Metodologías de desarrollo de software

Breve descripción:

Las metodologías de desarrollo de software proponen un conjunto de procesos y actividades que deben ser desarrolladas por el equipo de desarrollo de software para realizar un trabajo organizado que sea fácil hacerle seguimiento y de esta forma establecer planes de mejora en busca de una mejor calidad de los productos y servicios que se desarrollan.

Mayo 2024

Grupo de Ejecución de la Formación Virtual

Tabla de contenido

Introducción

El mundo del desarrollo de software es complejo y variado, con una amplia gama de metodologías y marcos de trabajo diseñados para enfrentar los retos que supone la creación de sistemas efectivos y eficientes. En este componente, se exploran las principales metodologías y marcos de trabajo que han moldeado la ingeniería de software moderna.

- **Metodologías de desarrollo de software**

  Aquí se examina el enfoque estructurado para el desarrollo de software, incluyendo la comprensión de requisitos, diseño, implementación, pruebas y mantenimiento. Discutiremos cómo estas metodologías ayudan a gestionar la complejidad y asegurar la calidad en la entrega de productos de software.

- **Marcos de trabajos tradicionales**

  Este apartado se sumerge en las metodologías clásicas como el modelo en Cascada y el Proceso Racional Unificado (RUP). Analizaremos cómo estos enfoques secuenciales y predecibles pueden proporcionar una estructura sólida para el desarrollo de software, a la vez que consideramos sus limitaciones en entornos de cambio rápido.

- **Marcos de trabajo ágiles**

  La agilidad es la clave de las metodologías modernas, permitiendo a los equipos responder a la imprevisibilidad y al cambio constante. Exploraremos las prácticas y principios de marcos ágiles como la Programación Extrema (XP), el Desarrollo Rápido de Aplicaciones (RAD) y Scrum, que promueven la colaboración, adaptabilidad y entrega continua.

- **Planeación de proyectos de software**

  Desde la asignación de recursos hasta la gestión de riesgos y la definición de cronogramas, la planificación es vital para el éxito y la sostenibilidad de cualquier proyecto de software.

Con estas bases establecidas, se adentrará en el corazón de cada metodología y marco de trabajo, buscando no solo entender su propósito y prácticas, sino también cómo y cuándo implementarlas de manera efectiva en los proyectos de software.

1. Metodologías de desarrollo de software

Es importante aclarar que existe un gran número de definiciones sobre lo que es una metodología, para evitar cualquier confusión en este material se utilizará la definición dada por Maida y Pacienzia (2015), quienes indican que:

Una metodología hace referencia a un conjunto de procedimientos genéricos y lógicos que se usan para alcanzar un objetivo particular utilizando un conjunto de habilidades y conocimientos.

Las metodologías de desarrollo de software siempre parten de un componente teórico y, cuando son usadas por los equipos de trabajo, conllevan a la utilización de un conjunto de técnicas y métodos que al final determinarán las tareas generales y específicas que se deberían realizar para alcanzar un objetivo.

Existen diferentes tipos de metodologías de desarrollo de software que fueron ideadas pensando en problemas particulares presentados en la industria en contextos específicos, por lo cual es importante conocer sus diferentes características y contrastarlas con las necesidades particulares a las que se enfrenta a la hora de desarrollar un producto o servicio. Es decir, cada una de estas tiene ventajas y enfoques que pueden ser reutilizados en diferentes momentos.

Existen dos grandes clasificaciones de metodologías de desarrollo de software que se agrupan generalmente como marcos de trabajo tradicionales o marcos de trabajo ágiles que se presentan a continuación.

2. Marcos de trabajos tradicionales

Para el desarrollo de un buen producto de software se debe iniciar con un excelente proceso de planificación y gestión durante todas las etapas y actividades que involucran transformar una idea o requerimiento en un producto o servicio que será utilizado por un cliente particular.

"Los marcos de trabajo o metodologías tradicionales se caracterizan por centrar la mayor parte de su esfuerzo en la planificación y control del proceso, lo que conlleva a una documentación exhaustiva y precisa de los artefactos que describen los requisitos y los modelos del sistema en las etapas iniciales del desarrollo del proyecto" (Maida y Pacienzia, 2015).

Lo anterior supone que este tipo de enfoques son óptimos en proyectos en los cuales los requisitos están plenamente identificados y delimitados, y donde no se espera que se produzca ningún cambio en lo establecido mientras el proyecto es finalizado.

A continuación, se describen algunas metodologías que se enmarcan en los marcos tradicionales de desarrollo de software.

2.1. Cascada

Este es uno de los modelos genéricos más ampliamente conocidos en la ingeniería de software y se deriva de procesos de ingeniería de sistemas más generales (Royce, 1970). Este modelo plantea un proceso lineal donde las actividades de desarrollo de un producto o servicios de software se agrupan en un conjunto de fases sucesivas. Estas son desarrolladas una única vez y los resultados de cada fase son la entrada requerida para la siguiente, ninguna fase puede iniciar si la fase anterior no

ha sido finalizada, generalmente mediante un formalismo que puede ser un documento.

Según Sommerville (2005), el modelo en cascada se compone de cinco etapas principales que se asocian con actividades fundamentales en el proceso de desarrollo de software, las cuales son:

- **Requerimientos**

  Se lleva a cabo la recopilación y análisis detallado de las necesidades del sistema o software. Se definen los objetivos, funciones, interfaces de usuario y demás necesidades del cliente o mercado.

- **Diseño**

  Se establece cómo será construido el sistema para cumplir con los requerimientos identificados. Incluye la arquitectura del sistema, la selección de patrones de diseño, y la planificación de interfaces y experiencias de usuario.

- **Implementación**

  En esta fase, se escribe el código fuente del software, basado en el diseño previamente establecido. Cada componente es desarrollado y probado individualmente antes de ser integrado en el sistema mayor.

- **Verificación**

  Esta etapa implica probar el sistema completo para validar que cumple con los requerimientos especificados. Incluye pruebas unitarias, de integración, de sistema y de aceptación del usuario.

- **Mantenimiento**

  Una vez que el software está en operación, esta fase aborda la corrección de errores, la actualización de funciones y la optimización del rendimiento. También se pueden incorporar nuevos requerimientos según sea necesario.

Continuando con el tema, se presenta un comparativo que resume las principales ventajas y desventajas de este modelo.

Ventaja

- La definición clara de fases permite el desarrollo de una estructura sencilla que es ideal para proyectos sencillos y cortos.
- Siguiendo este modelo se genera una muy buena documentación del proceso y es posible definir hitos claros.
- Es más fácil realizar la estimación de costos al inicio del proyecto.
- Es fácil elaborar cronogramas de trabajo con base al desarrollo de las actividades de cada fase.

Desventaja

- No se acopla muy bien a proyectos complejos con múltiples equipos trabajando en paralelo, ya que las fases normalmente se traslapan y es difícil diferenciarlas.
- Es difícil introducir cambios en el transcurso del proyecto.
- Los usuarios finales y clientes normalmente son integrados al final del proceso, lo que impide tener realimentación y ajustes en etapas tempranas.

