**METODOLOGÍAS DE DESARROLLO Y REQUERIMIENTOS DE SOFTWARE**

JAVIER FELIPE FEO CORTÉS

HENDERSON STEVEN MORENO URIBE

RAFAEL LEONARDO SANABRIA SÁNCHEZ

EDWIN DUVIAN SANTANA QUIROGA

SERVICIO NACIONAL DE APRENDIZAJE SENA

CENTRO DE ELECTRICIDAD, ELECTRÓNICA Y TELECOMUNICACIONES

ANÁLISIS Y DESARROLLO DE SISTEMAS DE INFORMACIÓN

BOGOTÁ D.C.

2016

**TABLA DE CONTENIDO**

[METODOLOGÍA DE DESARROLLO 3](#_Toc468491232)

[ENFOQUE DE METODOLOGÍA DE DESARROLLO 4](#_Toc468491233)

[METODOLOGÍAS DE DESARROLLO POPULARES EN EL PRESENTE 6](#_Toc468491234)

[REQUERIMIENTOS 8](#_Toc468491235)

[ESTÁNDARES PARA EL DESARROLLO DE REQUERIMIENTOS 10](#_Toc468491236)

[SCRUM 12](#_Toc468491237)

[ACTIVIDADES DE TRANSFERENCIA DEL CONOCIMIENTO 15](#_Toc468491238)

[BIBLIOGRAFÍA 18](#_Toc468491239)

# METODOLOGÍA DE DESARROLLO

La metodología de desarrollo de software consiste en un conjunto de pasos organizados, con el objetivo de controlar y facilitar el proceso del desarrollo de un sistema de información.

Precisamente, las metodologías de desarrollo fueron hechas como respuesta a la dificultad que suponía desarrollar un sistema de información. Las metodologías de desarrollo usadas en esa época, conocidas como las **metodologías de desarrollo tradicionales** aún suponían un problema para el desarrollo de sistemas de información, dado a que, son consideradas como burocráticas, y porque se dedicaba mucho tiempo a seguir la metodología, suponiendo el retraso del desarrollo entero de la aplicación. (Delgado, 2008).

Según Delgado (2008), estos son los problemas de las metodologías tradicionales:

* Retraso en la entrega del producto final.
* Altos costes de mantenimiento del sistema, causante de su abandono.
* Defectos del software, los cuales son muchos, causando el desuso.
* Requisitos no solucionados por el producto final.
* Cambio del problema al cual el software estaba destinado y la no adaptación.
* El software, aunque tenga buenas utilidades, no resuelve el problema principal.
* El abandono de los desarrolladores del proyecto.

Como respuesta a los problemas presentes en las metodologías de desarrollo tradicionales, se crearon otras metodologías las cuales eliminan una cantidad considerable de documentación y se enfocan más en la programación. Estás varían según el tamaño del proyecto, la cantidad de gente trabajando, el tiempo disponible, etc. Estas metodologías son llamadas como **metodologías de desarrollo ágiles**.

Entre las metodologías de desarrollo tradicionales se encuentran:

* RUP (Rational Unified Process).
* MSF (Microsoft Solution Framework).
* Win-Win Spiral Model.
* Iconix.

Entre las metodologías de desarrollo ágiles se encuentran algunas como:

* FDD (Feature Driven Development).
* XP (Extreme Programming).
* Scrum.
* DSDM (Dynamic Systems Development Method).
* ASD (Adaptive Software Development).

# ENFOQUE DE METODOLOGÍA DE DESARROLLO

Cada metodología de desarrollo contiene un elemento llamado **“enfoque”**, los enfoques, acompañados de las metodologías de desarrollo, busca modelar el proceso de desarrollo de software, mediante una secuencia de pasos organizados.

Los enfoques poseen varios tipos los cuales son:

* Lineal: los pasos se ejecutan uno tras otro.
* Iterativo: no hay orden estricto de pasos, estos se pueden repetir para el mejoramiento del producto.

