia.

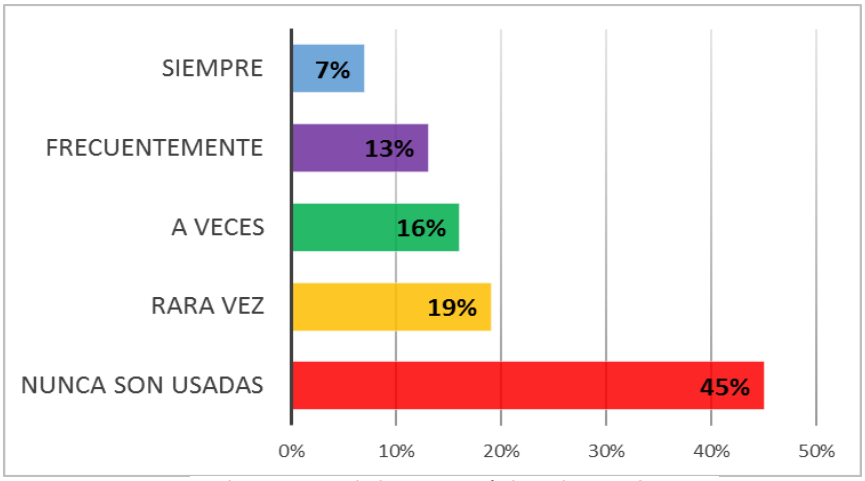


Figura 9. Uso de las características de un software

d. Inadecuada competencia (interna y externa):

i. Competidores internos: son todos los recursos (humanos y/o materiales) que deben ser compartidos para la elaboración de dos o más proyectos en una misma empresa, en este caso la deficiencia en los procesos hace que los recursos se desperdicien y por tanto la empresa tenga perdidas económicas o de clientes.

ii. Competidores externos: son todas las empresas contra las que tiene que competir una organización para ganar un cliente. Los procesos deficientes limitan la participación de la empresa en el mercado.

2. ***Problemas relacionados con el personal*:** aunque los procesos estén bien diseñados y la planeación sea adecuada si el personal no tiene las habilidades necesarias y/o no está capacitado, éste presentará problemas en aplicar los procesos de planeación y generación del producto, afectando, entre otras cosas, a las entradas y las salidas de los procesos, ya que la información proporcionada por el cliente puede ser interpretada de manera inadecuada por el personal y generar un producto que puede ser un desecho o que no cumple con las funciones para lo cual estaba planeado.

3. ***Problemas relacionados con la parte técnica y la infraestructura*:** si no se cuenta con la infraestructura necesaria para la elaboración del producto esto generará problemas de incumplimiento en las expectativas del cliente.

Como se puede observar la mayor parte de los problemas se presentan por la falta de sistematización y establecimiento de procesos. Esto es evidente debido a que el modelo en cascada basa sus etapas en la recolección de requerimientos, análisis, diseño, codificación, pruebas e implementación, y se olvida del resto de los elementos que intervienen en el desarrollo de software, ignora la gestión o administración del proyecto que es la encargada de planear, organizar, controlar y dirigir la creación del producto.

Tomando como base la información anterior, y realizando un análisis sobre los elementos básicos que intervienen cuando se desarrolla software (***FIGURA 10***), se puede concluir lo siguiente:

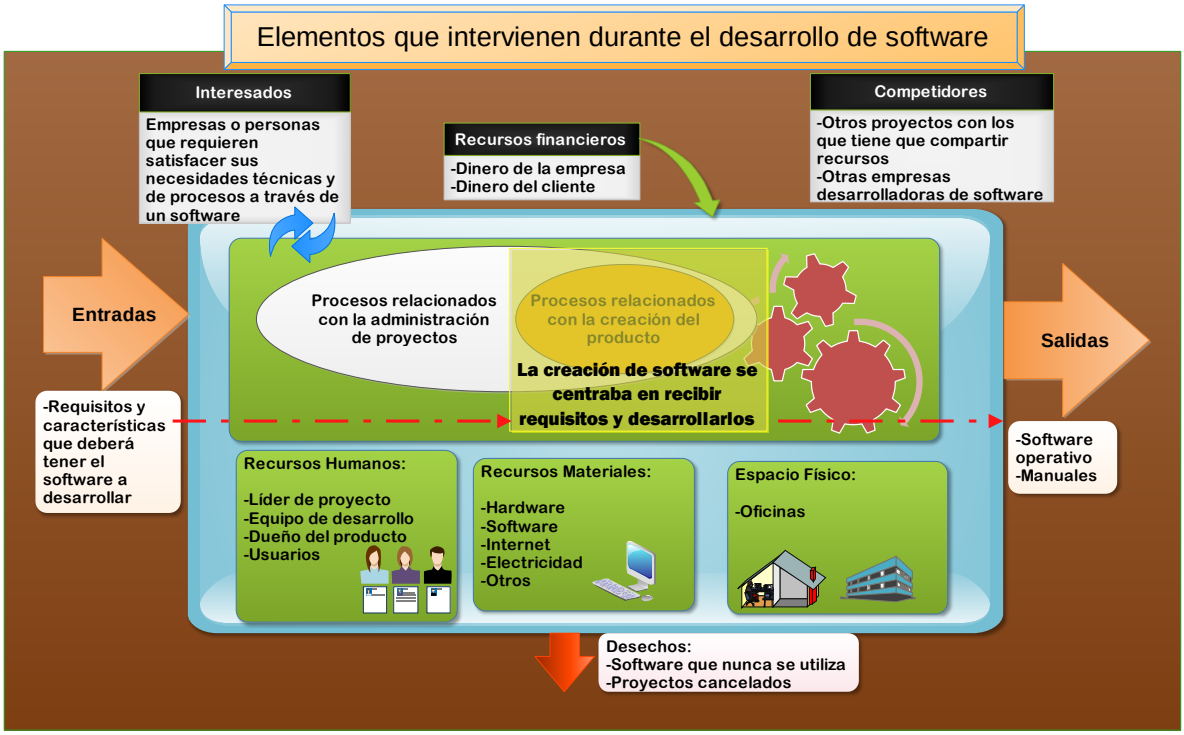


Figura 10. Elementos base para el desarrollo de software. Elaboración propia.

Las empresas desarrolladoras de software no establecen procesos adecuados para generar software de calidad, centran sus esfuerzos en el desarrollo del producto (codificar y corregir). Las siguientes estadísticas confirman esto:

(Jiménez & Orantes, 2012) indican:

- El 87% de las empresas no utilizan ningún modelo de gestión y aseguramiento de la calidad, por tanto, no verifican que el software que se desarrolla sea útil al cliente, Esto es congruente con las estadísticas (***FIGURA 9***), que indican que el 45% de las características desarrolladas nunca se utilizan.
- Del 13% de las empresas que utilizan algún marco de trabajo para la gestión de proyectos y buenas prácticas para el desarrollo de software, el 37% utiliza RUP o UP, el 26% XP, el 16% SCRUM.
- 78% utilizan lluvia de ideas para la recopilación de requerimientos.
- El 82% de las empresas tienen de 1 a 10 trabajadores.
- El tiempo de desarrollo de los proyectos, el 20% de un mes, el 47% es de 2 a 3 meses, el 9% de 4 a 7 meses, el 22% de 8 a 12 meses y el 2% más de un año.

(Gutiérrez & Gutiérrez, 2007) obtuvieron los siguientes resultados:

- El 71% de las empresas utiliza un proceso o metodología para gestionar proyectos de los cuales:
    - En primer lugar, prácticas y técnicas basadas en experiencia propia.
    - En segundo lugar, procesos ágiles, XP, SCRUM o RUP.
    - Tercer lugar, modelos y normas establecidas, CMM, ISO, PMBOK.
- 29% de las empresas piensan que el desarrollo de software es un proceso creativo que no puede seguir un método.
- Modelos para la administración de la organización 71% ninguno, 22% CMMI, 6% MoProsoft, 1% ISO 12207.

Las debilidades en la industria del software se dividen en dos partes (González Bañales, 2006):

1. Procesos para la administración de la organización.
    - Los procesos no aseguran el cumplimiento de los requisitos.
    - No se recolectan y analizan los datos relacionados al producto (procesos de medición).
    - No se cuenta con una cultura organizacional que tenga una visión, una misión y valores.
    - Las habilidades y conocimientos del personal son precarias.

2. Procesos de desarrollo de software.
    - Gestión del riesgo: no se identifica, ni se mitiga continuamente el riesgo del proyecto a lo largo de su ciclo de vida.
    - Administración de la configuración: no se mantiene la integridad del producto, no se asegura que en el producto final estén todos los componentes de la versión apropiada.
    - Verificación y validación de requerimientos: no se confirma que cada producto de trabajo y/o servicio de software refleje apropiadamente la especificación establecida, el producto llega a no ser utilizado o aceptado.

De la información anterior se puede concluir lo siguiente:

- El modelo en casada no es el adecuado para la creación del producto.
- No se consideran todos los elementos que intervienen durante el desarrollo de software, lo cual da como resultado una visón parcial e incompleto del problema que se debe resolver.
- Las empresas no utilizan métodos para gestionar los procesos de la organización, en otras palabras, no cuentan con procesos sistematizados ya probados (mejores prácticas[4]) que permitan generar productos de calidad.

1.2 Identificación del problema

Como ya se observó existen dos problemas fundamentales dentro del desarrollo de software:

1. El modelo en cascada no es útil para crear software, debido principalmente a que éste tiene las siguientes características (Pressman, 2010):

    a. *Aunque la industria se mueve hacia la construcción basada en componentes, la mayor parte del software se construye para un uso individualizado.* Los clientes por lo general sólo tienen una visión sobre lo que desean obtener, por tanto, es difícil determinar desde un principio todas las características, esto implica que durante el desarrollo del producto existirán múltiples cambios. La siguiente gráfica representa el costo al realizar cambios durante la utilización del modelo en cascada.

[4] Una práctica es un método o técnica utilizada para llevar a cabo una parte de un proceso y describe cómo se realiza. Las mejores prácticas son aquellas técnicas o métodos que permiten incrementar la satisfacción del cliente al incorporar su uso en los procesos.

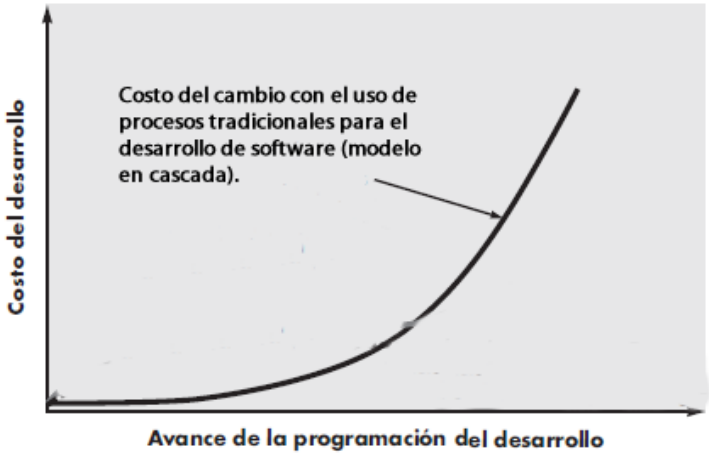


Figura 11 Costo de los cambios en función del tiempo transcurrido en el desarrollo

El modelo en cascada no está diseñado para desarrollar productos que cambian de manera radical durante todo el proceso de construcción.

b. *El software se desarrolla o modifica con intelecto; no se manufactura en el sentido clásico.* El modelo en cascada está diseñado para desarrollar productos donde las características son claras desde el principio. En el caso del software, se crean diferentes versiones[5] antes de obtener el producto final. Al ser el modelo en cascada lineal, trata de evitar el cambio, no permite manejar diferentes versiones, ya que los costos y el tiempo se incrementan de manera significativa.

La mayor parte de las empresas tiene problemas con la gestión y establecimiento de procesos. Lo que provoca el fracaso de los proyectos de desarrollo de software.

1.3. Alternativas de solución

Una de las alternativas al modelo en cascada, que surgió como resultado de la problemática anterior es el *modelo Iterativo e Incremental.*

Cuando se desarrolla software con este modelo, el proyecto se planifica en diversos bloques temporales llamados iteraciones, los cuales se pueden entender como *miniproyectos* (generalmente con una duración de 2 a 4 semanas), en todas las iteraciones se repite un proceso de trabajo similar (de ahí el nombre “iterativo”) para proporcionar un resultado completo sobre el producto final, de manera que el cliente pueda obtener los beneficios del proyecto de forma incremental.

[5] Las versiones por las que pasa un software son las siguientes: *alpha*, esta versión es utilizada para encontrar los problemas y errores que pueda tener el sistema; *beta*, muestra cómo se verá la versión final, se utiliza para depurar fallos, los cuales suelen ser menos graves que en la versión anterior; *Release Candidate (Candidata a definitiva)*, la versión está completa y prácticamente será el mismo que en la versión definitiva, a menos que aparezca algún error no detectado en las pruebas anteriores; *Versión definitiva:* no tiene fallos, es la versión que se utilizará de manera general.

En cada iteración el *equipo evoluciona* el producto (hace una entrega incremental) a partir de los resultados completados en las iteraciones anteriores, añadiendo nuevos objetivos/requisitos o mejorando los que ya fueron completados. Un aspecto fundamental para guiar el desarrollo iterativo e incremental es la priorización de los objetivos/requisitos en función del valor que aportan al cliente.

El desarrollo iterativo e incremental fue ideado por Harlan Mills en 1980. Surgió como un enfoque para reducir la repetición de trabajo en el proceso de desarrollo y dar oportunidad de retrasar la toma de decisiones en los requisitos hasta adquirir experiencia con el sistema.

Etapas del modelo iterativo e incremental:

1. Recolección de los requisitos, se genera una lista con todas las características que el cliente desea satisfacer con la elaboración del software.
2. De la lista de requisitos, el cliente determina cuáles son las características que tienen mayor prioridad para su elaboración.
3. Los requisitos seleccionados pasan por una serie de fases para crear el primer incremento:
    - Análisis de los requisitos seleccionados.
    - Diseño.
    - Codificación.
    - Pruebas/integración.
    - Implementación.
    - Documentación.
4. Durante la ejecución de la iteración no se permite la realización de cambios en los requisitos seleccionados.
5. Al final de cada iteración se entrega un incremento (una versión del producto completamente funcional).
6. El cliente evalúa la versión y realiza las sugerencias sobre las correcciones que desea realizar y selecciona el siguiente grupo de requerimientos para desarrollar las nuevas funcionalidades (estos requisitos se toman de la lista elaborada en el punto 1).
7. Se repiten los pasos 2 a 6 hasta llegar al producto deseado.

Este modelo es útil para el desarrollo de software, ya que no es necesario especificar a priori “todos” los requisitos del software, sino que el proceso ayudará a ir descubriendo paso a paso los requisitos a partir de cada nueva entrega.

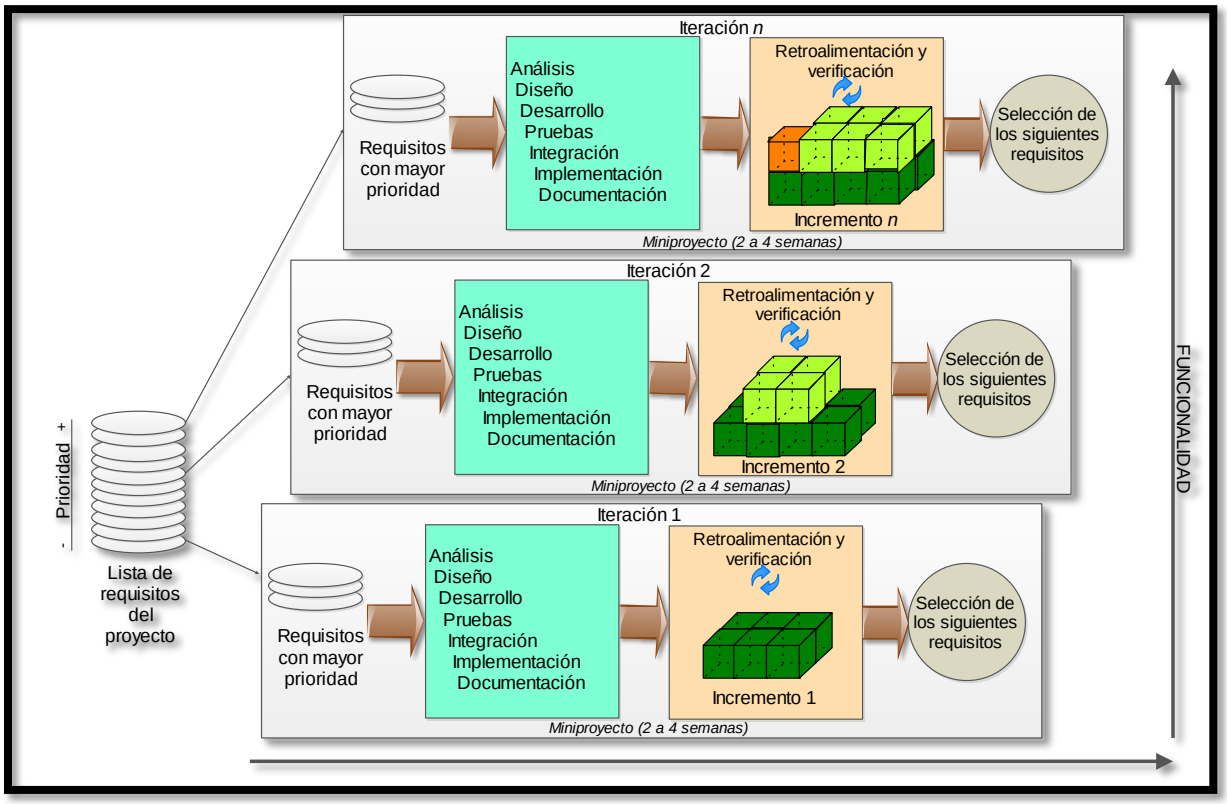


Figura 12. Modelo iterativo e incremental. Elaboración propia.

El desarrollo iterativo e incremental (DII), se basa en construir el producto conforme se va conociendo, se van integrando nuevas características y ajustes con cada entrega. A diferencia del modelo en cascada, esta alternativa al tener periodos cortos de entrega, permite identificar errores de manera temprana y realizar cambios a un menor costo.

En la siguiente grafica se puede observar una comparación entre el costo de realizar cambios utilizando el modelo iterativo e incremental contra el modelo en cascada (Pressman, 2010).

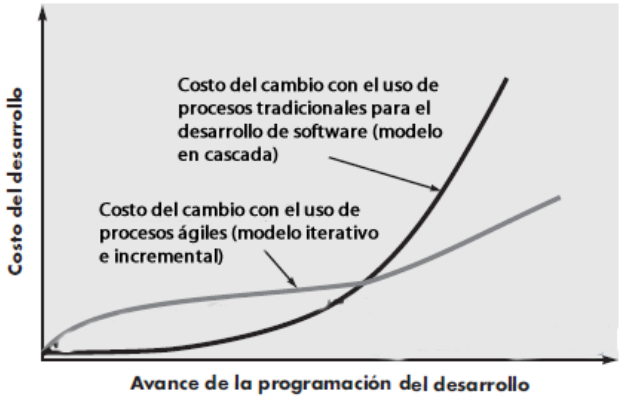


Figura 13. Costo del cambio del modelo iterativo incremental vs modelo en cascada.

También el riesgo disminuye con el uso del modelo iterativo e incremental, como se muestra en la siguiente gráfica (***FIGURA 14***):

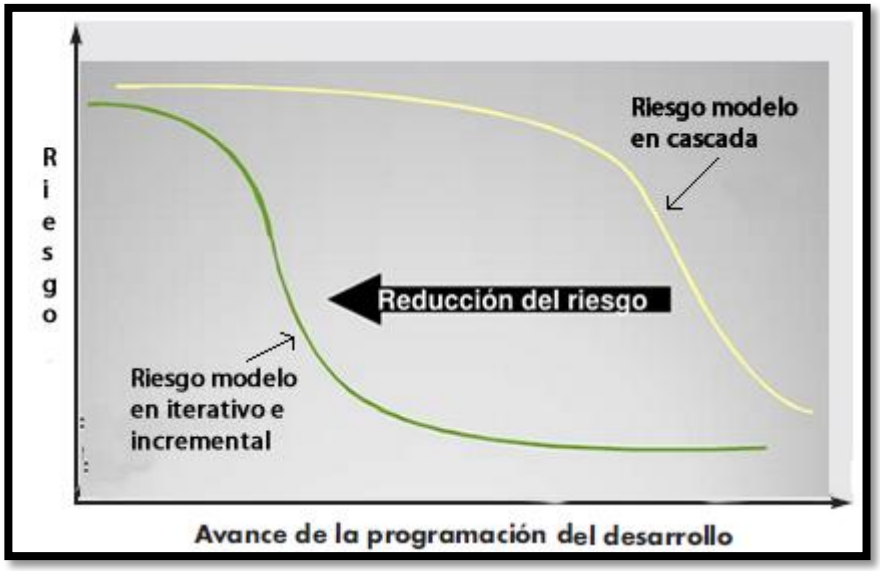


Figura 14. Riesgo en el modelo iterativo incremental vs modelo en cascada.

Al igual que la propuesta del modelo en cascada, el Desarrollo Iterativo Incremental (DII) inicio en 1970, el problema de la popularidad de uno sobre el otro radicó en la difusión que se le dio a cada uno, las herramientas del DII surgieron dentro de una empresa privada (IBM), mientras que el modelo en cascada fue difundido de manera masiva por la publicación del Dr. Royce y por la adopción del Departamento de los Estados Unidos como norma para el desarrollo de software.

Existen distintas marcos de trabajo basados en el modelo iterativo e incremental entre las que se encuentran:

- **Lean Software Development (LD):** El término de Desarrollo de Software Lean (LD) tiene origen en un libro del mismo nombre, escrito por Mary Poppendieck y Tom Poppendieck. El libro presenta los principios Lean adaptados al desarrollo de software, así como un conjunto de 22 instrumentos y herramientas.

- **Metodologías Crystal:** El nombre de metodologías Crystal viene de que cada proyecto de software puede caracterizarse según dos dimensiones, tamaño y criticidad, al igual que los minerales se caracterizan por dos dimensiones, color y dureza. Esta es una de las bases de las metodologías Crystal: hay una metodología para cada proyecto, por ejemplo Crystal Clear (3 a 8 personas, es decir proyectos pequeños) y Crystal Orange (25 a 50 personas). Crystal Clear es la única que se ha desarrollado por completo (Cockburn, 2004).

- **Proceso Unificado (UP):** Es un marco de desarrollo iterativo e incremental que se caracteriza por estar dirigido por casos de uso, y utilizar cuatro fases para el desarrollo de

software: inicio, elaboración, construcción, transición (Jacobson, Booch, & Rumbaugh, 1999).

- **Scrum:** Es un marco de trabajo (framework) para el desarrollo de proyectos complejos, 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 (Schwaber & Sutherland, 2003).

- **Extreme Programming (XP):** Es un estilo para el desarrollo de software centrado en la aplicación de prácticas de programación y trabajo en equipo. Se basa en cinco valores: comunicación, simplicidad, retroalimentación, coraje y respeto (XP, 2014).

  XP promueve las siguientes buenas prácticas: *trabajo en equipo*, todos forman parte del equipo, incluidos el cliente y el responsable del proyecto; *planeación continua,* basada en historias de usuario, y priorización; *revisión del cliente*, propone sus propias pruebas para validar las versiones de cada iteración; v*ersiones pequeñas (incrementos)*, deben ser versiones que ofrezcan algo útil al usuario final y no trozos de código que no tengan sentido para el usuario; *diseño simple,* hacer siempre lo mínimo imprescindible de la forma más sencilla posible; *programación en parejas*; d*esarrollo guiado por las pruebas automatizadas*, cuantas más pruebas se hagan, mejor; *mejora del diseño*: mientras se codifica, debe mejorarse el código; *integración continua*, deben tenerse siempre un ejecutable del proyecto en funcionamiento y en cuanto se finaliza una nueva pequeña funcionalidad, debe integrarse y probarse; *el código es de todos,* cualquiera puede modificar el código; *normas de codificación,* debe haber un estilo común de codificación (no importa cuál), de forma que parezca que ha sido realizado por una única persona; *metáforas*, hay que buscar frases o nombres que definan cómo funcionan las distintas partes del programa, de forma que sólo con leer los nombres se pueda identificar de que se trata el programa. Un ejemplo claro es el "recolector de basura" de java y; *ritmo sostenible*: se debe trabajar a un ritmo que se pueda mantener indefinidamente. Esto quiere decir que no debe haber días muertos en que no se sabe qué hacer, ya que esto ocasiona que posteriormente se tenga que trabajar horas extra. Al tener claro semana a semana lo que debe hacerse, hay que trabajar duro en ello para conseguir el objetivo.

- **Método para el Desarrollo de Sistemas Dinámicos (DSMD):** Este método fue desarrollado como un marco de trabajo para aplicar el Desarrollo Rápido de Aplicaciones (RAD). La idea fundamental detrás de DSDM es que en lugar de fijar los requisitos del producto y, a continuación, predecir el tiempo y los recursos para alcanzar la funcionalidad preestablecida, es preferible fijar el tiempo, los recursos, luego ajustar la funcionalidad y la cantidad de requisitos que se requieran.

DSDM tiene 5 fases, las dos primeras fases son secuenciales y se realizan una sola vez, las últimas tres fases son iterativas e incrementales (Abrahamsson, Salo, & Ronkainen, 2002): *1)estudio de viabilidad,* se determina si es viable desarrollar el proyecto; *2)estudio de la empresa,* son analizadas las necesidades de la empresa con respecto a la tecnología que necesita implementar*; 3)iteración del modelo funcional*: define la forma en que se trabajará con cada iteración, muestra cómo se planificarán los contenidos para cada una. Se crea la lista de funciones priorizadas. Se identifican los requisitos no funcionales y se indica que documentos se obtendrán en cada iteración; *4) diseñar y construir la iteración*: cada iteración produce un prototipo funcional del sistema, el cual ha sido diseñado, codificado y probado, el prototipo debe cumplir con los requisitos mínimos necesarios, los usuarios los revisan y realizan la retroalimentación; *5) implementación*, es la transferencia del entorno de desarrollo al entorno real. Se capacita a los usuarios y se entregan los manuales de usuario.

Según la 8th anual state of agile survey 2014, indica que de las 100 empresas encuestadas (que utilizan como base principal el modelo iterativo e incremental), el 73% utilizan como marco de trabajo Scrum con algunas variantes, por ejemplo, Scrum combinado con el tablero Kanban o con prácticas de XP.

El modelo iterativo e incremental y los marcos de trabajo que lo toman como base, han hecho más eficiente y eficaz el desarrollo de software, ya que los cambios durante el desarrollo del producto se pueden manejar de mejor manera, disminuyendo el riesgo de que el proyecto fracase, aunque este modelo tiene una debilidad, la cual radica en que éste se centra en recibir requisitos, priorizarlos, e iniciar con las iteraciones y la entrega de incrementos, en otras palabras se centra en la ejecución y desarrollo del producto, olvidando los procesos de gestión (planear, controlar, dirigir y coordinar).

1.4 Propuesta de solución

Como se indicó en el párrafo anterior el modelo iterativo e incremental se ocupa solamente de la ejecución y desarrollo del producto, dejando de lado los procesos de gestión. Estos procesos pueden definirse por medio de la planeación. Planear según (Ackoff R. , 2002) es diseñar un futuro deseado así como los medios efectivos para lograrlo. Es un proceso que implica tomar y evaluar decisiones interrelacionadas antes de que sea necesaria la acción.

(Negroe, 2005) propone como base metodológica a la planeación como un proceso de conducción, en donde aplicando las propiedades del enfoque de sistemas, y al paradigma cibernético, identifica dos subsistemas: el conducente o de gestion y el conducido o productivo.

El subsistema productivo es el encargado de cumplir con la misión y los objetivos que tiene el sistema, los cuales pueden consistir en prestar servicios o producir bienes. En este caso particular el subsistema productivo sería la codificación e implementación del software, dentro de una

organización, ya que al producirlo se estaría cumpliendo con los objetivos que el cliente desea satisfacer.

El subsistema de gestión se encarga de las funciones que se requieren para dirigir, organizar y controlar las actividades productivas de los servicios o bienes producidos. Estas funciones hacen posible la operación y desarrollo del subsistema productivo. Este subsistema es el que se analizará y desarrollará como parte de esta investigación.

Dentro del subsistema de gestión (Negroe, 2005) propone una estructura para el proceso de planeación la cual se divide en, diagnóstico (planteamiento del problema), prescripción (solución del problema), instrumentación (planes), planeación de la implementación (ejecución) y control (evaluación y adaptación).

En la estructura del proceso de planeación, el modelo iterativo e incremental se integraría en las etapas de ejecución y control. Eliminando de esta manera la debilidad detectada, en donde el modelo solo se centra en el desarrollo del producto. De esta manera al aplicar un enfoque de sistemas se pueden identificar las etapas que debe seguir para el desarrollo de software y obtener como resultado un método.

Como se mencionó en la problemática, cuando se desarrolla software, el resultado siempre es un producto único, ya que el cliente inicia con una visión sobre lo que desea y esta se va materializando conforme se va construyendo. Al tener esta característica (de ser un producto único) surge la necesidad de desarrollar software como proyectos, el PMBOK (PMI, 2013), define a un proyecto como un esfuerzo temporal emprendido para crear un producto único. Es temporal ya que tiene una fecha de inicio y una fecha de fin.

Desde un enfoque de sistemas, se puede definir a un proyecto como un conjunto interrelacionado de partes que recibe como *entradas* las necesidades de un cliente, las cuales por medio de un proceso de trasformación, genera como salida un producto único (software).

Por tanto, la propuesta consiste en crear un método, que tome como base metodológica a la planeación como proceso de conducción y al modelo iterativo e incremental, para determinar las etapas que debe seguir un equipo de trabajo, para gestionar un proyecto de software y obtener como resultado un producto de calidad, esto quiere decir, software técnicamente bien diseñado y alineado con los objetivos que el cliente busca satisfacer (***FIGURA 15***). En cada etapa del método se integrarán buenas prácticas diseñadas para el desarrollo de software.