- Existen fallos que solo son detectados cuando el sistema entra en funcionamiento, lo que puede ser desastroso para un proyecto que sigue este modelo.

El modelo en cascada define cuatro grupos de roles típicos, los cuales se mencionan a continuación:

- **Desarrolladores**

  Es el rol más importante en la metodología cascada, ya que son los encargados directos de la creación de código.

- **Testers**

  Encargados de encontrar fallas en los productos finales y retornar el software a los desarrolladores para arreglar todos los defectos.

- **Analista del negocio**

  Encargado de la realización de estrategias de negocio que le permitan al producto software alcanzar popularidad en el mercado digital.

- **Administrador del proyecto**

  Es responsable de la calidad final del software. Administra el proyecto y lo subdivide en tareas entre los miembros del equipo.

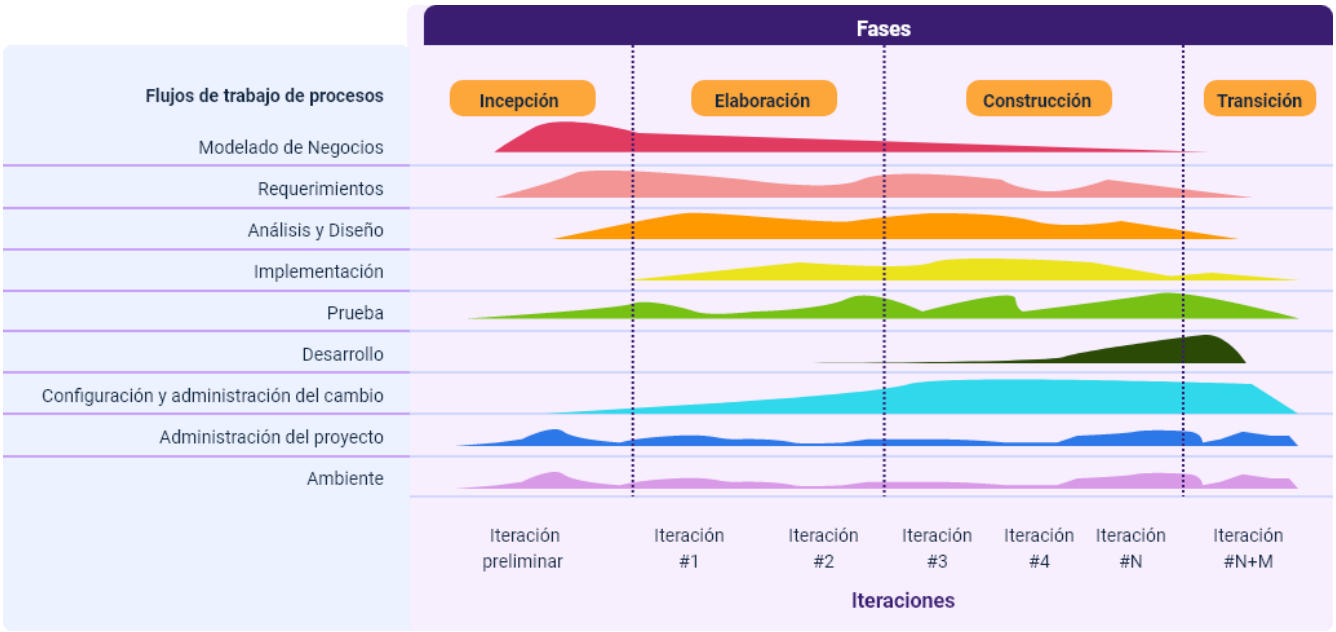
2.2. Proceso Racional Unificado - RUP

RUP es una sigla en inglés que corresponde a Rational Unified Process, el cual es un proceso de desarrollo de software tradicional que se basa en el modelo en cascada y fue desarrollado por la empresa Rational Software, actualmente propiedad de IBM. Esta metodología se enfoca en la arquitectura y es guiada por casos de uso (requerimientos) (Kruchten, 2003).

RUP divide el proceso de desarrollo en cuatro grandes fases. Dentro de ellas, se realizan iteraciones donde se desglosan, con mayor o menor intensidad, un conjunto de disciplinas según la fase en la que se está.

A continuación, se presentarán las fases y disciplinas propuestas por RUP.

Figura 1. Fases y disciplinas de RUP



A continuación, se describen las fases y su impacto en el proceso, por lo que se le invita a revisar el siguiente recurso de aprendizaje.

- **Inicio**

  La primera gran fase definida por RUP es la de inicio, en la cual se abordan actividades principalmente enfocadas en la comprensión del problema y el tipo de tecnología a utilizar, por lo que hay una gran carga en actividades relacionadas con la disciplina de modelado del negocio y especificación de requisitos.

- **Elaboración**

  La segunda fase es la elaboración, donde se centra la mayor parte del esfuerzo en la definición general de la arquitectura del sistema y el refinamiento de requisitos y modelado del negocio. RUP generalmente se apoya en el lenguaje de modelado UML para el modelado del negocio y descripción de la arquitectura.

- **Construcción**

  La tercera fase corresponde a la construcción, donde se centran en las actividades relacionadas con la construcción del producto. Normalmente, esta fase está constituida por varias iteraciones, donde en cada una de ellas se desarrolla un subconjunto de requerimientos que normalmente están especificados como casos de uso.

- **Transición**

  La cuarta fase se desarrolla principalmente las disciplinas de pruebas y despliegue, es decir, las actividades encaminadas a garantizar que el producto esté listo para entrega a sus usuarios.

Es importante señalar que, dentro de cada una de las fases del RUP, se desarrollan varias disciplinas, y el esfuerzo y tiempo invertidos en cada una tendrán un enfoque diferente dependiendo de la fase del proyecto en la que se esté.