Los enfoques de desarrollo más populares en la actualidad son:

* Modelo cascada: este modelo organiza los pasos de tal manera que cada fase inicia justo cuando termina otra, de manera de que las fases del desarrollo del sistema de información sean representados en forma de una cascada.
* Modelo incremental: este modelo rompe el objetivo primario del proyecto entre pequeños segmentos con el objetivo de controlar los posibles riesgos que se puedan presentar en el desarrollo de la aplicación (Software development process, s.f.). Existen tres variantes de este modelo, las cuales pueden consistir en que:
  1. El proyecto se divida en una serie de varias cascadas, donde cada cascada es reservada para una parte del software, o
  2. La parte que comprenda los requerimientos sea definida totalmente antes de proceder a una serie de desarrollos en cascada para cada parte del software, o
  3. El concepto inicial del software a desarrollar, los requerimientos, y el desarrollo como tal son definidos por una cascada, seguida por un desarrollo incremental y culminando con la implementación del sistema de información, completamente funcional (Software development process, s.f.).
* Prototipado: el prototipado consiste en el desarrollo de la aplicación mediante varias versiones de la aplicación no completas, las cuales sirven para la demostración de una funcionalidad de la aplicación. Estas versiones de la aplicación, son llamadas como “**prototipos**”. El modelo de prototipado se rige bajo unos principios:
  1. El modelo de prototipado no puede ser usado como único, dado a que este no es un modelo de desarrollo completo, este debe ser acompañado junto a otros modelos de desarrollo, como el modelo espiral, cascada, o el RAD.
  2. El proyecto se puede romper en varios segmentos para reducir el riesgo que pueda presentar el proyecto.
  3. El cliente es involucrado en el desarrollo del proyecto, lo cual aumenta la probabilidad de la aceptación del cliente de la implementación final.
  4. Mientras que algunos prototipos son desarrollados con la expectativa de ser descartados, en algunos casos es posible la evolución desde el prototipo, hasta el sistema funcional (Software development process, s.f.).
* Rapid application development (RAD): este modelo, según Tutorialspoint (s.f.), otorga prioridad al prototipado sobre la planeación. Desde que la planeación del proyecto es mínima, el desarrollo se puede ajustar fácilmente a los cambios en los requerimientos. Aunque el desarrollo del proyecto pueda ser rápido y efectivo, requiere alta experiencia en los desarrolladores, así como el uso de varias herramientas como por ejemplo, los generadores de código. También requiere que los prototipos puedan ser reusados.
* Modelo espiral: el modelo en espiral combina los principios del modelo RAD y el modelo cascada junto a los principios del diseño Top-down y Bottom-up (Software development process, s.f.). Los principios básicos por los cuales rige este modelo son, según el CMS (2008):
  1. Enfocarse en los riesgos y disminuir el riesgo que puede conllevar el proyecto rompiéndolo en varios segmentos, proveer mayor facilidad en el cambio durante el proceso del desarrollo y evaluar la posibilidad de continuación del desarrollo del proyecto.
  2. El desarrollo se divide entre varios ciclos, los cuales consisten en la misma secuencia de pasos y los cuales representan una parte del producto funcional.
  3. Cada ciclo se divide en cuatro fases, las cuales son:
     1. Determinar objetivos, alternativas y restricciones para la iteración.
     2. Evaluar alternativas, identificar y resolver riesgos.
     3. Desarrollar y verificar objetivos de la iteración.
     4. Planificar la siguiente iteración.
  4. Comenzar cada ciclo con la identificación de los *stakeholders* y sus condiciones conseguidas, y finalizar cada ciclo con una revisión y un compromiso.