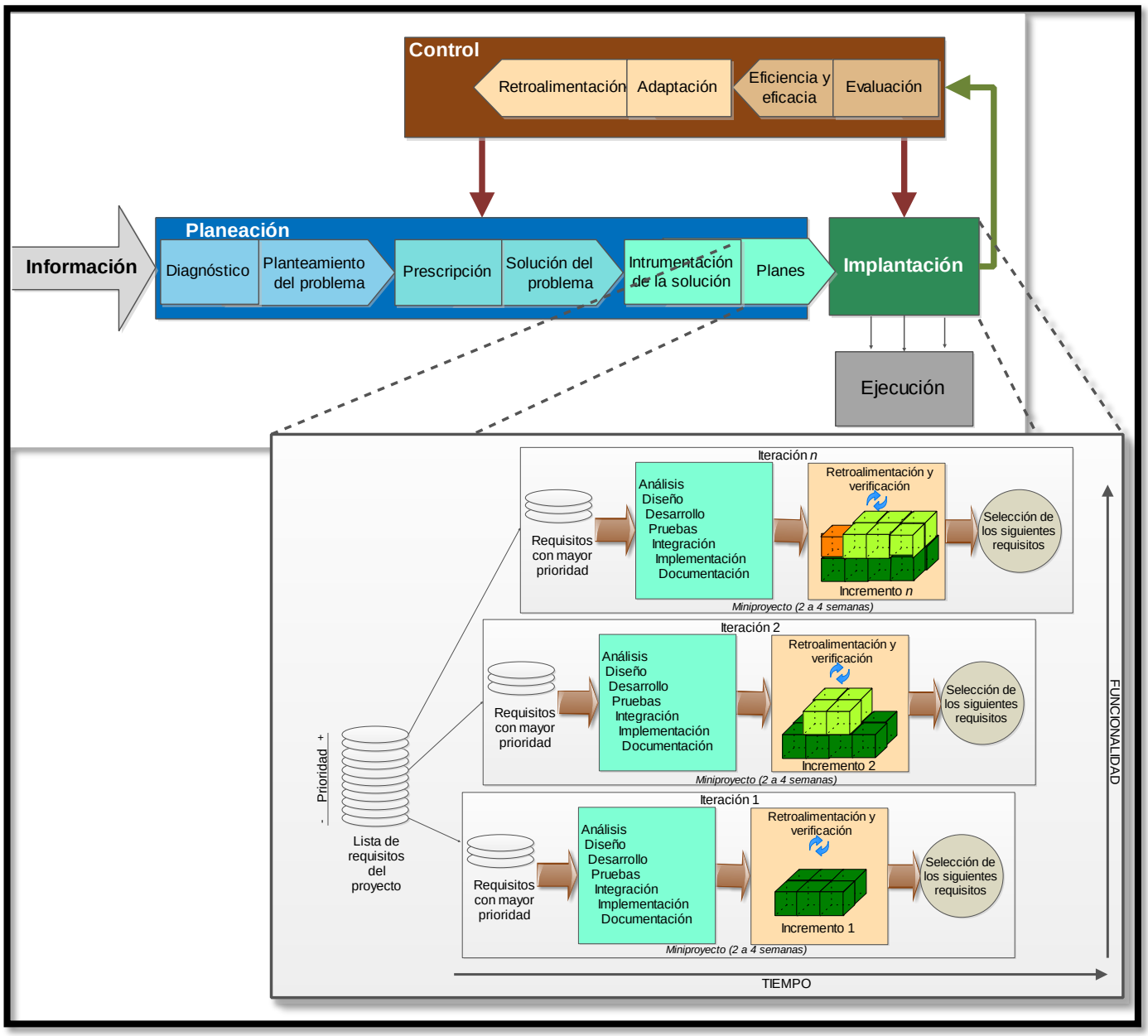


Figura 15. Propuesta de solución.

1.5 Establecimiento de objetivos

1.5.1 Objetivo general

Crear un método basado en la planeación como proceso de conducción y el modelo iterativo e incremental, para sistematizar los pasos que debe seguir un equipo de trabajo para planear, coordinar, dirigir y controlar un proyecto de desarrollo de software.

1.5.2 Objetivos específicos

- Identificar y definir la problemática que enfrenta el proceso general para la gestión de proyectos de software.
- Utilizar las propiedades del enfoque de sistemas para identificar los elementos que intervienen en la gestión de un proyecto de software.
- Realizar un análisis en cada uno de los elementos identificados para determinar sus principales características.
- Identificar los pasos a seguir para administrar un proyecto de software tomando como base a la planeación como proceso de conducción y al modelo iterativo e incremental.
- Proponer técnicas que permitan aplicar el método propuesto.

1.6 Hipótesis

Si los integrantes de un equipo de trabajo utilizan un método (basado en la planeación como proceso de conducción y el modelo iterativo e incremental) que les indique los pasos que debe seguir para planear, coordinar, controlar y dirigir un proyecto de desarrollo de software, tendrá mayor posibilidad de éxito, al obtener como resultado un producto técnicamente bien diseñado y alineado con los objetivos que el cliente busca satisfacer.

1.7 Alcance de la investigación

Se pretende generar una alternativa de solución, que de manera integral permita a un equipo de trabajo, implementar un método que indique las etapas que deben seguir para administrar un proyecto de software, la solución no pretende ser una panacea, sino una herramienta que sea útil para coordinar, planear, dirigir y organizar los diferentes elementos que intervienen cuando se desarrolla software.

Para crear el método se utilizarán las propiedades del enfoque de sistemas, el enfoque cibernético, el modelo iterativo e incremental y la planeación como proceso de conducción. En cada etapa del método se propondrán técnicas y herramientas para su ejecución.

Con esta propuesta se pretende cubrir las necesidades detectadas por (Jiménez & Orantes, 2012):

- Desarrollar un método que permita gestionar proyectos con equipos de trabajo que van de 3 a 10 personas.
- Que sea útil para desarrollar proyectos de corta duración de 2 a 3 meses.
- Incluir procesos que permita realizar un análisis sobre la factibilidad para desarrollar un proyecto.
- Incluir procesos de gestión y aseguramiento de la calidad.

Con el método propuesto, también se pretende mitigar algunos de los problemas (***FIGURA 7***) que frecuentemente se encuentran en el desarrollo de proyectos de software, sobre todo los relacionados con la falta de gestión e implementación de procesos.

1.8 Conclusiones

Desarrollar software implica, creatividad, complejidad y cambio constante. Los equipos de trabajo dentro de las empresas deben estar preparados para crear productos de calidad, para cumplir los objetivos que los clientes pretenden alcanzar.

Por tanto, no es posible que las empresas sigan desarrollando software sin tener claro cuáles son los objetivos del producto que se desean obtener, e implementando modelos lineales e inflexibles respecto al cambio después de sus primeras fases (modelo en cascada). Al no adaptarse este modelo al desarrollo de software, las empresas optan por definir procesos basados en la experiencia (prueba y error), debido a la falta de un proceso adecuado para la gestión de proyectos de software. Como las estadísticas indican, el resultado son productos de baja calidad que terminan siendo desechados o utilizados de manera parcial.

Los encargados para guiar a los equipos de trabajo (administradores y líderes de proyecto), deben tener claro cuáles son los elementos que intervienen en el desarrollo de software, dejar de centrarse solamente en recibir requisitos y codificar.

De los párrafos anteriores, se desprende la necesidad de esta investigación, que tiene como objetivo: elaborar un método para la gestión de proyectos de software, que incluya el diagnóstico de la situación, el estado deseado del sistema que se desea construir, el análisis del entorno donde se implantará el sistema, etc. Para lograr este objetivo, la investigación tomará como marco de referencia el enfoque de sistemas, el paradigma cibernético, la base metodológica de la planeación como un proceso de conducción, el modelo iterativo e incremental y el uso de técnicas que son consideradas mejores prácticas para la gestión y desarrollo de software.

Capítulo 2. El enfoque sistémico en la gestión de proyectos de software

Objetivos del capítulo:

- ✓ Definir los conceptos básicos del enfoque de sistémico y del enfoque cibernético.
- ✓ Analizar la estructura de los sistemas y señalar sus elementos.
- ✓ Aplicar el enfoque de sistemas a la gestión de proyectos de software.
- ✓ Definir la estructura de un proyecto de software como un sistema e identificar sus principales elementos.

2.1 Introducción

A mediados del siglo XX, con la invención de la computadora y el internet, se generó un cambio radical dentro la sociedad, se inició *la revolución del conocimiento,* al lograrse la manipulación lógica de símbolos (datos), mediante la automatización (el remplazo de la mente). La automatización de procesos sustituyó a los procesos tradicionales en incontables áreas, obteniendo una inmensa economía de tiempo y a menudo de costo (Drucker, 2002).

La tecnología terminó pensando ya no en máquinas sueltas, sino de *sistemas*. Por ejemplo, una máquina de vapor, un automóvil o un receptor de radio caían dentro de la competencia de un ingeniero especializado. Pero cuando se trataba de proyectiles, vehículos espaciales, sistemas de información, sólo podían crearse con metodologías heterogéneas: mecánica, electrónica, química, computación, etc. Se hizo necesario, un *enfoque de sistemas* (Bertalanffy, 2006).

La complejidad que presentan los problemas que enfrentan las empresas actualmente, hace que el conocimiento especializado sea insuficiente y cada vez son más extendidas y frecuentes las situaciones problemáticas en cuya solución deben participar diversas disciplinas. Los proyectos de desarrollo de software son un ejemplo, no se requiere solamente de especialistas en computación, intervienen, diseñadores gráficos, matemáticos, pedagogos, sólo por mencionar algunos, esto depende de los requerimientos solicitados por los clientes.

Por tanto, se crea la necesidad de la aplicación de un Enfoque Sistémico a la Gestión de Proyectos de Software, ya que un enfoque parcial no es eficiente. En el presente trabajo, se emplea este enfoque debido a las bases conceptuales que presenta para visualizar a un sistema como un todo, es decir, se interesa por el desempeño total del sistema y el desempeño de cada uno de los elementos que lo componen. El Enfoque de Sistemas parte de la definición precisa del propósito de cada sistema, para luego analizar las alternativas de proceso que permiten alcanzar tal objetivo; examinar en cada una de ellas los elementos que se necesitan para ser controlados; y obtener un

sistema de información que permita mantener la fluidez de la información para la tomada de decisiones (Cordoba, 1979).

2.2 El Enfoque de Sistemas

El primero en enunciar la Teoría General de Sistemas fue el Biólogo Ludwig Von Bertalanffy, en 1951 publicó en la revista Human Biology, un artículo denominado “Teoría General de los Sistemas: un nuevo intento de aproximación a la unidad de las ciencias”, el cual fue considerado como el punto de partida para el desarrollo del Enfoque de Sistémico.

Un sistema es un conjunto de dos o más elementos interrelacionados que interactúan entre sí para lograr un objetivo común. Un sistema satisface tres condiciones (Ackoff R. , 2002):

a) El comportamiento de cada elemento tiene un efecto en el comportamiento del todo.
b) El comportamiento de los elementos y sus efectos sobre el todo son interdependientes.
c) De cualquier manera que se formen subgrupos de los elementos, cada uno tiene un efecto sobre el comportamiento del todo y ninguno tiene un efecto independiente sobre él. Los elementos de un sistema están, a tal punto, conectados, que no pueden formarse subgrupos de ellos que sean independientes.

Subsistema: son todos los elementos contenidos del sistema que están interrelacionados, y que pueden ser considerados como sistemas, dependiendo del objeto de estudio.

Los sistemas poseen **límites**, todos los subsistemas que estén dentro de los límites pertenecen al sistema, todo aquello que esta fuera y que puede influir sobre el comportamiento de éste, es el **medio ambiente**, también llamado **entorno o suprasistema.** Si se observa a cualquier sistema en la naturaleza, se puede notar que no existe en forma aislada, sino que está siempre interrelacionado a un ambiente con el que interactúa, el cual posibilita su existencia y le da sentido, es decir, siempre se tiene al sistema y su medio ambiente (Bertalanffy, 2006).

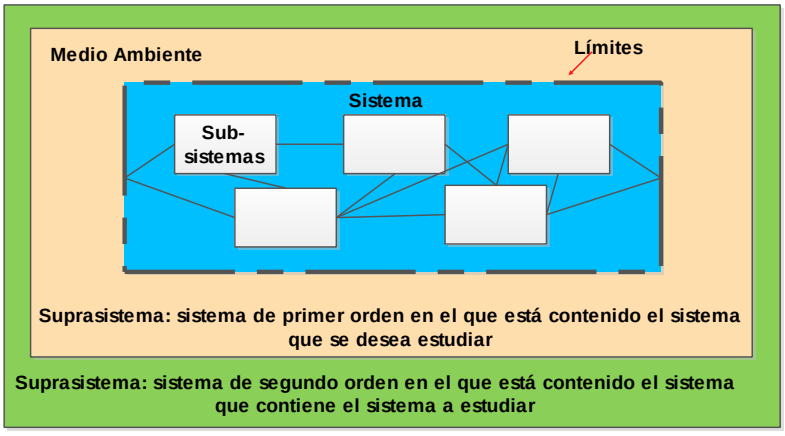


Figura 16. Definición de sistema. Elaboración propia.

Por tanto, un sistema es un todo que no puede dividirse en partes independientes. Cada sistema tiene propiedades que se pierden cuando se separa del sistema, todo sistema posee algunas partes –las esenciales- que ninguna otra de sus partes tiene.
Todo en el universo se puede conceptuar como un sistema y, por tanto, es susceptible de análisis como tal: la célula, el átomo, el cuerpo humano, un ojo, el sol, una empresa, una institución. Cada sistema tiene una función o misión; llámese ser humano, computadora o animal; existe para cumplir objetivos determinados.

En 1954 Kenneth Boulding escribió el artículo titulado "La teoría general de sistemas y la estructura científica". Este artículo es considerado de gran relevancia porque revolucionó el pensamiento científico y administrativo, ya que desarrolló la siguiente clasificación para los sistemas:

- Primer nivel. Estructuración "estática".
- Segundo nivel. De "relojería" o mecánico.
- Tercer nivel. Cibernético o de equilibrio.
- Cuarto nivel. Estructura de autorreproducción.
- Quinto nivel. Genético social.
- Sexto nivel. Animal.
- Séptimo nivel. Humano.
- Octavo nivel. Las estructuras sociales
- Noveno Nivel. Sistemas trascendentes

La clasificación anterior se divide en dos subgrupos (sistemas cerrados o abiertos), según el grado de interacción que tengan con su medio ambiente.

Un sistema *cerrado*: es aquel que se conceptualiza de tal modo que no tienen ninguna interacción con ningún elemento que no está contenido en él. No presentan intercambio con el medio ambiente que los rodea, son herméticos a cualquier influencia ambiental. No reciben ningún recurso externo. Son aquellos sistemas cuyo comportamiento es totalmente determinístico y programado (***FIGURA 17***). Un ejemplo puede ser un reloj. Los primeros tres niveles de la clasificación de Kenneth pertenecen a este tipo de sistemas.

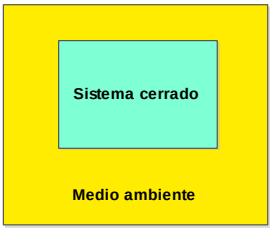


Figura 17. Sistema cerrado

Un sistema abierto: es el que intercambia información, energía o materia con su medio ambiente, es decir aquellos cuyas entradas se originan en el ambiente y cuyas salidas se vuelcan a él (***FIGURA 18***). Los niveles cuarto a séptimo pertenecen a este tipo de sistemas.

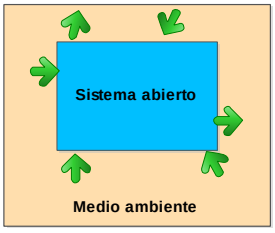


Figura 18. Sistema abierto

Los sistemas abiertos, son estudiados por la **cibernética**. Esta disciplina fue impulsada por Norbert Wiener en 1942, la cual tiene como objetivo "el control y comunicación en los organismos vivos y en las máquinas", por otro lado, el académico A. N. Kolmogorov dice que "la cibernética se ocupa de estudiar los sistemas de cualquier naturaleza capaces de recibir, conservar, transformar información y utilizarla para su propia dirección y regulación".

La cibernética investiga el comportamiento de la relación mutua del sistema con el medio ambiente, definiendo los siguientes conceptos:

- Considera que un sistema abierto puede representarse como un **proceso de transformación**, entrada > transformación > salida (***FIGURA 19***). La representación gráfica se hace por medio del concepto de caja negra[6].

Figura 19. Sistema abierto como modelo de transformación.

- **Entropía**. Es el grado de desorden. En los sistemas abiertos biológicos y sociales la entropía puede ser contenida y ser transformada en **entropía negativa** debido a que el sistema obtiene insumos (entradas) de su medio ambiente, que le permite evitar la decadencia.

- **Homeóstasis**. El concepto de estado estable está estrechamente ligado al de entropía negativa. Un sistema abierto podría llegar a un estado en que el sistema se mantiene en equilibrio dinámico por medio del flujo continuo de materiales, energía e información.

- **Retroalimentación**. El concepto de retroalimentación es importante para entender de qué manera un sistema mantiene un estado estable (homeóstasis). La cibernética se basa en la retroalimentación negativa, la cual es una entrada de información que indica que el sistema debe reajustarse hacia un nuevo estado para estabilizarlo.

[6] En el enfoque de sistemas, una caja negra, es un elemento que es estudiado desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno. En otras palabras, interesa su forma de interactuar con el medio que le rodea.
- Entrada: es todo lo que recibe el sistema de su medio, información, energía, materiales.
- Salida: el resultado final de la operación de un sistema.
- Proceso de transformación: serie de pasos estructurados para lograr un objetivo.

Los conceptos anteriores se pueden comprender mediante el siguiente ejemplo: una empresa que toma del medio ambiente insumos como entrada (entropía), a través de un proceso de trasformación, obtiene como salidas, un producto. Para que el sistema no muera y se mantenga en equilibrio (homeostasis), la empresa vende su producto en el medio ambiente en el que está inmerso, para obtener recursos, conseguir nuevamente insumos, para seguir produciendo. La retroalimentación identifica los posibles problemas que existen y permite a la empresa tomar decisiones para volver a estabilizar el sistema.

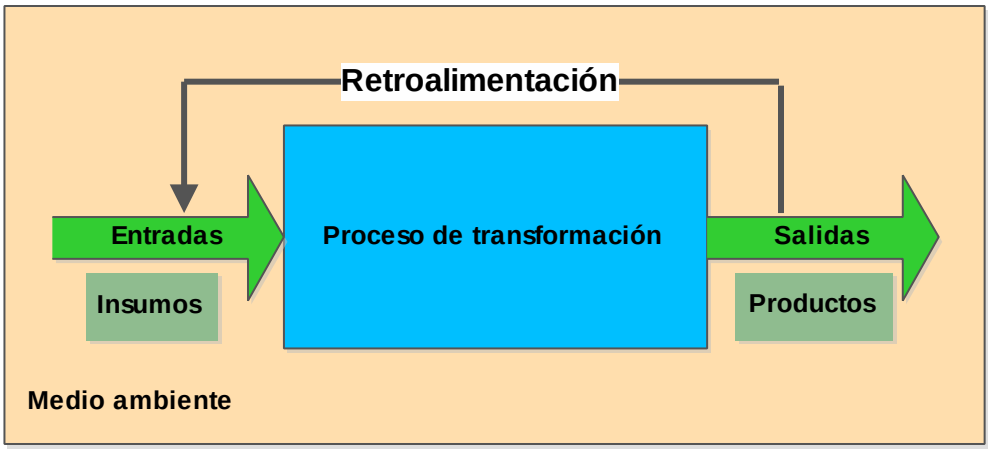


Figura 20. Enfoque cibernético. Elaboración propia.

2.3 Proyecto de Software

Desde el punto de vista del Enfoque Sistémico, "un proyecto se entiende como un sistema abierto y adaptativo, una totalidad cuyos elementos se interaccionan en su conjunto y con su entorno para alcanzar un objetivo o fin previamente establecido" (Olmedo, 2009).

"Todo proyecto es un proceso encaminado a la búsqueda razonada para resolver en el presente, con base en el pasado, pero para el futuro, cierta necesidad individual o social en un entorno específico y en un tiempo determinado" (Olmedo, 2009).

"Un proyecto es un esfuerzo temporal emprendido para crear un producto, servicio o resultado único. Es temporal ya que tiene una fecha de inicio y una fecha de fin", (PMI, 2013).

Si se visualiza a un proyecto con un sistema, éste tendrá cuatro elementos básicos (Hernández, 2006):

1. **Entradas o insumos**: abastecen al sistema de lo necesario para que cumpla su misión; por ejemplo, capital, personal, materia prima.

2. **Proceso de transformación:** es la transformación de los insumos, de acuerdo con ciertos métodos propios, con sistemas que son subsistemas; ejemplo: para el desarrollo de software son los procesos que se deben seguir para para transformar los requisitos que el cliente solicita en software, algunos subsistemas pueden ser: diagnóstico, análisis de requisitos, diseño, etc.

3. **Salidas o producto:** es el resultado del proceso, en este caso se obtiene software.

4. **Retroalimentación**: *retro* proviene del latín y significa hacia atrás. Administrativamente, retroalimentación —o retroinformación— significa recibir la evaluación o aceptación de los productos o servicios por el medio ambiente para corregir procesos; en la práctica, es el análisis de los resultados en relación con la aceptación del usuario, cliente o consumidor respecto de lo que produce el proyecto.

   La retroalimentación proporciona información que indica si todo va acorde a los objetivos y metas, o si deben reajustarse uno o varios procesos. En el desarrollo de software, esta información se obtiene por medio de métricas, ITIL en su versión V3 indica dentro de sus buenas prácticas que deben tomar en cuenta tres tipos de métricas:
   - *Métricas técnicas*: responden a la pregunta ¿el producto, servicio o resultado satisface las necesidades de desempeño que se solicitaron?, estas métricas están relacionadas directamente con los componentes del servicio.
   - *Métricas de servicio*: responde a la pregunta ¿se cumplió con las expectativas del cliente?
   - *Métricas del proyecto*: responde a la pregunta ¿qué elementos son claves para el éxito del proyecto y cómo pueden medirse?

Todo sistema forma parte de otro mayor llamado macrosistema, suprasistema o ecosistema, el cual es un subsistema de un suprasistema. Por ejemplo, un proyecto es un sistema que forma parte de una organización[7], y ésta, a su vez, forma parte de la sociedad, y así sucesivamente.

[7] Una organización: según (Ackoff R. 1992) “es un sistema con algún propósito, el cual es parte de uno o más sistemas con algún propósito en el cual algunas de sus partes (las personas, por ejemplo) tienen sus propios propósitos”.

En otras palabras una organización (Gil, 2010):
- Tiene un propósito definido, cuenta con una visión, una misión, objetivos y metas.
- Tiene una estructura deliberada, por ejemplo puede tener una estructura jerárquica.
- Es un sistema abierto, quiere decir que interactúa con el medio ambiente.

Está compuesto por:
- Medios: son las alternativas con las que cuenta la empresa para llegar a su objetivo, por ejemplo, estrategias, planes, procesos, actividades.
- Recursos: 1) suministros: materiales, aprovisionamientos, energía y servicios; 2) las instalaciones y el equipo; 3) el personal; 4) dinero.

Debido a la interacción que un proyecto tiene con su medio ambiente, se clasifica como un sistema abierto, siendo capaz de modificar su ambiente, de acuerdo con sus necesidades o intereses para alcanzar el objetivo planteado.

Según (PMI, 2013) un proyecto de caracteriza por:

- **Ser temporal:** lo que significa que su ciclo de vida está acotado en un lapso predeterminado, con un inicio y un final totalmente definidos con anterioridad. El rango entre dichos eventos representa la duración de un proyecto. Evidentemente, un proyecto termina cuando se alcanzan los objetivos planteados inicialmente; o bien, cuando dichos objetivos no pueden ser alcanzados y el proyecto tiene que ser cancelado.

- **Dar resultados únicos:** Todo proyecto lleva implícita una propuesta de producción de un bien, servicio o resultado único independientemente del tipo de proyecto, dado que la inversión de recursos asignada es única, al igual que las condiciones del proceso, las condicionantes del bien, servicio o resultado y las circunstancias de su medio ambiente, que siempre serán únicas, particulares e irrepetibles.

- **Es teológico**: todo sistema persigue un fin. En el caso de los proyectos de software existen tres elementos que deben ser identificados de manera clara:

  La visión: describe de manera general lo que se quiere lograr.
  Responde a las preguntas:
  - ✓ *Producto*: ¿Cómo el producto va apoyar a la empresa o estrategia de la organización?, ¿Qué estrategias específicas de la empresa se van a cumplir?, ¿Qué es lo que debe hacer el producto?
  - ✓ *Cliente*: ¿Quién utilizará el producto?
  - ✓ *Necesidad*: ¿Por qué necesita el cliente el producto?, ¿Qué características son fundamentales para el cliente?
  - ✓ *Competencia*: ¿Cómo se compara el producto con productos similares?
  - ✓ *Diferenciación*: ¿Qué diferencias tendrá el producto respecto a la competencia?

  ***Objetivo*:** son logros que se proponen en un plazo determinado, el o los objetivos ayudan a alcanzar la visión.
  Para construir un objetivo se puede seguir el siguiente procedimiento:
  - ✓ El objetivo siempre inicia con un verbo en infinitivo (hacer, desarrollar, trabajar, crear. etc.).
  - ✓ Contestar a las preguntas: ¿Qué?, ¿Cómo?, ¿Para qué?, ¿Quién?, ¿Cuándo

La diferencia entre una empresa y una organización, es que la primera tiene como objetivo principal la obtención de utilidades. El intercambio de productos o servicios por dinero.

(tiempo o periodo)?, ¿Dónde?.

- Ejemplo:
  - Determinar (verbo en infinitivo).
  - Los requisitos para el desarrollo del software X *(¿Qué?)*
  - Mediante un taller de historias de usuario (*¿Cómo?)*
  - Para identificar las necesidades del cliente *(¿para qué?)*
  - Deberán participar el dueño del producto, un usuario y el equipo de desarrollo *(¿Quién?)*
  - La fecha X *(¿Cuándo?)*
  - En la sala de conferencias de la empresa *(¿Dónde?)*

***Metas*:** son logros a corto plazo, ayudan a cumplir con los objetivos. Son descripciones que indican cómo se logrará un objetivo, por tanto son medibles. Siguiendo con el ejemplo algunas metas serían:

- Identificar al dueño del producto y a los usuarios que participarán en el taller de historias de usuario.
- Solicitar la sala de conferencias para la fecha X.
- Determinar el material que se utilizará en el taller y los costos del mismo.

La representación gráfica de un proyecto utilizando el concepto de caja negra quedaría de la siguiente manera (***FIGURA 21)***:

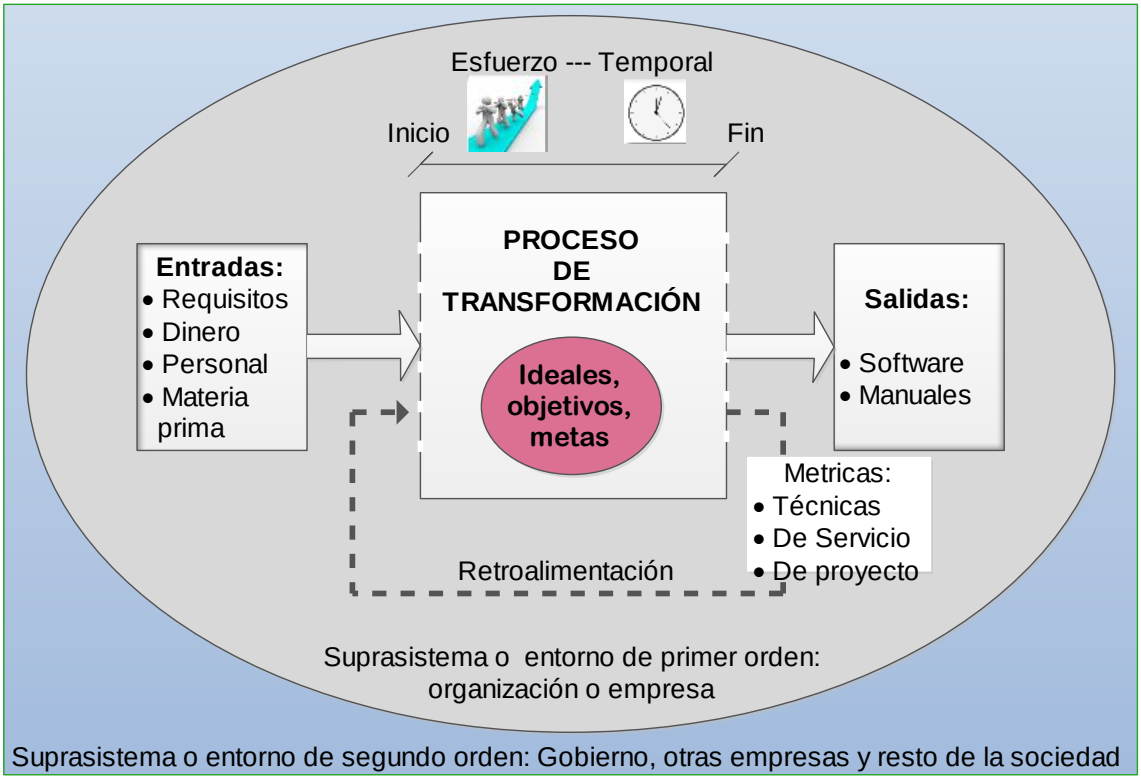


Figura 21. Proyecto como un sistema. Elaboración propia.

2.4 Medio ambiente de un proyecto

El medio ambiente de un proyecto está constituido por todos los elementos externos, sociales, políticos, jurídicos, económicos y físicos, que directa o indirectamente influyen en el desarrollo de un proyecto y después en su operación, afectándolo de manera positiva o negativa.

Un proyecto al ser un sistema abierto adaptativo, puede influir activamente en su medio ambiente, tanto de forma previsible como entrópica, modificando directa o indirectamente y adaptando las condiciones que prevalecían antes de su existencia. Es por ello que en la evaluación de un proyecto, sobre todo en el nivel de factibilidad, se establecen dos escenarios: el primero, considerando solamente las condiciones del medio ambiente, sin la interacción del proyecto; y el segundo, analizando el medio ambiente con el resultado esperado del proyecto de desarrollo determinado (Moreno Jiménez, 2013).