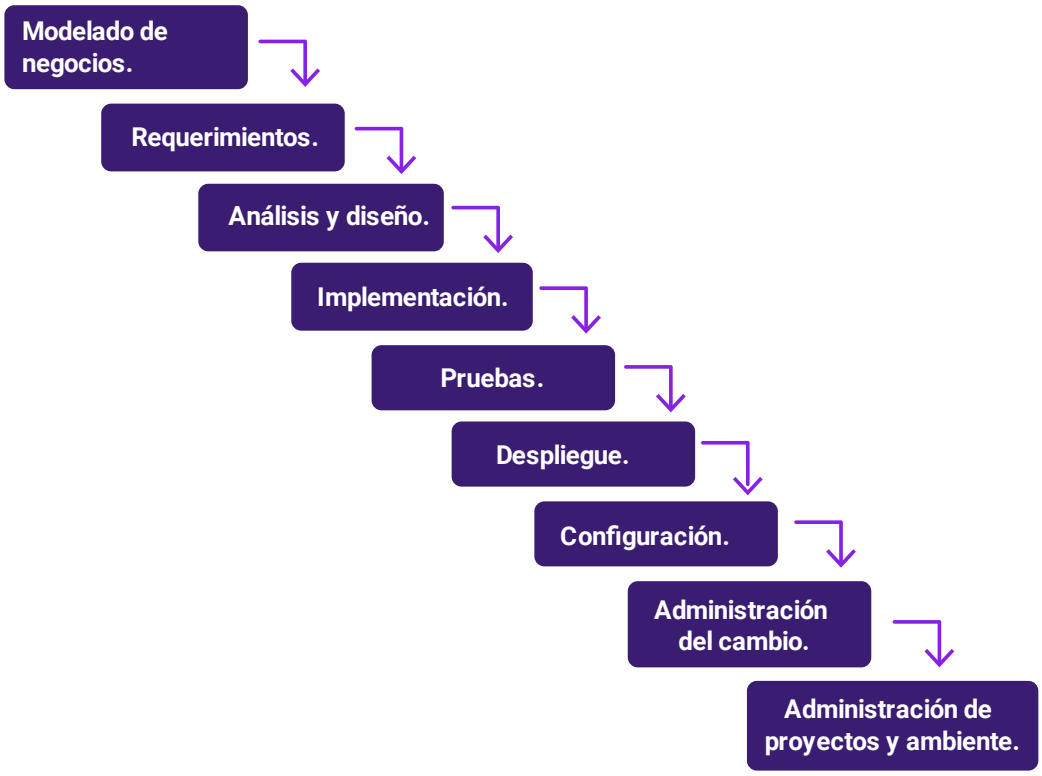
Otro aspecto relevante de esta metodología es que se sustenta en un conjunto de artefactos que se elaboran con el objetivo de especificar el proceso de análisis y diseño que respalda la construcción del software. Entre los artefactos más comunes se incluyen:

- Documento de visión.

- Documento de especificación de requisitos.
- Diagramas de casos de uso.
- Modelos conceptuales (clases y entidad relación).
- Diagramas que representan la vista de implementación como:
    - Diagramas de secuencia.
    - Diagramas de estados.
    - Diagramas de colaboración, entre otros.

Las disciplinas, por otra parte, representan un conjunto de actividades relacionadas con un área específica del proyecto y están inspiradas en el modelo en cascada. RUP establece las siguientes disciplinas según Kruchten (2003):

Figura 2. Fases en el desarrollo de proyectos



A continuación, se describen cada una de las disciplinas mencionadas y cómo estás aportan en el proceso.

RUP propone una categorización de roles encargados de la realización de actividades dentro de cada una de las disciplinas que son:

- **Analistas**
  - Analistas de procesos de negocio.
  - Diseñadores del negocio.
  - Analistas del sistema.
  - Especificador de requisitos.
  - Diseñadores de interfaces de usuario o similares.
- **Desarrolladores**
  - Arquitectos de software.
  - Diseñador de bases de datos.
  - Desarrollador backend y frontend.
  - Cualquier otro rol relacionado con procesos de codificación o integración de código.
- **Testers**
  - Diseñadores de pruebas.
  - Implementadores de pruebas.
- **Encargados y otros actores**

  Incluye a todos los roles mencionados anteriormente en las categorías de Analistas, Desarrolladores y Testers, reflejando su importancia en la gestión y operación de proyectos.
  - Artistas gráficos.

- Administradores de sistemas.
- Especialista en herramientas.
- Stakeholders.
- Cualquier otro rol no especificado anteriormente.

Existen también metodologías catalogadas dentro de los marcos de trabajo tradicionales. Sin embargo, el propósito de este texto es resaltar las características principales de las más destacadas en la industria del software. Para aquellos interesados en profundizar sobre las ventajas y desventajas de estas y otras metodologías, se recomienda revisar la sección de materiales complementarios.

3. Marcos de trabajo ágiles

Los marcos de trabajo ágiles, también conocidos como metodologías ágiles para el desarrollo de software, surgen como alternativa para gestionar proyectos en los que no es posible definir completamente los requerimientos y sus estimaciones en la fase inicial, o cuando se requiere adaptabilidad a lo largo del desarrollo del software. (Maida y Pacienzia, 2015).

Estas metodologías ofrecen un conjunto de pautas y principios diseñados para facilitar y dar prioridad a la entrega de productos por encima de los procesos de documentación exhaustiva, simplificándolos y promoviendo la participación activa del cliente desde las primeras etapas del proyecto.

El origen de las metodologías ágiles se remonta al año 2001 con la publicación del Manifiesto Ágil para el Desarrollo de Software, que establece cuatro valores fundamentales. (Manifiesto Ágil, 2001):

- Individuos e interacciones sobre procesos y herramientas.
- Software funcionando sobre documentación extensiva.
- Colaboración con el cliente sobre negociación contractual.
- Respuesta ante el cambio sobre seguir un plan.

Adicionalmente a los valores ágiles anteriormente listados, el manifiesto ágil establece 12 principios ágiles para materializar los valores definidos (Manifiesto Ágil, 2001):

a) Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software de valor.

b) Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo. Los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente.
c) Entregar con frecuencia software que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia en los períodos breves.
d) Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.
e) Construcción de proyectos en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y procurándoles confianza para que realicen la tarea.
f) La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.
g) El software que funciona es la principal medida del progreso.
h) Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.
i) La atención continua a la excelencia técnica enaltece la agilidad.
j) La simplicidad como arte de maximizar la cantidad de trabajo que se hace es esencial.
k) Las mejores arquitecturas, requisitos y diseños emergen de equipos que se autoorganizan.
l) En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta en consecuencia.

3.1. Programación Extrema - XP

XP, abreviatura de Extreme Programming, es un marco de desarrollo de software ágil diseñado para producir software de alta calidad en entornos con requisitos muy cambiantes. Se enfoca en contextos que presentan riesgos asociados a plazos fijos, la implementación de tecnologías emergentes y equipos de trabajo reducidos localizados en el mismo lugar.

En XP se definen cinco valores según Beck y Andrés (2004):

- **Comunicación**

  El desarrollo de software depende del trabajo en equipo, haciendo crucial la transferencia de conocimientos y el uso de medios de comunicación adecuados. Se sugiere la discusión cara a cara con herramientas que faciliten dibujar o escribir, como los tableros.

- **Simplicidad**

  Se refiere a realizar solo lo estrictamente necesario, evitando desperdicios. Las tareas deben ser ejecutadas de manera que resulten fáciles de entender para otros, concentrándose únicamente en los requisitos conocidos.

- **Retroalimentación**

  Permite identificar áreas de mejora y la revisión constante de las prácticas actuales, de modo que se puedan establecer procesos de mejora continua.

- **Coraje**

  Implica enfrentarse a situaciones desafiantes para el equipo, como problemas organizacionales o intentar implementar funcionalidades de

formas innovadoras cuando las convencionales no funcionan, y aceptar comentarios constructivos.