# METODOLOGÍAS DE DESARROLLO POPULARES EN EL PRESENTE

1. **Scrum**

Se conoce como *Scrum*, como marcos de desarrollo ágiles caracterizados, según el artículo en Wikipedia “Scrum (desarrollo de software)” (s.f.), por:

1. Se desarrolla de manera gradual o incremental en cambio de la planificación y creación de un producto totalmente terminado.
2. Se basa la calidad de los resultados por las cualidades que muestran las personas a  vez de procesos completados por empleados.
3. Se desarrollan ideas y estas se pueden añadir de forma libre y que no conlleve dentro de fases de secuencias o en cascadas.

**2. Programación Extrema (XP)**

Nace de una idea del señor Kent Beck, exactamente del libro *Extreme Programming Explained: Embrace Change (1999).*Es considerada la metodología para el desarrollo del software que mejor satisface las condiciones bajo la que se somete los requisitos en la vida del desarrollo de un proyecto (Programación extrema, s.f.).

Ya gracias a su fácil adaptación al ambiente en desarrollo, los procesos son mejorados en cualquier fase de los procesos de la creación de nuevo software u otros sistemas relacionados.

**3. Proceso Unificado Racional (RUP)**

Este proceso conocido también *Rational Unified Process* en inglés, fue creado por la empresa [*Rational Software*](https://es.wikipedia.org/wiki/Rational_Software), está adquirida por IBM. Se le considera junto al lenguaje [UML](https://es.wikipedia.org/wiki/Lenguaje_unificado_de_modelado), el modelo estándar con más uso para tareas como: análisis, diseño, implementación y documentación orientada a objetos (Proceso Unificado Racional, s.f.).

No es un software rígido, usa su conjunto de herramientas en la adaptación y maleabilidad para diferentes proyectos según la necesidad que estos presentes.

Existen varios software como soporte de este, para sumar más características a esta metodología de desarrollo.

**4. Proceso Unificado Ágil (AUP)**

Versión simplificada del Proceso Ágil Unificado (RUP). Busca de manera más óptima mejorar y hacer más ágil los procesos basados en RUP con estas técnicas: desarrollo dirigido a pruebas (test driven development, TDD), modelado ágil, gestión de cambios ágil, y refactorización de bases de datos para mejorar productividad (Agile Unified Process, s.f.).

Las metodologías de desarrollo ágiles como Scrum, XP y AUP, vienen siendo las más usadas por empresas de desarrollo pequeñas, aunque también son usadas si se busca desarrollar un proyecto pequeño y de bajo costo, si no se tiene mucho tiempo disponible, o si los requerimientos son probables de que cambien durante el transcurso del desarrollo del sistema de información.

La metodología de desarrollo tradicional más usada es precisamente el RUP, el cual es usado por empresas las cuales disponen de mucho personal, para proyectos grandes, caros y los cuales lleven mucho tiempo. Este tipo de proyectos requieren suma seriedad por parte de los *stakeholders* cuando se trate de requerimientos, dado a que un cambio en los requerimientos puede obligar la creación de más documentación.

# REQUERIMIENTOS

En la ingeniería de software, un requerimiento, también llamado como “**requisito**”, es definido como una parte del contenido del sistema de información el cual se tiene previsto de que sea incluido en el producto final y funcional. El requerimiento puede significar una función que el sistema de información realice para que este pueda ser usado por el usuario, una función de la cual el sistema de información no depende pero que pueda afectarlo, o una función que no afectará el estado del sistema de información.

Sommerville (2006), divide la definición de requerimiento de software, dado a que el término no es usado dentro de una manera consistente, en unos casos puede significar una representación abstracta de tipo *high-level* de un servicio el cual el sistema de información debe hacer, y en otros casos puede significar una representación detallada, exacta y formal de una función del sistema.

