Capítulo 4

Especificación del software y enfoques ágiles

Mónica Elizabeth Páez Padilla
Diego Javier Portilla Martínez

Resumen

En este capítulo, se tratan asuntos fundamentales que abarcan desde la definición del software hasta la implementación de enfoques ágiles, con un enfoque particular en el marco de trabajo Scrum. Se destaca la relevancia de no confiar únicamente en metodologías y procesos para producir software de alta calidad, sino también en una precisa especificación y la aplicación de prácticas ágiles. El texto se enfoca en la etapa de especificación del software, incluyendo diseño, implementación, validación y evolución. También explora los enfoques ágiles en el desarrollo de software, detallando su idoneidad y el Manifiesto Ágil. Luego, se analizan los componentes clave de Scrum, incluyendo roles, responsabilidades, interacción, justificación de negocio y artefactos. Finalmente, se profundiza en los eventos esenciales de Scrum, incluyendo el Scrum Diario, la retrospectiva y el refinamiento del Product Backlog.
Palabras claves: Scrum, Product Backlog, Manifiesto Ágil, software.

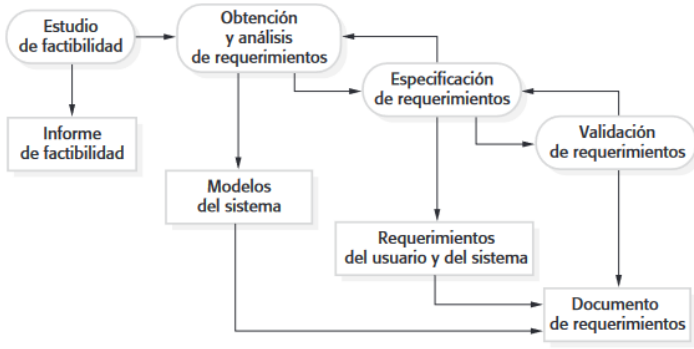
Paáz Padilla, M.E., y Portilla Martinez, D.J. (2023). Especificación del software y enfoques ágiles. En M.E. Páez Padilla (ed). *Análisis y diseño de sistemas.* (pp. 169-238). Religación Press. http://doi.org/10.46652/religacionpress.89.c85

4.1 Especificación del software

La elaboración de la especificación de software o la disciplina de ingeniería de requisitos comprende el procedimiento de comprender y definir los servicios necesarios del sistema, además de identificar las limitaciones en su funcionamiento y desarrollo. La ingeniería de requisitos es una fase especialmente crucial en el ciclo de desarrollo de software, ya que los fallos en esta etapa inevitablemente desencadenan problemas posteriores tanto en el diseño como en la implementación del sistema.

El proceso de ingeniería de requisitos (representado en la figura 11) se concentra en la elaboración de un documento de requisitos acordado que exprese las expectativas de los stakeholders del proyecto sobre lo que el sistema debería lograr. En general, estos requisitos se dividen en dos niveles de detalle. Los usuarios finales y los clientes solicitan un resumen de alto nivel de los requisitos, mientras que los desarrolladores del sistema requieren una descripción más detallada de los mismos (Sommerville, 2009).

Figura 11. Proceso de ingeniería de requerimientos.



Nota. Como se muestra en la figura anterior la especificación de requisitos de software consiste en una exposición exhaustiva del funcionamiento del sistema que será construido.

De acuerdo con (James, 2007) dentro del proceso de ingeniería de requisitos, se pueden identificar cuatro actividades principales:

Estudio de factibilidad: Se realiza una evaluación para determinar si las necesidades identificadas por el usuario son factibles de abordar con las tecnologías actuales de software y hardware. Esta evaluación considera si la propuesta del sistema conducirá a una relación costo-beneficio favorable desde una perspectiva empresarial y si su desarrollo puede ser gestionado dentro de las restricciones financieras existentes. Es esencial que esta evaluación de viabilidad sea rápida y económica. El resultado debe tener un impacto en la decisión de proceder o no a un análisis más detenido.

Obtención y análisis de requerimientos: Este procedimiento implica deducir los requisitos del sistema a través de la observación de sistemas ya en funcionamiento, conversaciones con usuarios y posibles proveedores, análisis de tareas y similares. Esto podría involucrar la creación de uno o varios modelos de sistemas y prototipos, con el propósito de aumentar la comprensión del sistema que se está por describir.

Especificación de requerimientos: Involucra la tarea de documentar la información recopilada durante la fase de análisis en un informe que define un conjunto de requerimientos. Este informe abarca dos categorías de necesidades: los requisitos del usuario, que ofrecen descripciones generales de los requisitos del sistema dirigidas al cliente y al usuario final del sistema; y los requisitos del sistema, que proporcionan una explicación detallada de las capacidades que se deben proporcionar.

Validación de requerimientos: Esta etapa se encarga de verificar que los requisitos sean lógicos, razonables y completos. En este proceso, es inevitable detectar errores en el documento de requisitos. Como resultado, es necesario ajustarlos para resolver dichos problemas.

Por supuesto, las fases en el proceso de requisitos no siguen una secuencia estrictamente lineal. El análisis de requisitos continúa durante la definición y especificación, y a lo largo del proceso, surgen nuevos requisitos; por lo tanto, las actividades de análisis, definición y especificación están interconectadas. En enfoques ágiles, como la programación extrema, los requisitos se desarrollan de manera incremental según las prioridades del

usuario, y la obtención de requisitos proviene de los usuarios que forman parte del equipo de desarrollo (Sommerville, 2011).

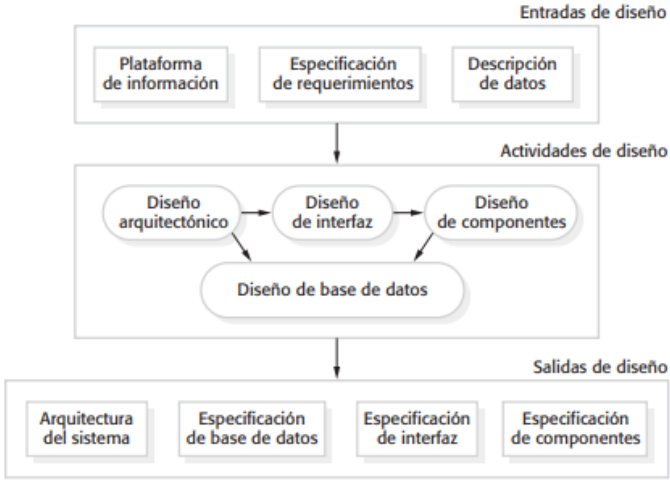
4.1.1 Diseño e implementación del software

La fase de desarrollo e implementación del software comprende el proceso de transformar una descripción del sistema en un sistema funcional. Siempre implica los procedimientos de diseño y programación del software y, en algunas ocasiones, puede requerir modificaciones en la descripción del software si se sigue un enfoque de desarrollo incremental (Sulbarán, 2018).