- **Respeto**

  Es fundamental que los miembros del equipo se respeten mutuamente para facilitar una comunicación efectiva y un trabajo en equipo armónico. Cada integrante contribuye significativamente al logro del objetivo común.

Además de sus valores, XP se distingue por establecer un conjunto de 12 prácticas de desarrollo de software. Aunque estas prácticas pueden adoptarse de manera independiente, su importancia se magnifica cuando se implementan conjuntamente (Jeffries, 2011).

- **El juego de la planificación**

  Consiste en una reunión de planificación donde el equipo de desarrollo y el cliente discuten y priorizan las características a desarrollar, promoviendo una comprensión mutua de los requisitos.

- **Pequeños lanzamientos**

  Se recomienda realizar iteraciones cortas con funcionalidades reducidas para facilitar la prueba con el cliente, permitiendo obtener retroalimentación temprana y frecuente.

- **Metáfora**

  Esta práctica enfatiza la importancia de utilizar nombres coherentes y comprensibles para cualquier identificador en el sistema, facilitando así la comprensión global del proyecto.

- **Diseño simple**

  Implica desarrollar un código que solamente haga lo que es necesario en la forma más sencilla posible, evitando duplicaciones y reduciendo el número de métodos y clases.

- **Pruebas**

  Sugiere el empleo de técnicas de prueba constantes antes de proceder con la escritura del código, como por ejemplo el TDD (Desarrollo Dirigido por Pruebas).

- **Refactorización**

  Se centra en la eliminación de elementos innecesarios, desacoplando componentes y reduciendo redundancias para mantener el código limpio, entendible y fácil de modificar.

Los roles fundamentales establecidos por este marco de trabajo ágil son los siguientes: cliente, programador, coach, tester y manager.

- Los clientes son los encargados del establecimiento de las prioridades del proyecto y las necesidades puntuales.
- Los programadores se encargan de transformar esos requerimientos en bloques de código funcional, son el centro del marco de Extreme Programming.
- Los testers se encargan de la aplicación de pruebas para garantizar la calidad de los productos y servicios desarrollados.
- El coach es una figura encargada de brindar asesoría a los miembros del equipo y son los que definen el rumbo del proyecto.

- El manager se encarga de la coordinación de actividades, del aseguramiento de los recursos requeridos para el proyecto y quien tiene la responsabilidad de la comunicación externa.

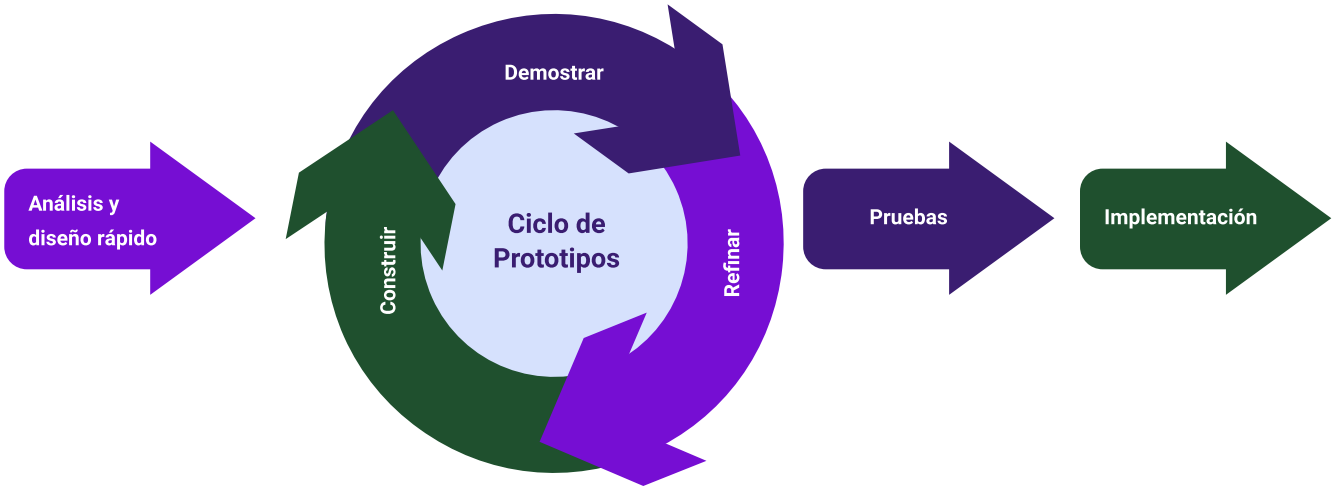
3.2. Desarrollo rápido de aplicaciones - RAD

RAD (Desarrollo Rápido de Aplicaciones) es una metodología ágil de desarrollo de software que se enfoca en la rápida creación de aplicaciones a través de iteraciones frecuentes y retroalimentación constante. Fue propuesta por James Martin en 1991. Algunas características fundamentales de RAD son:

- Mayor flexibilidad y adaptabilidad a cualquier ajuste que deba realizarse durante el proceso de desarrollo.
- Iteraciones rápidas que reducen el tiempo de desarrollo y mantienen un ritmo de entrega acelerado.
- Se fomenta la reutilización de código.
- Mejor gestión del riesgo, ya que las partes interesadas discuten y abordan cualquier vulnerabilidad mientras se construyen los productos.

A continuación, se describen las fases propuestas en RAD.

Figura 3. Fases definidas en RAD



Nota: Tomada de Blog Capterra (2019).

El proceso de desarrollo en cinco fases clave: desde la definición precisa de los requisitos, pasando por la creación y afinamiento de prototipos, hasta las pruebas rigurosas y el lanzamiento cuidadosamente preparado del producto final.

- **Primera fase**

  Definición y finalización de requisitos:

  - Definir los objetivos del proyecto.
  - Establecer las expectativas.
  - Acordar los plazos.
  - Presupuestar el proyecto.

- **Segunda fase**

  Construcción de prototipos:

  - Construir prototipos.

- Validar y refinar prototipos con la retroalimentación de los usuarios.
- Aprobar los prototipos para pasar a la siguiente fase.

- **Tercera fase**

  Transformación en modelos funcionales:

  - Desarrollar modelos funcionales a partir de los prototipos aprobados.

- **Cuarta fase**

  Pruebas:

  - Realizar pruebas exhaustivas para asegurar que cada elemento funciona de manera individual y en conjunto.

- **Quinta fase**

  Lanzamiento:

  - Ejecutar todas las actividades necesarias para el lanzamiento del producto, incluyendo la carga inicial de datos y la capacitación de usuarios.

Según HKSAR (2009), los roles más importantes en la metodología RAD incluyen: el facilitador, el escriba, el equipo SWAT, el administrador del modelo, el administrador de la base de datos, el equipo de planificación del taller, el equipo de diseño de usuario, el equipo de soporte de construcción y el equipo de transición. A continuación, se describen las características más destacadas de cada rol.