Sommerville (2006), divide el término “requerimiento de software” en:

* **Requerimientos de usuario**: son representaciones, hechas en un lenguaje natural el cual puede ir acompañado de diagramas, de un servicio el cual se espere, sea incluido en el sistema de información, y el cual pueda ser usado, generalmente, bajo unas restricciones.
* **Requerimientos del sistema**: son definiciones de funciones, servicios y restricciones del sistema de información hechas bajo un lenguaje formal, estas definiciones indican de manera exacta sobre qué va a ser implementado en el sistema de información. Los requerimientos del sistema pueden hacer parte del contrato entre el stakeholder y los desarrolladores.

Los requerimientos de software son clasificados en requisitos funcionales, no funcionales, y de dominio, según Sommerville (2006), se definen como:

* **Requerimientos funcionales**: los requerimientos funcionales describen lo que el sistema de información debe hacer, esto puede variar según el tipo de software desarrollado, los stakeholders, o el enfoque tomado por los desarrolladores al hacer los requerimientos, de vez en cuando, un requerimiento funcional puede indicar lo que el sistema no debe hacer.
* **Requerimientos no funcionales**: son requerimientos que no se encuentran directamente relacionados a ciertas funcionalidades del sistema, generalmente son restricciones del sistema o funciones que el sistema ofrece al usuario.
* **Requerimientos de dominio**: los requisitos de domino son considerados como aquellos que están relacionados con la parte de la funcionalidad del sistema, más relacionado que con la parte de las necesidades específicas de los stakeholders. Requerimientos de este tipo pueden volverse requerimientos funcionales, pueden restringir otros requerimientos funcionales, o pueden indicar sobre cómo se deben llevar a cabo algunas labores de computación en específico.

# ESTÁNDARES PARA EL DESARROLLO DE REQUERIMIENTOS

Para el desarrollo de requerimientos, se utilizan ciertos estándares, los cuales dependen de la metodología de desarrollo y el enfoque, por ejemplo, aquellas metodologías pesadas como el RUP, requieren de estándares más estrictos, por lo cual es más complicado y toma más tiempo realizar la parte de requerimientos, mientras que las metodologías ágiles como Scrum, utilizan estándares que permiten realizar los requerimientos de manera más resumida y con menos restricciones.

El IEEE (*Institute of Electrical and Electronics Engineers*, en español: Instituto de Ingenieros Eléctricos y Electrónicos), es una organización mundial que se encarga de la estandarización, así como la investigación en las áreas relacionadas con la electricidad y la electrónica.

El IEEE ha distribuido dos estándares de desarrollo de requerimientos, los cuales han tenido una gran aceptación en sus tiempos:

* IEEE 830/1998: este conjunto de normas tiene como objetivo la reglamentación de la creación de los llamados “SRS” (*Software requirements specification*, en español: Especificación de requerimientos de software), el SRS se trata de una descripción del sistema de información en desarrollo, este contiene los requerimientos funcionales y no funcionales (Software requirements specification, s.f.).

Según el IEEE, el SRS sirve como una herramienta de comunicación entre los stakeholders y los proveedores, sus objetivos consisten en (Software requirements specification, s.f.):

* + Facilitar revisiones
  + Describir el alcance del trabajo
  + Proveer una referencia a los desarrolladores.
  + Proveer un *framework* para testear los casos de uso primario y secundario.
  + Incluir *features* para los requerimientos del cliente. El término *feature*, según el IEEE 829 (2008), se define como una “Característica distinguible de un software (ej., desempeño, portabilidad, o funcionalidad).”
* IEEE 1233/1998: este conjunto de normas, a diferencia del IEEE 830, reglamenta la creación de los llamados “SyRS” (*System Requirements Specification*, en español: Especificación de requerimientos del sistema), el SyRS consiste en una colección estructurada de información el cual contiene los requerimientos de un sistema (System requirements specificaction, s.f.). El IEEE 1233 comparte muchas similitudes al IEEE 830.

El ISO (*International Organization for Standardization*, en español: Organización Internacional de Normalización), es un organismo internacional, el cual se compone de los representativos de varias organizaciones de estandarización nacionales. La organización se encarga de promover la estandarización de las actividades comerciales, industriales y propietarios, a nivel mundial (International Organization for Standardization, s.f.).