El diseño de software se define como una representación de la estructura del software a ser implementado, los modelos y estructuras de datos utilizados por el sistema, las conexiones entre los componentes del sistema y, ocasionalmente, los algoritmos empleados. Los diseñadores no llegan de inmediato a un diseño definitivo, sino que desarrollan el diseño de manera iterativa. Conforme avanzan en el diseño, añaden formalidad y detalles, al mismo tiempo que corrigen y ajustan diseños previos.

La figura 12 es un modelo general del proceso de diseño en la cual las etapas del proceso de diseño siguen un orden lineal, pero en la práctica, las actividades de diseño están estrechamente conectadas entre sí. En todos los procesos de diseño, es inevitable que se produzca retroalimentación entre las etapas, lo que da lugar a una revisión continua y a la mejora del diseño.

Figura 12. Modelo general del proceso de diseño.



Nota. La figura 12 representa un esquema conceptual de esta metodología, que muestra las aportaciones al proceso de diseño, las acciones llevadas a cabo durante el proceso y los documentos que se originan como resultados de dicho proceso.

La mayoría del software interactúa con otros sistemas de software, como sistemas operativos, bases de datos y middleware. Estos sistemas forman la "plataforma de software" en la que el software se ejecutará. La información acerca de esta plataforma es fundamental para el proceso de diseño, y los diseñadores deben determinar cómo integrarla con el entorno de software. La especificación de requerimientos describe la funcionalidad prevista del software, así como los requisitos de rendimiento y fiabilidad. Si el sistema debe trabajar con datos existentes, entonces la descripción de esos datos se incluirá en la especificación de la plataforma, de lo contrario, la organización de los datos se convertirá en un elemento de entrada para el proceso de diseño.

Las actividades en el proceso de diseño son diferentes según el tipo de sistema que se esté desarrollando. Por ejemplo, los sistemas en tiempo real necesitarán un diseño relacionado con la gestión del tiempo, mientras que los sistemas que no involucren bases de datos no requerirán un diseño de base de datos (Senn, 1997).

En la figura anterior, se representan cuatro tareas que podrían formar parte del proceso de diseño para sistemas de información:

En el libro Arquitectura de Sistemas de Información (Aguilar, 2005), afirma que:

Diseño arquitectónico: Se determina la configuración general del sistema, los elementos primordiales, sus conexiones y distribución.

Diseño de interfaces: Se establecen las interfaces entre los elementos del sistema. Estas descripciones de interfaz deben ser detalladas para que un componente pueda operar sin que otros requieran información sobre cómo está construido.

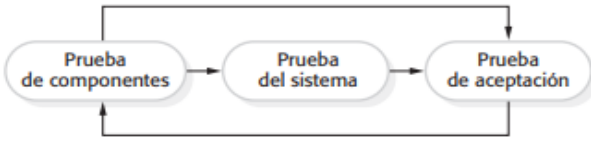
Diseño de componentes: Se enfoca en cada uno de los elementos del sistema y se elabora su funcionamiento. Esto puede implicar la creación de una descripción funcional o la lista de modificaciones en un componente reutilizado.

Diseño de base de datos: Se planifican las estructuras de datos del sistema y cómo se guardarán en una base de datos.

Estas acciones resultan en diversos productos de diseño, que también se ilustran en la figura. La forma y el nivel de detalle de estos productos pueden variar, desde documentos minuciosos hasta diagramas. En metodologías ágiles, estos productos pueden ser directamente integrados en el código del programa.

Los métodos de diseño estructurado se desarrollaron en las décadas de 1970 y 1980, lo que condujo al surgimiento del UML y al enfoque de diseño orientado a objetos. Estos métodos implican la creación de modelos gráficos y, en muchos casos, la generación de código a partir de estos modelos. El Desarrollo Basado en Modelos (MDD) representa una evolución de estos métodos, donde se generan modelos de software en varios niveles de abstracción. El diseño de programas para implementar el sistema se deriva de los procesos de desarrollo del sistema. Las herramientas de desarrollo de software pueden crear una estructura básica del programa a partir del diseño, y luego los programadores completan los detalles (Sommerville, 2011).

Figura 13. Etapas de Pruebas.



Nota. La depuración implica establecer hipótesis sobre el comportamiento del programa y ponerlas a prueba para identificar y corregir errores. Herramientas interactivas son útiles para depurar, mostrando valores de variables y rastreo de instrucciones ejecutadas.

La programación es una tarea individual y no sigue un proceso uniforme. Algunos programadores empiezan trabajando en componentes que conocen bien y luego avanzan hacia los menos familiares, mientras que otros siguen un enfoque contrario. A medida que desarrollan el código, las pruebas revelan fallos que deben ser corregidos, en un proceso conocido como depuración.

4.1.2 Validación de Software

La validación de software, o en un sentido más amplio, su verificación y validación (V&V), se lleva a cabo para demostrar que un sistema cumple con sus especificaciones y las expectativas del cliente. La principal técnica empleada en la validación es realizar pruebas del programa, donde el sistema se ejecuta utilizando datos de prueba simulados. Además de las pruebas, la validación también puede involucrar actividades de verificación, como inspecciones y revisiones en cada etapa del proceso de desarrollo de software, desde la definición de requisitos por parte del usuario hasta el desarrollo del programa. Debido a que las pruebas son predominantes, la mayoría de los costos asociados con la validación se incurren durante la implementación y en las fases posteriores (Lewis, 2011).

A menos que se trate de programas pequeños, no se debe realizar pruebas en los sistemas como una entidad única. En la figura 13 se representa un proceso de pruebas en tres etapas, donde en primer lugar se prueban los componentes individuales del sistema, luego se procede con las pruebas del sistema en su

conjunto y, finalmente, se realizan pruebas con los datos reales del cliente. Idealmente, los defectos en los componentes se descubren tempranamente en el proceso, mientras que los problemas de interconexión se identifican al integrar el sistema. Sin embargo, a medida que se encuentran defectos, es necesario depurar el programa, lo que podría requerir repetir otras fases del proceso de pruebas. Los errores en los componentes individuales pueden manifestarse durante las pruebas del sistema. Por lo tanto, este proceso es iterativo, con información que fluye desde etapas posteriores hacia las fases iniciales del proceso.

En palabras de Massol y Husted (2003), las etapas en el proceso de pruebas incluyen:

Prueba de desarrollo: Los desarrolladores del sistema realizan pruebas en los elementos que componen el sistema. Cada elemento se somete a pruebas de forma individual, sin que otros elementos del sistema estén presentes. Estos elementos pueden ser unidades simples, como funciones o clases de objetos, o grupos coherentes de estas entidades. Generalmente, se emplean herramientas de automatización de pruebas, como JUnit que facilitan la repetición de las pruebas de los componentes cuando se introducen nuevas versiones de estos.