- **Facilitador**

  Encargado de asegurar los objetivos y de los talleres de captura de requerimientos, prepara materiales, y actúa como mediador en la resolución de conflictos.

- **Escriba**

  Responsable de documentar los resultados de los talleres de captura de requerimientos y registrar la información de todo el proceso.

- **Equipo SWAT**

  Equipo encargado del diseño y construcción del sistema.

- **Administrador del modelo**

  Coordinador del desarrollo y mantenimiento de las arquitecturas y modelos resultantes del proyecto.

- **Administrador de bases de datos**

  Responsable del rendimiento, integridad y seguridad de la base de datos de la organización.

- **Equipo de planificación de los workshops**

  Asisten en la definición de requerimientos y el alcance del sistema propuesto.

- **Equipo de diseño de usuario**

  Proveen información detallada sobre las funciones del negocio y procesos afectados por el sistema propuesto.

- **Equipo de soporte de construcción**

  Se aseguran de que las necesidades del usuario final se cumplan en el sistema completo, ofreciendo conocimiento detallado de las actividades del negocio y retroalimentación sobre la usabilidad del sistema al equipo SWAT.

- **Equipo de transición**

  Desarrollan las tareas necesarias para preparar y llevar el sistema a entornos de producción.

3.3. Scrum

Scrum es un marco de trabajo ágil de muy amplio uso en la industria del software que se fundamenta en los valores y principios ágiles definidos en (Manifiesto Ágil, 2001) y donde se definen tres pilares fundamentales según (SCRUMstudy, 2013) los cuales se describen a continuación:

Transparencia

- Hace referencia a que cualquier proceso de Scrum puede ser conocido por cualquiera. Esto es posible por medio de eventos como:
- Las reuniones de revisión y reuniones diarias.
- Artefactos como la pila de producto.
- Cronogramas de lanzamiento.
- Documentos de visión del proyecto.
- Instrumentos de seguimiento, como: el burndown chart o el tablero de Scrum (Scrum board).

Inspección

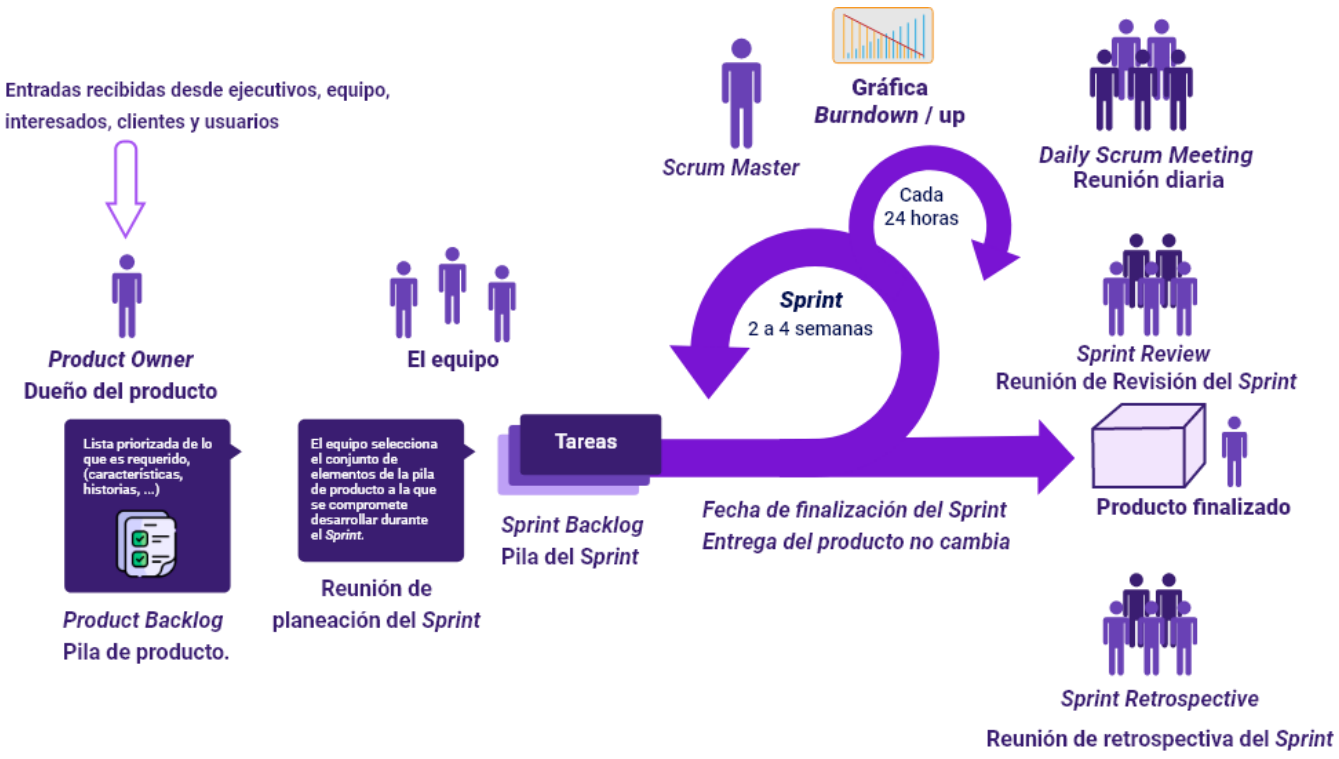
- Permite que cualquiera pueda estar enterado de las actividades realizadas por otros y en general conocer el estado actual de los procesos.

Adaptación

- Por medio de la transparencia y la inspección es posible fijar actividades de mejoras que permitan modificar todo tipo de proceso en pro de lograr más altos estándares de calidad.

Adicionalmente, este marco de trabajo ágil está estructurado por un conjunto de roles, eventos y artefactos que pueden ser presentados a continuación:

Figura 4. Marco de trabajo Scrum



Dentro de los roles hay una división en dos categorías fundamentales:

Los roles centrales

- Hacen referencia a los requeridos obligatoriamente para la creación de un producto están altamente comprometidos y de los cuales depende el éxito o no de un proyecto.

Los roles no centrales

- Refieren a todo el personal interesado en el proyecto pueden interactuar con el equipo, pero no son los responsables del éxito del mismo, dentro de esta categoría entran los stakeholders, directivos, gerentes, marketing, asesores, etc.

Por otro lado, hay tres roles centrales dentro del marco de trabajo de Scrum (SCRUMstudy, 2013) que se describen a continuación:

- **Dueño del producto (Product Owner)**

  Persona con amplio conocimiento en el negocio del cliente, sus necesidades y las tendencias del mercado para el área específica. Encargado de maximizar el valor de negocio entregado al cliente y es el único responsable del control del Product Backlog (requerimientos) y su priorización. Representa al cliente en algunos procesos de demostración de avances y determina cuándo aprobar o no una entrega.
- **Scrum Master**

  Rol que se encarga de facilitar los procesos al interior del equipo de trabajo removiendo cualquier impedimento y apoyando procesos de empoderamiento personal. Debe velar porque los elementos propios del marco de trabajo Scrum se apliquen de manera correcta.
- **Equipo de desarrollo (Developer Team)**

  Responsables de la transformación de los requerimientos en código ejecutable a ser usado por el cliente. También responsables de la planificación de las iteraciones, establecimiento de características para tener en cuenta en la verificación de un requerimiento terminado y

presentación de avances a los clientes. Generalmente es un equipo autoorganizado y autogestionado.