Los principales elementos del medio ambiente que deben considerarse cuando se desarrollar un proyecto de software son:

- **Restricciones:** como ya se mencionó un proyecto es temporal, ya que debe ser desarrollado en un tiempo preestablecido, esto es una restricción[8], ya que si no se inicia o finaliza el proyecto en las fechas acordadas, se corre el riesgo de fracasar.

  Existen muchas restricciones para un proyecto, pero tradicionalmente se utilizan tres variables (costo, tiempo y alcance), como las más relevantes para el éxito o fracaso de un proyecto, estas restricciones, también conocidas como "triple restricción", deben mantenerse balanceadas para que el proyecto pueda seguir adelante ya que si una de ellas se modifica las otras se ven afectadas.

  El PMBOK en su versión 5, indica que además de la triple restricción, se debe considerar la calidad, los riesgos y los recursos como restricciones dentro de un proyecto.

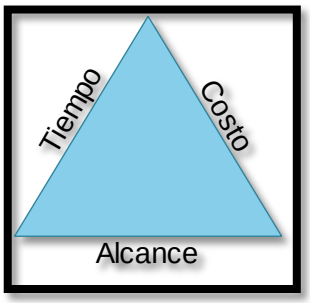


Figura 23. Tripe restricción.

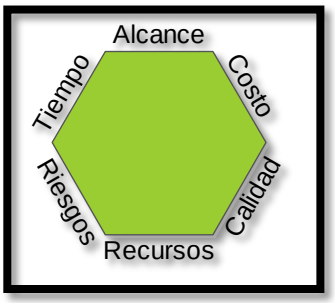


Figura 22. Restricciones según el PMBOK.

[8] Una restricción es cualquier condicionante, limitante, impedimento que se presenta en un proyecto y que además tiene la siguiente particularidad: cualquier cambio que experimente, provocará cambios en otros condicionantes del mismo proyecto.

- *Alcance*: del producto, son todas las características y funciones que definen al producto, servicio o resultado; alcance del proyecto, todo el trabajo necesario para producir los resultados esperados y los procesos utilizados para producirlos.
- *Costo*: es el presupuesto para el proyecto, incluye todos los gastos necesarios para concluir el proyecto.
- *Tiempo*: cantidad de días, meses o años necesarios para terminar el proyecto. Determina cuando inicia y cuando termina el proyecto.
- *Calidad*: la obtención de los resultados esperados del proyecto según las expectativas de los interesados. La calidad debe enfocarse en *la utilidad* y la *garantía* en el caso del desarrollo de software.
- *Recursos*: son todos los suministros: materiales, aprovisionamientos, energía y servicios; las instalaciones y el equipo; y el personal.
- *Riesgos:* son eventos inciertos que, si suceden, tienen un efecto por lo menos en uno de los objetivos del proyecto (tiempo, costo, calidad, alcance, recursos).

El Método de Desarrollo de Sistemas Dinámicos (DSDM) indica que en los proyectos que utilizan el paradigma en cascada fijan el alcance y tratan de estimar los costos y el tiempo, mientras que los proyectos que utilizan el modelo iterativo incremental lo hacen al revés, primero determinan los recursos con los que se cuenta, fijan un tiempo determinado y estiman los requisitos en base a las prioridades del cliente (***FIGURA 24***) (DSDM, 2014).

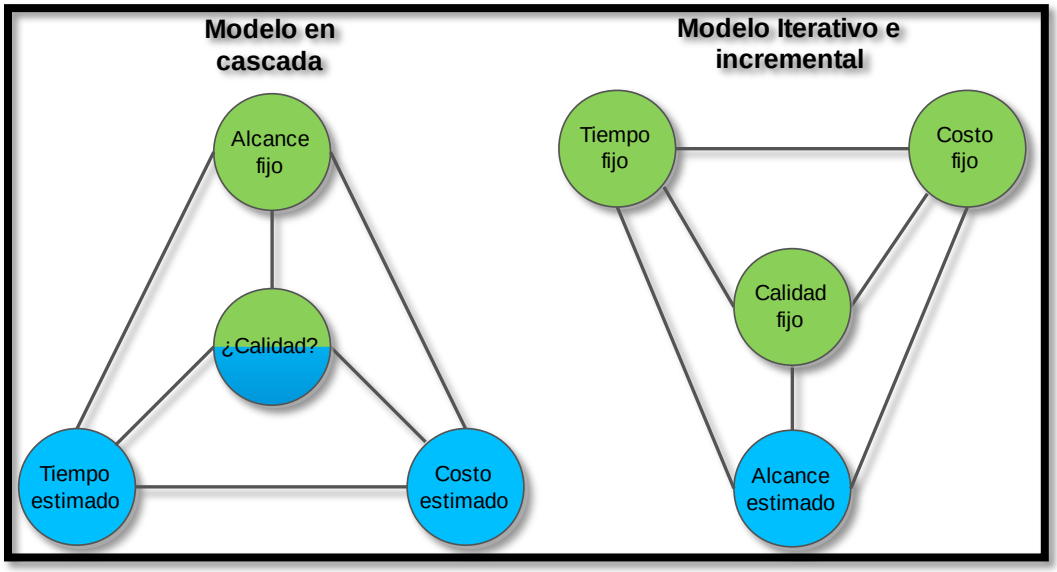


Figura 24. Restricciones fijas y estimadas según el modelo en cascada y el modelo iterativo e incremental

El DSDM no considera al *alcance* como una restricción debido a que aplica el principio de Pareto dentro del proceso de desarrollo de software, indica que el 80% del beneficio que obtendrá el cliente de un software proviene del 20% de los requisitos solicitados, así DSDM comienza implementando primero este 20% de requisitos para cumplir con el 80% de las

necesidades del cliente, *lo que es suficientemente bueno mientras los usuarios estén involucrados en el proceso de desarrollo* y en una posición de asegurar que el 20% restante no causará serias consecuencias a la empresa u organización. Por tanto el alcance puede variar sin que se afecte al resto de las variables.

- **Riesgos**: Un riesgo es algo desconocido que, si se produce, afecta en forma negativa los objetivos del proyecto. El riesgo representa el impacto potencia de todas las amenazas u oportunidades que podrían afectar el logro de los objetivos del proyecto (Lledó, 2013).
  Un riesgo que se ha materializado, se considera como un problema.

Componentes del riesgo	Definición	Ejemplo
Causa / origen	Factor o condición que determina el riesgo.	Obtener un permiso para realizar una modificación en un *servidor*, ya que el acceso al lugar es restringido.
Probabilidad	Todo riesgo tiene una probabilidad de ocurrencia.	10% de que el permiso se retrase.
Impacto	Es el daño que ocasiona a los objetivos del proyecto (tiempo, costo, calidad, alcance, etc.).	Alto, ya sin el permiso no se pueden iniciar las pruebas del producto, lo que retrasaría la entrega.

Tabla 2. Componentes del riesgo.

Como tal, el riesgo que se presenta en el ejemplo de la (***TABLA 2***) es que *el departamento que otorga el permiso, demore más tiempo de lo previsto.*

Cuando se ha detectado un riesgo existen distintas maneras de abordarlo:

- *Mitigar:* esta actividad consiste en reducir a un nivel tolerable el impacto o la probabilidad de un evento de riesgo.
- *Trasferir:* esto implica trasladar el impacto y la responsabilidad del evento de riesgo a un tercero.
- *Aceptar:* es la decisión de no actuar directamente para responder al riesgo.
- *Eliminar:* implica modificar el plan del proyecto para eliminar la amenaza.

- **Proveedores:** son terceras personas encargadas de abastecer a un proyecto del material necesario (bienes o servicios) para que el producto final funcione. Son todos los servicios o materias primas que requieren ser comprados a otras empresas para poder crear el producto principal.

Los proveedores se pueden clasificar de la siguiente manera:

- *Estratégicos:* proporcionan bienes o servicios vitales para el funcionamiento del producto final, son todos los proveedores que brindarán los insumos necesarios para que el software que se desea obtener funcione de manera correcta, se debe manejar

información confidencial, por tanto el trato debe ser directo y se deben tener claros los términos del contrato. Por ejemplo servicios de almacenamiento para bases de datos.

- *Tácticos:* insumos esenciales para que el equipo de trabajo realice sus actividades y logre los objetivos previstos, ejemplo, equipo de cómputo, licencias de software, internet, etc.
- *Operacionales:* insumos básicos que son fáciles de conseguir, por ejemplo, artículos de papelería.

- **Infraestructura:** en un proyecto de software se refiere al lugar (ubicación) en donde trabajará el equipo del proyecto, así como al *layout* o esquema de distribución de los recursos (escritorios, sala de juntas, etc.) dentro de la oficina.

  Los *recursos* se refieren a todo el mobiliario e instalaciones que son necesarias para que el equipo del proyecto realice su trabajo de manera eficaz y eficiente.

  La metodología Crystal Clear desarrollada por (Cockburn, 2004) sugiere que un equipo de trabajo debe cumplir con la propiedad "*osmotic communication*", su traducción al español, sería que todo el equipo se encuentre en una misma ubicación física, para lograr una comunicación cara a cara. Esto según Cockburn permite que, por ejemplo, si una persona realiza una pregunta, otros en la habitación pueden participar en la convesación o continuar con su trabajo.

  La "*osmotic communication*" hace que el costo de las comunicaciones sea baja y la tasa de retroalimentación sea alta, por lo que los errores se corrigen de manera extremadamente rápida y el conocimiento llega a todas las personas. Este tipo de comunicación es recomendado para equipos de trabajo pequeños, ya que si el equipo es muy grande es más difícil mantener el sentido de la "*osmotic communication*".

  El mantener al equipo del proyecto en un mismo espacio físico propicia:

  - La comunicación cara a cara.
  - Resolver problemas en equipo.
  - Ser consciente de lo que otros están trabajando.
  - Pedir ayuda con una tarea.
  - Apoyar a los demás en sus tareas

  Este tipo de comunicación conduce a determinar la forma en la que debe estar organizado el mobiliario y las instalaciones de la oficina. Se recomienda tener un cuarto denominado,

cuarto de proyecto o “sala de guerra”, el cual se define como un lugar físico donde se encuentra todo el equipo del proyecto.

Los escritorios deben estar ubicados de tal manera que las paredes queden libres, tal como se muestra en la (***FIGURA 25***), se recomienda utilizar escritorios y sillas móviles así como laptops y tablets para poder moverse fácilmente.

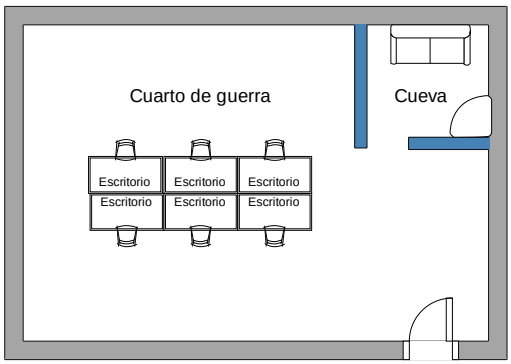


Figura 25. Sala de guerra y cueva. Elaboración propia.

De ser posible se recomienda crear una “cueva”, un lugar aislado donde los miembros del equipo de trabajo puedan tomar breves descansos o alejarse del bullicio del cuarto de guerra.

El propósito de dejar las paredes libres es que estas se puedan utilizar colocando pizarras o si se trata de una construcción nueva, utilizar materiales como pintura especial o lámina para que los muros puedan utilizarse como pizarrones, de esta manera se pueden dibujar los diseños, procesos, etc. o colocar hojas de rotafolio o post-it, con tareas, listas de riesgos, la visión, etc. La finalidad es que la información este a la vista de todo el equipo de trabajo.

Cuando se crean diseños sobre las paredes, es recomendable tener cámaras digitales, o celulares que permitan tomar fotografías, para documentar y tener un respaldo de la información que se ha ido desarrollando.

Figura 26. Ejemplos de salas de guerra.

- **Gobierno**: es prudente determinar que reglamentos, normas, leyes que deben ser cumplidas para el desarrollo de un proyecto.
- **Geografía**: factores que pueden afectar debido a la ubicación del equipo de trabajo y de los clientes.
- **Sindicatos**: organizaciones que cuentan con reglamentos, normas, etc. para la protección de sus trabajadores.
- **Sociedad**: son factores positivos o negativos, donde la sociedad puede influir participando y apoyando un proyecto o impidiendo que se realice.
- **Competidores externos**: este factor afecta sobre todo cuando se concursa para ganar un proyecto, por ejemplo en una licitación, se deben considerar las reglas de operación y al resto de las empresas con las que se competirá.
- **Productos sustitutos**: se debe determinar si el producto que se creará tiene un producto sustituto.

2.5 Cultura organizacional

La cultura organizacional es "la forma característica de pensar y hacer las cosas en una empresa" (Garcia & Dolan, 1997). Por otro lado (Serna, 1997) indica que "la cultura es el resultado de un proceso en el cual los miembros de la organización interactúan en la toma de decisiones para la solución de problemas inspirados en principios, valores, creencias, reglas y procedimientos que comparten y que poco a poco se han incorporado a la empresa".

La cultura organizacional es un elemento del medio ambiente que afecta directamente al proyecto, por tanto es importante definirla dentro de una empresa dedicada al desarrollo de software.

En 2001 un grupo de 17 expertos se reunieron en Utah, para analizar la problemática referente al desarrollo de software. Uno de los resultados obtenidos fue la definición una alternativa para el desarrollo de software denominado *Movimiento Ágil.* Según Jim Highsmith uno de los expertos de la reunión, define *Agile o Agilidad* como, "la capacidad de crear y responder al cambio con el fin de obtener ganancias en un entorno empresarial turbulento".

El *Movimiento Ágil* propone que las organizaciones dedicadas al desarrollo de software deben seguir, 12 principios plasmados en lo que fue denominado *Manifiesto Ágil* (Beck, y otros, 2014):

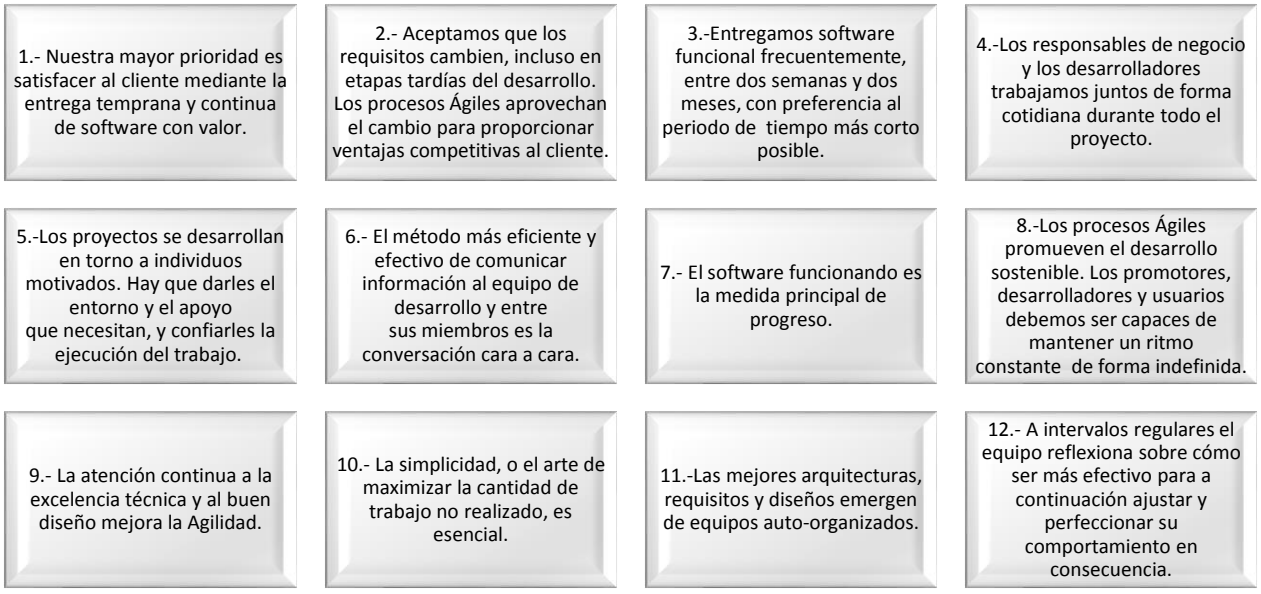


Figura 27. Manifiesto ágil

Además de los principios el manifiesto también propone cuatro valores:

1. A los **individuos y su interacción**, por encima de los procesos y las herramientas.
2. El **software que funciona**, por encima de la documentación exhaustiva.
3. La **colaboración con el cliente**, por encima de la negociación contractual.
4. La **respuesta al cambio**, por encima del seguimiento de un plan.

La organización debe tener las siguientes características según (Craig, 2004):

Características de la organización
1. Funcionalidad cruzada:
Definición: voluntad y capacidad para trabajar en diferentes tipos de tareas del desarrollo del producto. *Características*: • Los títulos no tienen cabida en un equipo ágil, las capacidades y habilidades son las que importan. • Trabajar para ampliar conocimientos. • No trabajar solo en tareas que ya conoce. • Tratar de aprender algo nuevo en cada iteración. • Ayudar a otros que se han topado con un obstáculo. • Ayudar a alguien con algún problema es una buena manera de aprender. • Ser flexible, esto ayuda a equilibrar la carga de trabajo.
2. Auto-organización:
Definición: capacidad y responsabilidad de cada miembro del equipo, para determinar cómo se realizará el trabajo para elaborar el servicio. *Características que debe tener cada miembro del equipo de desarrollo*:

• Comprometerse con las metas que cada miembro del equipo ha seleccionado desarrollar. • Identificar sus tareas de manera clara • Estimar el esfuerzo necesario para los requisitos y tareas relacionadas • Enfocarse en la comunicación cara a cara. • Ser claro. • Participar activamente • Los miembros del equipo deciden qué hacer, cuándo y cómo.
3.- Autogestión:
Definición: que sean las personas quienes puedan cumplir sus objetivos gracias a sus propios esfuerzos y decisiones. *Características:* • Permitir el liderazgo en todos los miembros. Cada miembro podrá dirigir en diferentes tareas, esto dependerá de la experiencia y habilidades de cada miembro. • Confiar en los procesos y herramientas ágiles. • Informar con regularidad los avances. • Crear un acuerdo de equipo sobre cómo se trabajará. • Crear un documento donde cada quien expone las expectativas que tiene. • Averiguar lo que le funciona al equipo
4.- Comportamiento maduro:
• Tomar la iniciativa, en lugar de esperar a que alguien le diga que hacer • Los éxitos y fracasos son del equipo, los problemas que surjan se deben resolver en equipo. Cuando se tiene éxito es porque el grupo cumplió • Confiar en la capacidad de tomar buenas decisiones
5.- Equipos de desarrollo de tamaño limitado
• Se recomienda tener como mínimo tres miembros en el Equipo de Desarrollo ya que en equipos más pequeños se pueden encontrar limitantes en cuanto a las habilidades necesarias para el desarrollo del producto. • Tener más de nueve miembros en el equipo, requiere demasiada coordinación. Grupos grandes generan demasiada complejidad y tienden a formar subgrupos. • Los roles de Dueño de Producto y Líder de Proyecto no cuentan en el cálculo del tamaño del equipo a menos que también estén contribuyendo en el desarrollo del producto.

Tabla 3. Características de la organización.

2.6 Entidades directamente involucradas en un proyecto de software

Son todas las personas que se verán directamente afectadas por el resultado del proyecto. Dentro de un proyecto se debe identificar a los interesados clave, a los cuales se les reconocerá por el rol[9] asignado, estas personas conformarán el *equipo de trabajo* y estarán encargadas de participar en el desarrollo del proyecto durante todo su ciclo de vida.

Según (Shwaber & Sutherland, 2013) los roles que debe tener un equipo para desarrollar un proyecto de software deben ser:

Dueño del producto
Definición:
• Es un experto en las necesidades que se desean cubrir con el proyecto. • Conoce la estrategia de la empresa. • Entiende las necesidades de las personas que van a utilizar el producto final.

[9] Un rol es un conjunto de responsabilidades, actividades y autoridades otorgadas a una persona o grupo de personas. Una persona o grupo puede tener múltiples roles.

- Es una única persona.
- Alienar los objetivos del producto con los de la organización o empresa.

Funciones:

- Desarrolla la estrategia y dirección del proyecto.
- Tiene la decisión final sobre los objetivos y metas.
- Entiende y reúne las necesidades de los grupos de interés en el proyecto.
- Asume la responsabilidad del presupuesto, entiende el tipo de retorno de inversión que le dará el producto.
- Decide sobre la fecha de lanzamiento del producto.
- Acepta o rechaza el trabajo de una *caja de tiempo* (sprint), entiende los requisitos y se asegura que las características funcionen bien.
- Presenta los logros que se deberán tener al de una *caja de tiempo*, esto lo hace antes de que el Equipo de Desarrollo presente el trabajo realizado.

Es la única persona responsable de gestionar la Lista del Producto (los requisitos):

- Expresa claramente los elementos de la Lista del producto.
- Da prioridad a los elementos de la Lista del Producto para alcanzar los objetivos y metas de la mejor manera posible.
- Asegura que la Lista del Producto es visible, transparente y clara para todos, y que muestra aquello en lo que el equipo trabajará.
- Se asegura que el Equipo de Desarrollo entiende los elementos de la Lista del Producto al nivel necesario.

Características

- Es preferible que haya trabajado en proyectos similares en el pasado.
- Toda la organización debe respetar sus decisiones.
- El Dueño del Producto podría respetar los deseos de un comité en la Lista de Productos, pero aquellos que quieran cambiar la prioridad de un elemento de la Lista deben hacerlo por medio del Dueño.
- Pacientes, especialmente con las preguntas, ya que deben responder a todas las dudas que el Equipo de Desarrollo tenga.
- Ser pragmáticos.
- Deben saber negociar.
- Saber tomar de decisiones.
- Flexibles.

Tabla 4. Rol - Dueño del Producto

Equipo de desarrollo
Definición:
• Son los profesionales que desempeñan el trabajo de entregar un Incremento de producto "terminado", que potencialmente se puede poner en operación al final de una *caja de tiempo*.
Personal:
• Programadores. • Probadores • Diseñadores • Analistas Una sola persona puede ser la encargada del diseño, la programación, y las pruebas del producto.
Funciones:
Del equipo: • Son *auto-organizados*: el Equipo de Desarrollo determina sus propias tareas y como desean completarlas. • *Funciones cruzadas*: el Equipo de Desarrollo debe estar dispuesto a aprender nuevas cosas durante el proyecto.

De un Miembro del Equipo:

- Es experto en una de las áreas de desarrollo
- *Auto-organizado*: seleccionan las tareas que van a desarrollar, toma la iniciativa, son curiosos y saben eliminar obstáculos.
- *Multifuncional*: voluntariamente contribuye en áreas fuera de su dominio. Disfruta aprender de nuevas habilidades. Comparte sus conocimientos.
- Sabe trabajar en equipo.

Características

- Nadie indica al Equipo de Desarrollo cómo convertir elementos de la Lista del Producto en Incrementos de funcionalidad potencialmente desplegables.
- El marco de referencia Scrum indica que no se deben reconocen títulos para los miembros de un Equipo de Desarrollo, todos son Desarrolladores, independientemente del trabajo que realice cada persona; no hay excepciones a esta regla.
- Tampoco se reconocen sub-equipos en los equipos de desarrollo, no importan los dominios particulares que requieran ser tomados en cuenta, como pruebas o análisis de negocio; no hay excepciones a esta regla.
- Los Miembros individuales del Equipo de Desarrollo pueden tener habilidades especializadas y áreas en las que estén más enfocados, pero la responsabilidad recae en el Equipo de Desarrollo como un todo.
- Los Equipos de Desarrollo son multifuncionales, contando como equipo con todas las habilidades necesarias para crear un Incremento de producto

Tabla 5. Rol – Equipo de desarrollo

Líder de Proyecto
Definición:
• Persona responsable de proteger al Equipo de Desarrollo de las distracciones de la organización. • Es un facilitador dentro del proyecto.
Funciones:
• Entender la planificación del producto en un entorno empírico. Entrenar al equipo para que siga los principios ágiles. • Ayudar a eliminar los obstáculos del proyecto, tanto de forma reactiva como proactiva. • Fomentar una estrecha relación entre los actores y el equipo. Evita tener preferencias sobre miembros del equipo. • Facilita la creación de consensos. • Protege al equipo de las distracciones de la organización. Es firme acerca de la necesidad del equipo para centrarse en la *caja de tiempo*. • Entender y practicar la agilidad. • Ser un líder, no un jefe o gerente. Respecto al Equipo de Desarrollo: • Guiar al Equipo de Desarrollo en ser auto-organizado y multifuncional. • Ayudar al Equipo de Desarrollo a crear productos de alto valor. • Protege al equipo de las distracciones de la organización. Es firme acerca de la necesidad del equipo para centrarse en la *caja de tiempo*. Respecto a la organización: • Liderar y guiar a la organización en la adopción de prácticas ágiles. • Ayudar a los empleados e interesados a entender y llevar a cabo prácticas ágiles. • Motivar cambios que incrementen la productividad del Equipo.
Características
• Capacidad de comunicación. • Experto en técnicas ágiles. • Debe tener influencia sobre la organización y resolver problemas rápidamente. • Elocuente y diplomático.

Tabla 6. Rol – Líder de Proyecto.

Según (Cohn, User Stories Applied - For Agile Software Develpment, 2004) cuando se desarrolla software es importante incluir por lo menos a un usuario real en el equipo para el desarrollo del proyecto.

Usuario Final
Definición:
• Personas que utilizarán directamente el producto final (software). • Son expertos en las actividades que serán automatizadas, ya que son quienes las realizan.
Funciones:
• Proporcionar información clave acerca del producto y su uso. • Proponer requisitos para el desarrollo del producto.
Características
• Trabajar en equipo. • Ser proactivos y propositivos.

Tabla 7. Rol – Usuario Final.

En caso de no poder integrar a un Usuario Final al desarrollo del proyecto Cohn indica que se debe seleccionar un Proxy, el cual se define como intermediario entre el usuario final y el Equipo de Desarrollo. Un Proxy puede ser el Dueño del Producto.

Según (Layton, 2012) cuando se está desarrollando un proyecto que utilizará métodos ágiles y la organización aún se está adaptando a ellos, se sugiere tener a un Mentor en Agilidad.

Mentor en Agilidad
Definición:
• Es una persona con experiencia en la aplicación de técnicas Ágiles.
Funciones:
• Es un mentor y no es parte del equipo de trabajo. • Es una persona ajena a la organización.
Características
• Experto en la aplicación de técnicas ágiles.

Tabla 8. Rol – Mentor en Agilidad.

Esquema del equipo del proyecto:

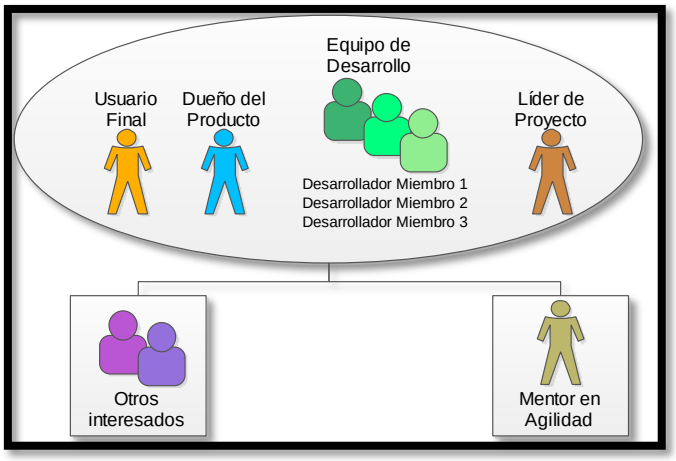


Figura 28. Esquema del equipo de trabajo.

2.7 Software

Según (Drucker, 2002) el software surge como la reorganización del trabajo tradicional por medio de la automatización de procesos.

Conceptos básicos (Solórzano, Millán, Solórzano, & Sánchez, 2012):

Algoritmo: se define como el conjunto ordenado de pasos (instrucciones o pasos dados secuencialmente, desde un inicio hasta un fin) para resolver un problema dado (son instrucciones dadas en el tiempo), sin ambigüedad alguna en un tiempo finito. Los algoritmos son escritos con lenguajes de programación.

Lenguaje de programación: la computadora expresa sus acciones elementales mediante código binario (ceros y unos). Para poder comunicarse con la computadora existen dos lenguajes de programación: *el de bajo nivel* (instrucciones dadas utilizando dígitos binarios) y *el de alto nivel* (constituido por un conjunto de instrucciones dadas con palabras en inglés). Por tanto un lenguaje de programación, es el conjunto de instrucciones y su sintaxis (conjunto de reglas para su correcta escritura), para su utilización por medio de una computadora digital.

Automatización: proviene de la palabra griega automatos, que significa "semejante a la forma en la que la mente trabaja". La automatización se hace por medio de una máquina o mecanismo (hardware), diseñado para seguir un patrón determinado y una secuencia repetitiva de operaciones respondiendo a instrucciones predeterminadas (software), sustituyendo el esfuerzo físico humano o la rutina por la observación o toma de decisiones.