Pruebas del sistema: Los elementos del sistema se agrupan para constituir el sistema en su totalidad. Esta acción tiene como finalidad detectar posibles errores que puedan surgir de interacciones no anticipadas entre los componentes, así como problemas de conectividad entre los mismos. Además, se busca verificar que el sistema cumple con sus requisitos tanto funcionales como no

funcionales, y se evalúan las propiedades emergentes del sistema. En sistemas de gran envergadura, este proceso puede constar de múltiples etapas, en las cuales los componentes se organizan en subsistemas que son evaluados de manera individual antes de ser integrados en el sistema definitivo.

Pruebas de aceptación: Esta etapa representa el último paso en el proceso de pruebas antes de que el sistema se considere apto para su uso operativo. En esta fase, el sistema se somete a pruebas utilizando datos proporcionados directamente por el cliente del sistema, en lugar de emplear datos simulados para las pruebas. Las pruebas de aceptación tienen la capacidad de poner de manifiesto errores y omisiones en la definición de los requisitos del sistema, ya que los datos reales ponen a prueba el sistema de manera diferente en comparación con los datos de prueba simulados. Además, estas pruebas permiten identificar problemas relacionados con los requisitos, en situaciones en las que las capacidades del sistema no satisfacen eficazmente las necesidades del usuario o cuando el rendimiento del sistema resulta insuficiente.

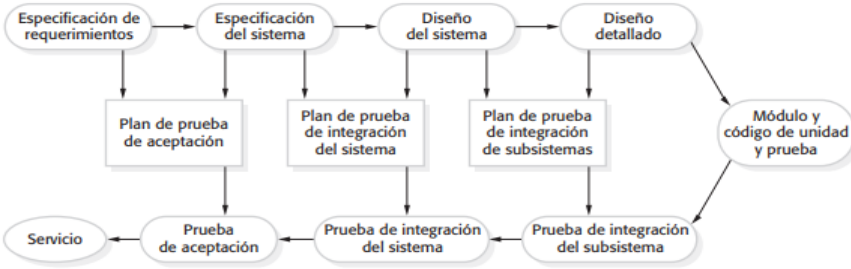
En general, los procesos de desarrollo y pruebas de componentes están estrechamente relacionados. Los programadores crean sus propios conjuntos de datos de prueba y evalúan el código incrementalmente a medida que lo desarrollan. Este enfoque tiene sentido desde un punto de vista económico, ya que el programador tiene un conocimiento profundo del componente y es el más adecuado para crear casos de prueba.

En un enfoque de desarrollo incremental, cada fase adicional debe someterse a pruebas a medida que se diseña, y estas

pruebas se basan en los requisitos específicos de esa fase. En la programación extrema, las pruebas se elaboran juntamente con los requisitos antes de iniciar el desarrollo. Este enfoque ayuda a los revisores y a los desarrolladores a comprender los requisitos y garantiza que no haya demoras en la creación de casos de prueba.

En situaciones en las que se utiliza un proceso de desarrollo planificado, como en el desarrollo de sistemas críticos, las pruebas se llevan a cabo mediante un conjunto de planes de prueba predefinidos. Un equipo de revisores independientes trabaja de acuerdo con estos planes, que se generan a partir de las especificaciones y el diseño del sistema (Aguilar, 2005).

Figura 14. Probando fases en un proceso de software dirigido por un plan.



Nota. La figura anterior ilustra la relación entre los planes de prueba en las etapas de desarrollo y pruebas, a menudo conocida como el "modelo V" de desarrollo (puede girarse para formar una "V" y así distinguirlo).

A veces, las pruebas de aceptación se denominan "pruebas alfa". Los sistemas personalizados se desarrollan exclusivamen-

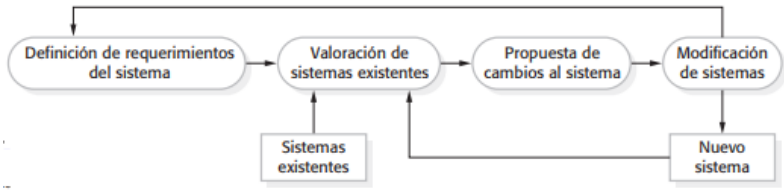
te para un cliente. El proceso de prueba alfa continúa hasta que tanto el desarrollador como el cliente están de acuerdo en que el sistema entregado cumple satisfactoriamente con los requisitos.

Una vez que un sistema se considera un producto de software, a menudo se recurre a un proceso de prueba conocido como "prueba beta". Esto implica ofrecer el sistema a algunos usuarios potenciales que están dispuestos a probarlo. Estos usuarios proporcionan comentarios sobre cualquier problema a los desarrolladores. Este proceso de retroalimentación somete el producto a un uso real y permite detectar errores que los constructores del sistema no anticiparon. Después de esta fase de retroalimentación, el sistema se ajusta y se lanza nuevamente, ya sea para más pruebas beta o para su lanzamiento al público en general.

4.1.3 Evolución de Software

La flexibilidad inherente a los sistemas de software es una de las principales razones por las cuales el software se está incorporando cada vez más en sistemas extensos y complejos. Una vez que se toma la decisión de fabricar hardware, efectuar modificaciones en su diseño conlleva un costo significativamente alto. Sin embargo, en cualquier momento durante o después del proceso de desarrollo del sistema, es posible realizar cambios en el software. Incluso las modificaciones sustanciales resultan más económicas que las equivalentes en el hardware del sistema. A lo largo de la historia, ha existido una clara separación entre el proceso de desarrollo de software y el proceso de evolución del software, que se refiere al mantenimiento de este.

Figura 15. Evolución del sistema.



Nota. La figura anterior muestra el proceso evolutivo, en el cual el software se modifica de manera constante a lo largo de su ciclo de vida, de acuerdo con los cambiantes requerimientos y necesidades del cliente.

El desarrollo de software se considera una actividad creativa en la que se crea un sistema de software a partir de una idea inicial y mediante un proceso de trabajo. No obstante, a veces, el mantenimiento del software se percibe como tedioso y carente de interés. A pesar de que, en la mayoría de los casos, los costos de mantenimiento superan con creces los costos iniciales de desarrollo, algunas veces los procesos de mantenimiento se consideran menos desafiantes que la creación original del software. La distinción tradicional entre desarrollo y mantenimiento está perdiendo relevancia. Es altamente improbable que cualquier sistema de software sea completamente innovador, por lo que es más apropiado ver el desarrollo y el mantenimiento como un continuo. En lugar de considerarlos como procesos independientes, tiene más sentido concebir la ingeniería de software como un proceso integrado (James, 2007).