El ISO/IEC 29110, consiste en una serie de estándares llamados “*Systems and Software Engineering – Lifecycle Profiles for Very Small Entities (VSEs)*” (en español: Ingienería de Sistemas y Software – Perfiles de Ciclo de Vida Para Entidades Muy Pequeñas (VSEs)” estos estándares fueron desarrollados junto al Working Group 24 (WG24) del Subcomité 7 (SC7), del Joint Technical Committee 1 (JTC1) del ISO, junto a la International Electrotechnical Comission (IEC) (ISO 29110, s.f.). Su objetivo es la mejora de la calidad de los productos y servicios, así como mejorar el desempeño de los procesos (Dominguez, Lizana & Saavedra, 2013), esto, enfocado para las empresas pequeñas.

# SCRUM

Es un modelo, el cual fue definido por Ikujiro Nonaka y Hirotaka Takeuchi a principios de los años ochenta, en la época cuando algunas principales empresas de tecnología empezaron a usar ciertos procesos de producción: Fuji-Xerox, Canon, Honda, NEC, Epson, Brother, 3M y Hewlett-Packard.  
  
En estudios comparativos con la nueva manera de ejecutar trabajo en equipo, con el adelanto en preparación de melé (Scrum en inglés) de los profesionales en  deporte Rugby, a partir de lo cual quedó como la definición “scrum” para referirse a esta.  
  
Aunque esta idea de trabajo surgió en empresas de productoras de tecnología, es usable para cualquier tipo de proyecto con requisitos inestables y para los que requieren rapidez y flexibilidad, situaciones frecuentes en el desarrollo de determinados sistemas de software.  
  
En 1995, Ken Schwaber presentó “Scrum Development Process” en OOPSLA 95 (Object-Oriented Programming Systems & Applications conference) (SCRUM Development Process), un marco de reglas para desarrollo de software, basado en los principios de Scrum, y que él había empleado en el desarrollo de Delphi, y Jeff Sutherland en su empresa Easel Corporation (Scrum (desarrollo de software), s.f.).

En un desarrollo en el que se use la metodología Scrum, según el artículo en Wikipedia “Scrum (desarrollo de software)” (s.f.), intervienen los siguientes roles:

**Product Owner**

El *Product Owner* representa la opinión del cliente, Se asegura de que el equipo Scrum trabaje de forma adecuada desde la perspectiva del negocio. El Product Owner escribe historias de usuario, las prioriza, y las coloca en el Product Backlog.

**Scrum Master (o Facilitador)**

El Scrum es facilitado por un Scrum Master, cuyo trabajo primario es eliminar los obstáculos que impiden que el equipo alcance el objetivo del sprint. El Scrum Master no es el líder del equipo (porque ellos se auto-organizan), sino que actúa como una protección entre el equipo y cualquier influencia que le distraiga. El Scrum Master se asegura de que el proceso Scrum se utiliza como es debido. El Scrum Master es el que hace que las reglas se cumplan.

**Equipo de desarrollo**

El equipo tiene la responsabilidad de entregar el producto. Es recomendable un pequeño equipo de 3 a 9 personas con las habilidades transversales necesarias para realizar el trabajo (análisis, diseño, desarrollo, pruebas, documentación, etc.).

**Roles Auxiliares**

Usados esta clase de roles para complementar procesos en desarrollo de proyectos.

**Stakeholders (Clientes, Proveedores, Vendedores, etc.)**

Son las personas que hacen posible el proyecto y para quienes el proyecto producirán el beneficio acordado que justifica su desarrollo. Sólo participan directamente durante las revisiones del "sprint"

**Administradores (Managers)**

Son los responsables de establecer el entorno para el desarrollo del proyecto.