Proceso: es un conjunto estructurado de actividades interrelacionadas, diseñado para cumplir un objetivo específico. Toma una o dos entradas definidas y las convierte en salidas definidas. (ITIL[10] v3), indica que todo proceso comparte las siguientes características:

[10] ITIL versión 3: Biblioteca de Infraestructura de Tecnologías de la Información (ITIL®, por sus siglas en inglés) es un conjunto de buenas prácticas para la Gestión de Servicios Informáticos. Es propiedad del gobierno de Reino unido y es de libre uso, no está ligado a prácticas comerciales. El sitio oficial del ITIL es *http://www.itil-officialsite.com/.*

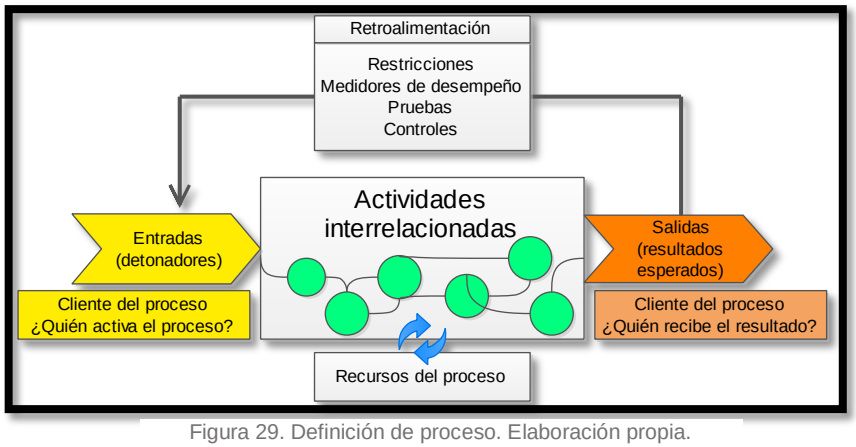


Figura 29. Definición de proceso. Elaboración propia.

- *Es medible*: el proceso es impulsado por el desempeño y puede ser evaluado de forma relevante. El enfoque de métricas varía dependiendo de la audiencia; por ejemplo, los usuarios podrían enfocarse en la facilidad para utilizarlo, el tiempo de respuesta, etc.
- *Entrega de resultados específicos (salidas)*: los resultados deben ser identificables, medibles y constatables.
- *Responden a eventos específicos (entradas)*: tienen detonadores específicos que hacen que el proceso se active. Por ejemplo al ingresar a un sistema por medio de un usuario y contraseña, al dar clic en el botón de enviar, se activa un proceso para verificar los datos y autentificar al usuario.
- *Tienen un cliente final*: es el receptor del resultado final. El proceso debe cubrir las expectativas del cliente. Un ejemplo sería el proceso para ingresar a un correo electrónico (por medio de un nombre y una contraseña), el usuario el ingresar sus datos (entradas), al presionar el botón acceder (detonador), el proceso realizaría las actividades necesarias para determinar si genera la salida esperada (ingresar al correo), o si retroalimenta al proceso para indicarle que algo falló.

El software se define: es un conjunto de instrucciones escritas en *un lenguaje de programación de alto nivel* que indica, paso a paso (algoritmo), las acciones para resolver un problema dado y que pueda ser traducido al lenguaje de la computadora. El conjunto de instrucciones es conocido como programa fuente. El proceso de escribir y codificar programas se denomina *programación*, y las personas que se especializan en esta labor se llaman *programadores*. En otras palabras el software es la automatización de procesos, que sustituyen las actividades rutinarias que un ser humano realizara de manera cotidiana

Tipos de software:

- *Software de sistemas*: es un conjunto de programas generalizados que administran los recursos de la computadora, como el procesador central, los enlaces de comunicaciones y los dispositivos periféricos. Está constituido por los sistemas operativos, el software de comunicaciones, los compiladores e intérpretes. Ejemplos: sistemas operativos de Windows, iOS, Linux para computadora, sistemas operativos Android, Symbian, iOs para celulares y tabletas,

- *Software de aplicación*: describe los programas que se escriben para los usuarios o son escritos por ellos, con el fin de aplicar la computadora a una tarea específica. Existen diversos tipos de lenguajes de programación de alto nivel para crear este tipo de software los cuales se clasifican en (Solórzano, Millán, Solórzano, & Sánchez, 2012):

Lenguajes de programación de alto nivel
***Lenguajes orientados a procedimientos*:** científicos (análisis estadísticos, manejo de matrices, etc), para negocios (nóminas, balances, estados financieros, etc.), de aplicaciones múltiples (tanto científicos como de negocios). Ejemplos: FORTRAN, Java, C, HTML, JavaScript, etc.
***Lenguajes orientados a problemas*:** se conocen como lenguajes de paquetería, y están diseñados para resolver los problemas de un área particular de aplicación. Ejemplos: WORD, SAS, COGO, etc.
***Lenguajes de consulta*:** generan automáticamente el procedimiento de solución del problema, es decir el programador sólo indica qué hacer y no como hacerlo, instrucción por instrucción. Ejemplos: DATA EASE, EASYTRIEVE, etc.
***Generadores de aplicaciones*:** en este tipo de lenguajes, las necesidades de un sistema se indican seleccionando funciones previamente programadas, sin la necesidad de dar instrucciones a nivel de procedimientos. Ejemplo: ORACLE, MySQL, PostgreSQL, SQL server.

Tabla 9. Lenguajes de programación de alto nivel

2.8 Conclusiones

Con base en el enfoque sistémico y en el enfoque cibernético, se pudo conceptualizar a un proyecto como un sistema e identificar sus cuatro elementos básicos: 1) entradas o insumos, 2) al proceso de transformación, 3) las salidas o productos, 4) y la retroalimentación. Se determinó que los proyectos son sistemas abierto ya que intercambian información a su medio, además de tener la capacidad de modificarlo según sus propios objetivos. Un proyecto puede ser identificado por ser teleológico (cuenta con una visión, objetivos y metas); contar con un tiempo finito para ser desarrollado; y por obtener productos únicos, en este caso software funcional.

Se identificó que la retroalimentación es como un elemento vital dentro del desarrollo de proyectos, ya que es la encargada de obtener información de su medio ambiente para determinar si un producto es aceptado o rechazado, con la finalidad de corregir procesos y mantener al sistema en equilibrio (homeóstasis), en otras palabras, evitar que un proyecto fracase debido a deficiencias en el proceso de transformación. Para la obtención de información se recomendó el uso de métricas (técnicas, de servicio y de proyecto) para controlar y monitorear un proyecto.

Al conceptualizar al proyecto como un sistema se estableció que, su suprasistema es una organización, y que los principales elementos de su medio ambiente son:

- *La cultura organizacional:* es uno de los elementos más importantes ya que define la forma característica de pensar y hacer las cosas en una empresa, por tanto dependiendo de los principios y valores que se apliquen, es como se puede lograr el éxito o fracaso de los proyectos. Se recomendó el uso de los 12 principios y 4 valores que propone el movimiento ágil.
- *Las restricciones (calidad, costo y tiempo):* para los proyectos de software se recomienda fijar el tiempo y los costos mientras que el alcance puede ser modificado.
- *Los proveedores:* las terceras personas encargadas de abastecer la materia prima para el desarrollo de software.
- *Los riesgos:* se definió lo que es un riesgo, y las distintas formas de abordarlo.

Otro elemento importante dentro del desarrollo de proyectos de software son los interesados, se detectaron tres roles principales,1) dueño del producto,2) líder de proyecto, equipo de desarrollo, y dos roles que pueden ser útiles durante el proceso de desarrollo de un proyecto, usuario final y mentor ágil.

Se desarrolló la definición de software, y se determinó que su principal objetivo es la automatización de procesos.

Como se puede observar aplicar el enfoque de sistemas y el enfoque cibernético a la administración de proyecto de software permitió tener una visión holística, y de esta manera identificar los elementos que componen a un proyecto, así como los factores ambientales que intercambian información con éste, los cuales pueden afectarlo de manera positiva o negativa.

Capítulo 3. El modelo iterativo incremental y la planeación en el proceso de conducción

Objetivos del capítulo:

- ✓ Analizar el proceso de transformación de un proyecto de software para determinar sus subsistemas.
- ✓ Describir el proceso que propone el modelo iterativo e incremental para el desarrollo de software.
- ✓ Identificar la estructura de la planeación para el proceso de conducción.
- ✓ Determinar la relación entre el modelo iterativo e incremental y la planeación como proceso de conducción.

3.1 Introducción

En el capítulo anterior, al conceptualizar a un proyecto como un sistema abierto, se determinó que uno de sus elementos es el *proceso de transformación*, el cual es el encargado, por medio de métodos propios, de transformar los insumos que su medio ambiente le proporciona para obtener un producto. Uno de los objetivos de este proceso es obtener como salidas software, que satisfaga las necesidades de un cliente. Hasta el momento se ha definido a este proceso como una caja negra, en donde, entran requisitos, personas, dinero, etc., y se obtiene como salidas software en funcionamiento.

Para entender el funcionamiento de este proceso de trasformación, conceptualizado como un sistema, se requiere conocer los subsistemas que lo componen. En este sentido, el enfoque cibernético, permite distinguir en cualquier sistema, aplicando el proceso de descomposición[11], dos subsistemas principales: el de gestión o conductor y el productivo o conducido, junto con sus dos tipos de relaciones fundamentales, las de información y las de ejecución (Gelman & Negroe, La planeación como un proceso básico en la conducción, 1982).

Otro autor que considera dentro del proceso de transformación que existe un subsistema de gestión es (Hernández, 2006), indica que su función principal es conducir al proyecto (sistema) para lograr sus objetivos.

[11] Proceso por descomposición funcional.- Se basa en la descomposición funcional del sistema en subsistemas que, a su vez, son sistemas y cuyas funciones, tanto individuales como en conjunto, aseguran el funcionamiento del sistema. La aplicación sucesiva de este procedimiento permite trabajar en diferentes niveles de desagregación del sistema. Por ejemplo, el sistema se descompone en subsistemas, éstos en partes, las últimas en componentes y, finalmente, los componentes se desmiembran en elementos, que se consideran como las unidades indivisibles en un estudio correspondiente. Este proceso es deductivo porque parte del sistema hacia el elemento, donde las características de los subsistemas, partes, etc., se deducen de las del sistema (Gelman, 2005).

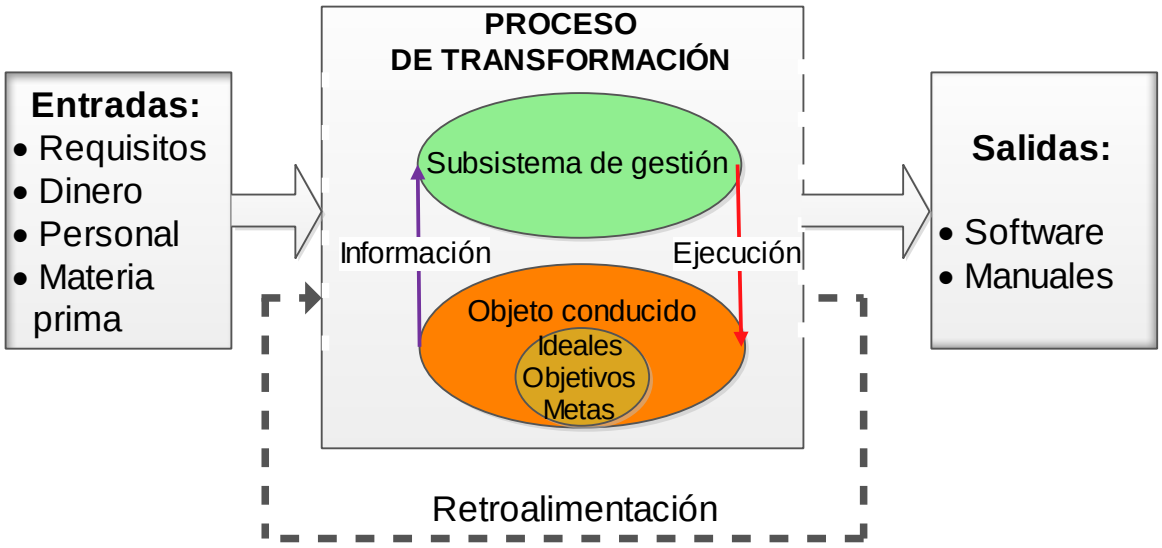


Figura 30. Subsistemas del proceso de transformación. Elaboración propia.

A principios del siglo XX, Henri Fayol, empresario francés, propuso por primera vez que la gestión tiene cinco funciones: planear, organizar, mandar, coordinar y controlar. Actualmente esas funciones se han reducido a cuatro (Robbins & Coulter, 2010):

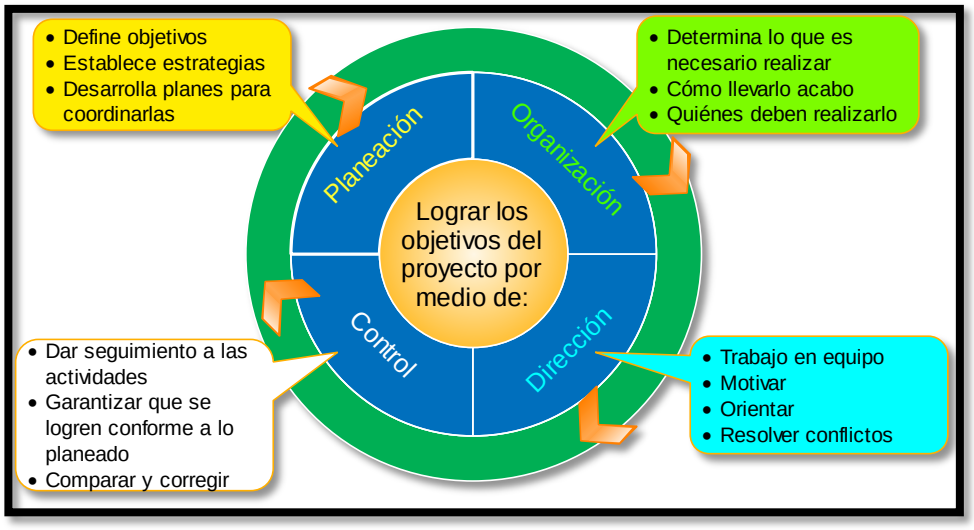


Figura 31. Funciones de la gestión.

Por tanto las funciones del subsistema de gestión son las de planear, organizar, dirigir y controlar el proceso de conducción para el desarrollo de un proyecto. Por tanto surge la necesidad de terminar la estructura que debe tener este proceso para poder aplicarlo en el subsistema productivo para lograr los objetivos que este persigue.

3.2 Análisis del proceso de conducción en los proyectos de software

El desarrollo de proyectos de software, ha tomado como proceso de conducción para cumplir con las funciones del subsistema de gestión, al modelo iterativo e incrementar debido a las ventajas que presenta cuando se deben crear productos en donde no se tienen claros todos los requisitos desde el principio, y su flexibilidad para la realización de cambios dentro del producto (Abrahamsson, Salo, & Ronkainen, 2002). El modelo propone los siguientes pasos:

1. *Recolección de los requisitos*: generar una lista con todas las características de alto nivel que describan de manera general cuáles son los procesos que el cliente desea automatizar por medio de un software.

Figura 32. Recolección de requisitos

2. *Priorización:* de la lista de requisitos obtenida, el cliente ordena las características de mayor a menor prioridad.

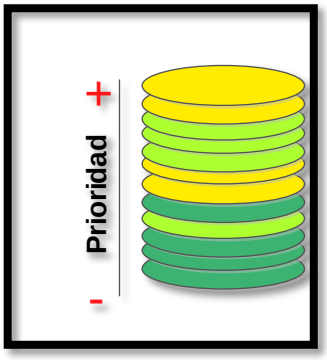


Figura 33. Lista de requisitos priorizada

3. *TimeBox*[12] *(caja de tiempo):* se fija un periodo de tiempo donde se desarrollará una iteración[13] y al finalizar se entregará como resultado un incremento. El TimeBox puede ser de una a cuatro semanas, una vez establecido el periodo se recomienda no modificarlo durante todo el proyecto.

[12] En la metodología Scrum al TimeBox se le conoce como Sprint.
[13] Una *iteración* es el acto de repetir un proceso con la finalidad de alcanzar un objetivo. Los resultados de una iteración generan un *incremento,* el cual se utiliza como punto de partida para la siguiente iteración.

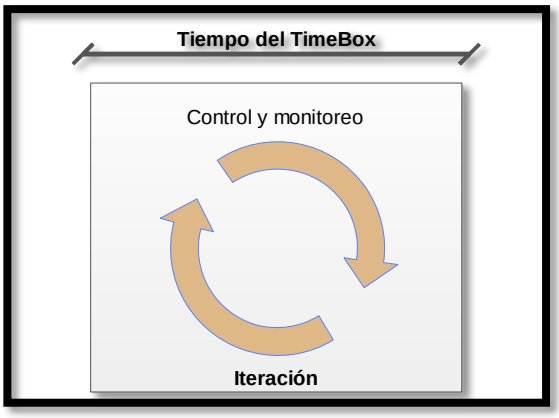


Figura 34. TimeBox

4. *Estimación*: el equipo de trabajo toma de lista de requisitos priorizados la cantidad que puede ser desarrollada durante el TimeBox.

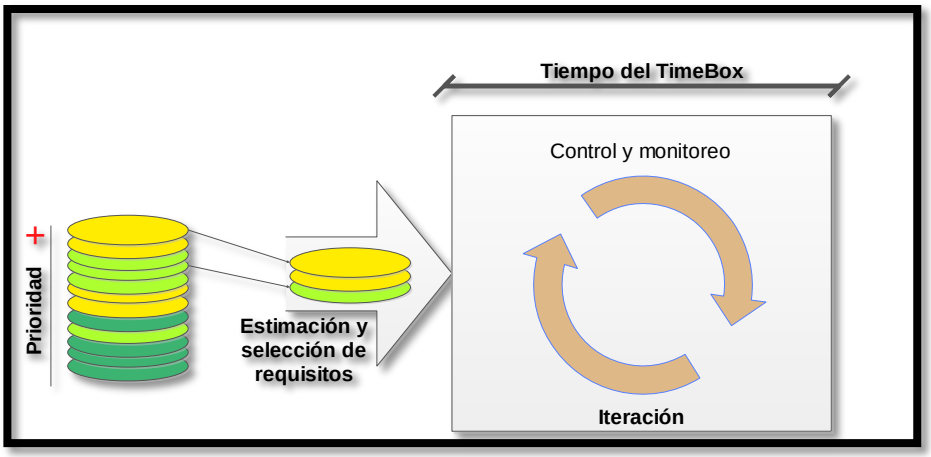


Figura 35. Estimación y selección de requisitos.

5. *Iteración*: los requisitos seleccionados pasan por una serie de fases:
   - *Análisis de los requisitos seleccionados*: se obtiene el mayor detalle posible de los requisitos seleccionados.
   - *Diseño*: se elabora un prototipo de alto nivel, sobre los requisitos seleccionados.
   - *Pruebas*: el cliente determina que pruebas debe pasar cada requisito.
   - *Programación:* se codifican las pruebas para posteriormente codificar las funcionalidades, esto se hace así ya que se recomienda trabajar con el desarrollo basado en pruebas (TDD).
   - *Integración*: si es la primera iteración, se muestra al cliente el incremento obtenido, a partir de la segunda iteración el incremento se integra con los resultados de las iteraciones anteriores.
   - *Pruebas de integración*: es la fase en la cual los módulos individuales de software son combinados y probados como un grupo.

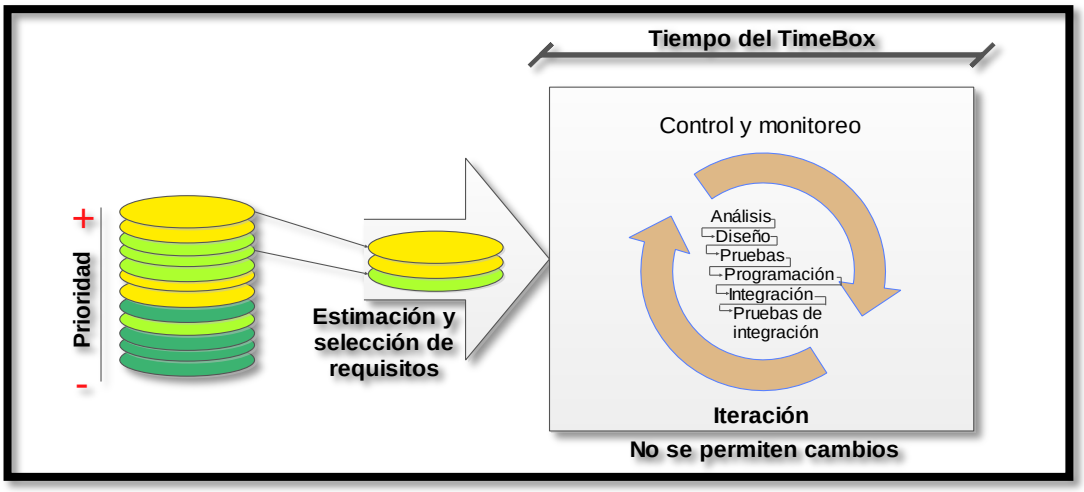
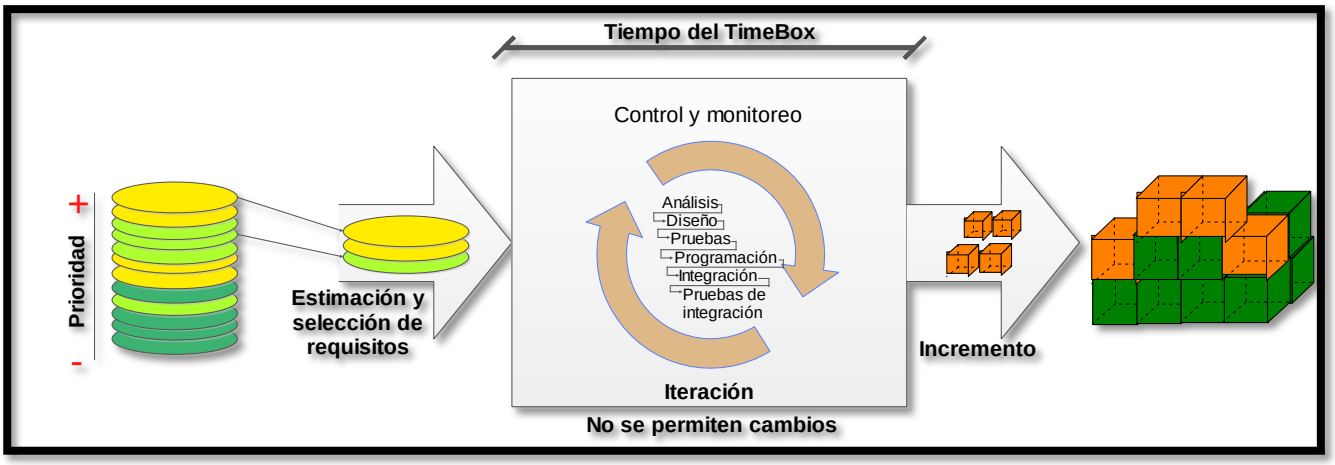


Figura 36. Fases de una iteración.

Reglas de la iteración: durante la ejecución de la iteración los requisitos seleccionados se congelan, no se permiten cambios hasta su finalización.

6. *Incremento*: al final de cada TimeBox se entrega una versión del producto completamente



funcional.

Figura 37. Incremento.

7. *Retroalimentación*: en cada incremento, el cliente evalúa la versión y realiza las sugerencias sobre las correcciones que desea realizar y selecciona el siguiente grupo de requerimientos para desarrollar las nuevas funcionalidades (estos requisitos se toman de la lista elaborada en el punto 1).

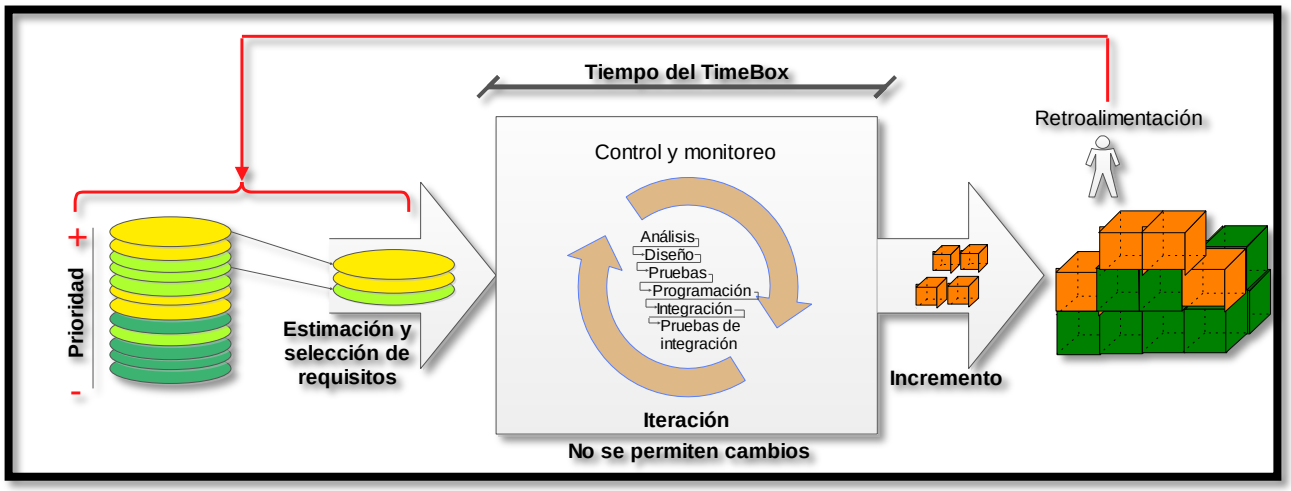


Figura 38. Retroalimentación en el desarrollo iterativo e incremental. Elaboración propia.

8. *Nueva iteración*: Se repiten los pasos 2 al 6 hasta llegar al producto deseado.

El proceso propuesto por el modelo iterativo e incremental para el proceso de conducción, inicia con la generación de una lista de requisitos, por ejemplo, si una persona requiere un software para administrar las ventas de una tienda de abarrotes, el cliente comienza a describir lo que él cree que necesita, y esto es tomado como las necesidades que requieren ser cubiertas, posteriormente se inicia con el desarrollo de los primeros requisitos, según la prioridad.

Como se refirió el desarrollo de software genera mucha incertidumbre, ya que no se tienen claras todas las características que el sistema tendrá, o cual será el resultado final, el modelo iterativo e incremental, lo que hace es disminuir esa incertidumbre al tomar solo una pequeña parte de los requisitos solicitados por el cliente, desarrollarlos en un corto tiempo, presentarlos como un incremento, y reajustar el producto mediante la retroalimentación del cliente. Es por medio de la retroalimentación que aumenta la experiencia del conducente y desarrolla un proceso de aprendizaje.

La interpretación del procedimiento de aprendizaje se presenta a través de las siguientes etapas (Gelman, 2005):

- Planteamiento de la hipótesis con base en la experiencia y conocimientos alcanzados.
- Realización de experimentos para su validación.
- Evaluación de resultados a fin de cambiar, ajustar y desarrollar la hipótesis.

Cada etapa está relacionada con el proceso de conducción que propone el modelo iterativo e incremental:

- Detección y planteamiento del problema, basándose en la experiencia y en la información disponible.
- Toma de decisiones y ejecución.

- Análisis de los resultados a fin de modificar las decisiones, e inclusive reformular el problema.

Este proceso es útil cuando el cliente conoce realmente la problemática que desea resolver. Pero de manera general, no puede afirmarse que el cliente estará plenamente seguro de lo que cree necesitar, y el conducente tampoco podrá estar seguro de recibir la información suficiente para la toma de decisiones y estructuración de las posibles alternativas de solución. Esto quiere decir que pueden generarse productos que no resolverán la problemática que tiene el cliente, debido a que nunca se analizó, lo que implica que, resolverán problemas de manera aislada o se presentarán soluciones que servirán sólo para un corto plazo.

Al considerar el proceso de toma de decisiones presentado, se observa una falta de marco teórico mediante el cual se facilite el planteamiento del problema, la búsqueda de solución y el establecimiento de criterios que permitan evaluar y seleccionar las decisiones más adecuadas.

Por tanto, se puede decir que este proceso es parcial, restringido y con la carencia de actividades teóricas, ya que presenta una postura puramente empirista, esto lo confirman (Schwaber & Sutherland, 2003) autores de marco de trabajo Scrum basado en el modelo iterativo e incremental y uno de los más utilizados para el desarrollo de software, ellos definen este marco de trabajo útil para el desarrollo de proyectos complejos, ya que 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 (Schwaber & Sutherland, 2003).

Como solución a este tipo de modelos, (Gelman, 2005) propone un proceso de conducción que establezca un estado deseado del objeto conducido, así como ciertos criterios que sirvan para selecciona y organizar las actividades adecuadas que contribuyan al cambio actual deseado.

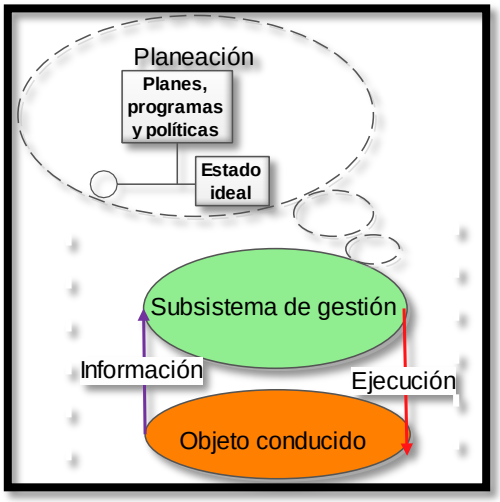


Figura 39. Paradigma de la conducción planificada.

Con base en el análisis anterior se justifica la planeación como una actividad adicional, que constituye una herramienta de apoyo en el proceso de conducción, y que visualiza y especifica el objeto conducido, los objetivos de la conducción y las actividades que permiten realizar el cambio, de manera directa. Los elementos proporcionados por la planeación enriquecerán al modelo iterativo e incremental, ya que proporcionarán un marco conceptual cuya necesidad se ha enfatizado, así como las bases y criterios teóricos, que permite ampliar la experiencia y tratar de tomar decisiones con información suficiente, y tener la posibilidad de prever y prevenir problemas futuros (Gelman, 2005).

3.3 La planeación como un proceso de conducción

Planear según (Ackoff R. , 2002) es diseñar un futuro deseado así como los medios efectivos para lograrlo. Es un proceso que implica tomar y evaluar decisiones interrelacionadas antes de que sea necesaria la acción.

La planeación tiene tres características principales:

1. Es algo que se hace antes de emprender una acción, es una toma de decisiones anticipada. Es un proceso para decidir qué hacer antes de que sea necesaria una acción.
2. La planeación es necesaria cuando el estado futuro que deseamos incluye un conjunto de decisiones interdependientes. Por tanto se requiere de un sistema de toma de decisiones.
3. La planeación es un proceso dirigido a producir uno o más estados futuros deseados y cuya materialización no es probable a menos de que se haga algo. La planeación se ocupa de evitar acciones incorrectas y reducir el número de oportunidades que no se aprovechan.