Autoevaluación 13

1. **¿Cuál es el propósito de la actividad de «Estudio de factibilidad» en la ingeniería de requisitos?**
    a. Definir los requisitos del usuario.
    b. Documentar los requisitos del sistema.
    c. Evaluar si las necesidades son factibles y rentables.
    d. Verificar que los requisitos sean lógicos.
2. **¿Qué implica la actividad de «Obtención y análisis de requerimientos» en la ingeniería de requisitos?**
    1. Documentar los requisitos del sistema.
    2. Verificar que los requisitos sean lógicos.
    3. Deducir los requisitos del sistema a través de la observación y análisis.
    4. Evaluar la viabilidad financiera del proyecto.
3. **¿Qué tipo de necesidades abarca la actividad de «Especificación de requerimientos» en la ingeniería de requisitos?**
    1. Requisitos de usuario y requisitos de hardware.
    2. Requisitos de software y requisitos de seguridad.
    3. Requisitos del usuario y requisitos del sistema.
    4. Requisitos de rendimiento y requisitos de mantenimiento.
4. **¿Cuál es el propósito de la actividad de «Validación de requerimientos» en la ingeniería de requisitos?**
    a. Evaluar la viabilidad del proyecto.
    b. Documentar los requisitos del sistema.
    c. Verificar que los requisitos sean lógicos y razonables.

d. Identificar las necesidades de hardware.

5. **¿Cómo se relacionan las actividades de análisis, definición y especificación de requisitos en el proceso de requisitos?**
    1. Son actividades completamente independientes.
    2. Siguen una secuencia estrictamente lineal.
    3. Están interconectadas, y el análisis de requisitos continúa durante la definición y especificación.
    4. Se llevan a cabo en etapas separadas del proyecto.
6. **¿Cuál es el propósito del diseño de software en el proceso de desarrollo?**
    a. Documentar los problemas en el sistema.
    b. Transformar una descripción del sistema en un sistema funcional.
    c. Evaluar la viabilidad financiera del proyecto.
    d. Generar código directamente desde la descripción del sistema.
7. **¿Qué tipo de sistemas pueden requerir un diseño relacionado con la gestión del tiempo en el proceso de diseño?**
    1. Sistemas de información.
    2. Sistemas en tiempo real.
    3. Sistemas de base de datos.
    4. Sistemas de desarrollo incremental.
8. **¿Qué tarea del proceso de diseño se enfoca en establecer las interfaces entre los elementos del sistema?**
    a. Diseño arquitectónico.
    b. Diseño de componentes.

c. Diseño de interfaces.
d. Diseño de base de datos.

9. ¿Qué método de diseño implica la creación de modelos gráficos y, en algunos casos, la generación de código a partir de estos modelos?

a. Diseño estructurado.
b. Desarrollo Basado en Modelos (MDD).
c. Desarrollo orientado a objetos.
d. Desarrollo incremental.

10. ¿Qué proceso se lleva a cabo cuando los programadores corrigen los fallos que se revelan durante las pruebas del código?

1. Depuración.
2. Diseño de componentes.
3. Diseño de base de datos.
4. Evaluación de la viabilidad.

11. ¿Cuál es una de las principales ventajas del software en comparación con el hardware en términos de flexibilidad?

1. El software es más económico de producir.
2. El software no requiere mantenimiento.
3. Las modificaciones en el software son más económicas que en el hardware.
4. El software es más rápido en términos de procesamiento.

12. ¿Qué se entiende por evolución del software en términos de ingeniería de software?

a. La creación inicial de un sistema de software.
b. La fase de desarrollo de un sistema de software.
c. El proceso de mantenimiento y actualización del software.
d. La retirada de un sistema de software obsoleto.

13. ¿Cuál es la principal diferencia entre el desarrollo de software

y el mantenimiento del software?

1. El desarrollo de software es más económico que el mantenimiento.
2. El mantenimiento del software se considera menos desafiante.
3. El desarrollo de software es una actividad creativa, mientras que el mantenimiento es tedioso.
4. No hay diferencias significativas entre el desarrollo y el mantenimiento.

14. ¿Por qué la distinción tradicional entre desarrollo y mantenimiento del software está perdiendo relevancia?

1. Porque el mantenimiento del software se ha vuelto más caro que el desarrollo.
2. Porque los procesos de desarrollo y mantenimiento son independientes.
3. Porque la mayoría de los sistemas de software son completamente innovadores.
4. Porque se considera más apropiado ver la ingeniería de software como un proceso integrado.

15. ¿Cuál es una razón importante para considerar la evolución del software como un proceso continuo?

a. Para reducir los costos de mantenimiento.
b. Para evitar la creación de software completamente innovador.
c. Porque el mantenimiento del software es más económico que el desarrollo.
d. Para que el desarrollo y el mantenimiento estén mejor separados.

16. ¿Por qué la flexibilidad inherente al software es una ventaja en sistemas extensos y complejos?

a. Porque el software es más barato que el hardware.
b. Porque el software no requiere mantenimiento.
c. Porque las modificaciones en el software son más económicas que en el hardware.
d. Porque el software es más rápido que el hardware.

17. ¿Qué se entiende por evolución del software en términos de ingeniería de software?

1. La creación inicial de un sistema de software.
2. La fase de desarrollo de un sistema de software.
3. El proceso de mantenimiento y actualización del software.
4. La retirada de un sistema de software obsoleto.

18. ¿Cuál es una de las razones por las que a veces el mantenimiento del software se percibe como tedioso?

1. Porque es más económico que el desarrollo original del software.
2. Porque los costos de mantenimiento son significativamente más bajos que los del desarrollo.
3. Porque no es necesario realizar cambios en el software con el tiempo.
4. Porque se considera menos desafiante que la creación original del software.

19. ¿Por qué la distinción tradicional entre desarrollo y mantenimiento del software está perdiendo relevancia?

a. Porque el mantenimiento del software se ha vuelto más caro que el desarrollo.
b. Porque los procesos de desarrollo y mantenimiento son independientes.

c. Porque la mayoría de los sistemas de software son completamente innovadores.
d. Porque se considera más apropiado ver la ingeniería de software como un proceso integrado.

20. ¿Qué ventaja principal ofrece la flexibilidad del software en comparación con el hardware?

1. El software es más económico de producir.
2. El software permite realizar cambios más económicos.
3. El software no requiere mantenimiento.
4. El software se crea de manera más rápida.

4.2 Introducción a los enfoques ágiles

La estrategia ágil en el ámbito del desarrollo de software tiene como objetivo principal la entrega continua de sistemas de software operativos mediante iteraciones rápidas.

