**7.1.4 Análisis de Alternativas y Justificación**

**7.1.4.1 Modelo en Cascada**

Este modelo toma los procesos fundamentales de las actividades de especificación, desarrollo, validación, y evolución y las representa como fases separadas tales como especificación de requerimientos, diseño de software, implementación, pruebas, entre otras. [1]

Las etapas que constituyen a este modelo son:

1. **Análisis y Diseño de Requerimientos:** Los servicios, restricciones, y objetivos del sistema son establecidos por consulta con los usuarios del sistema. Son definidos en detalle y sirven como especificaciones del sistema.
2. **Diseño de Sistema y de Software:** Establece los requerimientos de hardware y/o software por medio del establecimiento de una arquitectura general del sistema. El diseño de software involucra identificar y describir las abstracciones fundamentales del software y sus relaciones.
3. **Implementación y pruebas de unidades:** El diseño es implementado como un set de programas o unidades de programa. La prueba de unidades involucra la verificación de cada unidad.
4. **Integración y pruebas de sistema:** Las unidades de programa individuales son integradas y puestas a prueba como un sistema completo para asegurar que los requerimientos de software se hayan cumplido. Después de pasar las pruebas, el sistema de software se entrega al cliente.
5. **Operación y mantenimiento:** Se trata de corregir errores que no fueron descubiertos en etapas anteriores, mejorando el sistema a medida que nuevos requerimientos son descubiertos.

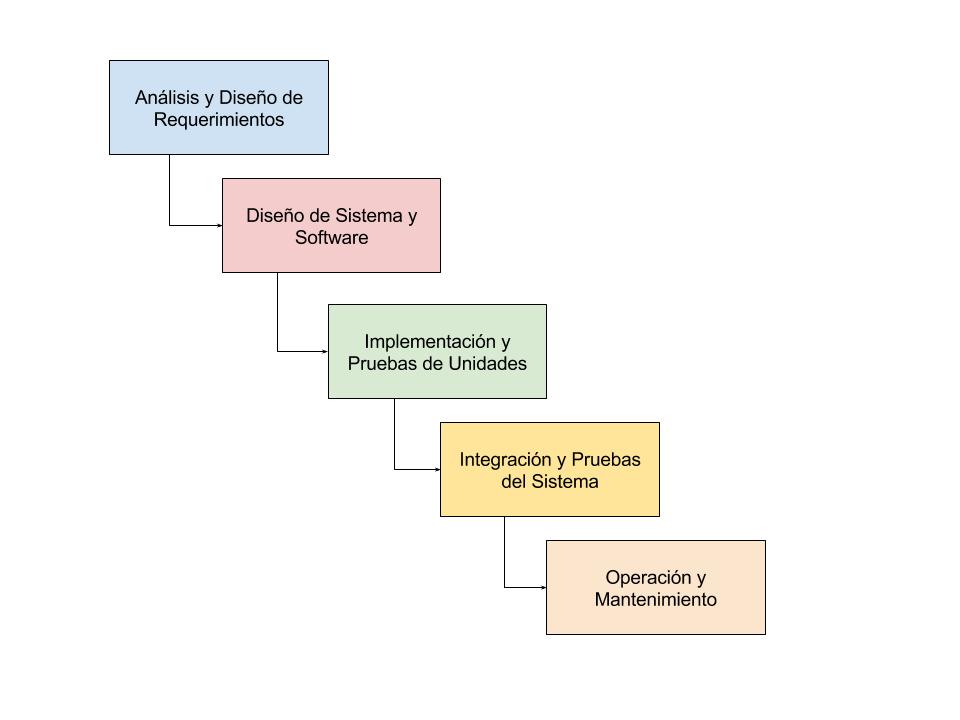
****

Figura [XX]: Modelo de Ciclo de Vida en Cascada

En principio, el resultado de cada fase es uno o más documentos que son aprobados. Como se puede ver en la *Figura [XX]: Modelo de Ciclo de Vida en Cascada*, las fases son en serie y una fase no debe comenzar hasta que la anterior termine. En practica, las fases se sobreponen y alimentan información entre ellas. [1]

Este modelo solo se debe usar cuando los requerimientos son comprendidos y no están propensos al cambio radical durante el desarrollo del sistema. [1] Adicionalmente, requiere de una documentación rigurosa y posible repetición de trabajo en cada iteración por lo cual puede incurrir en un mayor volumen de trabajo. Adicionalmente, el cambio en el trascurso del proyecto puede generar trabajo excesivo por lo cual se optó por descartar este modelo.