Como se observa, el proceso de transformación para el desarrollo de proyectos de software tiene dos subsistemas:

El subsistema conducente o de gestión: consiste en un proceso de cambio controlado del objeto conducido según cierto objetivo, es decir, sirve para seleccionar y realizar la trayectoria adecuada de cambio.

El subsistema conducido o productivo es el encargado de cumplir con la misión y los objetivos que tiene el sistema en el suprasistema, los cuales pueden consistir en prestar servicios o producir bienes.

El subsistema de gestión tiene dos formas de operar:

1. *De forma correctiva*: intenta que el subsistema productivo se mantenga en un estado homeostático o que se optimice la operación, realizando operaciones inmediatas acordes a la situación actual, la información disponible y la experiencia de los tomadores de decisiones.
2. *De forma planificada*: establece un futuro deseado del subsistema productivo y se encarga de llevarlo del estado actual al estado deseado, en otras palabras, conducir el cambio de manera controlada del subsistema productivo por medio de la planeación.

El subsistema de gestión consta de cuatro subsistemas (Negroe, 2005) (***FIGURA 40***):

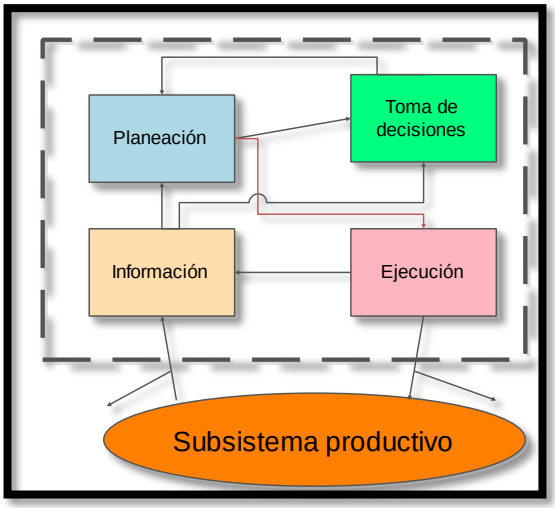


Figura 40. Representación funcional del sistema de gestión

1. *Información*: proporciona datos sobre el estado actual del objeto a conducir por medio de indicadores relevantes que provengan no únicamente del objeto conducido, sino además de otros sistemas vinculados. El subsistema de información debe captar, generar, seleccionar, transmitir y procesar la información para el proceso de toma de decisiones.

   En el caso de la planeación, se requiere adicionar información obtenida para la conducción actual, esto quiere decir que se debe tener información sobre lo que está sucediendo con el objeto conducido, y de esta manera tener un subsistema que pueda ser empleado como retroalimentación del sistema.
2. *Toma de decisiones:* analiza la información para determinar qué acciones deben ser llevadas a cabo para conducir el sistema productivo.
3. *Planeación:* es un proceso que apoya la conducción del sistema, ya que permite visualizar y prever el cambio y con ello definir los objetivos a alcanzar y las actividades que se deben realizar para darles cumplimiento.
4. *Ejecución:* lleva a cabo las acciones planeadas.

Enseguida se describe la estructura del proceso de planeación que propone (Gelman, 2005):

Gelman aplica el enfoque sistémico a la planeación, conceptualizándola como un sistema. Identifica tres subsistemas:

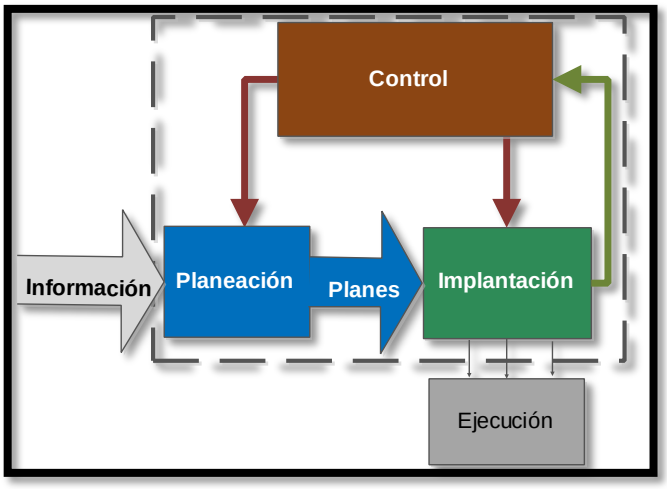


Figura 41. Estructura de la planeación.

1. *Planeación (Planes):* producir planes con sus diferentes elementos (objetivos, políticas, metas, actividades).
2. *Implantación:* define cómo serán ejecutados los planes.
3. *Ejecución:* realización de los planes.
4. *Control*: Ackoff lo considera como el diseño de un procedimiento que permite prever o detectar los errores o fallas de un plan y la forma de prevenirlos y corregirlos

El subsistema de planeación lo descompone en tres etapas:

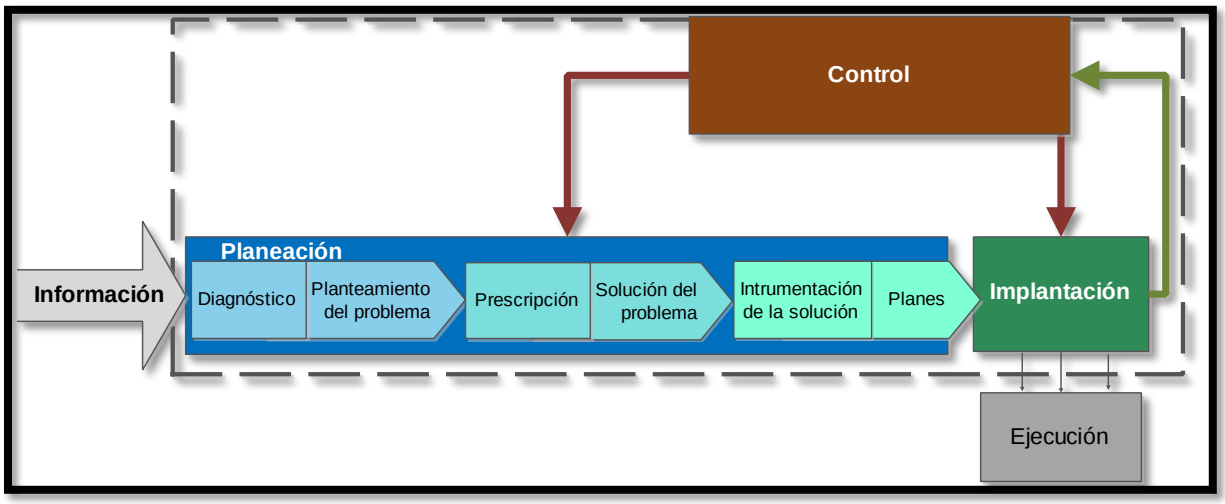


Figura 42. Estructura del subsistema de planeación

- *Diagnóstico*: trata de detectar, definir y plantear los problemas que se quieren resolver a través del proceso de conducción del objeto.

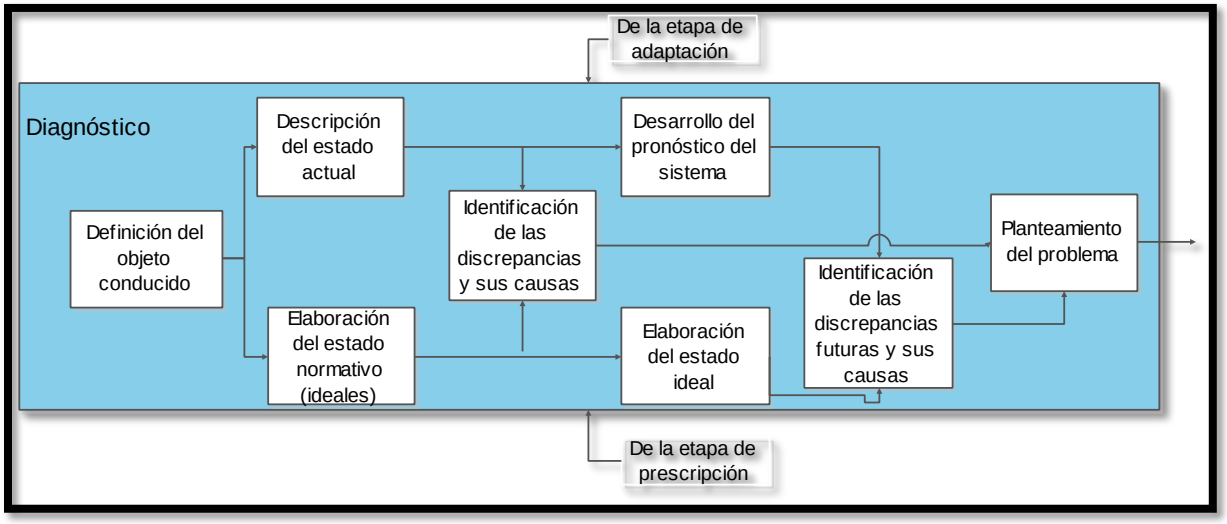


Figura 43. Estructura de la etapa de diagnóstico.

- *Prescripción*: trata de dar solución al problema planteado, mediante el análisis de las distintas alternativas factibles (con sus restricciones o limitaciones).

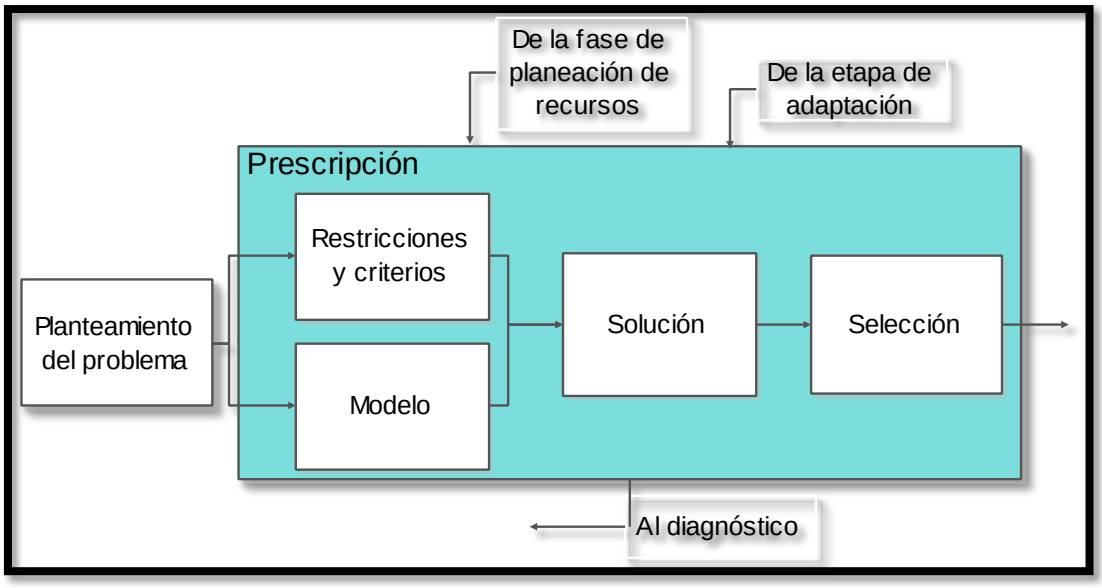


Figura 44. Estructura de la etapa de prescripción.

- *Instrumentación de la solución*:

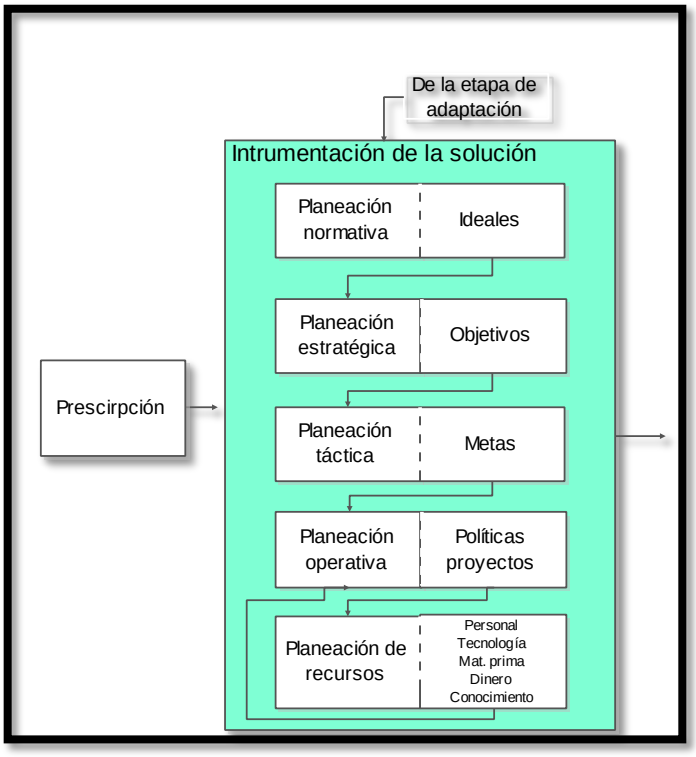


Figura 45. Estructura de la etapa de instrumentación de la solución

El subsistema de control se divide en dos etapas:

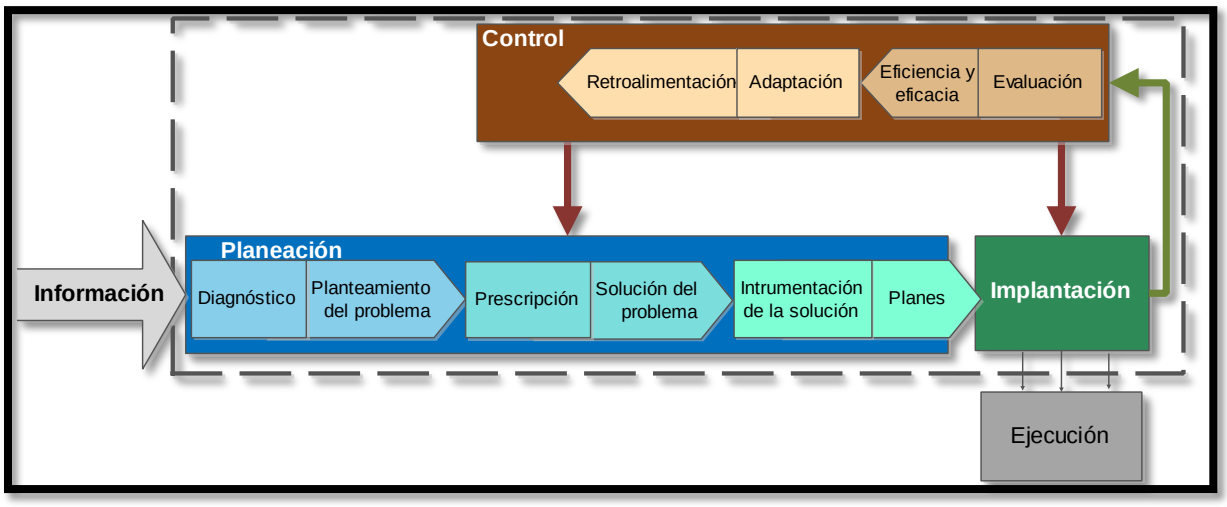


Figura 46. Estructura del subsistema de control.

- *Evaluación*: observa la eficacia y eficiencia de los planes ejecutados, para determinar si se han cumplido las metas y objetivos.
- *Adaptación*: recibe información del subsistema de evaluación, para retroalimentar al sistema y determinar si debe ajustarse.

3.4 Integración del modelo iterativo e incremental en la estructura del proceso de planeación

Al aplicar la planeación como proceso de conducción en un proyecto de desarrollo de software, permite a los interesados realizar como primer paso un diagnóstico, para identificar el objeto conducido por medio de la descripción de su estado actual, para posteriormente plantear un futuro deseado, hacer una comparación entre ambos estados para identificar las discrepancias y sus causas, y obtener como resultado la problemática que debe resolverse.

Una vez que se identificó la problemática se debe realizar una prescripción, esto es, identificar las posibles alternativas de solución, para seleccionar la que más se adecúe a las necesidades que se deben cubrir.

Con la propuesta de solución, la siguiente etapa es la instrumentación de la solución, definir la estrategia (qué se debe hacer), la táctica (cómo se planea hacer), y la operación (las políticas que se deberán seguir), la definición de estos elementos permite determinar los recursos que será necesario utilizar para el desarrollo del proyecto.

Los planes elaborados sirven como entrada del subsistema de implantación, dentro de la estructura propuesta, Gelman, sólo hace referencia a Ackoff, mencionando que esta actividad consiste en el diseño de los procedimientos para tomar decisiones dentro de la implantación, así como de su organización para poder realizar el plan.

Por último, se propone un subsistema de control, el cual contiene dos etapas: la evaluación, encargada de verificar si la implantación es eficiente[14] y eficaz[15], la información obtenida es transmitida a la etapa de adaptación la cual se encarga de retroalimentar al sistema de planeación e implantación, para mantener el sistema en homeostasis.

Como se observa el subsistema de implantación, no fue descrito por Gelman, debido a que su propósito era obtener un proceso general de planeación, pero en este caso en particular se está buscando un proceso que este enfocado a la conducción de proyectos de desarrollo de software, por tanto se propone que el subsistema de implantación, tome las etapas que propone el modelo iterativo e incremental, para aplicar los planes obtenidos del subsistema de planeación.

Como se puede observar en la (***FIGURA 47***) la estructura de planeación puede ser complementada con el modelo iterativo e incremental, para la gestión de proyectos de software.

[14] La eficiencia se ocupa de obtener los mejores resultados con el mínimo de recursos (hacer las cosas bien).
[15] La eficacia se encarga de lograr los objetivos que el proyecto persigue (hacer las cosas correctas).

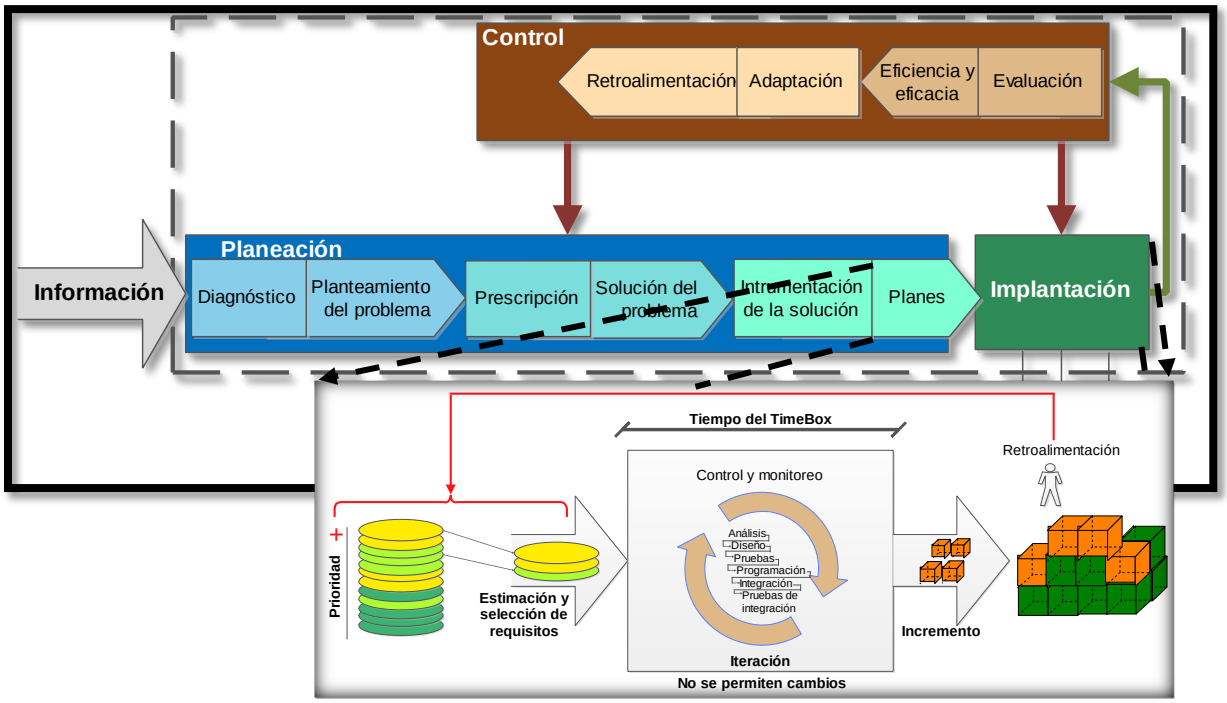


Figura 47. Integración del modelo iterativo e incremental en la estructura de planeación como proceso de conducción.

3.5 Conclusiones

En el presente capítulo, se determinó que el proceso de transformación de un proyecto de software, contiene dos subsistemas:

- *El subsistema de gestión*: donde sus funciones son planear, organizar, dirigir y controlar el proceso de conducción de un proyecto, para que este logre sus objetivos.
- *El subsistema conducido*: es el encargado de cumplir con la misión y los objetivos que tiene el sistema en el suprasistema, los cuales pueden consistir en prestar servicios o producir bienes. En este caso su objetivo es desarrollar software que este técnicamente bien diseñado y que cumpla con las expectativas del cliente.

Se analizó al subsistema de gestión para el proceso de conducción de proyectos de software utilizando el modelo iterativo e incremental, y se determinó que presentaba una visión parcial y restringida, ya que su enfoque es completamente empírico, lo cual indica, que carece de un marco teórico que facilite el planteamiento del problema, la búsqueda de soluciones y el establecimiento de criterios que permitan evaluar y seleccionar las decisiones más adecuadas.

Para dotar al modelo iterativo e incremental de un marco teórico, se propuso a la planeación como proceso de conducción, la cual propone una estructura compuesta por tres subsistemas: el de planeación, dividido en tres etapas 1) realizar el diagnóstico (estado actual y futuro del objeto conducido), 2) la prescripción (identificación de alternativas de solución), y 3) de instrumentar la

solución (definir los planes que requiere el proyecto); el de control, dividido en dos etapas: 1) evaluación y 2) adaptación; el de implantación, en este subsistema se propuso la integración de modelo iterativo e incremental, ya que es el encargado de la organización para poder realización de los planes diseñados en el subsistema de planeación.

Se obtuvo como resultado un subsistema de gestión que utiliza un proceso de conducción basado en la planeación y el modelo iterativo incremental que permite: planear, controlar, dirigir y organizar el proceso de trasformación de un proyecto de software.

Capítulo 4. Método para la gestión de proyectos de software

Objetivos del capítulo:

- ✓ Identificar los pasos a seguir para administrar un proyecto de software tomando como base la planeación y al modelo iterativo e incremental como proceso de conducción.
- ✓ Conceptualizar los pasos como un método.
- ✓ Proponer técnicas que permitan aplicar el método propuesto.

4.1 Introducción

El diseño de un método surge de la necesidad que tienen las empresas por gestionar sus proyectos de software de manera sistematizada, como se definió en la problemática, la mayor parte de los problemas que se presentan cuando se desarrolla software, surgen por la falta de definición de procesos para seguimiento al desarrollo del producto.

Al conceptualizar al proyecto como un sistema, se determinó que uno de sus elementos es el proceso de trasformación, en el capítulo anterior, se analizó este elemento y se determinó que está compuesto por el subsistema de gestión y el subsistema conducido. Se definió al primero como encargado de planear, organizar, dirigir y coordinar al sistema conducido, el cual contiene los objetivos que el proceso de transformación persigue.

Para determinar los subsistemas del sistema de gestión, se definió una estructura basada en la planeación y el modelo iterativo incremental para el proceso de conducción de un proyecto de software. Esta estructura se tomará como base para el diseño del método, el cual tendrá como finalidad permitir a un equipo de trabajo, identificar los pasos que deben seguir para gestión de un proyecto de software desde su inicio hasta su cierre. En cada uno de los pasos del método se sugerirán técnicas para su aplicación.

Un **método,** se define como una serie o conjunto de pasos ordenados y sistematizados que conducen a un fin. El *método* representa los *pasos a seguir* para el logro de un objetivo, las *técnicas* son las *habilidades y herramientas* para aplicar los pasos del método (***Figura 48***).

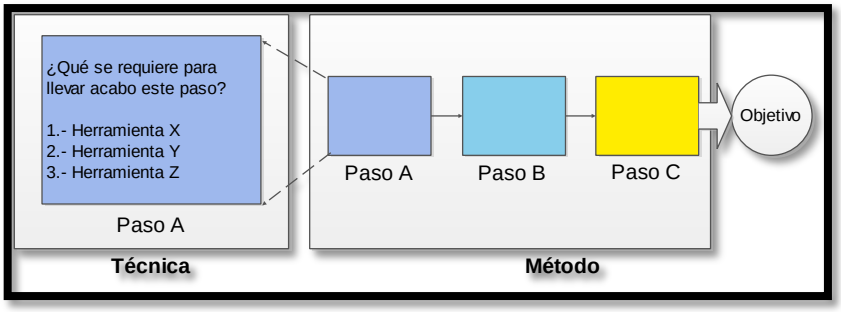


Figura 48. Relación entre método y técnica. Elaboración propia.

4.2 Estructura del método para la gestión de proyectos de software

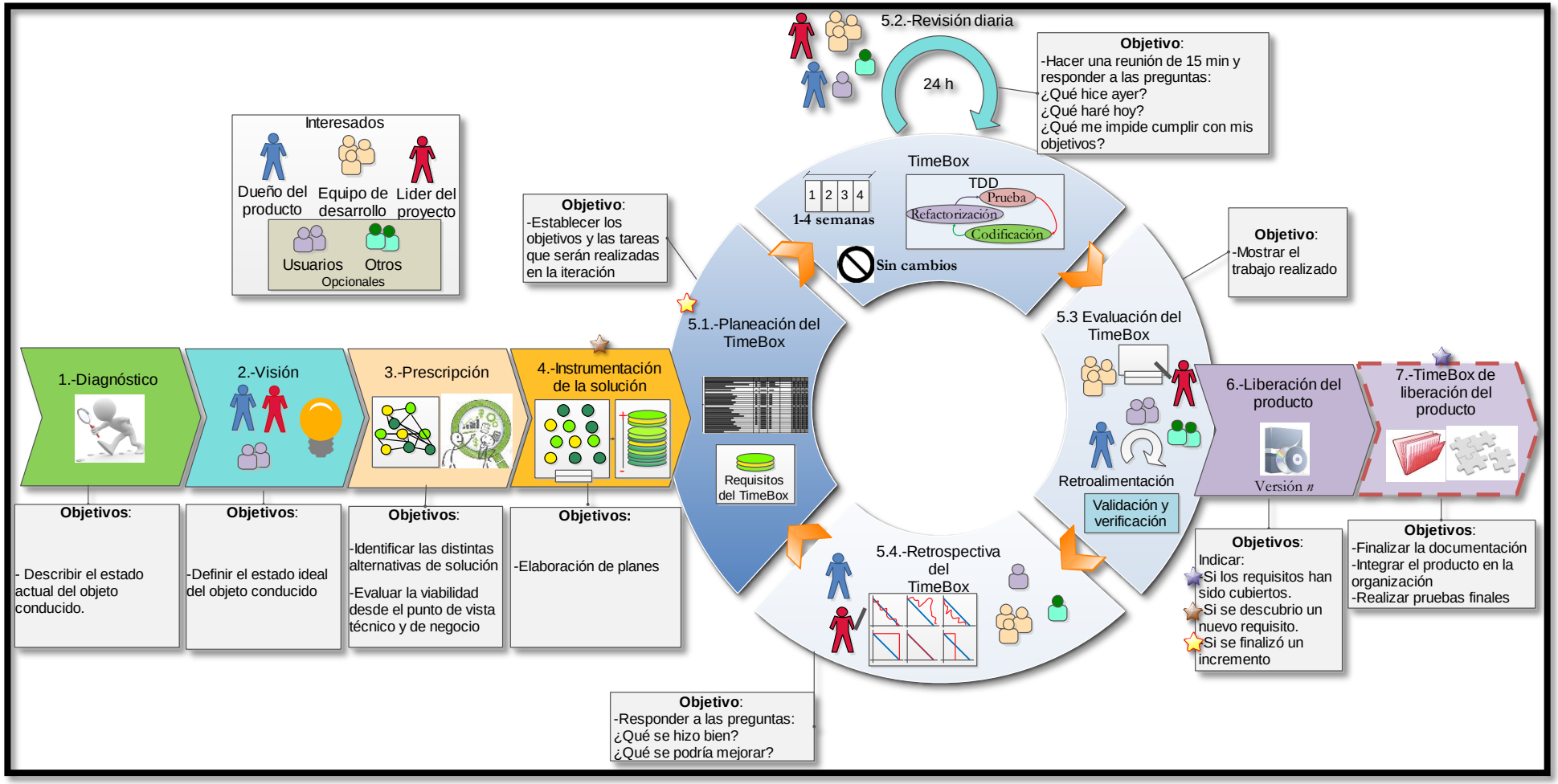


Figura 49. Método para la administración de proyectos de software. Elaboración propia.

En la ***FIGURA 49*** se tiene la representación gráfica del método propuesto para la gestión de proyectos de software, la estructura es la siguiente:

- El sistema de planeación está dividido en 4 pasos:
    1. Diagnóstico
    2. Visión
    3. Prescripción
    4. Instrumentación de la planeación.

- Para determinar los pasos del sistema de implantación, se tomó como referencia al marco de trabajo Scrum (Schwaber & Sutherland, 2003), se dividió en 4 pasos:
    5.1. Planeación del TimeBox
    5.2. Revisión diaria
    6. Liberación del producto
    7. TimeBox de liberación del producto

- El sistema de control está divido en dos pasos:
    5.3. Evaluación del TimeBox
    5.4. Retrospectiva del TimeBox

4.3 Descripción de los pasos del método para la gestión de proyectos de software y técnicas para la ejecución

4.3.1 Paso 1: Diagnóstico

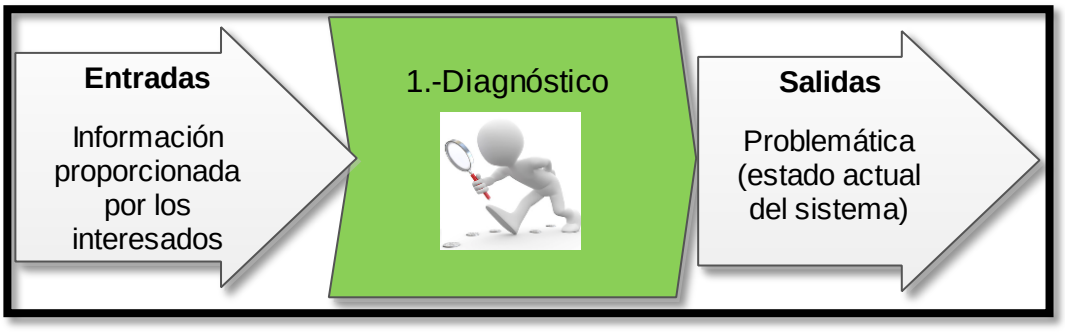


Figura 50. Diagnóstico

Objetivo:

- Definir el estado actual del sistema conducido.