Un desarrollo de software bajo la metodología Scrum posee el siguiente ciclo de vida:

1. **Redacción de historias de usuario**: las historias de usuario, las cuales hacen parte de los requerimientos de un sistema de información, son escritas bajo los criterios que son manejados por la misma metodología Scrum, además, se les asigna una referencia.
2. **Estimación de historias de usuario**: las historias de usuario, después de haber sido redactadas, son estimadas mediante una práctica llamada “***planning poker***”, donde los involucrados en el proyecto le otorgan a una historia de usuario una cierta cantidad de puntos el cual puede ser de 0.5 hasta 100, aunque el valor adecuado es de 0.5 hasta 8. Los puntos simbolizan la cantidad de días en el que se va a atender la historia de usuario.
3. **Priorización por valor y riesgo**: las historias de usuario son organizadas de acuerdo a su prioridad, es decir, se definen que historias de usuario se atenderán primero.
4. **Product backlog**: el product backlog constituye las historias de usuario debidamente organizadas y priorizadas, estas se preparan para ser organizadas en varias fases llamadas “***sprints***”.
5. **Junta de planeación del Sprint**: se organizan los *sprints*, los cuales consisten en series de historias de usuario que representan una parte del sistema de información, los cuales poseen una cantidad de puntos (días) definidos por la estimación de las historias de usuario, esto es realizado para la disminución de los riesgos.
6. **Sprint**: el sprint como tal se lleva a cabo, los desarrolladores deberán atender las historias de usuario contenidas en el sprint durante el tiempo contemplado, definido por los puntos. Durante el sprint, se llevan a cabo los “***Daily Scrum***”, juntas diarias donde se evalúa el progreso del sprint y se solucionan problemas relacionados con las historias de usuario.
7. **Junta de revisión del Sprint**: es una junta realizada generalmente al final de cada sprint, donde se reúnen todos los involucrados en el desarrollo del proyecto, los desarrolladores muestran el progreso de proyecto y se le da explicación a las historias de usuario no realizadas, adicionalmente, se puede realizar el compromiso del siguiente sprint (mayra13, 2015)
8. **Retrospectiva del Sprint**: posteriormente de ser realizada la revisión del sprint, se realiza otra junta, de pequeño tamaño, donde se discute acerca de los nuevos requerimientos, los cambios (en caso de que existan), o lo que se debe modificar o eliminar para el avance del cliente (mayra13, 2015).

Una historia de usuario, en la metodología Scrum:

Es la representación de un requerimiento, lo que en los métodos tradicionales seria la realización de casos de uso de manera detallada.  
Una historia de usuario representa un requerimiento con valor importante para el desarrollo del sistema, se define como una breve descripción de una necesidad del usuario y las pruebas que determinan la finalización de la historia descrita de una manera entendible para cualquier usuario del sistema. Las HU son redactadas sobre papel pequeño (fichas bibliográficas, por ejemplo), esto facilita la administración de los requerimientos sin la necesidad de realizar gran cantidad de documentos, de la misma forma, facilita la realización del cambio al sistema (mayra13, 2015).

Al redactar la historia de usuario, debe estar presente un rol de la aplicación, la función la cual va a realizar, así como la razón por la cual se busca implementar la función.

Yo como "rol", quiero, necesito o deseo "función" para "razón o causa"

Ejemplo 1:

“Yo como administrador, quiero una función de restricción para los usuarios que no cumplan con las reglas de mi servidor.”

Ejemplo 2:

“Yo como usuario final, deseo una función de verificación de dos pasos, la cual me exija introducir un código el cual sea enviado a mi teléfono móvil, para poder ingresar a la aplicación.”

Para redactar una historia de usuario se deben tener en cuenta los siguientes criterios:

* La historia de usuario debe contener la menor cantidad de lenguaje técnico posible, esto para que pueda ser entendido por los stakeholders.
* La historia de usuario debe ser corta, para poder facilitar el proceso de priorización.
* La historia de usuario en sí debe ser una parte pequeña del sistema de información, la cual pueda ser priorizada en un rango de 0.5 a 8 puntos durante el planning poker. Si la historia de usuario posee una estimación mayor a 8 puntos, es mejor dividirla en historias más pequeñas.