**7.1.4.2 Modelo en Espiral**

Boehm, en su paper *“Spiral Development: Experience, Principles, and Refinements”*, lo define inicialmente como una familia de procesos de desarrollo de software caracterizados por la iteración de un set de procesos elementales de desarrollo y de manejo de riesgo apara activamente reducirlo. También lo describe como un generador de modelos de proceso, orientado a riesgos. Como tal, este tiene dos características principales: un acercamiento cíclico para el crecimiento de un sistema mientras se disminuye el riesgo, y un conjunto de hitos de ancla para asegurar que compromiso de los *stakeholders* y llegar a soluciones mutuamente satisfactorias.

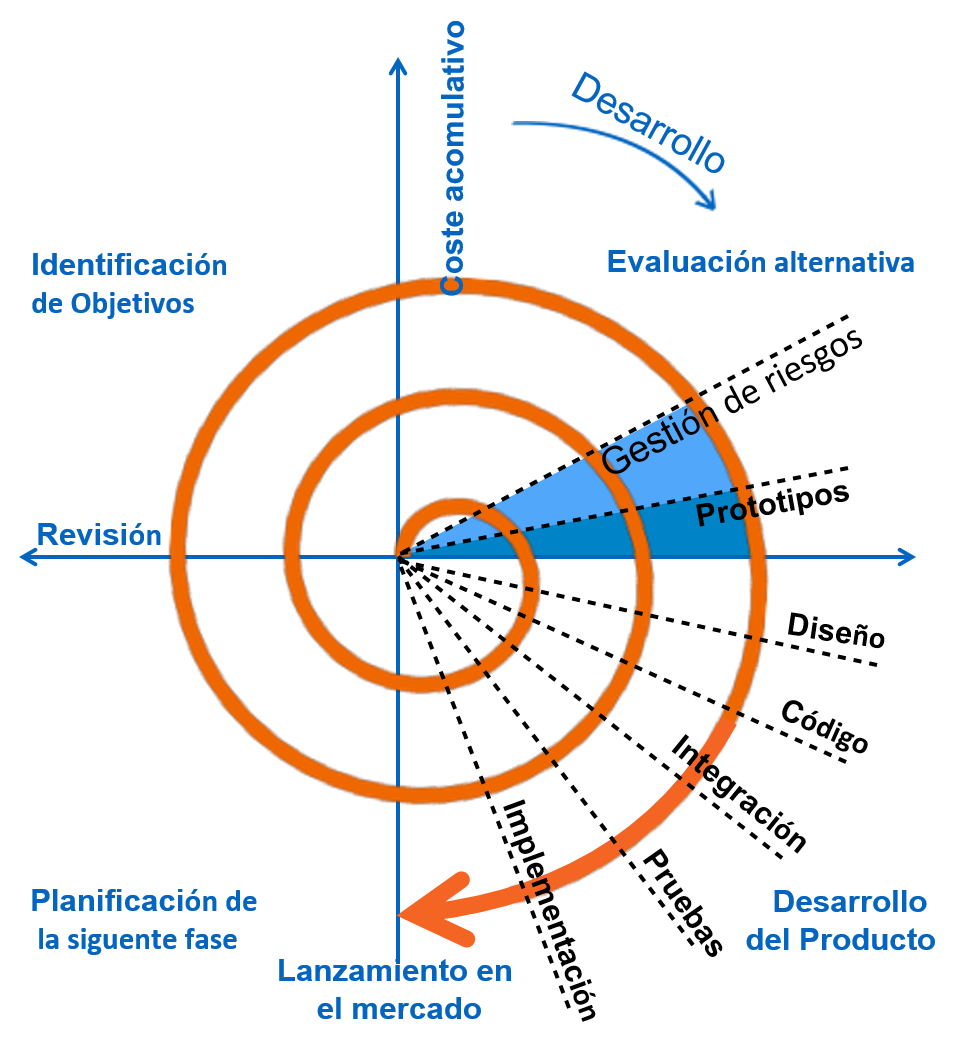
****

Figura [XX]: Modelo de Ciclo de Vida en Espiral

Como se puede ver en la *Figura [XX]: Modelo de Ciclo de Vida en Espiral,* este está dividido en cuatro cuadrantes. Cada cuadrante es una fase del desarrollo. Con cada iteración se ejecuta cada fase para generar un crecimiento en el sistema y una disminución de riesgos. Primero se ejecuta la fase de identificación de objetivos. Es aquí donde se hace un levantamiento de requerimientos. Posteriormente se hace la fase de evaluación de alternativas que hace vez es un análisis de riesgos donde se sugieren alternativas para la mitigación de riesgo y se implementan. Después se desarrolla el producto. Por último, se evalúa con el cliente el resultado de la iteración y se planifica la siguiente iteración para repetir el proceso otra vez.