Metas:

- Describir la problemática del negocio que se aborda.
- Identificar los procesos y las actividades que serán modificados.
- Identificar a las personas que están directamente involucrados en los procesos y actividades que serán modificadas.

Roles involucrados:

- Líder del proyecto.
- Dueño del producto.

Proceso:

1. *Describir la situación actual***:** es una breve declaración, textual y concisa de la situación actual del sistema conducido.

2. *Elaboración de un diagrama de context*o (Yurdon, 2014)**:** El diagrama de contexto muestra a través de flujos de datos las interacciones existentes entre los agentes externos y el sistema conducido, sin describir en ningún momento la estructura del sistema que se desea obtener.

En este tipo de diagrama, el sistema de información debe representarse como un único proceso de muy alto nivel con entradas y salidas hacia los agentes externos que lo limitan, de forma equivalente a una caja negra.

Teniendo en cuenta que este diagrama debe de ser comprensible, no es posible representar todos los flujos de datos del sistema en él, sino más bien debe representarse en él una visión general del sistema desde la perspectiva de los propietarios de sistemas siguiendo dos lineamientos básicos:

- Representar únicamente los flujos de datos que tengan algo que ver con el objetivo principal del sistema.
- Utilizar flujos de datos compuestos que representen a aquellos que sean similares.

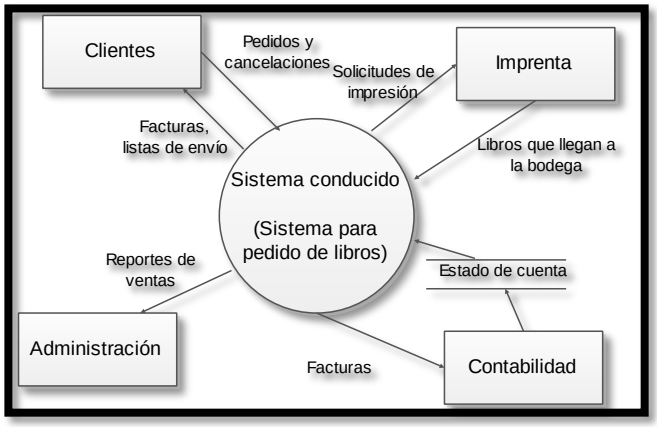


Figura 51. Ejemplo de diagrama de contextos.

Fuente: (Yurdon, 2014)

3. *Identificar de los procesos a modificar*: con el diagrama de contextos se determinan los elementos que tienen relación directa con el sistema conducido y por tanto se pueden identificar los procesos que serán modificados, la descripción de estos procesos se puede hacer por medio de una figura rica o un mapa conceptual.

4. *Identificar de los interesados***:** determinar quiénes son las personas responsables de desarrollar los procesos que tienen relación directa con el sistema conducente.

4.3.2 Paso 2: Visión

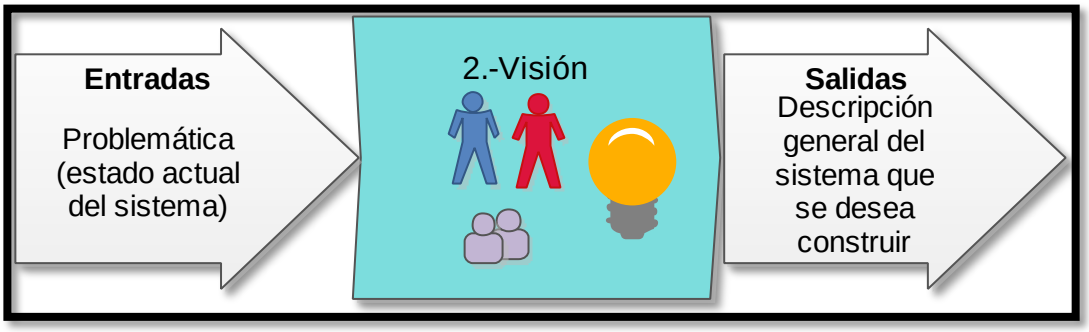


Figura 52. Visión

Objetivo:

- Definir el estado ideal del objeto conducido

Metas:

- Definir los objetivos del sistema conducido.
- Identificar si los objetivos están alineados a la estrategia de la organización.

Roles involucrados:

- Líder de proyecto.
- Dueño del producto.
- Usuarios.

Proceso (Layton, 2012):

1.- ***Desarrollar el objetivo del producto***. Respondiendo a las siguientes preguntas:

- Sobre el producto
  - ¿Cómo el producto va a apoyar a la empresa o estrategia de la organización?
  - ¿Qué estrategias específicas de la empresa se van a cumplir?
- Sobre el cliente
  - ¿Quién usará el producto?

- Sobre la necesidad detectada:
    - ¿Por qué necesita el cliente el producto?
    - ¿Qué características son fundamentales para el cliente?
- Competencia
    - ¿Cómo se compara el producto con productos similares?
- Diferenciación
    - ¿Qué diferencias tiene con la competencia?

2.- Crear la visión del producto

Se puede usar la siguiente plantilla (Chasm, 1991)

PARA: los clientes del Banco XYZ **(clientes objetivo)**
QUIENES: quieren tener acceso desde sus móviles a la banca en línea **(necesidad)**
LA: aplicación *banca móvil YYY* por el Banco XYZ **(nombre de la aplicación)**
ES UNA: aplicación móvil que puede ser descargada desde la Apple Store y el Market de Android, para ser utilizada en los teléfonos inteligentes y las tabletas **(categoría del producto)**
A DIFERENCIA DE: la banca tradicional, la banca móvil permite operaciones desde la casa o la oficina **(competidores)**
NUESTRO PRODUCTO PERMITE: a los usuarios acceso inmediato las 24 hrs del día para realizar operaciones financieras desde lugares donde exista internet o servicio 3G **(beneficios)**
ESTO APOYA NUESTRA ESTRATEGIA DE LA COMPAÑÍA: para proporcionar servicios bancarios a los clientes de forma rápida, segura, y en cualquier lugar **(valor de la propuesta)**

Figura 53. Plantilla para elaborar la visión de un proyecto.

Consejos:

- La redacción debe estar en presente para imaginar que el producto está en uso.
- Evite generalidades en la visión, por ejemplo:
    - Hacer que los clientes estén contentos.
    - Vender más productos.
- Es preferible no utilizar tecnicismos, por ejemplo:
    - Usar la versión X de Java.
    - Crear una aplicación en Java.

3.- Validar la declaración de la visión con los participantes, respondiendo a las siguientes preguntas:

- ¿La visión proporciona una descripción convincente de cómo el producto cumple con las necesidades del cliente?
- ¿El objetivo es alcanzable?
- ¿La declaración es consistente con las estrategias de la empresa?

Se debe preguntar directamente a los interesados:

- ¿La visión incluye todas las necesidades que el producto debe cumplir?

Se debe preguntar el equipo de desarrollo:

- ¿Es claro el producto que se debe desarrollar?

El líder de proyecto debe entender la visión y determinar cuáles serán los posibles obstáculos y riesgos.

4.-Finalizar la declaración de la visión

Se debe entregar una copia a todas las partes interesadas, el Dueño del producto es responsable de esto.

4.3.3 Paso 3: Prescripción

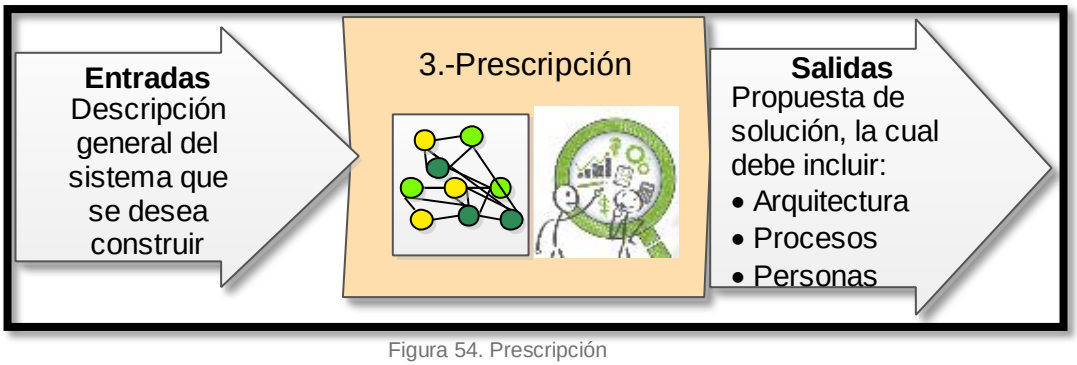


Figura 54. Prescripción

Objetivo:

- Identificar las distintas alternativas de solución
- Evaluar la viabilidad desde el punto de vista técnico y de negocio

Metas:

- Responder a la pregunta: ¿Qué curso de acción se deberá tomar?

Roles involucrados:

- Dueño del producto
- Líder de proyecto
- Usuarios
- Equipo de trabajo
- Expertos

Proceso:

Se recomienda usar la técnica de análisis de problemas propuesta por (Kepne & Tregoe, 1983), el cual propone los siguientes pasos:

1. Enunciar la decisión.
2. Desarrollar objetivos.
3. Clasificar objetivos en OBLIGATORIOS Y DESEADOS.
4. Dar peso a los objetivos DESEADOS.
5. Filtrar las alternativas a través de los objetivos OBLIGATORIOS.

6. Comparar alternativas contra los objetivos DESEADOS.
7. Identificar consecuencias adversas.
8. Tomar la elección mejor balanceada.

Para la toma de decisiones se debe considerar según las buenas prácticas de ITIL V3:

- Las necesidades del cliente.
- Las arquitecturas:
  - Hardware
  - Software
  - Información
  - Datos
  - Entornos
  - Aplicaciones
- Los procesos
  - De negocio
  - Técnicos
  - Administrativos

4.3.4 Paso 4: Instrumentación de la solución

Figura 55. Instrumentación de la solución

Objetivo:

- Elaboración de planes

Metas:

- Identificar la lista de requisitos
- Priorizar los requisitos
- Determinar las herramientas a utilizar
- Definir las políticas de trabajo
- Planear los recursos

Roles involucrados:

- Dueño del producto
- Líder de proyecto
- Usuarios
- Equipo de trabajo
- Expertos

Proceso:

1) *Identificar la lista de requisitos que el cliente requiere satisfacer:*

Se recomienda utilizar las historias de usuario para la recolección de requisitos del cliente:

Para la identificación de las características y temas, se sugiere el uso de *historias de usuario*:

¿Qué son? Una descripción simple de un requisito del producto

(Frente de la tarjeta)	(Reverso de la tarjeta)
TÍTULO: Transferir dinero entre cuentas *Nombre de la historia* **QUIENES:** Carol *Nombre de usuario o persona* **QUIERO:** Verificar los fondos de mis tarjetas y transferir dinero a cuentas de otras personas *Descripción de la acción* **DE MANERA QUE:** Pueda completar la transferencia y ver nuevamente los fondos de mis cuentas *Beneficio que se genera* Alexei	**Pruebas de aceptación** **CUÁNDO:** la transferencia es exitosa *Describir acción* **ENTONCES**: entrega un número de folio con la fecha y hora de la transferencia *Describir acción*

Se puede usar el siguiente formato:

Figura 56. Plantilla para la elaboración de una historia de usuario

Para entender cómo usar la historias de usuario se recomienda el libro *User Stories Applied - For Agile Software Develpment* (Cohn, User Stories Applied - For Agile Software Develpment, 2004).

¿Cómo saber si una historia está bien escrita?

Por medio del *método INVEST*, diseñado por Bill Wake, una historia debe:

I	**Independiente**	En la mediad de lo posible, una historia no debe depender de ninguna otra para ser implementada
N	**Negociable**	Una buena historia es negociable No es un contrato explícito para las características Una buena historia captura la esencial no los detalles
V	**Valor**	Demuestra el valor que el requisito le brinda a la empresa
E	**Estimable**	La historia debe ser descriptiva, concisa. Para que los desarrolladores puedan calcular el esfuerzo que realizarán.
S	**Pequeña**	La historia no debe ocupar más de la mitad de una iteración
T	**Comprobable**	Se puede validar fácilmente y los resultados son definitivos. Se debe cumplir la frase: "Entiendo lo que quiero por tanto podría escribir una prueba para el requisito". Si un cliente no puede escribir una prueba quiere decir que la historia no es lo suficientemente clara.

Figura 57. Método INVEST

2) *Estimación de las historias de usuario:*

Las estimaciones sólo son realizadas por el equipo de desarrollo, el dueño del producto sólo responde a las dudas que tenga el equipo, pero jamás debe calcular el esfuerzo y costo de las historias.

Termino	Descripción
Esfuerzo	Es la facilidad o dificultad de crear un requisito
Estimación	Puede ser un valor numérico (monetario o de tiempo) que indica la cantidad de esfuerzo aplicado
Priorizar	El valor de ese requisito con respecto a otros
Valor	Beneficios que un requisito o requisitos aportan a la organización
Dependencia	Un requisito es precursor de otro
Puntos de historia	Un punto de historia es una cantidad de tiempo relativa, asignada por el equipo de desarrollo, por ejemplo un punto de historia puede equivaler a un día de trabajo.

Tabla 10. Clasificación de las historias de usuario.

Para la estimación de historias de usuario existen dos técnicas:

El método más usado es el *Planning Poker:*

Pasos:

a) Proporcionar a cada miembro del equipo de desarrollo una baraja de cartas de póker. Son una serie de cartas enumeradas según la secuencia Fibonacci, para asignar la cantidad de tiempo que un equipo de desarrollo tarda en desarrollar una historia de usuario.

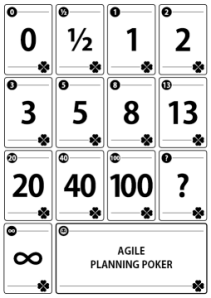


Figura 58. Ejemplo de una baraja de Planning Poker

b) A partir de una historia de usuario simple, los jugadores deciden cuánto vale para tomarla como línea base.

c) El dueño del producto lee una historia de usuario de alta prioridad para los jugadores.

d) Cada jugador elige una carta que representa su estimación del esfuerzo que involucra la realización y la colocan boca abajo.

e) Todos muestran la tarjeta a la vez.

f) Si los jugadores tienen distintos tipos de historia, es tiempo de discusión

-Los jugadores con las puntuaciones más altas y más bajas explican por qué esa puntuación.

-El dueño proporciona la información necesaria

-Se vuelve a votar (por lo general el proceso no se repite más de tres veces)

-Si no hay acuerdo el líder de proyecto sirve como mediador para determinar si la historia debe dividirse o debe definirse nuevamente

g) Se repiten todos los pasos para cada historia de usuario.

Participantes

El líder de proyecto coordina y es un facilitador

El dueño del producto proporciona información acerca de las historias, no estima.

Los desarrolladores estiman el nivel de esfuerzo requerido para las historia de usuario.

Consejos

- Recuerde que las estimaciones incluyen: diseño, programación, pruebas e integración.
- Un equipo de desarrollo gasta aproximadamente 10% de su tiempo estimando y resstimando.
- Este método sirve cuando se tienen hasta 20 historias de usuario

Inconvenientes

Si se tienen muchas historias el resultado es lento.

Para entender mejor la planeación por Poker se recomienda el libor Agile Estimating and Planing (Cohn, 2005).

La segunda manera de estimar es por medio del método Affinity y se utiliza cuando se tienen más de 20 historias de usuario.

Pasos:

a. Sin tomar más de un minutos los desarrolladores tabulan la historias en:

Tamaño	Puntos
Una historia Epica (no puede ser incluida en el TimeBox)	8 pts
Una historia grande	5 pts
Una historia mediana	3 pts
Una historia pequeña	8 pts

Tabla 11. Ejemplo de asignación de puntos a histotiras de usuario

b. Sin tomar más de 30 segundos por historia los desarrolladores colocan las historias según su tamaño, se puede utilizar un muro para hacer esto.

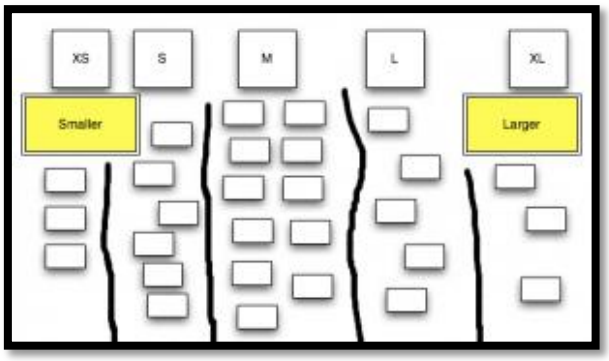


Figura 59. Clasificación de historias de usuario según el método Affinity

c. Tomando máximo 60 min por cada 100 historias, se realizan y discuten los cambios entre las historias.

d. El propietario del producto revisa la categorización y realiza los comentarios pertinentes.

e. Cuando la estimación del equipo de desarrollo difiere en más de un tamaño se debe discutir la historia.

3) *Priorizar los requisitos:*

Se recomienda usar el método MoSCoW.

Los requisitos deben priorizarse según las siguientes categorías:

M	**Must (Debe tener)**	Requisito que tiene que estar implementado en la versión final del producto No son negociables, si al menos uno no se entrega el proyecto se considera un fracaso.
S	**Should (De ser posible deben ser incluidos)**	Requisito de alta prioridad que en la medida de lo posible debería ser incluido en la solución final, pero que llegado el momento y si fuera necesario, podría ser prescindible si hubiera alguna causa que lo justificara.
C	**Colud (Podría)**	Requisito deseable pero no necesario, se implementaría si hubiera posibilidades presupuestarias y de tiempo.
W	**Won´t (De ninguna manera)**	Hace referencia a requisitos que están descartados para el proyecto actual pero que en versiones siguientes puede ser incluido.

0. Método MoSCoW

4) *Determinación de los plazos de alto nivel de los requisitos*

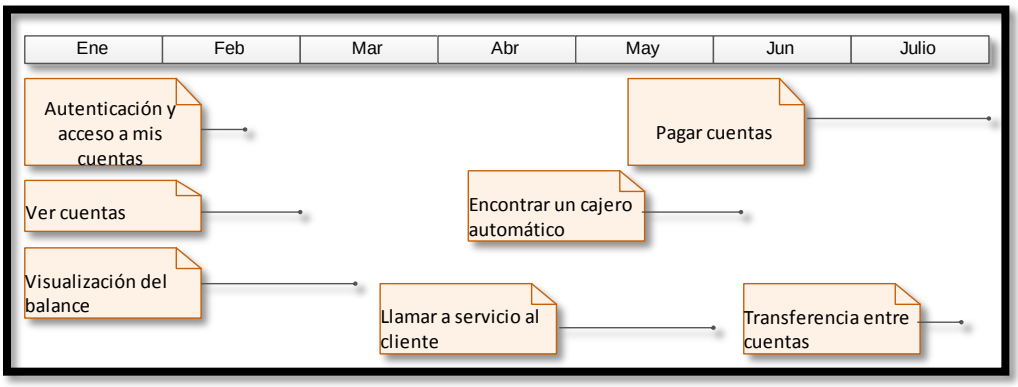


Figura 61. Ejemplo de un calendario de alto nivel

5) Crear una tabla con la lista de requisitos según su prioridad:

ID	Historia de usuario	Tipo	Status	Valor
121	Como administrador: quiero poder vincular cuentas de usuario con sus perfiles, para que los clientes puedan acceder a sus nuevas cuentas mediante su celular	Historia	No iniciada	5
113	Como cliente: quiero ver los balances de mis cuentas, para saber cuánto dinero tengo en cada cuenta.	Historia	No iniciada	3
97	Como visitante: quiero contactar al banco, para realizar preguntas sobre diferentes temas	Historia	No iniciada	2

Tabla 12. Ejemplo de una lista de requisitos.

6) Determinará las herramientas que serán utilizadas para el desarrollo del producto y la forma en la que será dividida para la pila de requisitos según su prioridad.

Selección de herramientas

Se recomienda utilizar herramientas como pizarras, post-it, rotafolios, plumones, hojas de colores, cuando se tiene la posibilidad de tener a todo el equipo junto, en caso de no ser así se recomienda: el uso de videoconferencias, mensajería en tiempo real, redes sociales de trabajo y wikis[16].

Para seleccionar el uso de una herramienta se debe analizar poniendo atención a los siguientes puntos:

- ¿Cuál es el propósito de la herramienta?
- ¿Qué problemas específicos resuelve?
- No elegir algo más complicado de lo que se necesita
- No invertir en herramientas costosas antes de tener suficiente madures en el uso de técnicas ágiles.
- Antes de adquirir una herramienta se debe verificar si se cuenta con los recursos necesarios para trabajar con ella.

7) Planeación de recursos:

Se identificarán:

a) Los suministros: materiales, aprovisionamientos, energía y servicios.
b) Las instalaciones y el equipo.
c) El personal, que participará directamente en el proyecto.
d) El dinero, que serán necesario para la implantación del proyecto.

[16] Un Wiki (del hawaiano wiki wiki, «rápido») es un sitio web colaborativo que puede ser editado por varios usuarios. Los usuarios de una wiki pueden crear, editar, borrar o modificar el contenido de una página web, de una forma interactiva, fácil y rápida; dichas facilidades hacen de una wiki una herramienta efectiva para la escritura colaborativa.

4.4.5 Paso 5.1: Plan del TimeBox (Sprint)

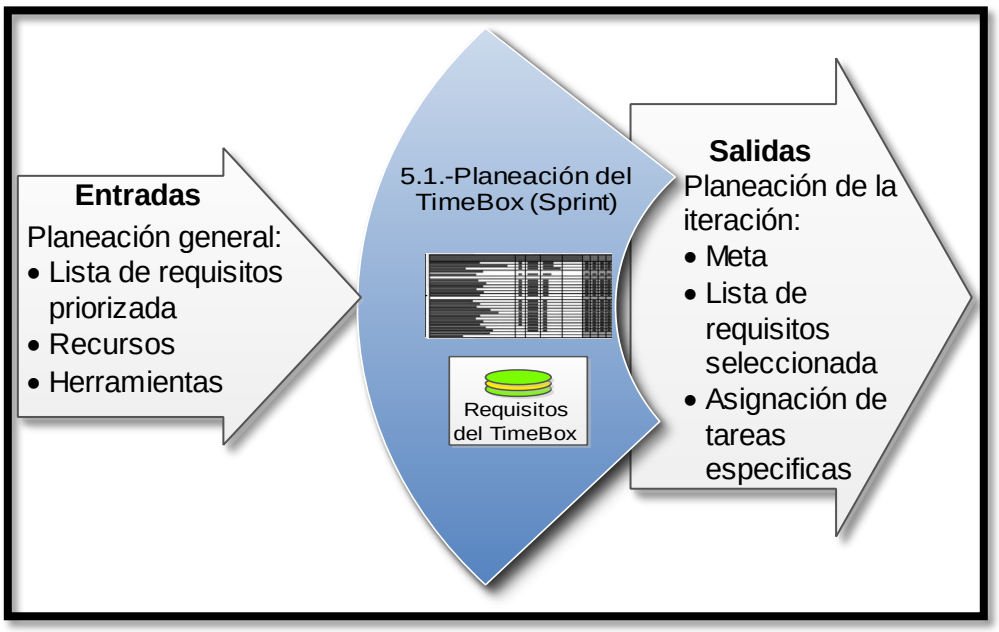


Figura 62. Planeación del TimeBox

Objetivo:

- Establecer los objetivos y las tareas que serán realizadas en la iteración

Metas:

- Realizar la planeación del TimeBox, debe contener:
  - La meta del TimeBox
  - Las tareas a realizar
  - ¿Cómo se completarán esas tareas?

Roles involucrados:

- Dueño del producto
- Líder de proyecto
- Equipo de trabajo

Requisitos previos:

Lista de requisitos o backlog:

- Lista de historias priorizadas.
- La estimación de esfuerzo relativo de cada historia.
- Las tareas necesarias para desarrollar cada historia.

El esfuerzo en horas para completar cada tarea. Las tareas deben ser máximo de 1 día

Duración:

La planificación del TimeBox se calcula de 2 horas por cada semana de trabajo

Proceso:

La reunión se divide en dos:

1.- Establecimiento de objetivos y selección de historias

a. Identificar el objetivo del Time Box, por ejemplo:

Como cliente de la banda móvil quiero:

- Acceder a mi cuenta
- Ver los saldos de la cuenta
- Ver transacciones anteriores

b. Revisar las historias de usuario y las estimaciones relativas, por ejemplo:

- Iniciar sesión a mis cuentas
- Ver saldos de cuenta
- Ver transacciones pendientes
- Ver transacciones anteriores

c. Determinar lo que el equipo puede desarrollar en el sprint

2.- Creación de tareas para para la lista de requisitos seleccionados para el TimeBox

a. Romper las historias de usuario en tareas, las cuales deben abarcar: diseño, desarrollo, codificación, pruebas e integración. Por ejemplo:

Historia de usuario	Iniciar sesión en mis cuentas	Horas
Tareas	• Crear una pantalla de autenticación de un usuario y contraseña, con un botón enviar	10
	• Crear una pantalla de error para que el usuario vuelva ingresar sus datos	16
	• Crear una pantalla que de la bienvenida al usuario	5
	• Refactorizar código para dispositivos móviles	20

Figura 63. Desagregación de una historia de usuario

Para la definición de tareas se recomienda usar el Método SMART.

S	**Especifica**	Todo el mundo debe entender de qué se trata lo que se debe hacer
M	**Medible**	Se debe preguntar ¿Se sabe qué debe hacer para que se diga que está terminada? ¿Qué pruebas debe pasar? ¿El código esta revisado?
A	**Realizable**	El equipo de trabajo debe estar seguro que puede realizarla
R	**Relevante**	Debe ser relevante para que el cliente la priorice
T	**TimeBox**	Debe tener una duración especifica

Figura 64. Método SMART

b. Revise nuevamente que el equipo pueda completar las tareas en el tiempo disponible del TimeBox

c. Cada miembro del equipo debe seleccionar las tareas a cubrir

Ejemplo de la planeación de un TimeBox:

La aplicación XYZ para banca móvil – TimeBox 1 **Fecha del TimeBox:** 14 al 25 de julio de 2014	
Objetivo: Como cliente del banco, quiero ingresar a mi cuenta, y poder ver mis balances y las transacciones pendientes.	
Número de días a trabajar:	9
Alexei (35 hrs)	63
Enrique (35 hrs)	63
Lidia (35 hrs)	63
Total	189

Tarea	Prioridad	Status	Responsable	Aprobado	L	Ma	M	J	V	L	Ma	M	J	V
Historia de usuario: Iniciar sesión en mis cuentas														
Crear una pantalla de autenticación de un usuario y contraseña, con un botón enviar	1	Completada	Alexei		3	3	4	0	0	0	0	0	0	0
Crear una pantalla de error para que el usuario vuelva ingresar sus datos	3	Completada	Lidia		4	3	4	5	0	0	0	0	0	0
Crear una pantalla que de la bienvenida al usuario	2	En proceso	Enrique		2	2	1	0	0	0	0	0	0	0
Refactorizar código para dispositivos móviles	1	En proceso	Alexei		0	0	0	4	4	4	4	4	0	0

Figura 65. Ejemplo de la planeación de un TimeBox

4.4.6 Paso 5.2: Reuniones diarias y TimeBox

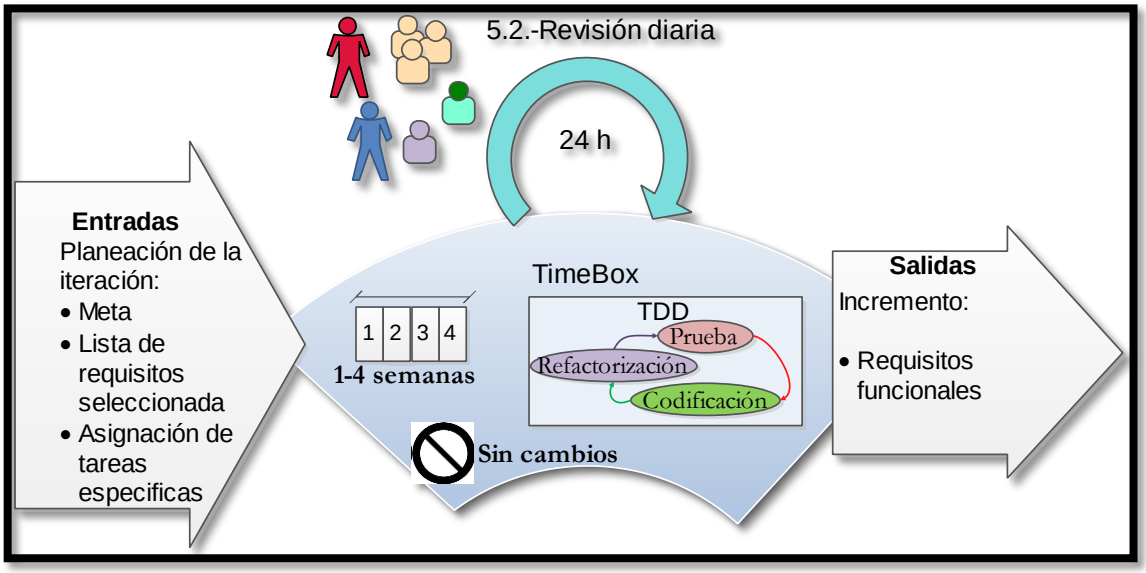


Figura 66. Reuniones diarias

Objetivos de las reuniones diarias:

Hacer una reunión de 15 min y responder a las preguntas:

- ¿Qué hice ayer?
- ¿Qué haré hoy?
- ¿Qué me impide cumplir con mis objetivos?