No obstante, la expresión "metodología ágil" puede ser engañosa, ya que sugiere que el enfoque ágil es la única manera de encarar el desarrollo de software. En realidad, la metodología ágil no proporciona instrucciones precisas sobre cómo llevar a cabo el desarrollo de software. Más bien, representa una mentalidad orientada a la colaboración y a los flujos de trabajo, y establece un conjunto de principios que dirigen nuestras decisiones en lo que respecta a qué hacer y cómo hacerlo.

Específicamente, las metodologías ágiles de desarrollo de software se centran en proporcionar fragmentos funcionales de software en un corto período de tiempo para mejorar la satisfacción del cliente. Estas metodologías adoptan enfoques flexibles y fomentan la colaboración en equipo para lograr mejoras continuas. En resumen, el desarrollo ágil de software implica que equipos pequeños y autónomos de desarrolladores y representantes de negocios se reúnan regularmente en persona a lo largo del ciclo de vida del desarrollo de software. La metodología ágil aboga por una aproximación simplificada a la documentación de software y acepta los cambios que puedan surgir en diversas etapas del ciclo de vida en lugar de resistirse a ellos (Hat, 2023).

4.2.1 ¿Cuándo y por qué enfoques ágiles?

El sentido común, que rara vez es común, sobre todo entre profesionales que han invertido años en trabajar con sistemas, estructuras y protocolos predefinidos tras finalizar sus estudios, se convierte en un componente esencial dentro del nuevo enfoque ágil para gestionar proyectos. Aunque se etiqueta como "metodología ágil", esta perspectiva no impone un conjunto rígido de reglas para la ejecución de proyectos. En lugar de ello, representa un modo de pensar acerca de la colaboración y los flujos de trabajo, estableciendo un conjunto de valores que guían las decisiones sobre qué hacer y cómo llevarlo a cabo.

Las metodologías ágiles para el desarrollo de software buscan entregar segmentos operativos de sistemas en un tiempo breve, otorgando prioridad a la satisfacción del cliente. Estas metodologías adoptan enfoques adaptables y trabajo en equipo para lograr mejoras en curso. Se centran en equipos autoorganizados que se reúnen con regularidad a lo largo del ciclo de vida del desarrollo. La metodología ágil favorece la simplificación de la documentación y la aceptación de cambios en lugar de resistirlos.

Las metodologías ágiles fomentan tres objetivos esenciales en los proyectos: proporcionar valor, minimizar desperdicios y mejorar continuamente. Aunque estos principios parezcan evidentes, a menudo son pasados por alto en la gestión convencional de proyectos. También se vinculan con el ciclo de Deming (Planificar, Hacer, Verificar y Actuar) y promueven una mentalidad de mejora continua.

El Triángulo de Hierro, que engloba Costo, Tiempo, Alcance y Calidad, juega un papel crucial en las metodologías ágiles. Aquí, los aspectos de Costo y Tiempo se mantienen constantes, permitiendo que el Alcance sea el elemento variable para lograr resultados de alta calidad.

Se menciona el "Cono de Incertidumbre", que apunta a que la incertidumbre disminuye conforme avanza el proyecto. Al combinar estos conceptos, como si estuvieran mezclando ingredientes en un recipiente, se enfatiza la importancia de tener objetivos claros, un enfoque iterativo e incremental y la capacidad de adaptarse en la gestión de proyectos.

En síntesis, el enfoque ágil aporta claridad a conceptos que a menudo son pasados por alto, promoviendo una mentalidad colaborativa, adaptable y orientada a la mejora constante en la gestión de proyectos (Dremyn, 2020).

4.2.2 Manifiesto y Principios Agiles

Después de examinar los valores propuestos por las metodologías ágiles, exploraremos los 12 principios en los que se basan (Sentrio, 2021).

1. Satisfacer al cliente mediante la entrega temprana y continua

El primer principio del Manifiesto Ágil se enfoca en la satisfacción del cliente y destaca cómo una entrega temprana y constante de software valioso es esencial para lograrla. Esto aumenta

las probabilidades de cumplir con las demandas de los clientes y acelera el retorno de la inversión.

En un contexto en constante cambio, retrasar la entrega de software resulta insatisfactorio para los clientes. Dado que los usuarios utilizan el software en una amplia variedad de actividades y buscan cambios inmediatos, la demora no es una opción viable. Por lo tanto, el equipo de desarrollo debe realizar entregas más ágiles que agreguen valor a los usuarios.

Además, las entregas frecuentes y rápidas permiten que los clientes reciban valor en menor tiempo y con mayor frecuencia. Esto también facilita la retroalimentación temprana por parte del cliente, lo que disminuye la posibilidad de tener que realizar cambios significativos en etapas posteriores.

2. Aprovechar el cambio como ventaja competitiva

El segundo principio de la declaración promueve la adaptación a los cambios en beneficio del cliente, permitiendo ajustes en los requisitos incluso en las etapas finales de un proyecto.

En las prácticas tradicionales de gestión de proyectos, introducir cambios en los requisitos al final del proceso solía implicar un aumento del alcance y, por ende, un incremento de los costos. Sin embargo, en los enfoques ágiles, los equipos reconocen la potencial utilidad de estos cambios para los clientes y responden de manera eficaz a ellos.

Dada la naturaleza en constante evolución del entorno, resulta sumamente desafiante prever con exactitud los requisitos definitivos de un software. Destinar recursos a un producto que, al ser entregado a los usuarios, ya no resulta relevante para ellos carece de sentido para una empresa. Por lo tanto, aceptar y abrazar los cambios proporcionará al cliente una ventaja competitiva al abordar las necesidades actuales de sus usuarios.

3. Entregar valor frecuentemente

El tercer principio del Manifiesto Ágil profundiza en la noción de entrega continúa presentada en el primer principio fundamental. Específicamente, se refiere a la importancia de proporcionar actualizaciones más pequeñas del software en intervalos más cortos.

Estas entregas de menor escala requieren un menor tiempo de planificación y disminuyen la probabilidad de errores en su desarrollo. Además, un aumento en la frecuencia de entregas conlleva una mayor retroalimentación por parte del cliente, lo que previene la necesidad de solicitar cambios significativos a los desarrolladores en etapas posteriores.

Este principio no solo sigue siendo relevante en la actualidad, sino que ha ganado más importancia en los últimos años. Las versiones se liberan semanalmente o incluso diariamente.

4. Cooperación negocio-desarrolladores durante todo el proyecto

Este principio aboga por abolir las divisiones que existen entre los equipos de negocio y desarrollo de software, con el propósito de mejorar la comprensión y la colaboración recíproca, en aras de lograr resultados más efectivos.