Por un lado, este acercamiento otorga un fuerte control de documentación, la posibilidad de agregar funcionalidad en momentos avanzados del desarrollo y una producción temprana de software. Sin embargo, esto con lleva a un alto costo para su uso y requiere de conocimiento y dominio altamente especifico en análisis de riesgos. Lo anterior, genera una dependencia en el análisis de riesgos por lo cual el éxito del proyecto depende de la misma. Adicionalmente no es muy eficiente en proyectos pequeños. [2]

A causa del tamaño del grupo, cantidad de recursos, falta de experiencia suficiente en el análisis de riesgo y de la rigurosidad de este modelo, se concluye que el modelo no es acorde al proyecto.

**7.1.4.3 Modelo en V**

El modelo en V es una variación del modelo en cascada que muestra cómo se relacionan las actividades de prueba con el análisis y el diseño. [3] Este modelo promueve el uso de pruebas constante durante el desarrollo. Sin embargo, mantiene la linealidad del modelo en cascada.



Figura [xx]: Modelo de Ciclo de Vida en V [4]

Este modelo ofrece múltiples ventajas:

* Simple y fácil de usar
* Actividades de pruebas se ejecutan mucha antes de programar, lo cual ahorra tiempo.
* Rastreo de defectos proactivo
* Evita propagación de defectos
* Funciona en proyectos pequeños donde los requerimientos son fáciles de entender

Por otro lado, presenta las siguientes desventajas:

* Rígido y poco flexible
* No genera prototipos
* Si se efectúan cambios en medio del proceso, entonces los documentos de prueba y de requerimientos deben actualizarse

Este modelo ofrece ventajas en el área de pruebas y se enfoca en ello. Sin embargo, la falta de prototipos significa un alto riesgo para la entrega del producto ya que este podría no satisfacer las necesidades del cliente. Esta desventaja hace que este modelo no sea elegido para su uso en el proyecto.

**7.1.4.4 Modelo Diente de Sierra (Sawtooth)**

Todos los ciclos de vida previamente mencionados en esta sección trabajan bajo un mismo supuesto que es que los requerimientos no cambiaran drásticamente durante el desarrollo del proyecto. Lo que busca el modelo de diente de sierra es mostrar la percepción del sistema por parte del usuario y por parte del desarrollador por medio de diferentes niveles de abstracción en el tiempo [5]. Este modelo es en realidad una versión modificado del modelo en V que incluye intersecciones.