# ACTIVIDADES DE TRANSFERENCIA DEL CONOCIMIENTO

**Seleccionar la metodología de desarrollo que va a aplicar en el proyecto formativo.**

Se determinó que la metodología tipo Scrum, supone la mejor metodología de desarrollo para aquellas personas las cuales comiencen en el desarrollo de software, un desarrollo con la metodología Scrum supone menos documentación en caso de que los requerimientos hayan cambiado. Además, el desarrollo puede llevarse a cabo con pocas personas y poco presupuesto.

Como segunda metodología, se seleccionó el paradigma de programación orientada a objetos, ya que con esta es posible la creación de aplicativos web, algo difícil diferenciado a otros paradigmas como la programación estructurada.

**Elaborar los requerimientos según la metodología de desarrollo que va aplicar en el proyecto formativo:**

|  |  |
| --- | --- |
| **Referencia** | **Descripción** |
| US001 | Yo como **líder,** deseo administrar el rol instructor |
| US002 | Yo como **líder,** deseo administrar los roles de la aplicación |
| US003 | Yo como **líder,** deseo administrar tipos de documentos para poder crear usuarios. |
| US004 | Yo como **líder,** deseo crear de forma masiva, usuarios de rol aprendiz mediante el cargue de un archivo plano. |
| US005 | Yo como **líder,** deseo administrar las jornadas de formación. |
| US006 | Yo como **líder,** deseo asignar usuarios de rol instructor a rol jurado. |
| US007 | Yo como **líder,** del programa deseo administrar los trimestres de cada programa de formación. |
| US008 | Yo como **líder,** del programa deseo administrar las fichas de programas de formación. |
| US009 | Yo como **líder,** del programa deseo administrar los grupos de proyecto de todas las fichas. |
| US010 | Yo como **instructor,** deseo administrar los grupos de proyecto de cada ficha en la que doy formación. |
| US011 | Yo como **líder**, deseo agregar aprendices a los grupos de proyectos. |
| US012 | Yo como **instructor,** deseo agregar aprendices a los grupos de proyectos. |
| US013 | Yo como **líder,** deseo administrar los estados de formación de los aprendices en el área en la estoy encargado(a). |
| US014 | Yo como **líder,** deseo asignarle al instructor la ficha y el trimestre. |
| US015 | Yo como **líder,** deseo administrar los tipos de instructores |
| US016 | Yo como **líder,** deseo administrar los programas de formación, sus códigos y versiones. |
| US017 | Yo como **líder,** deseo administrar las competencias de un programa de formación. |
| US018 | Yo como **líder,** deseo administrar resultados de aprendizajes de las competencias. |
| US019 | Yo como **líder,** deseo asignar los resultados de aprendizaje a cada trimestre |
| US020 | Yo como **líder,** deseo administrar listas de chequeo para un programa de formación. |
| US021 | Yo como **líder,** deseo asignar una lista de chequeo a cada ficha. |
| US022 | Yo como **líder**, deseo administrar los ítems para la lista de chequeo. |
| US023 | Yo como l**íder,** deseo administrar valoraciones para poder evaluar los ítems de la lista de chequeo. |
| US024 | Yo como **jurado,** deseo evaluar los ítems de las listas de chequeo. |
| US025 | Yo como **jurado,** deseo agregar observaciones generales al grupo de proyecto durante la sustentación. |
| US026 | Yo como **jurado,** deseo agregar observaciones a cada ítem de la lista de chequeo de cada uno de los proyectos. |
| US027 | Yo como **usuario final,** deseo recuperar la contraseña de ingreso a la aplicación. |
| US028 | Yo como **usuario final**, deseo poder ingresar a la aplicación ingresando mis respectivos datos. |
| US029 | Yo como **usuario final,** deseo cerrar sesión para que otra persona no pueda usar mi sesión |
| US030 | Yo como **aprendiz**, deseo ver un informe de resultados de mi proyecto. |
| US031 | Yo como **instructor**, deseo ver un informe de resultados de los proyectos de la ficha, o fichas donde doy formación, de manera individual y de manera general. |
| US032 | Yo como **usuario final**, deseo poder actualizar mis datos como lo son, tipo y número de documento, etc. |
| US033 | Yo como **líder**, quiero fusionar fichas de los programas de formación. |
| US034 | Yo como l**íder** deseo Configurar el servidor del correo electrónico para que la aplicación pueda enviar correos. |
| US035 | Yo como **líder**, del programa deseo asignarle trimestres a las jornadas. |
| US036 | Yo como **líder**, deseo administrar los usuarios y sus roles. |
| US037 | Yo como **líder**, deseo visualizar las estadísticas de los proyectos del área. |