Históricamente, los responsables de negocios y los desarrolladores operaban en esferas separadas. Otros roles intervenían para traducir las ideas de los primeros a un lenguaje comprensible por los segundos. Con este principio, los promotores del Manifiesto Ágil buscan fomentar la interacción diaria entre ambos grupos, abordando cualquier malentendido previamente, compartiendo retroalimentación de manera mutua y, en última instancia, alineando sus intereses.

5. Construir proyectos en torno a individuos motivados

El quinto principio de Agile aboga por el fortalecimiento de la motivación entre los miembros del equipo de desarrollo, para que puedan ejecutar los proyectos de manera óptima.

La segunda parte de este enunciado se enfoca en un aspecto crucial: la confianza en el equipo para generar software de alta calidad de manera autónoma, contando con el entorno y el apoyo adecuados. Si los integrantes del equipo no participan en las decisiones de los proyectos en los que trabajan, no podrán conectar con el propósito de estos, lo que resultará en una disminución de su compromiso y rendimiento.

6. Utilizar la comunicación cara a cara

El sexto principio del manifiesto aborda la comunicación óptima para lograr proyectos exitosos: el diálogo en persona. Entre todas las modalidades de comunicación disponibles, el cara a cara es la más eficaz, ya que disminuye considerablemente los tiempos de respuesta y las posibilidades de malentendidos.

Sin embargo, la naturaleza de nuestro entorno laboral ha cambiado significativamente desde 2001. El trabajo a distancia se ha generalizado y han surgido numerosas herramientas que facilitan la comunicación y la colaboración a distancia. Esto no significa que este principio haya perdido relevancia en la actualidad, sino que ha evolucionado.

7. Software funcionando como medida de progreso

El séptimo principio ágil hace hincapié en que la medida fundamental para evaluar el progreso de un proyecto en las organizaciones debe ser la presencia de software en funcionamiento. Las actividades realizadas por el equipo de desarrollo que no contribuyen a la creación de un producto que responda a las demandas del cliente apenas o nada aportan en términos del verdadero avance del proyecto.

No importa cuántos errores el equipo haya corregido o cuánto código haya escrito si no se ha trabajado en pos de obtener un software que cumpla con las necesidades del cliente. En la actualidad, este principio tiene un alcance más amplio. Incluso si el

software funciona correctamente, si no se ha entregado al cliente, el equipo de desarrollo no ha logrado avanzar. No ha generado valor para el cliente hasta que el producto final esté en sus manos.

8. Promover y mantener un desarrollo sostenible

La octava premisa del manifiesto sostiene que en la operativa ágil se busca optimizar los métodos de trabajo para evitar excesos y lograr entregar con frecuencia al mercado soluciones de software que generen valor para los usuarios, sin necesidad de exponerse a esfuerzos desmesurados. En otras palabras, todas las partes involucradas en el proceso de desarrollo de software deben mantener un ritmo que sea viable para todos, evitando tensiones o presiones exageradas.

Por lo general, este principio se relaciona con la aplicación a equipos de desarrollo que están sobrecargados al proporcionar nuevas funcionalidades a los usuarios. Sin embargo, también puede aplicarse en sentido contrario. Si el equipo de desarrollo está liberando un número excesivo de nuevas funcionalidades para los usuarios, se deberá moderar la producción o invertir esfuerzos en capacitar a los usuarios en la utilización de estas características.

9. La excelencia técnica mejora la agilidad

El noveno enunciado del manifiesto enfatiza que otorgar importancia a los aspectos técnicos durante el proceso de desarrollo de un producto de software contribuye a la agilidad. Resulta más

factible realizar actualizaciones posteriores en el software si se ha construido de manera meticulosa y cuenta con un diseño sólido, en comparación con una construcción descuidada.

Con frecuencia, las organizaciones priorizan el time-to-market de sus productos por encima de la calidad del código. Esta perspectiva es comprensible, ya que la excelencia técnica no impacta directamente a los usuarios finales ni genera beneficios tangibles para el negocio. Sin embargo, si se relega, acabará afectando la velocidad, los plazos de entrega y la capacidad de mejorar el producto ante nuevas necesidades. En última instancia, se sacrificará la agilidad.

Un indicador estrechamente relacionado con la falta de priorización de un código de alta calidad es la deuda técnica. En este artículo, explicamos en qué consiste y por qué es importante considerarla en tus proyectos**.**

10. La simplicidad es fundamental

El décimo principio del Manifiesto Ágil presenta un enfoque altamente pragmático: adoptar la forma más simple de abordar una situación. El cliente no recompensa por el esfuerzo invertido, sino por la entrega de una solución que satisfaga sus necesidades.

Algunas estrategias para evitar esfuerzos superfluos incluyen automatizar tareas manuales, eliminar procedimientos redundantes y hacer uso de bibliotecas preexistentes. En última instancia, se trata de dirigir el tiempo y la energía del equipo hacia acciones que verdaderamente generen valor.

11. Equipos auto-organizados para generar más valor

La undécima premisa establece que los equipos que disfrutan de la libertad y confianza adecuadas son los que logran los resultados más sobresalientes.

Este principio guarda una estrecha relación con algunos de los conceptos presentados anteriormente en el manifiesto. Para lograr una comunicación y colaboración efectivas entre el negocio y los desarrolladores, para emplear el software en funcionamiento como medida de avance y para cimentar nuestros proyectos en torno a individuos motivados, resulta fundamental permitir que los equipos de desarrollo operen sin un control excesivo y se organicen internamente en todos los aspectos del proceso de desarrollo de software.

12. Reflexión y ajustes frecuentes del trabajo de los equipos

El último principio de Agile aborda la noción de mejora constante. Los equipos deben llevar a cabo revisiones periódicas de su trabajo con el propósito de ajustarlo y elevar su eficacia. Se trata de un concepto esencial que motiva a individuos, equipos y organizaciones a aspirar al éxito mediante la búsqueda continua de mejoras en lugar de conformarse.

4.2.3 Valores y Pilares de Scrum

Valores de la metodología Scrum

No podrás adquirir una comprensión completa de Scrum sin asimilar los cinco valores fundamentales de Scrum, los cuales están definidos en la Guía de Scrum: (Martins, 2023).

1.- Compromiso: El equipo Scrum opera como una entidad unificada, y la confianza entre sus miembros es esencial. Los integrantes del equipo Scrum se comprometen con el sprint durante su duración y se dedican a una mejora constante con el propósito de encontrar la solución óptima.

2.-Valentía: En el transcurso de un sprint Scrum, es posible que el equipo se enfrente a retos complejos que no tengan respuestas precisas. Los equipos Scrum poseen la valentía para plantear preguntas desafiantes y abiertas, y para responder con sinceridad con el objetivo de alcanzar la solución más adecuada.