Meta:

- Sirve para coordinar las actividades del día

Roles involucrados:

- Equipo de desarrollo

Duración:

15 minutos

Proceso

1. Todos se reúnen en un mismo lugar

2. Cada miembro del equipo responde a tres preguntas

- ¿Qué hice ayer? Ayer termine…
- ¿Qué haré hoy? Hoy voy a trabajar en…
- ¿Qué me impide cumplir con mis objetivos? Mis impedimentos son… (si existen)

3. Pautas a seguir para reuniones cortas

- Cualquiera puede asistir a la reunión pero solo el equipo de desarrollo y el dueño pueden hablar
- La atención debe estar centrada en las prioridades inmediatas
- Las reuniones son de coordinación y no de resolución de problemas
- Para evitar resolver problemas en la reunión
    - Anote los problemas en una pizarra
    - Resuelva los problemas solo con los involucrados después de la reunión

4. Seguimiento de progreso

Existen dos técnicas para seguir el progreso de un proyecto de software:

Gráfico burndown: Un diagrama burndown o diagrama de quema es una representación gráfica del trabajo por hacer en un proyecto. Usualmente el trabajo (puntos de historia u horas de trabajo) se muestra en el eje vertical y el tiempo en el eje horizontal (por lo general en días). Es decir, el diagrama representa una serie temporal del trabajo pendiente. Este diagrama es útil para determinar cuánto trabajo queda por hacer.

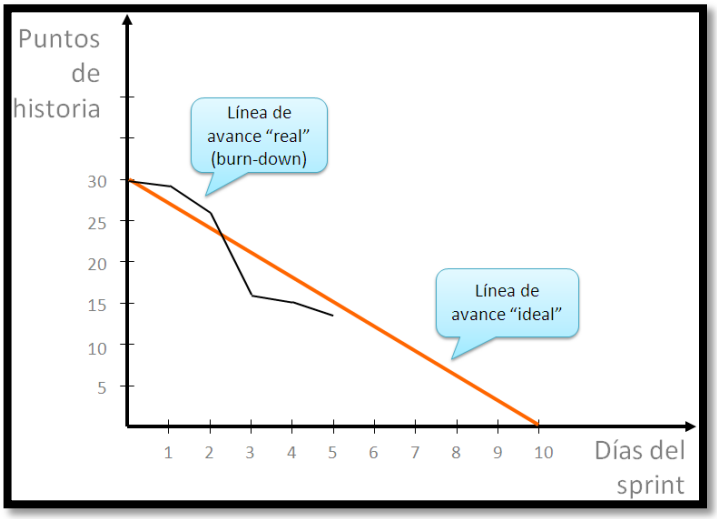


Figura 67. Ejemplo de un gráfico Burndown

- La línea naranja muestra el avance ideal que debe tenerse
- La línea negra muestra la realidad

¿Para qué sirve?

- Sirve para saber cómo está trabajando el equipo.
- Si la línea real está sobre la línea ideal, indica que tal vez hay exceso de historias y el cliente podría reducir la carga de trabajo.
- Al contrario si la línea real está debajo de la ideal quiere decir que se requieren agregar más historias

Lo que importa en un proyecto es saber cuánto falta no cuanto se ha invertido. Todos los gráficos utilizados deben estar en un lugar visible.

El Burndown debe actualizarse a diario.

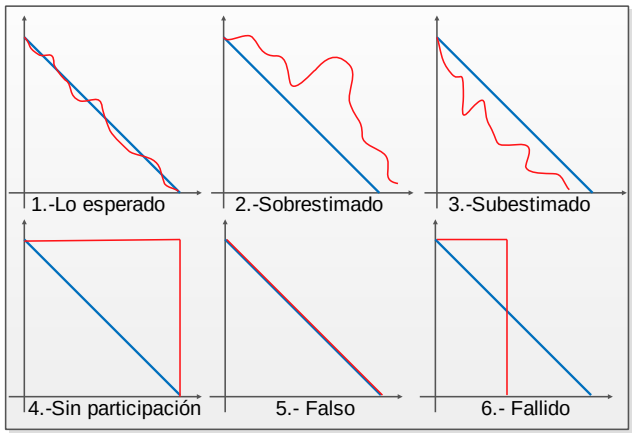


Figura 68. Gráficos burndown más comunes

I.- *Lo esperado*: así es como se espera que se comporte un gráfico burndown, esto indica que el equipo de desarrollo lo ha actualizado a diario, y las estimaciones fueron correctas.

II.-*Sobrestimado*: existen más historias de usuario de las que el equipo de desarrollo puede manejar, el líder de proyecto debe renegociar con el dueño del producto y determinar por qué no se está cumpliendo con los objetivos.

III.- *Subestimado:* durante la planeación del TimeBox eligieron menos historias de las que el equipo puede manejar, por tanto se puede renegociar para agregar más historias y realizar una entrega mayor.

IV.-*Sin participación:* el equipo de desarrollo nunca actualizó el gráfico, se debe determinar cuál fue la causa y si realmente se cumplió con los objetivos.

V.- *Falso:* ningún proyecto tiene un avance lineal, siempre existen pequeños picos, un gráfico como este indica que el equipo de trabajo está mintiendo o nunca lo actualizo.

VI.- *Fallido*: gráfico típico de un proyecto cancelado a la mitad de un TimeBox.

Otra técnica para visualizar el trabajo que se está realizando, es el *tablero Kanban.*

El tablero debe estar en un lugar visible para todos y debe contener las siguientes columnas:

- *Por hacer:* son todas las tareas del TimeBox que están pendientes de desarrollarse.
- *En progreso*: son las tareas que se están desarrollando. Si existen más de dos tareas quiere decir que no se está trabajando de manera conjunta.
- *Verificar:* son las tareas que están listas y deben ser revisadas por el dueño
- *Liberado: t*areas que el dueño a verificado

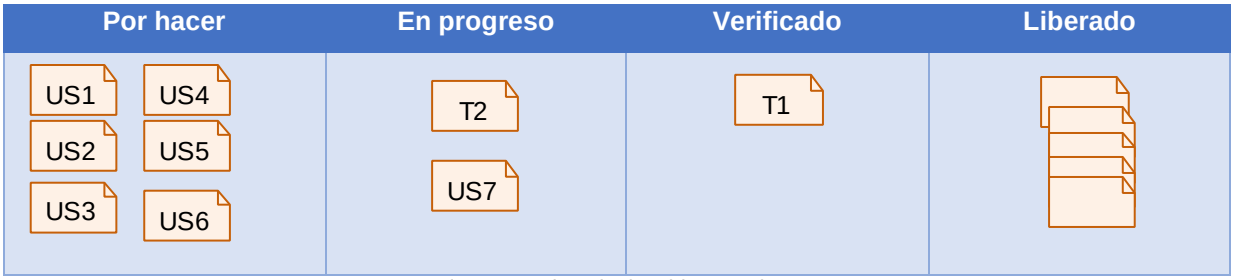


Figura 69. Ejemplo de tablero Kanban

Responsabilidad de los roles:

Equipo de desarrollo:

- Seleccionar las tareas de mayor prioridad y completarlas lo antes posible.
- Solicitar aclaraciones al dueño del producto cuando existan dudas sobre una historia de usuario.
- Llevar acabo revisiones por pares sobre el trabajo de otros.
- Colaborar con otros miembros del equipo de desarrollo para diseñar el enfoque de un caso de usuario específico.
- Desarrollo de la toda la funcionalidad, según lo acordado en la decisión de las tareas.
- Informe de actividades diario.
- Alertar al líder de proyecto sobre cualquier obstáculo que no pueda eliminar por cuenta propia.
- Cumplir con el objetivo del TimeBox.

Dueño del producto:

- Hacer las inversiones necesarias para mantener la velocidad de desarrollo a un alto nivel.
- Informar sobre el costo a las partes interesadas del producto.
- Elaborar las historias de usuario con el equipo de desarrollo para que estos últimos comprendan claramente lo que se desea obtener.
- Proporcionar aclaraciones inmediatas acerca de los requisitos solicitados.
- Revisar la funcionalidad completa de las historias de usuario.
- Añadir nuevas historias de usuario según sea necesario.
- Garantizar que las nuevas historias de usuario apoyen la visión del producto, el objetivo de liberación y la meta del TimeBox.

Líder de proyecto:

- Debe defender los valores y las prácticas agiles, entrenando al dueño del producto, al equipo de desarrollo y a la organización.
- Proteger al equipo de desarrollo de distracciones.
- Quitar obstáculos, tanto tácticamente para problemas inmediatos, como estratégicamente para problemas a largo plazo.
- Facilitar la creación de consenso dentro del equipo de desarrollo.

- Construir relaciones para fomentar una estrecha relación con las personas que trabajan con el equipo de desarrollo.

El TimeBox

Objetivo:

Analizar, diseñar, programar y probar los requisitos seleccionados para la iteración.

Se recomienda el uso del Desarrollo Basado en Pruebas:

Definición:

Es una técnica de diseño e implementación de software incluida dentro de la metodología XP, FDD no cubre todo el proceso de desarrollo de software, se centra en las fases de diseño y construcción del software. FDD se centra en tres pilares (Jurado, 2010):

1. La implementación de las funciones justas que el cliente necesita y no más.
2. La minimización del número de defectos que llegan al software en fase de producción.
3. La producción modular, altamente reutilizable y preparado para el cambio.

TDD responde a las preguntas:

¿Cómo lo hago?, ¿Por dónde empiezo?, ¿Cómo sé qué es lo que hay que implementar y lo que no?, ¿Cómo escribir un código que se pueda modificar sin romper su funcionalidad existente?

TDD tiene siete pasos:

1. Elegir un requisito: Se elige de una lista el requerimiento que se cree que nos dará mayor conocimiento del problema y que a la vez sea fácil de implementar.
2. Escribir una prueba: se comienza escribiendo una prueba para el requisito. Para ello el programador debe entender claramente las especificaciones y los requisitos de la funcionalidad que está por implementar.
3. Verificar que la prueba falle: Si la prueba no falla es porque el requerimiento ya estaba implementado o porque la prueba es errónea.
4. Escribir la implementación: Escribir el código más sencillo que haga que la prueba funcione.
5. Ejecutar las pruebas automatizadas: Verificar si todo el conjunto de pruebas funciona correctamente.
6. Eliminación de duplicación: el paso final es la refactorización, que se utiliza principalmente para eliminar el código duplicado. Se hacen pequeños cambios y luego se corren las pruebas hasta que funcione correctamente.
7. Actualización de la lista de requisitos: se actualiza la lista de requisitos, tachando el requisito implementado. Asimismo se agregan requisitos que se hayan visto como necesarios durante este ciclo y se agregan requerimientos de diseño.

4.4.7 Paso 5.3: Revisión del TimeBox

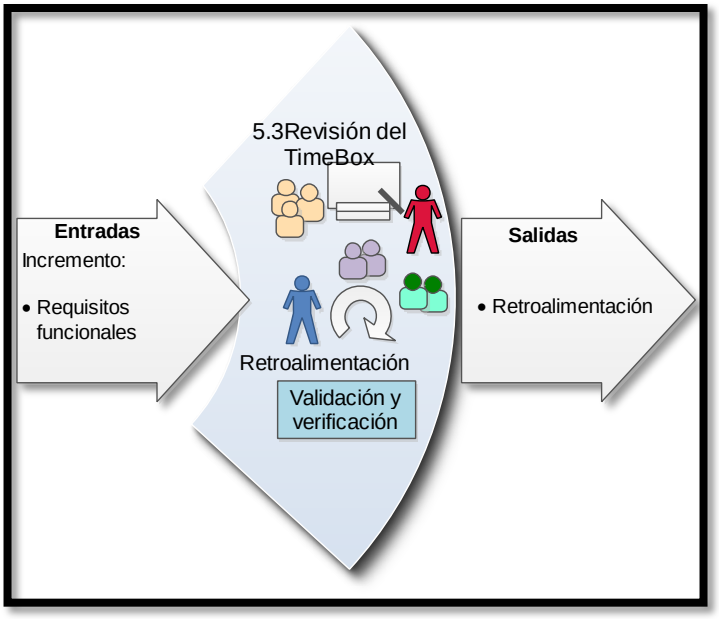


Figura 70. Revisión del TimeBox

Objetivos:

- Mostrar el trabajo realizado
- Historias de usuario realizadas.

Roles involucrados:

- Equipo de desarrollo
- Dueño del producto
- Líder del proyecto

Tiempo de preparación para la presentación:

- Menos de una hora

Duración de la reunión:

Por cada semana que dure el TimeBox se agrega como máximo una hora a la reunión.

Recomendaciones:

- No hay diapositivas en Power Point.
- Todo el equipo de desarrollo debe participar.
- Cualquiera puede asistir a una reunión de revisión.
- El dueño presenta el objetivo de la liberación, la meta sprint, las características que debieron haberse desarrollado.
- Los interesados pueden hacer preguntas y proporcionar información sobre un producto determinado.

- El dueño puede indicar lo que seguirá.

Tomar notas:

El líder de proyecto y el Dueño del producto toman notas, por ejemplo:

- La aplicación acepta caracteres especiales
- Se podría personalizar la aplicación guardando las preferencias del usuario

Productos obtenidos:

- Liberación de un incremento.

4.4.8 Paso 5.4: Retrospectiva del TimeBox

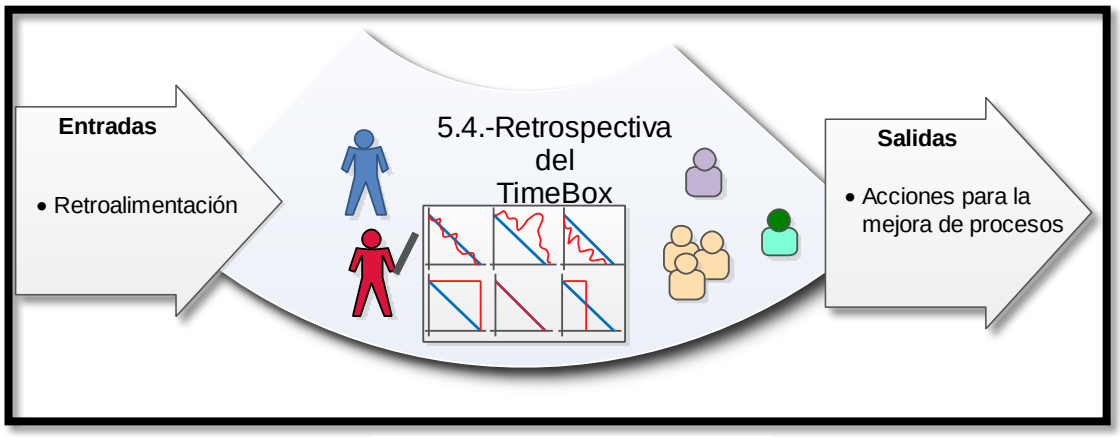


Figura 71. Retrospectiva del TimeBox

Objetivo

- Responder a las preguntas:
  - ¿Qué se hizo bien?
  - ¿Qué se podría mejorar?
- Definir mejoras en los procesos

Metas:

Determinar que mejoras pueden implantarse para que el equipo de trabajo sea eficaz y efectivo.

Roles involucrados:

- Equipo de desarrollo
- Dueño del producto

Descripción de la reunión:

Una retrospectiva es un encuentro de todo el equipo donde se analiza que salió bien durante el TimeBox y cuáles son las posibles mejoras que se pueden realizar.

Duración de la reunión:

No más de 45 min por cada semana de duración del TimeBox.

Preguntas clave:

- ¿Qué salió bien durante el Sprint?
- ¿Qué cambiarias y cómo?

Temas de discusión:

- Resultados obtenidos
- Relaciones: comunicación, cohabitación, trabajo en pareja.
- Procesos: desarrollo, revisión de código.
- Herramientas: ¿son adecuadas las herramientas utilizadas?
- Productividad: ¿Cómo se puede mejorar la productividad?

Proceso para una reunión de retrospectiva (Derby & Larsen, 2006):

1. *Prepare el escenario*

   Tener objetivos específicos de discusión
2. *Reunir datos*

   Discutir lo que salió bien y lo que se puede mejorar
3. *Generar ideas*

   Revisar los datos del TimeBox finalizado y aportar ideas de mejora
4. *Decidir qué hacer*

   Que acciones específicas se proponen para mejorar
5. *Cierre*

4.4.9 Paso 6: Liberación de un incremento

Figura 72. Liberación de un incremento.

Objetivo:

Indicar:

- Si los requisitos han sido cubiertos.

- Si se descubrió un nuevo requisito.
- Si se finalizó un incremento

Roles involucrados:

- Líder de proyecto
- Dueño del producto

Proceso:

- Si los requisitos han sido cubiertos: se pasa a la fase 7.
- Si se descubrió un nuevo requisito: se pasa la fase 3 para actualizar la hoja de ruta del producto.
- Si se finalizó un incremento: se pasa a la fase 4 para determinar cuál será el siguiente incremento.

4.10 Paso 7: TimeBox de liberación del producto

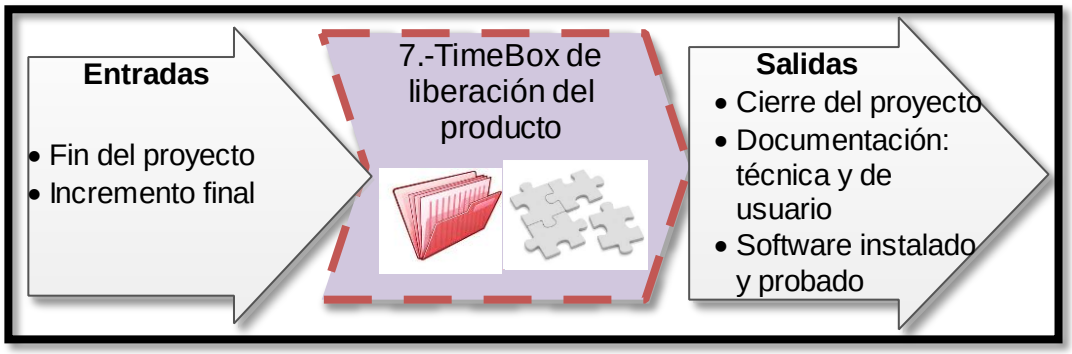


Figura 73. TimeBox de liberación del producto

Objetivos:

- Finalizar la documentación
- Integrar el producto en la organización
- Realizar pruebas finales

Roles involucrados:

- Dueño del producto
- Líder de proyecto
- Equipo de trabajo

Descripción:

- Es un TimeBox exclusivo para el cierre del proyecto.
- Finalización de procesos normativos.
- Notas finales de cambios al producto.
- No se desarrolla ningún requisito más.

Tipos de documentación:

Técnica:

- Debe ser sólo la esencial, por si un nuevo equipo de trabajo desea actualizar o cambiar el producto.
- Se crea durante todo el proyecto.

De usuario:

- Se debe esperar hasta terminar el producto para desarrollar los manuales de usuario, debido a los cambios que surgen durante la creación del producto.

4.4 Contratos

Los contratos son parte del medio ambiente del proyecto, pero se decidió colocarlos en este capítulo debido a que la descripción del tipo de contrato que se sugiere utilizar, se basó en el método desarrollado.

El manifiesto ágil indica en su valor número 3 la ***COLABORACIÓN CON EL CLIENTE SOBRE LA NEGOCIACIÓN CONTRACTUAL.*** Lo que plantea el manifiesto, es que siempre debe existir una buena relación entre cliente y proveedor, se debe pensar en términos de ganar-ganar. Pero para tratar de garantizar que los acuerdos se cumplan se debe crear un contrato, en el cual se establecen un conjunto de reglas por escrito que delimiten la forma en la que se trabajará en el proyecto.

En teoría, estas reglas se acuerdan libremente por ambas partes para crear condiciones óptimas que permitan completar con éxito el proyecto. En la práctica, los contratos se suelen ver como juegos competitivos, cuyo objetivo es dejar en desventaja al otro. Muchas organizaciones grandes y gobiernos tienen condiciones comunes que se deben aceptar en conjunto como prerrequisitos para hacer negocios con ellos.

Un contrato debe distribuir el riesgo y reflejar la confianza entre las partes. Debe responder a las preguntas:

- ¿Qué pasa cuando algo sale mal?
- ¿Quién paga cuando el proyecto es más difícil de lo que se esperaba?
- ¿Quién se beneficia si el proyecto acaba antes de lo previsto?
- ¿Cómo está estructurado? ¿Cuáles son las reglas básicas para el alcance de las entregas y la factura por ingresos?
- ¿Cómo se reparten los riesgos y los beneficios entre cliente y proveedor?
- ¿Cómo se gestionan los cambios a los requerimientos?

Según (Bermejo, 2012) algunos puntos que siempre deben aparecer en un contrato son:

1) Objetivos del proyecto y de la cooperación entre las organizaciones.
2) Un resumen de la estructura del proyecto: procesos de Scrum, ciclo de vida y funciones, clave, entre otros.
3) Personal clave: quién es responsable a escala operacional y jerárquica.
4) Pagos y facturación, incluyendo las cláusulas de bonos y penalizaciones.
5) Finalización normal y finalización antes de la fecha fijada en el contrato.
6) Detalles legales. En función de las leyes locales y costumbres legales, es posible que tenga que limitarse la responsabilidad civil, especificar la garantía o incluir otros textos que se necesitan legalmente.

Los puntos 2, 3, 4 y 5 determinan las reglas de juego para el proyecto. Si son las correctas, se tendrá una base sólida para un buen proyecto.

¿Es necesario incluir el alcance del proyecto en un contrato?

A pesar de que a menudo está presente, fijar el alcance en el contrato también lo hace inflexible. Si fuera posible, es mejor especificar cómo se va a gestionar el alcance (por ejemplo, con una lista de producto, contratos por *TimeBox* o *sprint*) y dejar los detalles operacionales fuera para que los gestione el equipo del proyecto.

Tipos de contratos

El contrato más común es el de *precio fijo / alcance fijo* (***TABLA 13***), este tipo de contrato no es útil para proyectos de software ya que toma como base el modelo en cascada, donde se fija el alcance y los cambios son difíciles de llevar acabo.

Precio fijo / Alcance fijo
Estructura:
• Se acuerdan los entregables desde el inicio. • Se entregan. • Se realiza el pago. • A la mayoría de los clientes les gusta este tipo de contrato porque les da la sensación de seguridad.
Cambios de alcance:
• El proceso de solicitud de cambios está pensado para limitar el alcance de los cambios. • Este proceso es costoso y los cambios en general no se pueden prever. • Como el cliente casi por definición quiere más alcance, resulta difícil acabar el proyecto. • El proveedor quiere que el cliente esté contento, por lo que el proveedor suele ceder. • La palabra etcétera es muy peligrosa para definir especificaciones en un requerimiento de precio fijado.
Riesgo:
• El riesgo es para el proveedor. Si la estimación es incorrecta, el proyecto pierde dinero. • Otros riesgos menos obvios surgen del juego de petición de cambios, en el que el proveedor negocia ingresos adicionales para cambios en el alcance. Si el proveedor subestima mal el esfuerzo adicional, o da un precio irreal muy bajo, las pérdidas pueden llegar a afectar a la propia existencia del proveedor, lo que también es un problema para el cliente.
Consejo:
Especificar los requerimientos funcionales con historias de usuario.

Tabla 13. Contrato precio fijo.

Para el desarrollo de proyectos basados en el modelo iterativo e incremental (Lacey, 2012) propone un tipo de contrato denominado *Rangos y Cambios*.

Rangos y cambios

Estructura:

Se manejan dos contratos:

1) El contrato de diagnóstico: que da al equipo la información que necesita para escribir un contrato sobre el proyecto completo. Se obtiene como resultado:

    - Determina los rangos.
    - Determina el costo y el cronograma de alto nivel.

2) El contrato del proyecto: incluye las cláusulas para proteger al cliente y al proveedor.

Contrato de diagnóstico:

Esta fase tiene una tarifa fija, con un único objetivo:
Que el equipo de trabajo pueda reunir suficiente información sobre el proyecto para determinar:

- Lo que puede ofrecer, una lista o pila de producto (lista de requisitos).
- En cuanto tiempo lo puede desarrollar (un plan de lanzamiento o sprint inicial).
- Cuánto cuesta cada sprint o caja de tiempo.

El contrato por lo general incluye a dos o tres miembros del equipo y tiene una duración de dos a cuatro semanas.

El primer resultado es la construcción de la cartera de productos, lo cual se traduce en un proceso de tres pasos:

1) *Identificar los tipos de usuarios o personajes*: el equipo debe determinar quiénes son los usuarios del sistema, cuáles son sus intereses, cómo van a utilizar el sistema, y cuáles son sus expectativas. Esto ayuda a las personas que escriben las historias a entender quiénes son los usuarios y poder crear una visión más centrada en el usuario. Se recomienda entrevistar al mayor número de usuarios del sistema como sea posible para comprender sus interacciones actuales, lo que les gusta y disgusta de la forma de trabajo actual, y cuál sería su mayor prioridad.

2) *Escribir historias de usuario*: el equipo de trabajo junto con el dueño del producto es responsable de escribir las historias de usuario. Es recomendable invitar a los clientes al taller historia de usuario junto con el equipo. Los miembros del equipo y los representantes de los clientes pueden intercambiar ideas y escribir historias.

3) *Estimar las historias*: el equipo de desarrollo prioriza la historias de usuario y estima los puntos de historia. Esto ayuda a las partes interesadas y a los clientes a comprender el costo de cada historia. El dueño del producto y los usuarios deben estar presentes durante estos tres pasos.

Costo y plan de lanzamiento

Una vez que el equipo ha trabajado con el dueño del producto y los usuarios para obtener información, el equipo trabaja de forma independiente para determinar el costo y el plan de lanzamiento. Este es un proceso de cuatro pasos:

1. *Determinar* la *velocidad del equipo*: cuántos puntos de historia puede trabajar por sprint. Si el equipo trabaja por primera vez de esta manera se debe dejar un rango considerable, ya que la velocidad del sprint se obtendrá hasta después del segundo sprint.
2. *Calcular el costo por sprint*: fijar el periodo de cada sprint (entre una a cuatro semanas), y en base a la velocidad calcular el costo.
3. *Construir un plan de lanzamiento*: el plan de alto nivel que indica cuanto tiempo tardará el desarrollo del proyecto.
4. *Establecer las opciones de pago*: un ejemplo puede ser pago por sprint.

Entregables de la fase de diagnóstico

- Los usuarios o personajes identificados para el sistema.
- La pila de requisitos priorizada.
- Un plan de lanzamiento.
- El contrato del proyecto.

Contrato del proyecto

En la fase final del diagnóstico se realiza un contrato de proyecto el cual hace referencia a la lista de requisitos y al plan de lanzamiento.

En el contrato se indica:

- El número aproximado de puntos de historia por sprint.
- El costo de los puntos de historia o del sprint.
- La forma de trabajo.
- La forma de entregar los productos al dueño del producto.
- Las condiciones en las cuales el dueño del producto puede rechazar un entregable.
- La forma de manejar los cambios.

Cambios
Los contratos pueden incluir:

- Aceptar la reorganización de la lista de requisitos, el dueño del producto puede cambiar la prioridad sin costo extra.
- Aceptar añadir nuevas historias, siempre y cuando el número total de puntos de historia de la lista de requisitos siga siendo la misma y el total de puntos de historia por sprint no sobrepase el rango determinado de lo que el equipo pueda entregar.

En una conferencia en 2008, Jeff Sutherland presento “Money for Nothing (cambios de forma gratuita)”, en donde sugiere incluir una cláusula denominada “Cambios de forma gratuita”, en esta cláusula se agregan los puntos anteriores, estipulándolos de la siguiente manera:

- El dueño del producto debe estar trabajando en todo momento con el equipo de desarrollo.
- El dueño del producto es responsable de asignar prioridad a la lista de requisitos.
- El dueño del producto puede añadir nuevas historias sin cargo, siempre y cuando se quite una historia de similar valor. Para entender este concepto se puede usar el ejemplo de una botella que puede contener un litro de cualquier líquido: agua, aceite, vinagre, crema, etc. Al inicial el proyecto el cliente decide agregar aceite y vinagre, durante el transcurso del proyecto el dueño del producto puede añadir sin costo agua a la botella, siempre y cuando se quite una cantidad igual de aceite de oliva o vinagre blanco. De lo contrario la botella se desbordará.

Claves para el éxito

Para completar el contrato se deben añadir cuatro elementos fundamentales:

1. *Disponibilidad del dueño del producto:* se debe incluir un número determinado de horas semanales en donde el dueño se comprometa a estar disponible. Los sprints cortos requieren disponibilidad casi diaria del cliente. Si el sprint es largo, se puede caer en que el cliente solo este al principio o al final del sprint. Se recomiendan sprints cortos para mantener involucrado al cliente.

2. *Ventana de aceptación:* es la cantidad de tiempo que tiene el cliente para aceptar la funcionalidad entregada de un sprint. La ventana más corta que se puede dar, es que el producto se acepte durante la *reunión de revisión* o *X número de horas después de la reunión de revisión*, en la cláusula se debe incluir que si el cliente no da una respuesta en el tiempo estipulado, se dará por sentado que el producto ha sido aceptado. Se recomienda una ventana de un día o menos ya que algunos trabajos no pueden seguir adelante sin la aceptación o modificación del producto.
    - Si el entregable es revisado fuera de la ventana de aceptación y se encuentra alguna falla, el trabajo que se generé, va en la pila de la lista de requisitos como una nueva historia de usuario (esto costará dinero al cliente, por lo que la disponibilidad del cliente es esencial).
    - Si dentro de la ventana de aceptación se detectan fallas o faltan requisitos que fueron acordados al inicio del sprint, el proveedor es responsable de corregir o agregar los requisitos que faltan.

3. *Establecimiento de prioridades:* una lista de requisitos debe ser revisada con regularidad para dar prioridad y estimar los puntos de historia. Se debe agregar al contrato el tiempo que se utilizará para analizar las de nuevas historias, la desagregación de historias grandes, y la ayuda que se brindará para determinar cualquier cambio en la prioridad.