Además de los roles, scrum define un conjunto de eventos con participantes y objetivos claros que se desarrollan en momentos particulares del flujo general de scrum, a continuación, se detalla cada uno de estos:

- **Sprint**

  Es el corazón de scrum, una iteración acotada entre 2 y 4 semanas donde se realiza un ciclo completo de actividades de análisis, diseño, construcción y pruebas para desarrollar una versión del producto potencialmente.

- **Planeación del sprint**

  Reunión realizada antes del inicio del sprint donde se definen los requerimientos a desarrollar, se detallan en tareas concretas, se estima el tiempo/esfuerzo y se distribuyen las responsabilidades.

- **Reunión diaria (Daily Meeting)**

  Reunión corta al inicio de cada día donde el equipo informa sobre su trabajo previo, las actividades del día y los problemas encontrados. Debe ser breve y alinearse con los pilares de transparencia e inspección.

- **Revisión del sprint**

  Reunión al final del sprint donde el equipo de desarrollo muestra los resultados. Para sprints de 4 semanas, se usan 4 horas para la revisión.

- **Reunión de retrospectiva**

  Última reunión después de la revisión del sprint para autoevaluar el desempeño del equipo durante el sprint, identificando y documentando

aprendizajes y respondiendo a preguntas clave sobre lo que se debe continuar, detener o mejorar.

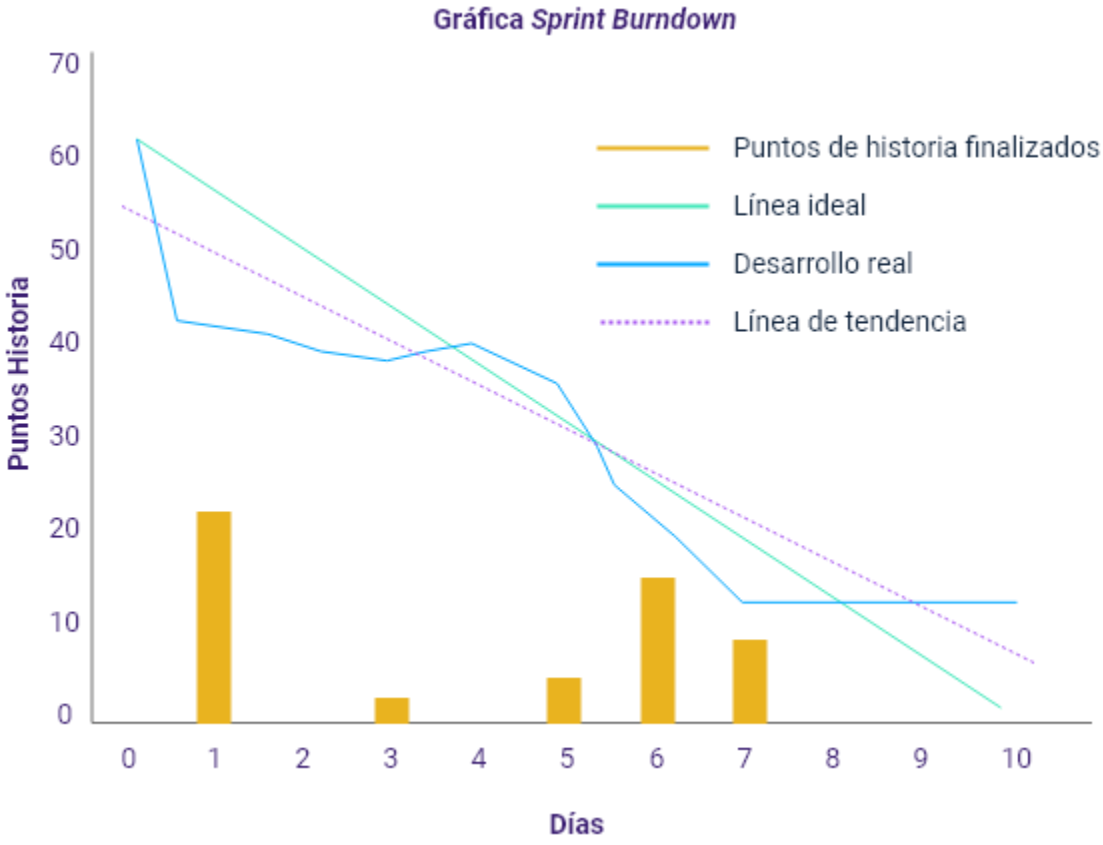
Finalmente, el marco de trabajo scrum define un conjunto de artefactos que permiten registrar y gestionar información clave para asegurar los tres pilares fundamentales y proveen información valiosa durante todo el proceso de desarrollo de software. Entre los artefactos representativos de este se encuentran los siguientes:

- **Pila de producto (Product Backlog):**
  Lista priorizada de requerimientos generalmente descritos en formato de historias de usuarios que representa todas las características del sistema a construir.
- **Pila del sprint (Sprint Backlog)**
  Lista de requerimientos seleccionados desde el product backlog por el equipo de trabajo para ser desarrollados durante un sprint particular. este es uno de los artefactos generados en la reunión de planeación del sprint.
- **Burndown Chart**
  Es un gráfico visual de dos ejes que muestra a los equipos la cantidad de trabajo pendiente por completar (eje Y) y el tiempo disponible para hacerlo (eje X). Este generalmente se realiza por cada sprint ubicando la cantidad de trabajo a realizar del sprint backlog (usualmente medido por puntos de historia u de horas de trabajo) en un tiempo 0 y por cada día finalizado se resta la cantidad de puntos de historia u horas de cada tarea completada.

Por otro lado, también es posible usar este mismo gráfico para representar el avance general del proyecto, ubicando en el eje Y la cantidad total de horas o esfuerzo

del Product Backlog y en el eje X el número de Sprints proyectados. Cada punto representa el total acumulado hasta ese Sprint y se une mediante una línea, lo que permite determinar visualmente si el flujo de trabajo está en una situación óptima o no respecto al tiempo restante para completar el Sprint actual.

Figura 5. Burndown Chart



Nota. Tomada de intl-blog.imgix.net (2019).

Tablero de Scrum (Scrumboard): es un elemento visual donde se integra la mayor parte de los elementos del marco de trabajo Scrum, en él se indica la carga de trabajo, el estado actual de cada una de las actividades y sus respectivos responsables. Este es un elemento que se sincroniza de manera permanente y facilita la implementación de los pilares de transparencia, inspección y adaptabilidad. Si bien se

aconseja el uso de un tablero, existen diferentes tipos de herramientas digitales que permiten la implementación de un tablero de Scrum.