3.-Enfoque: En cada sprint de Scrum, el equipo Scrum ejecutará tareas seleccionadas de una lista de pendientes del producto. La atención del equipo Scrum se concentra en las labores elegidas de dicha lista, las cuales resultarán en la entrega de sus productos al final de cada sprint.

4.-Actitud receptiva: No todo marchará sin contratiempos en el contexto de Scrum. Los miembros del equipo Scrum deben estar dispuestos a acoger nuevas ideas y oportunidades que contribuyan a su aprendizaje individual y a la mejora del producto o proceso.

5.-Respeto: La colaboración es un elemento esencial para comprender la esencia de Scrum, y para fomentar una colaboración sólida en el equipo, los integrantes deben tratarse entre sí con respeto, además de mostrar consideración hacia el Scrum Master y el proceso Scrum en sí.

Pilares de Scrum

La estructura de la metodología Scrum, como discutimos previamente, se cimienta en tres fundamentos: los eventos, los roles y los artefactos. Además, esta metodología opera en ciclos de trabajo llamados "sprints", los cuales tienen una duración predefinida que suele oscilar entre una semana y un mes. ¿Y por qué se establece un límite de un mes para la duración de los sprints? Esto se debe a que, al extender este período, aumenta el riesgo de cambios tanto en los requisitos del desarrollo como en el contexto en el que se desenvuelve. Al finalizar cada sprint, se genera un incremento que se suma al producto desarrollado hasta ese punto.

Cuando comprendes la esencia de Scrum, te das cuenta de que al iniciar un sprint es posible que tengas un conocimiento limitado, pero tienes la flexibilidad de adaptar tus procedimientos y necesidades en función de la información que vayas obteniendo durante el proceso del sprint (Martins, 2023).

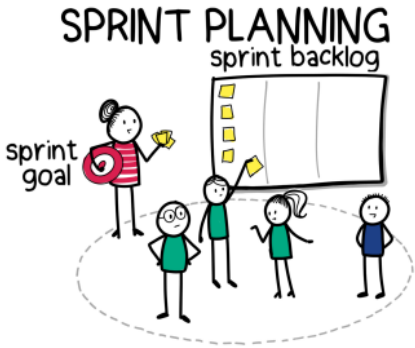
Eventos de Scrum

Entonces, ¿qué implica Scrum en realidad? ¿Cómo se desenvolverá tu equipo al adoptar Scrum? Así es como se desarrolla el proceso de Scrum:

1.-Organiza tu trabajo pendiente: Al iniciar un sprint de Scrum, el líder del equipo (también conocido como Scrum Master) determinará qué trabajo seleccionar de la lista de tareas pendientes, es decir, las labores que requieren ejecución. Para garantizar un sprint de Scrum efectivo, es esencial que el trabajo pendiente relacionado con el producto esté meticulosamente documentado en un solo lugar. Considera la posibilidad de utilizar una herramienta de gestión de proyectos para centralizar toda esta información.

2.-Sprint Planning: Lleva a cabo una reunión de planificación del sprint. Antes de iniciar el sprint de Scrum, es vital definir en qué se enfocará el equipo. Durante esta sesión de planificación del sprint, se evaluará qué parte del trabajo pendiente será el foco principal durante este sprint de Scrum en particular. Para comenzar, puedes utilizar nuestra plantilla gratuita para la planificación de Sprint.

Figura 16. Sprint Planning.



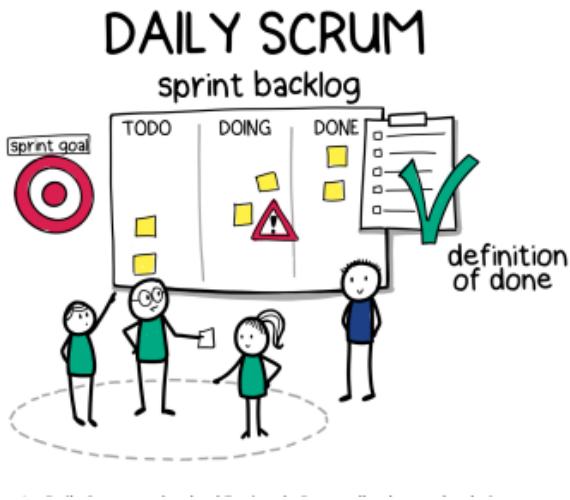
Nota. Durante el Sprint Planning se eligen los elementos requeridos del Product Backlog para alcanzar el Sprint Goal.

3.- Comienza tu sprint de Scrum: Por lo general, un sprint tiene una duración de dos semanas, aunque puedes optar por Sprint más cortos o largos según lo que resulte más adecuado para tu equipo. A lo largo del sprint, el equipo se dedicará a trabajar en las tareas pendientes establecidas durante la reunión de planificación del sprint.

4.- Daily Stand Up: Programa reuniones diarias de seguimiento de Scrum. La Reunión Diaria de Pie es un encuentro diario destinado al equipo de desarrollo, con una duración de 15 minutos. Estas reuniones diarias de seguimiento brindan la oportunidad de informar sobre las labores en progreso y abordar cualquier inconveniente inesperado que haya surgido. El objetivo primordial de estas reuniones es planificar el trabajo de las próximas horas y evaluar el progreso de las tareas. Si deseas optimizar

la efectividad de estas reuniones diarias de seguimiento, puedes utilizar nuestra plantilla gratuita diseñada para tal fin.

Figura 17. Daily Scrum.



Nota. La Daily Scrum es donde el Equipo de Desarrollo planea el trabajo colaborativo de las próximas 24 horas para lograr el Sprint Goal.

5.- Sprint Review: Exhibe tus logros durante la evaluación del sprint. Una vez que hayas completado el sprint de Scrum, tu equipo debe congregarse para llevar a cabo una evaluación del sprint. La revisión del sprint se extenderá durante un máximo de 4 horas en el caso de sprints de un mes. En este intervalo, tu equipo Scrum expondrá el trabajo que ha sido considerado como "Completado" para que los participantes lo aprueben o inspeccionen.

Figura 18. Sprint Review.

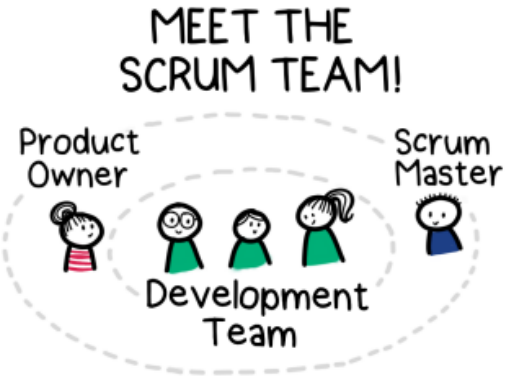
Nota. El propósito de la Sprint Review es inspeccionar el trabajo que se ha hecho hasta la fecha y decidir cuáles serán los siguientes pasos en base a lo que hemos aprendido

6. Sprint Retrospective: Dialoga y contempla durante la revisión retrospectiva del sprint. Al concluir cada sprint, dedica un momento para analizar cómo se desenvolvió y qué áreas podrían ser potenciadas en el futuro. Mantén en mente que en Scrum se abraza la idea de una mejora constante, por lo tanto, no dudes en experimentar con nuevos procedimientos o redefinir estrategias que consideras menos efectivas en el próximo sprint (James, 2007).