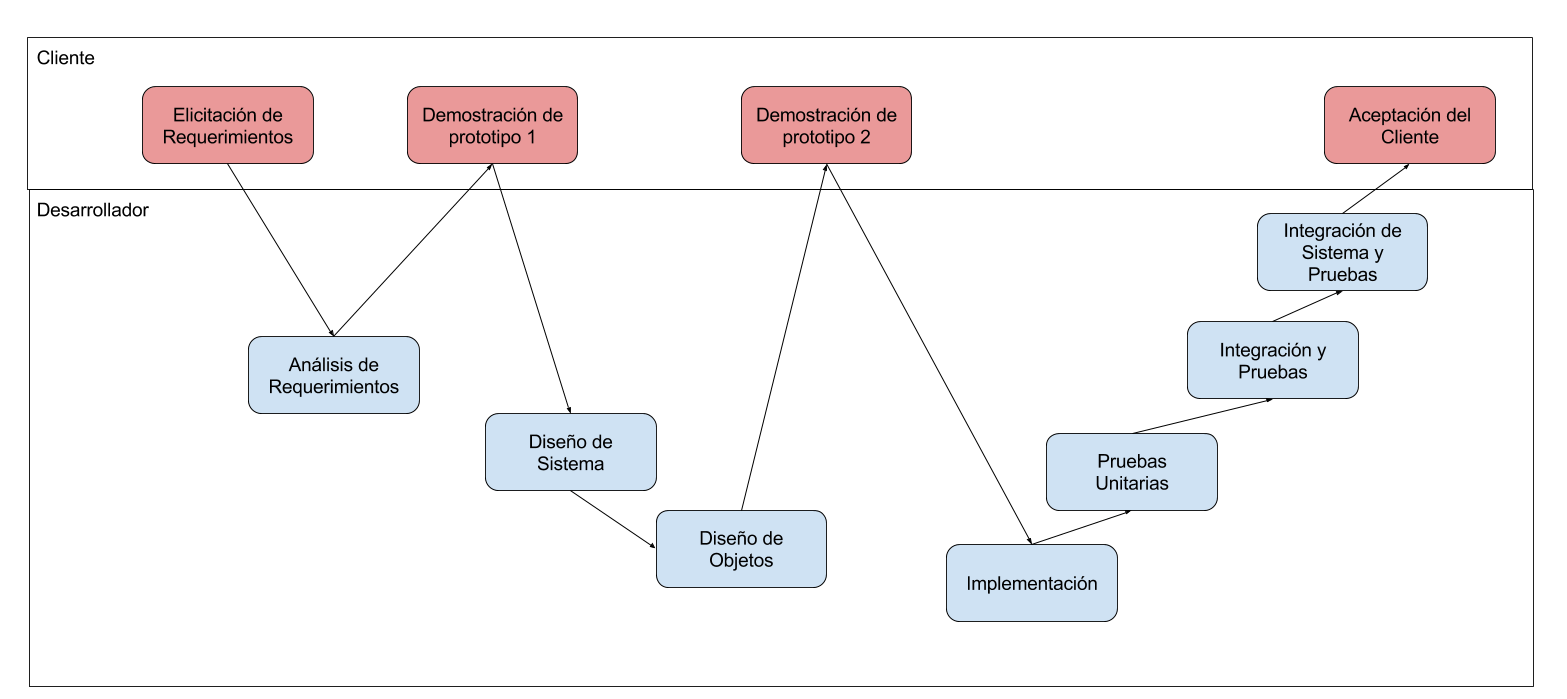


Figura XX: Modelo de Diente de Sierra

Al comienzo de este ciclo de vida, los desarrolladores y el cliente se encuentran en el mismo nivel de abstracción. A medida que el proyecto progresa, esto cambia. El usuario se mantiene a nivel de requerimientos mientras que el desarrollador se enfoca en factibilidad. El proceso de desarrollo de software debe garantizar que ambos puntos de vista coincidan al final del proyecto. El modelo lo logro introduciendo actividades adicionales que involucran al cliente desde su nivel de abstracción. Adicionalmente, en este modelo de ciclo de vida, se agrega dos demostraciones de prototipo lo cual ayuda a que el cliente este consiente del progreso y pueda tener una idea del funcionamiento del producto para así dar una retroalimentación.

Dado a que este modelo comparte características con el modelo en cascada, incurre en los mismos problemas del modelo en cascada: La naturaleza lineal del modelo y rigidez puede generar trabajo extra en caso tal que cambios grandes en los requerimientos se den durante el desarrollo. Adicionalmente, el hecho que las demostraciones se hagan en una etapa temprana puede generar una imagen muy diferente de producto final en el cliente. Es por estas razones que el grupo Active opto por no usar este modelo.

**7.1.4.5 Modelo Diente de Tiburón (Sharktooth)**

Este modelo es una versión refinada del modelo de diente de sierra. Agrega un nivel adicional de abstracción al modelo, el del gerente. Esto agrega actividades adicionales por parte del gerente. Se hace una distinción entre dientes “grande” y dientes “pequeños”. Los dientes grandes son las demostraciones a clientes y los dientes pequeños son las revisiones gerenciales y demostraciones a gerencia. Entonces las demostraciones podrían ser de vistas diferentes del sistema: Al cliente se le mostraría una demostración orientada a interfaces gráficas y funcionalidad, mientras que al gerente se le demuestra una demostración que muestre la factibilidad de las funcionalidades. Entonces los dientes pequeños son interacciones con el grupo gerencial y los dientes grandes con los clientes.