\*Las historias de usuario que fueron eliminadas so indicadas en color rojo.

# BIBLIOGRAFÍA

1. Delgado, E. (2008). *Metodologías de desarrollo de software. ¿Cuál es el camino?*. Recuperado de <http://www.monografias.com/trabajos60/metodologias-desarrollo-software/metodologias-desarrollo-software.shtml>
2. *Software development process*, (s.f.). En Wikipedia (en Inglés). Recuperado el 02 de Noviembre de 2016 de <https://en.wikipedia.org/wiki/Software_development_process>
3. Tutorialspoint, (s.f.). *SDLC – RAD* *Model*. Tutorialspoint. Recuperado de <https://www.tutorialspoint.com/sdlc/sdlc_rad_model.htm>
4. Centers for Medicare and Medicaid Systems [CMS]. (2008) *Selecting a Development Approach* (en Inglés). Department of Health & Human Services – USA [HHS]. Recuperado de <https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf>
5. *Scrum (desarrollo de software)*, (s.f.). En Wikipedia. Recuperado el 02 de Noviembre de 2016 de <https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)>
6. *Programación extrema*, (s.f.). En Wikipedia. Recuperado el 02 de Noviembre de 2016 de <https://es.wikipedia.org/wiki/Programación_extrema>
7. *Proceso Unificado Racional*, (s.f.). En Wikipedia. Recuperado el 02 de Noviembre de 2016 de <https://es.wikipedia.org/wiki/Proceso_Unificado_Racional>
8. *Agile Unified* Process, (s.f.). En Wikipedia. Recuperado el 02 de Noviembre de 2016 de <https://es.wikipedia.org/wiki/Agile_Unified_Process>
9. *Sommerville, I*. (2006). *Software Engineering*. Recuperado de <https://ifs.host.cs.st-andrews.ac.uk/Books/SE8/index.html>
10. *Software requirements specification*, (s.f.). En Wikipedia (en Inglés). Recuperado el 02 de Noviembre de 2016 de <https://en.wikipedia.org/wiki/Software_requirements_specification>
11. *System requirements specification*, (s.f.). En Wikipedia (en Inglés). Recuperado el 02 de Noviembre de 2016 de <https://en.wikipedia.org/wiki/System_requirements_specification>
12. *International Organization for Standardization*, (s.f.). En Wikipedia (en Inglés). Recuperado el 02 de Noviembre de 2016 de <https://en.wikipedia.org/wiki/International_Organization_for_Standardization>
13. *ISO 29110*, (s.f.). En Wikipedia (en Inglés). Recuperado el 02 de Noviembre de 2016 de <https://en.wikipedia.org/wiki/ISO_29110>
14. mayra13. (2015) *SCRUM Master*. Recuperado de <https://mayra13.gitbooks.io/scrum-master/content/index.html>