    Se debe ayudar a entender al cliente que se les permitirá ajustar la prioridad de las historias en el producto, pero que si lo hace puede tener un impacto en la cantidad de trabajo que el equipo puede lograr en una fecha determinada.

4. *Cláusulas de cancelación:* se debe indicar en el contrato en qué momento se puede cancelar un proyecto. Se recomienda permitir al cliente cancelar el contrato en cualquier momento con las siguientes condiciones:
    - Sutherland recomienda que si el cliente considera que se han desarrollado todas las funciones necesarias antes de finalizar el proyecto, el cliente puede cancelar, siempre y cuando se obtenga un estimado del resto de las características que se habían solicitado pagando al proveedor el 20% de esa cantidad por la cancelación.
    - Si el cliente desea cancelar, también se recomienda una cláusula que indique un periodo de tiempo para cerrar el proyecto.

Tabla 14. Contrato de rangos y cambios para proyectos que utilicen en desarrollo iterativo e incremental.

4.5 Conclusiones

En el presente capítulo se desarrolló el diseño de un método para la administración de proyectos de software, se tomó como base a la planeación y al modelo iterativo incremental como proceso de conducción. Se obtuvo como resultado un método con diez pasos.

Los primeros cuatro pasos son subsistemas del sistema de planeación, estos son los encargados de identificar el estado actual (diagnóstico), y el estado futuro deseado (visión), con ambos estados, se puede realizar un análisis para determinar la alternativa de solución que más convenga para resolver la problemática (prescripción), con la identificación de la solución, se procede a realizar los planes necesarios para iniciar con la ejecución del proyecto (instrumentación de la solución). Como resultado de este proceso, el sistema de planeación arroja un conjunto de planes, que contienen las políticas de trabajo, así como el alcance y los requisitos priorizados que deben ser desarrollados.

Estos planes son los insumos del sistema de implantación, para definir los pasos de este sistema se tomó como base el marco de trabajo Scrum el cual se fundamenta en el modelo iterativo e incremental, y utiliza mejores prácticas. Los pasos descritos inician con la planeación del TimeBox, se seleccionan de la pila de requisitos priorizados cierta cantidad para generar un incremento, posteriormente cuando se está realizando el desarrollo, se propone realizar reuniones diarias para dar seguimiento al desarrollo del producto y eliminar los posibles problemas que se puedan presentar, al finalizar la iteración, se realiza una evaluación del TimeBox y se verifica si se cumplido con los objetivos, si fue así se libera el incremento, en caso contrario se recibe la retroalimentación por parte del cliente y se toman los cambios para integrarlos a la siguiente iteración. Al finalizar cada ciclo, el equipo de trabajo realiza una retrospectiva para determinar qué salió bien y qué se puede mejorar, de esta manera se mantiene un proceso de mejora continua, tanto para el equipo de trabajo como para el producto.

Como se puede observar el sistema de control (evaluación y adaptación) se encuentran presentes en los pasos de evaluación del TimeBox y el de retrospectiva.

Por último se propone un paso, para realizar la liberación del producto final, permitiendo al equipo de trabajo, la realización de manuales y la corrección de problemas menores.

Con este método se cumple el objetivo de crear un proceso de conducción que permita gestionar proyectos de software.

Conclusiones generales

Se cumplió con los objetivos de la tesis planteados en el primer capítulo:

1. *Identificar y definir la problemática que enfrenta el proceso general para la gestión de proyectos de software.*

   Se determinó que el modelo en cascada no es adecuado para el desarrollo de proyectos de software, debido a su estructura lineal y a que evita el cambio. Al ser el software intangible es imposible determinar todas las características desde un inicio, siempre habrá cambios durante su desarrollo, por tanto se detectó la necesidad de utilizar como alternativa un proceso iterativo.

   Los principales problemas detectados debido al uso de un modelo lineal, están relacionados con el desarrollo de procesos, las empresas no cuentan con herramientas que les permitan conducir sus proyectos de manera adecuada, lo que da como resultado productos en donde el 45% de sus características nunca son utilizadas.

   Se detectó que un proceso meramente empírico como el que propone el modelo iterativo e incremental, es útil para el desarrollo de software, pero no es suficiente, debido a que solo se centra en la implantación, olvidándose de los elementos y las ventajas que proporciona la planeación.

   Por tanto se propuso integrar al modelo iterativo e incremental, a la estructura que propone la planeación como proceso de conducción, y de esta manera tener un enfoque holístico que permita a las empresas tener presentes todos los elementos que están involucrados en la gestión de proyectos de software.

2. *Utilizar las propiedades del enfoque de sistemas para identificar los elementos que intervienen en la gestión de un proyecto de software.*

   Se aplicó el enfoque sistémico a la gestión de proyectos de software, para conceptualizar a los proyectos como sistemas, lo que dio como resultado (***Figura 74***):

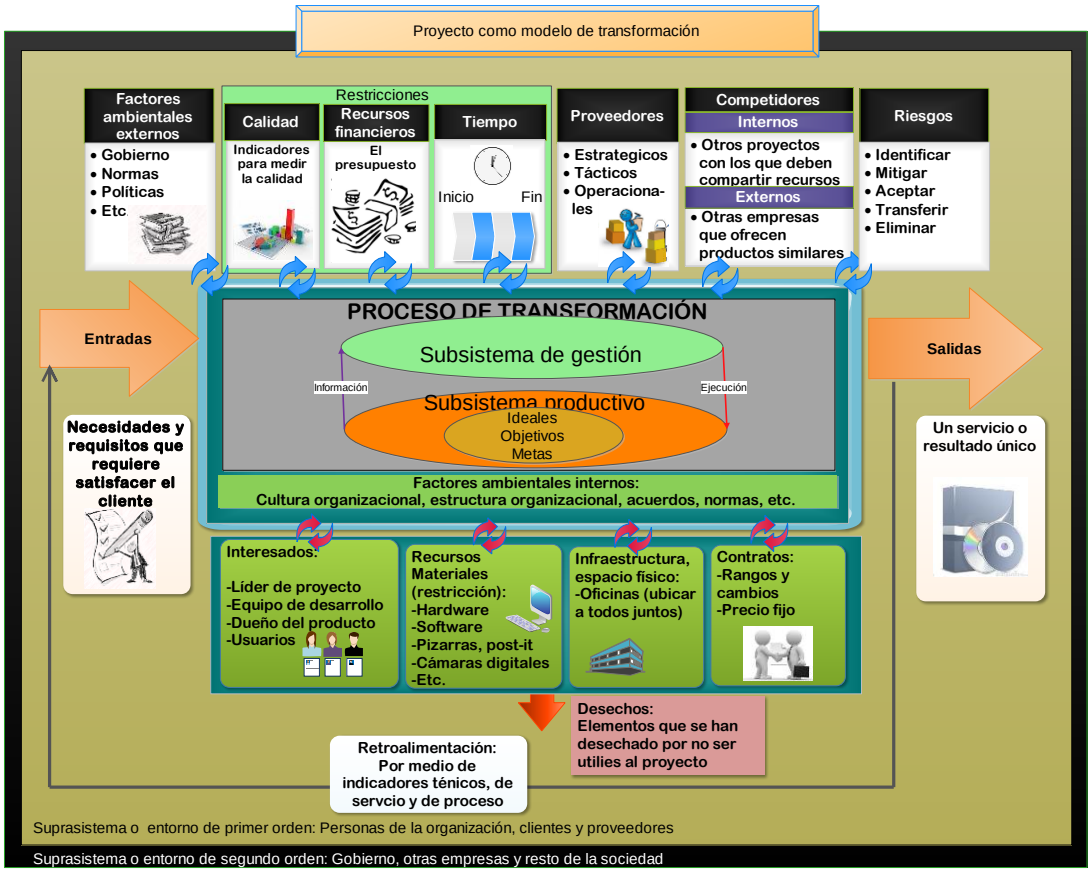


Figura 74. Proyecto como un sistema

Se determinó que un proyecto está constituido por cuatro elementos: entradas donde recibe los insumos que le proporciona el medio ambiente; un proceso de transformación, para el procesamiento de los insumos; salidas, productos únicos (software); y retroalimentación, lo que le permite mantener la homeostasis.

Los proyectos son identificados, por tener tres características: producir productos únicos, ser temporales, y teleológicos (persiguen un fin).

Al considerase a un proyecto como un sistema abierto, se identificó que los elementos del medio ambiente intercambian información con él, por tanto, se analizó al suprasistema para determinar qué elementos podían afectar de manera positiva o negativa al proyecto.

3. *Realizar un análisis en cada uno de los elementos identificados para determinar sus principales características.*

Se describieron las características de los elementos del medio ambiente que intercambian información con el proyecto:

- *Restricciones (calidad, costo y tiempo):* el desarrollo ágil fija el tiempo, los costos y la calidad mientras que el alcance puede ser modificado.
- *Proveedores:* las terceras personas encargadas de abastecer la materia prima para el desarrollo de software.
- *Riesgos:* se definió lo que es un riesgo, y las distintas formas de abordarlo.
- *Interesados:* se identificaron tres roles principales, dueño del producto, líder de proyecto, equipo de desarrollo, y dos roles que pueden ser útiles durante el proceso de desarrollo de un proyecto, usuario final y mentor ágil.
- *Factores ambientales:* se detectaron los principales elementos que pueden afectar de manera positiva o negativa el desarrollo del proyecto.
    - *Externos:* gobierno, geografía, sociedad, economía, política, clientes, proveedores, competidores.
    - *Internos:* cultura organizacional, reglas, normas y políticas internas, estructura organizacional y competidores.
- *Infraestructura y recursos:* se determinó como debe estar estructurada una oficina para el desarrollo de proyectos de software y cuáles son los principales recursos.
- *Contratos:* se definió qué es un contrato, así como los principales elementos para construcción de un acuerdo contractual.
- *Retroalimentación (métricas)*: todo proyecto debe ser retroalimentado con información obtenida por medio de métricas.

4. *Identificar los pasos a seguir para administrar un proyecto de software tomando como base a la planeación como proceso de conducción y al modelo iterativo e incremental.*

Al considerar a un proyecto como un sistema, y aplicando el enfoque cibernético se determinó que el proceso de trasformación contenía dos subsistemas, el de gestión y el conducido. Se analizó al subsistema de gestión para el proceso de conducción de proyectos de software, utilizando el modelo iterativo e incremental y se determinó que presentaba una visión parcial y restringida.

Para dotar al modelo iterativo e incremental de un marco teórico se propuso a la planeación como proceso de conducción, la cual propone una estructura compuesta por tres subsistemas: el de planeación, dividido en tres etapas 1) el diagnóstico (estado actual y futuro del objeto conducido), 2) la prescripción (identificación de alternativas de solución), y 3) de instrumentación de la solución (definir los planes que requiere el proyecto); el de control, divido en dos etapas 1) evaluación y 2) adaptación; el de implantación, en este subsistema se propuso la integración del modelo iterativo e incremental, ya que es el encargado de la ejecución de los planes diseñados en el subsistema de planeación.

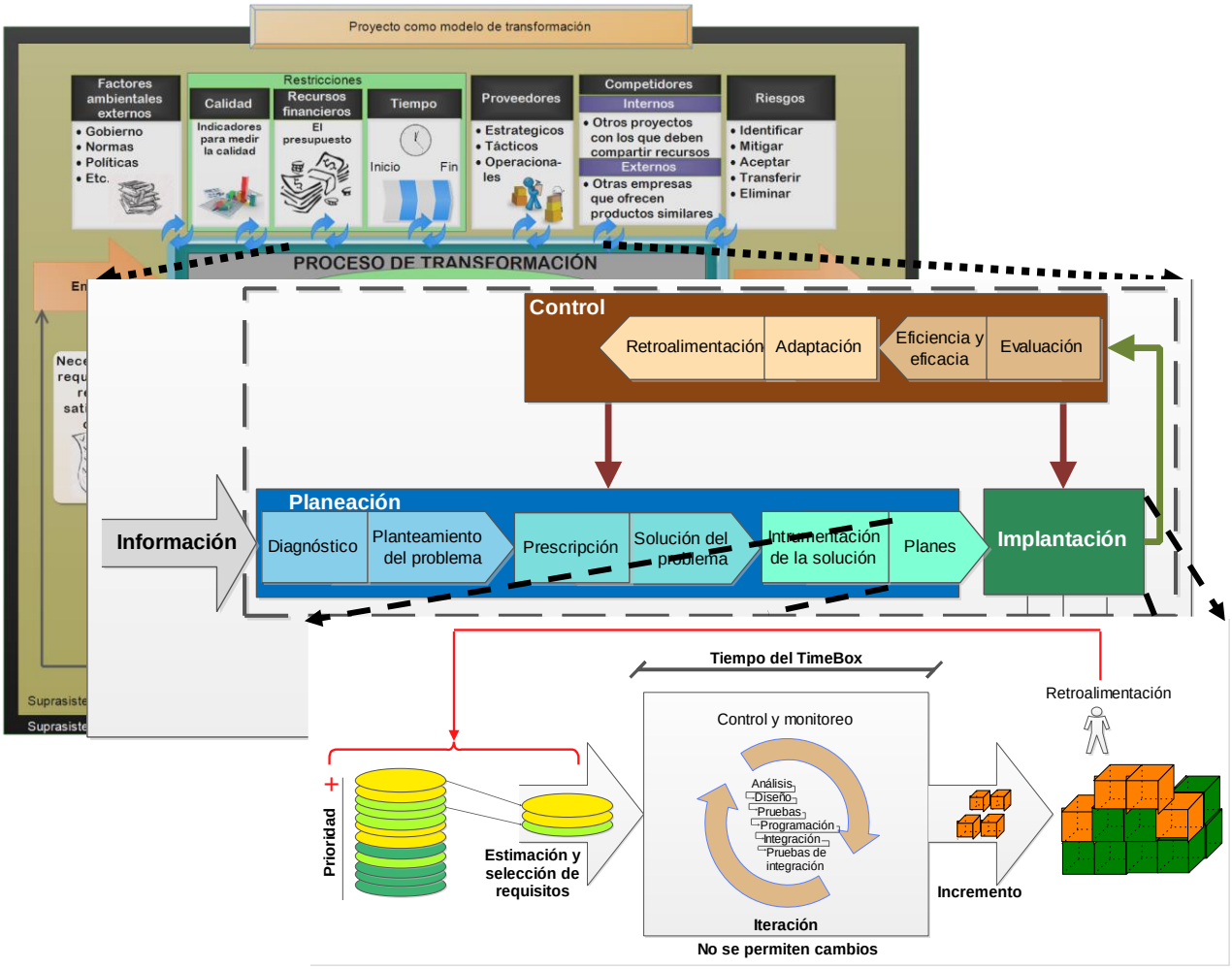


Figura 75. Base metodológica para el diseño del método. Elaboración propia.

Los puntos anteriores, permitieron cumplir con el objetivo principal:

Crear un método basado en la planeación como proceso de conducción y el modelo iterativo e incremental, para sistematizar los pasos que debe seguir un equipo de trabajo para planear, coordinar, dirigir y controlar un proyecto de desarrollo de software.

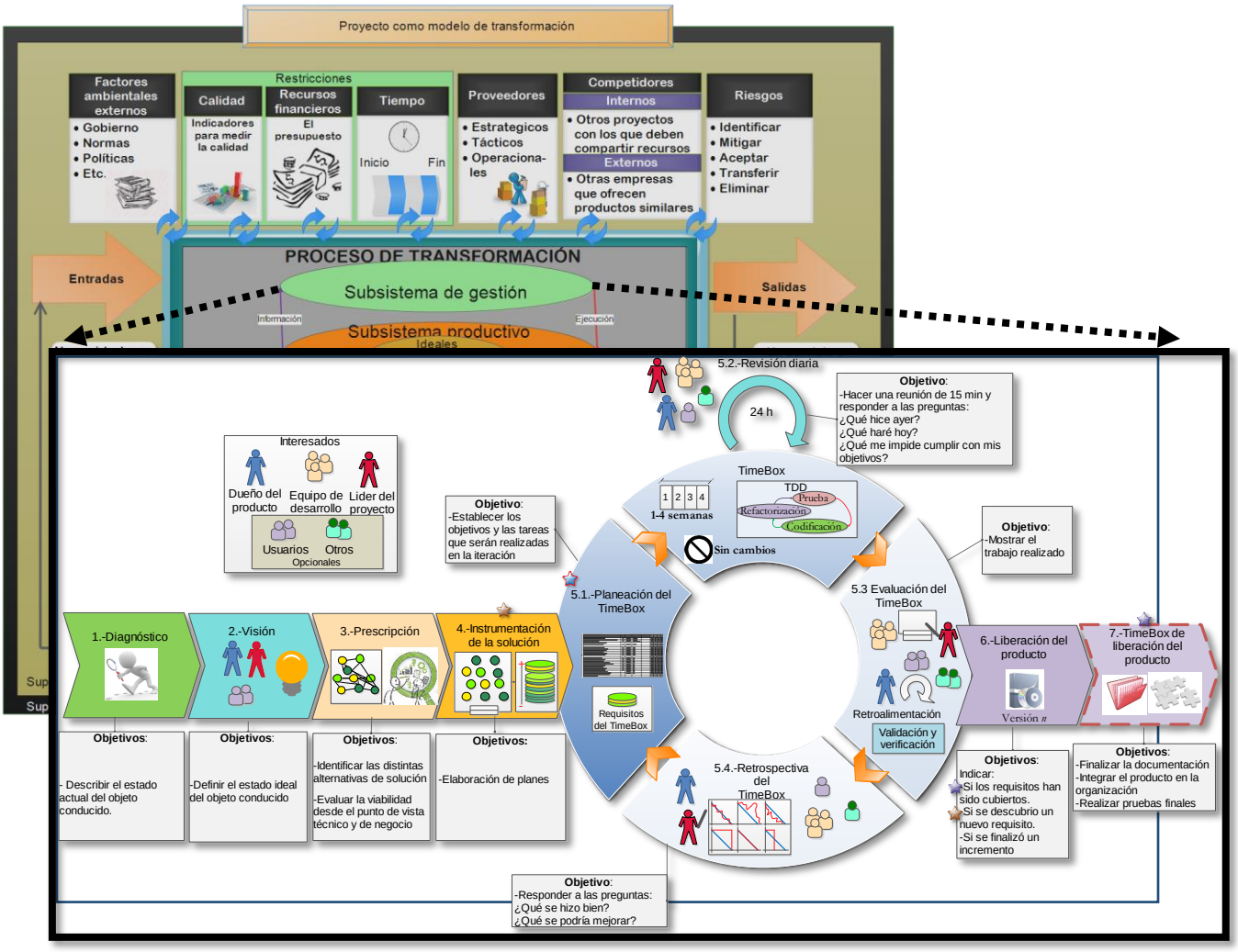


Figura 76. La gestión de proyectos de software: un enfoque sistémico.

De esta manera se dio un enfoque sistémico a la gestión de proyectos de software, al obtener una visión holística sobre los elementos que la constituyen, así como una sistematización, por medio de la definición de una serie de pasos que permiten al subsistema de gestión indicar a un equipo de trabajo como implementar y ejecutar un proyecto de manera eficaz y eficiente, para obtener productos que estén alineados con los objetivos que el cliente pretende alcanzar.

Líneas sugeridas para futuras investigaciones

La investigación que dio origen a la presente tesis podría continuar por medio de las siguientes líneas de investigación:

- Elaboración y publicación de un libro en español sobre la administración de proyectos de software.
- Se detectó que México no cuenta con una normatividad que permita a las dependencias gubernamentales licitar proyectos de software para la obtención de servicios, la mayor parte de las contrataciones se hacen por medio de contratos a precio fijo, a diferencia de Estados Unidos o Inglaterra, que exigen a las empresas seguir el modelo iterativo incremental para la creación de software.
- Desarrollar una investigación para determinar cuál es la problemática actual dentro de México en lo referente al desarrollo y administración de proyecto de software.
- Aplicar el método propuesto y determinar que mejoras podrían realizarse.

Bibliografía

Abrahamsson, P., Salo, O., & Ronkainen, J. (2002). *Agile software development methods, Review and analysis.* VTT.

Ackoff, R. (1992). *Planificación de la empresa del futuro.* México: Limusa.

Ackoff, R. (2002). *El paradigma de Ackoff, Una Administración sistémica .*

Albretch, K. (1994). *Todo al poder del cliente.* España: Paidos.

Aliance, S. (08 de 04 de 2014). *Scrum Aliance*. Obtenido de http://www.scrumalliance.org/why-scrum

Alliance, A. (17 de Abril de 2014). *Agile Alliance*. Obtenido de http://www.agilealliance.org/

Alliance, A. (20 de 06 de 2014). *Guide to Agile Practices*. Obtenido de http://guide.agilealliance.org/

Amaya, A. J. (2008). *Sistemas de información gerencial, Hardware-Software-Redes-Internet-Diseño.* Universidad Santo Tomas Bucaramanga.

Atern. (21 de 06 de 2014). *DSDM Arten Handbook.* Obtenido de http://www.dsdm.org/dig-deeper/book/dsdm-atern-handbook

Atwood, J. (07 de 08 de 2014). *Code Smells*. Obtenido de http://www.codinghorror.com/blog/2006/05/code-smells.html

Beck, K. (2004). *Extreme Programming Explained.* Person.

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., James, G., . . . Thomas, D. (17 de Abril de 2014). *Manifiesto por el Desarrollo Ágil de Software*. Obtenido de http://agilemanifesto.org/iso/es/

Bermejo, M. (2012). *Contratos ágiles.* España: Universidad Oberta de Cataluña.

Bertalanffy. (2006). *Teoria General de Sistemas.* Fondo de Cultura Económica.

Boehm, B. W. (1988). *A Spiral Model of Software Development and Enhancement.* TRW Defense Systems Group.

C230 Consultores. (2012). *Base de conocimiento sobre el PROSOFT 2.0 (con base en estudios previos).* México: Secretaría de Economía.

Chandramouli, S., & Dutt, S. (2012). *PMI Agile Certified Practitioner—Excel with Ease.* India: Pearson Education India.

Chasm, C. t. (1991). *Geoffrey A. Moore.* Harper Business Essentials.

Cockburn, A. (2004). *Crystal Clear A Human-Powered Metodology for Small Tems.*

Cohn, M. (2004). *User Stories Applied - For Agile Software Develpment.* Addison-Wesley.

Cohn, M. (2005). *Agile Estimating and Planing.* Addison-Wesley.

Cordoba, J. (1979). *Modelos y Tecnicas de Sistemas Aplicados a la Adminsitración de Proyectos.* Costa Rica: Instituto Nacional de Administración Publica (INAP).

Craig, L. (2004). *Agile & Iterative Development .* Pearson Education.

Derby, E., & Larsen, D. (2006). *Agile Retrospectives: Making Good Teams Great.* Pragmatic Bookshelf.

DOD-STD-2767. (1985). Military Standar, Defense System Software Develpment. *Departamento de Defensa de los Estados Unidos*, 61-70.

Dooley, J. (2011). *Software Development and Professional Practice.* Apress.

Drucker, P. (2002). *La gerencia en la sociedad futura.* Bogotá: Norma.

DSDM. (02 de 07 de 2014). *Fundamentos del DSDM*. Obtenido de http://www.dsdm.org/content/2-fundamentals

Economía, S. d. (2009). ACUERDO por el que se establece la estratificación de las micro, pequeñas y medianas empresas. *Diario Oficial de la Federación*.

Falgueras, B. C. (2002). *Ingeniería del software.* UOC.

Forsberg, K. (2005). *Visualizing Project Management.* John Wiley & Sons, Inc.

Garcia, S., & Dolan, S. (1997). *La Dirección por Valores.* Madrid: McGraw-Hill.

Garzás, J. (08 de 04 de 2014). *javiergarzas*. Obtenido de http://www.javiergarzas.com/2012/09/metodologias-crystal.html

Gelman, O. (2005). Papel de la planeación en el proceso de conducción. *Cuadernillo de divulgación, Facultad de Ingeniería* , 3-4.

Gelman, O., & Negroe, G. (1982). La planeación como un proceso básico en la conducción. *Revista de la Académia Nacional de Ingeniería* , 253-270.

Gil, E. Á. (2010). *Cómo crear y hacer funcionar una empresa.* ESIC .

Gómez, R. C., & Zornoza, C. C. (2002). *producto, Aprendizaje organizativo y sistemas complejos con capacidad de adaptación: implicaciones en la gestión del diseño de.* Universitat Jaume I.

González Bañales, D. (2006). *Estudio Exploratorio de los Factores Críticos de Éxito de la Industria Mexicana del Software y su Relación con la Orientación Estratégica de Negocio.* Valencia, España: Universidad Politecnica de Valencia.

González, D. (Julio de 2005). *Industria Mexicana del Software. Un estudio en cifras.* Obtenido de Software Guru: http://sg.com.mx/content/view/404

Gutiérrez, E., & Gutiérrez, A. (2007). *Diagnóstico de las Empresas Desarrolladoras de Software en México*. Obtenido de Software Guru: http://sg.com.mx/content/view/737

Hernández, Y. R. (2006). *Introducción a la Adminsitración - Teoria general adminsitrativa: origen, evolución y vanguardia.* México: MacGraw-Hill.

Herrscher. (2008). *Pensamiento Sistémico.* Granica.

IEEE, S. A. (1990). *729-1983 - IEEE Standard Glossary of Software Engineering Terminology.* The Institute of Electrical and Electronics Engineers.

IMIC, I. M. (31 de Marzo de 2013). *Catálogo Nacional de Costos en Linea.* Obtenido de Instituto Mexicano de Ingeniería de Costos: https://www.imic.com.mx/catalogo_nacional_costos.php

INEGI. (2009). *Censos económicos 2009. Micro, pequeña, mediana y gran empresa, estratificación de los establecimientos.* México: INEGI.

ISO, I. O. (2008). *ISO/IEC Standard 12207:2008:Software life-Cycle processes.* International Organization for Standardization.

ITAM. (2005). *LA INDUSTRIA DE SERVICIOS DE SOFTWARE EN MEXICO; DIAGNÓSTICO, PROSPECTIVA Y ESTRATEGIA.* México: Centros de Estudios de Competitividad.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *El proceso unificado de desarrollo de software.* Addison Wesley.

Jiménez, E., & Orantes, S. (2012). Metodologías híbridas para el desarrollo de software: una opción para México. *Revista Digital Universitaria*, 5.

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

Kepne, C., & Tregoe, B. (1983). *El nuevo directivo racional.* McGraw-Hill.

Kniberg, H., & Skarin, M. (2010). *Kanban y Scrum - Obteniendo lo mejor de ambos.* C4Media.

Lacey, M. (2012). *The Scrum Field Guide: Practical Advice for Your First Year.* Addison-Wesley Professional.

Larman, C. (2004). *Agile & Iterative Developmen, A Manager´s Guide.* Boston: Pearson Education.

Larman, C., & Basili, V. R. (2003). Iterative and Incremental Development: A Brief History. *Computer (Volumen: 36, Número: 6)*, 47-56.

Larman, C., & Vodde, B. (2010). *Practices for Scaling Lean and Agile Development: Large,Multisite, and Offshore Product Development with large-Scale Scrum.* Addison-Wesley.

Larman, C., & Vodde, V. (s.f.). *Practices for Scaling Lean and Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum.*

Layton, M. (2012). *Agile Project Management For Dummies.* Hoboken, New Jersey: John Wiley & Sons, Inc.

Lledó, P. (2013). *Director Profesional de Proyectos.* Canadá: Pablo Lledó.

Mall, R. (2004). *Fundamentals of Software Engineering.* Pretince Hall of India.

Moreno Jiménez, A. E. (2013). *CONTRIBUCIÓN METODOLÓGICA PARA LA MEJORA DE LA ADMINISTRACIÓN DE.* México: Facultad de Ingeniería.

Negroe, P. G. (2005). *Papel de la planeación en el proceso de conducción.* México: Facultad de Ingenieria.

Nizam, M., & Sahibuddin, S. (2011). Critical success factors for software projects: A comparative study. *Scientific Research and Essays Vol. 6(10)*, 2174-2186.

Ochoa, R. F. (1997). *Cuadernos de Planeación y sistemas No. 10. Método de los sistemas.* México: Facultad de Ingeniería, UNAM.

Olmedo, H. (. (2009). *Administración de Proyectos: una especialización en el ejercicio de la Arquitectura.* México: Facultad de Arquitectura.

Otero, A. S., & Grossi, M. M. (2005). *La Llamada Revolución Industrial: Siglos XVIII y XIX .* Caracas: Universidad Católica Andrés Bello.

Parnas, D. &. (1986). A rational design process: How and why to fake it. *Journal IEEE Transactions on Software Engineering*, Volume 12 Issue 2 , Pages 251 - 257 .

PMI, P. M. (2013). *A guie to the Project Management Body of Knowledge, Fith Edition.*

Pressman, R. (2010). *Ingeniería de Software.* Mc Graw Hill.

Ramesh, G. (2002). *Managing Global Software Projects.* McGraw-Hill.

Robbins, S. P., & Coulter, M. (2010). *Administración.* México: Pearson Educación.

Royce, W. (1970). Managing the development of large software systems. *IEEE WESCON 26 (August): 1–9*, 328-338.

Schwaber, K., & Sutherland, J. (2003). *La Guía Definitiva de Scrum: Las Reglas del Juego.* scrum.org.

Secretaría de Economía . (2009). *ACUERDO por el que se establece la estratificación de las micro, pequeii.as y medianas empresas.* México: Diario Oficial de la Federación.

Serna, H. (1997). *Gerencia Estratégica.* Colombia: 3R Editores.

Shwaber, K., & Sutherland, J. (2013). *La guía definitiva de Scrum.* scrum.org.

Solórzano, P., Millán, N., Solórzano, S., & Sánchez, V. (2012). *Fundamentos de computación, Panorama histórico y programación.* México: Facultad de Ingeniería, UNAM.

Sommerville, I. (2005). *Ingeniería del software.* Pearson Educación.

Standish, G. (2013). *CHAOS Manifiesto 2013.*

XP. (09 de 04 de 2014). *Los valores de la Programación Extrema*. Obtenido de http://www.extremeprogramming.org/values.html

Yurdon, E. (20 de Agosto de 2014). *Chapter 18*. Obtenido de The Environmental Model: http://yourdon.com/strucanalysis/wiki/index.php?title=Chapter_18#The_Context_Diagram