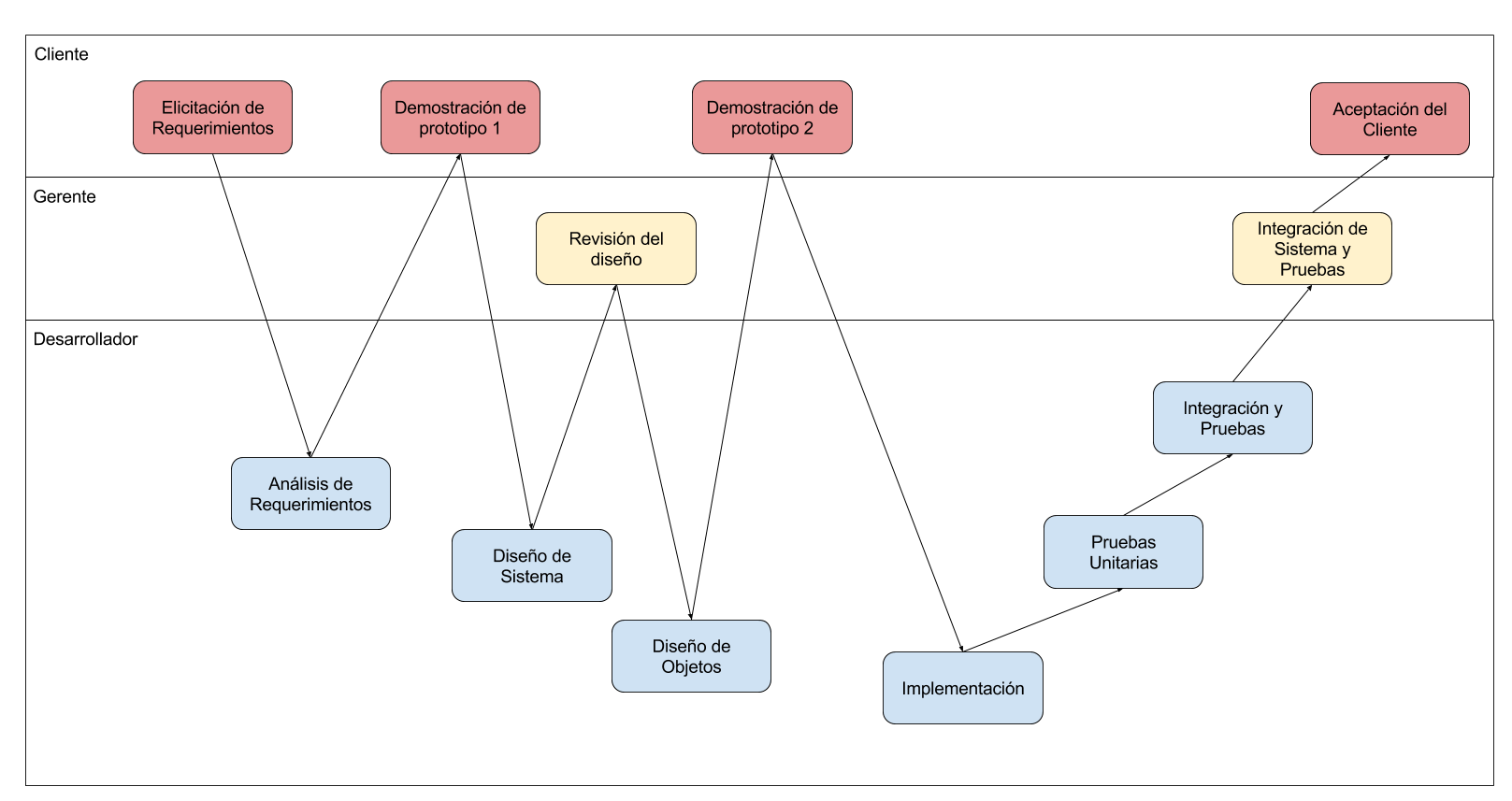


Figura XX: Modelo de Ciclo de Vida Sharktooth

A pesar de ser una expansión al modelo de diente de sierra, no soluciona los problemas que este presenta. Es por esto por lo que se descarta este ciclo de vida.

**7.1.4.6 Modelo RUP**

El Proceso Unificado de Rational (RUP, por sus siglas en inglés) es un ejemplo de un modelo de procesos moderno que se deriva del trabajo en el Lenguaje de Modelado Unificado (UML, por sus siglas en ingles), y de los asociados del *Unified Software Development Process* [1]. RUP reconoce que los modelos de procesos convencionales presentan una vista única del proceso. Por otro lado, RUP describe tres perspectivas [1]:

1. Una perspectiva dinámica, que muestra las fases del modelo en el tiempo
2. Una perspectiva estática, que muestra el proceso de actividades que se establecen
3. Una perspectiva práctica, que sugiere buenas prácticas a ser usadas durante el proceso

RUP es un modelo divido en fases que identifica cuatro fases en el proceso de desarrollo de software. A diferencia del modelo en cascada, que las fases son equivalentes a actividades del proceso, en este modelo las fases están más relacionadas con la empresa que con aspectos técnicos. La *Figura XX: Fases de RUP* muestra las fases que son:

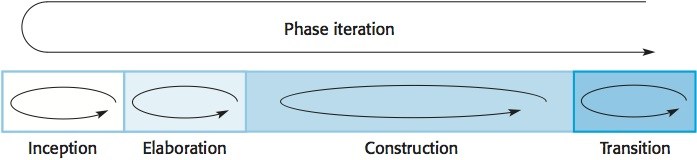


Figura XX: Fases de RUP

1. Concepción (*Inception*): El objetivo de esta fase es establecer los casos de negocio para el sistema. Se debe identificar las entidades externas que interactuaran con el sistema y definir las interacciones entres estos. Esta información se usa para definir el valor que el sistema le da al negocio. Si la contribución no es relevante, el proyecto se puede cancelar después de esta fase. [1]
2. Elaboración (*Elaboration*): El objetivo de esta fase es entender el dominio del problema, establecer un marco de trabajo para la arquitectura del sistema, desarrollar el plan de proyecto e identificar los riesgos clave del proyecto. Al final de esta fase, se debe generar un modelo de requerimientos para el sistema. Este puede ser un set de casos de uso, una descripción de la arquitectura y un plan de desarrollo del software. [1]
3. Construcción (*Construction*): Esta fase involucra el diseño, programación y pruebas. Partes del sistema son desarrolladas en paralelo e integradas durante esta fase. Esta fase debe generar un sistema de software funcional y su respectiva documentación, lista para entregar a los clientes. [1]
4. Transición (*Transition*): Esta fase se trata de mover el sistema de la comunidad de desarrollo a la comunidad de usuarios y hacerla funcionar en el ambiente real. [1] Al final de esta fase se debe contar con un sistema de software documentado, funcional en su ambiente de operación. [1]

Como se mencionó previamente, la perspectiva practica de RUP describe buenas prácticas de ingeniería de software que son recomendadas para uso de desarrollo de sistemas. Las seis practicas recomendadas según Sommerville son:

1. *Desarrolla software iterativamente.* Planea incrementos del sistema basado en prioridades del cliente y desarrolla las características del sistema de alta prioridad en las etapas iniciales del proceso de desarrollo.
2. *Gestiona requerimientos.* Explícitamente documenta los requerimientos del cliente y haz seguimiento de los cambios en estos. Analiza el impacto de los cambios en el sistema antes de aceptarlos.
3. *Has uso de arquitecturas basadas en componentes*. Estructura la arquitectura del sistema en componentes.
4. *Modela visualmente el software.* Usa modelos gráficos de UML para presentas vistas estáticas y dinámicas del software
5. *Verifica la calidad del software*. Asegura que el software cumpla con los estándares de calidad de la organización.
6. *Controla los cambios en el software*. Gestiona los cambios en el software por medio de un sistema de manejo de cambios y configuración de manejo de procedimientos y herramientas.

El RUP no es adecuado para todos los tipos de desarrollo [1]. Sin embargo, ofrece múltiples ventajas: la naturaleza iterativa le permite que el proceso crezca con cada iteración, divide en etapas el proceso y hace explicito lo que desarrolla en cada etapa. Este modelo se ve adecuado en proyectos medianos y grandes donde se requiere de un seguimiento robusto de procesos. Por otra parte, RUP puede ser un poco abrumador para proyectos pequeños y la vista gerencial de este modelo puede no ser adecuada para proyectos independientes, con grupos pequeños. Por estas razones se optó por no usar RUP para el proyecto.

# Bibliography

|  |  |
| --- | --- |
| [1] | I. SOMMERVILLE, Software Engineering (Ninth Edition), Adison Wesley, 2011. |
| [2] | ISQTB, "What is Spiral model- advantages, disadvantages and when to use it?," [Online]. Available: http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/. [Accessed 19 8 2017]. |
| [3] | "Ciclo de vida en "V" - INGENIERIA DE SOFTWARE," Weebly, [Online]. Available: http://ingsoftware.weebly.com/ciclo-de-vida-en-v.html. [Accessed 20 8 2017]. |
| [4] | ISTQB, "What is V-model- advantages, disadvantages and when to use it?," [Online]. Available: http://istqbexamcertification.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/. [Accessed 21 8 2017]. |
| [5] | "Software Life Cycle Models," Scribd, [Online]. Available: https://www.scribd.com/document/73647276/Software-Life-Cycle-Models. [Accessed 20 8 2017]. |
| [6] | B. Boehm, "Spiral Development: Experience, Principles, and Refinements," Software Engineering Institute, 2000. |