Figura 6. Tablero de Scrum

Adicionalmente es de vital importancia mencionar que entre los principales beneficios del marco de trabajo Scrum se encuentran los siguientes elementos:

a) Es posible gestionar las expectativas del cliente de manera regular, ya que, este puede y debe participar en las reuniones de revisión, por lo que, está enterado todo el tiempo del estado actual del proyecto.
b) El cliente puede obtener resultados importantes y utilizables desde las primeras iteraciones, ya que, la lista de producto está priorizada para ofrecer mayor valor en el menor tiempo posible y porque cada finalización de Sprint debe tener como resultado una versión totalmente funcional.

c) El proyecto puede iniciar con requerimientos de muy alto nivel y es fácil administrar los cambios.
d) La participación constante del cliente permite mitigar riesgos del proyecto desde sus primeras etapas.
e) Los procesos de retrospectiva permiten establecer actividades permanentes de mejora continua en función de las experiencias del equipo.

Un elemento clave en su formación es la planeación de proyectos de software que aunado con los contenidos ya vistos le permite tener un panorama amplio sobre el tema.

4. Planeación de proyectos de software

La planificación de proyectos de software comprende varias actividades destinadas a obtener una visión preliminar del software a desarrollar, lo que permite realizar estimaciones para determinar la viabilidad del proyecto en función de los recursos asignados.

De acuerdo con SCRUMstudy (2013), las actividades de planificación estipuladas en el marco de trabajo de Scrum se detallan en el siguiente recurso de aprendizaje:

Video 1. Planeación de proyectos de software

Enlace de reproducción del video

Síntesis del video: Planeación de proyectos de software
A continuación, se presentan los pasos que debe seguir para la planeación de proyectos de software. Creación de historias de usuario: En este proceso, que es responsabilidad del product owner, se crean las historias de usuarios con sus respectivos criterios de aceptación de forma que queden representados los requerimientos del sistema y sean comprendidos por los clientes y stakeholders. Estimación de historias de usuario: Este es un proceso realizado por el equipo de desarrollo, donde se valora el esfuerzo necesario para la realización de cada historia de usuario. Generalmente, hay un proceso previo en el cual el product owner explica y presenta cada una de las historias de usuario y aclara cualquier duda que surja. Así, cuando se realiza el proceso de estimación, todos son conscientes de las actividades que implican cada historia de usuario. Comprometer historias de usuario: En este proceso, el equipo de desarrollo identifica las historias de usuario a ser desarrolladas en el sprint. Esto se hace mediante la reunión de planificación del sprint. Identificar tareas: En este proceso, cada una de las historias de usuario se desagrega más detalladamente por medio de tareas. Estimar tareas: En este proceso, el equipo de desarrollo realiza una estimación de esfuerzo para la realización de la tarea de forma que es más sencillo asignar responsabilidades puntuales de acuerdo con las diferentes cargas del equipo. Crear el Sprint Backlog: El equipo de trabajo agrupa cada una de las tareas de las historias de usuario comprometidas en un artefacto llamado Sprint Backlog o pila del Sprint. Además, existen diferentes técnicas para la estimación de historias de usuario. Entre las más destacadas se encuentran las siguientes: Wideband Delphi,

esta es una técnica de estimación basada en consenso para determinar la cantidad de trabajo necesario y el tiempo requerido para el mismo. En esta, cada individuo realiza estimaciones individuales de las historias y luego son socializadas al grupo para analizar los factores que influyen en cada estimación. Posterior a esto, se realiza una nueva ronda de estimación individual, siguiendo el mismo procedimiento hasta que los valores de estimaciones se acercan lo suficiente o se logre un consenso.

Planning Poker es una técnica derivada de Wideband Delphi pero para la realización de las estimaciones se apoya en un conjunto de cartas especiales que vienen marcadas con números semejantes a la serie Fibonacci. Para el proceso de estimación, se lee una historia de usuario, cada miembro del equipo selecciona una de las cartas y, cuando se indica, todos al tiempo muestran sus cartas. Por otro lado, en el caso de haber diversidad muy representativa en los valores, se realiza una ronda donde los puntajes más altos y más bajos justifican sus decisiones. Después, se hace una segunda ronda de estimación. Luego, la estimación de cada historia se puede determinar por medio de consenso, democracia o promedio, según se haya definido en el grupo de trabajo.

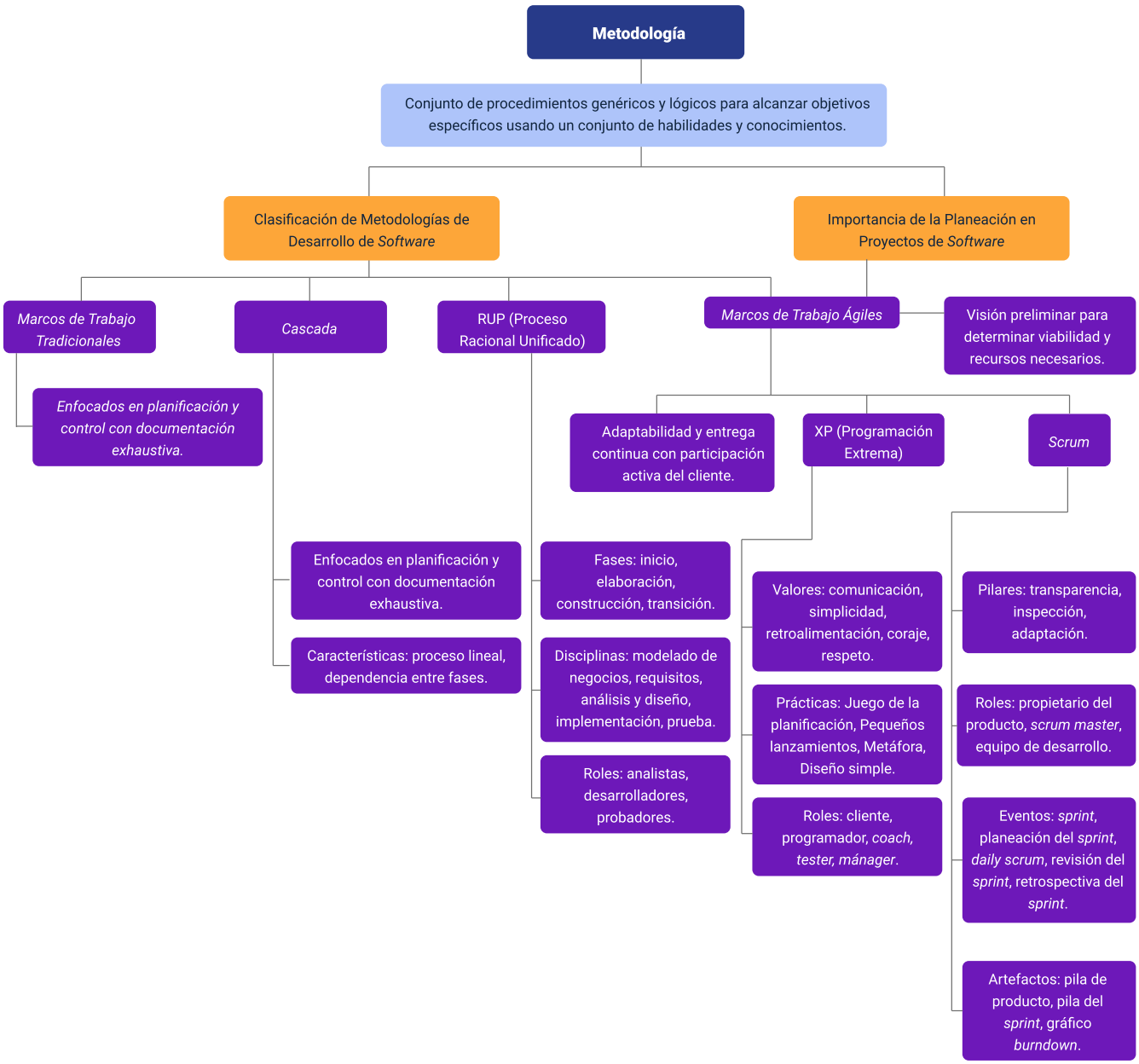
Puño de cinco: En esta técnica, se usan los dedos de la mano para expresar la estimación de cada miembro del equipo. Al igual que en el Planning Poker, se parte de una explicación inicial de la historia y luego se procede a realizar la votación. En caso de presentarse valores atípicos, se realiza una sesión corta donde se justifican los valores asignados para, posteriormente, realizar una nueva votación.

Como se puede ver, la planeación de proyectos de desarrollo del software requiere diseñar estrategias activas de participación de todos y cada uno de los

miembros del equipo para alcanzar los objetivos y los requerimientos del cliente en un plazo predefinido y acordado con el mismo.

Síntesis

A continuación, se presenta una síntesis de la temática estudiada en el componente formativo:



Material complementario

Tema	Referencia	Tipo de material	Enlace del recurso
¿Qué son las metodologías tradicionales en el desarrollo de software?	Henao, C. (2018). #1. ¿Qué son las metodologías tradicionales en el desarrollo de software?	Video	https://youtu.be/i8CPD1dW88k?si=n_wD0LM5WHg59CoO
¿Qué son las metodologías ágiles en el desarrollo de Software?	Henao, C. (2018b). #2. ¿Qué son las metodologías ágiles en el desarrollo de Software?	Video	https://youtu.be/fHKsufzM7qQ?si=9HPATMJdNFGJeYHW
SCRUM en 6 minutos | Metodologías Ágiles	Henao, C. (2018d). #3. SCRUM en 6 minutos | Metodologías Ágiles.	Video	https://youtu.be/HhC75IonpOU?si=9zcH8SvcdPyHmgwJ

Glosario

Iteración: hace referencia a un ciclo limitado por tiempo donde se ejecutan actividades de análisis, diseño, construcción y pruebas.

Product Owner: rol central de Scrum encargado de la gestión de la pila de producto y representante del cliente dentro del grupo de trabajo.

Scrum Master: rol central de Scrum encargado de facilitar el trabajo del equipo de desarrollo y de mantener la aplicación del marco de trabajo SCRUM.

Stakeholders: persona, organización o empresa interesada en el desarrollo del proyecto.

Referencias bibliográficas

Agilemanifesto.org. (2021). Manifiesto por el desarrollo ágil de software. Agilemanifesto.org. https://agilemanifesto.org/iso/es/manifesto.html

Beck, K., & Andrés, C. (2004b). Extreme Programming Explained: Embrace Change, 2nd Edition (The XP Series). Addison-Wesley.

Intl-blog.imgix.net. (2019). Burndown-chart.

Jeffries, R. (2011). What is Extreme Programming? Ronjeffries.com. https://ronjeffries.com/xprog/what-is-extreme-programming/

Kruchten, P. (2003). The Rational Unified Process: An Introduction. Addison-Wesley Professional.

Maida, E, G., Pacienzia, J. (2015). Metodologías de desarrollo de software [en línea]. Tesis de Licenciatura en Sistemas y Computación. Facultad de Química e Ingeniería Fray Rogelio Bacon. Universidad Católica Argentina. https://repositorio.uca.edu.ar/bitstream/123456789/522/1/metodologias-desarrollo-software.pdf

Man, M., Hafriz, M., Nural, A., Mohd, H., Maizura, N., Noor, M., Wan, W., Bakar, A., & Man, M. (2008). eWorks: Development of a Web Based Site Assessment Software for Construction Progress Project. Communications of the IBIMA, (5), 93-99.

Martin, J. (1991). Rapid Application Development. Macmillan Coll. Pngwing.com (s.f.) Marco de trabajo Scrum. https://www.pngwing.com/es/free-png-xqgjv

Royce, W. W. (1970). Managing the Development of Large Software Systems. Proceedings of IEEE WESCON, 26, 328-388.

SCRUMstudy. (2013). A Guide to the Scrum Body of Knowledge (SBOK Guide). VMEdu Inc.

Sommerville, I., Galipienso, M. I. A., y Martínez, A. B. (2005). Ingeniería del software. Pearson Educación.

Créditos

Nombre	Cargo	Centro de Formación y Regional
Milady Tatiana Villamil Castellanos	Responsable del Ecosistema	Dirección General
Olga Constanza Bermúdez Jaimes	Responsable de Línea de Producción	Centro de Servicios de Salud - Regional Antioquia
Jonathan Guerrero Astaiza	Experto Temático	Centro de Teleinformática y Producción Industrial - Regional Cauca
Paola Alexandra Moya Peralta	Evaluadora Instruccional	Centro de Servicios de Salud - Regional Antioquia
Andrés Felipe Herrera Roldán	Diseñador de Contenidos Digitales	Centro de Servicios de Salud - Regional Antioquia
Edwin Sneider Velandia Suárez	Desarrollador Fullstack	Centro de Servicios de Salud - Regional Antioquia
Edgar Mauricio Cortés García	Actividad Didáctica	Centro de Servicios de Salud - Regional Antioquia
Daniela Muñoz Bedoya	Animador y Productor Multimedia	Centro de Servicios de Salud - Regional Antioquia
Luis Gabriel Urueta Álvarez	Validador de Recursos Educativos Digitales	Centro de Servicios de Salud - Regional Antioquia
Margarita Marcela Medrano Gómez	Evaluador para Contenidos Inclusivos y Accesibles	Centro de Servicios de Salud - Regional Antioquia
Daniel Ricardo Mutis Gómez	Evaluador para Contenidos Inclusivos y Accesibles	Centro de Servicios de Salud - Regional Antioquia