Roles de Scrum

Para comprender a fondo qué implica Scrum, es esencial tener un conocimiento sólido de los roles principales que la metodología Scrum distingue, ya que estos desempeñan un papel fundamental en la implementación efectiva de este enfoque ágil:

Figura 19. Roles de Scrum.



Nota. Tres roles para equilibrar tres perspectivas: valor, calidad y proceso.

a. **Product Owner o responsable del producto:** Esta es la figura responsable del backlog del producto en espera (product backlog). Está estrechamente relacionado con las necesidades de los usuarios y se dedica a transmitir la perspectiva del usuario tanto a su equipo como a otros líderes implicados. Los efectivos encargados de produc-

tos aportan claridad acerca de qué debe ser abordado a continuación debido a su relevancia. En última instancia, deberían ser los encargados de decidir cuándo algo está preparado para ser entregado (preferiblemente, con una tendencia hacia entregas frecuentes).

b. **Scrum Master:** El Scrum Master es aquel individuo encargado de liderar los diversos acontecimientos de Scrum. Puedes concebirlo como el administrador del proyecto y el facilitador de Scrum. El Scrum Master tiene la responsabilidad de impulsar las reuniones diarias de seguimiento y coordinar las sesiones de planificación, revisión y evaluación retrospectiva del sprint.

c. **Equipo Scrum:** El grupo Scrum abarca a todos aquellos involucrados en el sprint en curso. Los integrantes del equipo deben asumir la responsabilidad de su propia organización y trabajar en colaboración para alcanzar el objetivo esencial de Scrum, que es la búsqueda constante de mejoras.

Artefactos de Scrum

Los elementos fundamentales de Scrum son esenciales para la comprensión de esta metodología. En el contexto de Scrum, un artefacto representa una construcción que generas, como una herramienta destinada a solventar un desafío. Se identifican tres elementos cruciales que definen la esencia de Scrum: el backlog

de producto, el backlog de sprint y el incremento del producto (Sulbarán, 2018).

a. Product Backlog, el trabajo pendiente del producto

El Backlog de Producto constituye el componente de Scrum que recopila el inventario de tareas por realizar. Quien supervisa este backlog, también conocido como la pila de producto, es el propietario del producto, quien se encarga de jerarquizar los elementos en esta lista. Es crucial entender que la presencia de elementos en la lista de tareas pendientes no implica necesariamente que el equipo los abordará; en lugar de eso, los elementos en el Backlog de Producto representan opciones que el equipo podría considerar durante un sprint de Scrum. Los responsables del proyecto tienen la responsabilidad de ajustar y actualizar de forma constante el trabajo pendiente en función de nueva información y la lista de requisitos obtenida de los clientes, el mercado o el equipo del proyecto.

b. Sprint Backlog, el trabajo pendiente del sprint

El Backlog de Sprint, también denominado pila de sprint, consiste en el conjunto de tareas por abordar durante el sprint en cuestión. En otras palabras, representa la serie de labores o elementos a los cuales tu equipo se ha comprometido durante el sprint de Scrum. Estos elementos son extraídos de la lista de tareas pendientes del producto durante la reunión de planificación del sprint y se incorporan al plan de trabajo específico para el sprint de tu equipo, si este existe.

Si bien es posible que no todas las tareas pendientes se culminen durante cada sprint, es poco común que añadas nuevas tareas a la lista de trabajo pendiente una vez que el sprint ha comenzado. Si observas que esto ocurre con frecuencia, dedica más tiempo a la fase de planificación del sprint para tener una comprensión más sólida de qué labores abordarás durante el transcurso del sprint.

c. Incremento del producto

El producto incrementado es lo que proporcionarás al concluir cada sprint. Este componente de Scrum puede englobar desde un nuevo producto o función, hasta mejoras o correcciones de errores, según lo requiera tu equipo. La presentación del incremento está programada durante la evaluación del sprint. En ese instante, la entrega del incremento estará sujeta a la opinión de los participantes de Scrum, en función de si consideran que está "Completado" o no.

4.2.4 Marco Cinefyn – Tipos de Dominio

Una manera potencialmente más comprensible de ilustrar el contexto en el cual Scrum resulta más efectivo está relacionada con la perspectiva sobre la complejidad del Marco Cynefin o Modelo Cynefin, introducido en el año 2000 por Dave Snowden.

El Marco o Modelo Cynefin contrasta las atribuciones de cinco categorías distintas de complejidad: simple, complicado, complejo, caótico y el área central de desorden. Vamos a explorar cada uno de estos dominios (Labs, 2019).

1. Dominio Simple: Dentro del Marco Cynefin, el Ámbito Simple es donde se trata con situaciones de simplicidad. Dave Snowden caracteriza este dominio como aquel en el que es sencillo discernir las causas y sus consecuencias. En general, la respuesta correcta es evidente, ampliamente reconocida y no objeto de debate. En este dominio, prevalecen las mejores prácticas y soluciones establecidas para problemas conocidos. Los procesos más efectivos aquí son aquellos que establecen una secuencia lógica de pasos y se ejecutan de manera reiterada. Ejemplos de esta categoría incluyen la producción en serie de un mismo artículo o la instalación uniforme de un sistema en diversos clientes. Si bien Scrum podría ser aplicable en este contexto, los procedimientos que involucran pasos claramente definidos resultan considerablemente más eficientes.

2. Dominio Complicado: Otro de los ámbitos contemplados en el Marco Cynefin es el Ámbito Complicado. En esta categoría se abordan problemáticas de complejidad, destacan las mejores prácticas y la intervención de expertos. Aquí, existe una multiplicidad de soluciones correctas para un mismo problema, aunque se requiere la intervención de expertos para identificarlas. Un caso típico en este contexto podría ser la resolución de problemas de rendimiento en software o bases de datos, la sincronización de semáforos en una intersección de tres avenidas o la optimización de la distribución logística de productos. Aunque es posible aplicar Scrum, no necesariamente sería la opción más eficaz para abordar estas situaciones. Aquí, un enfoque que involucre a expertos en el campo, quienes analicen la situación, exploren varias alternativas y propongan soluciones basadas en las mejores prác-

ticas, suele ser más adecuado. Una práctica común en este ámbito es el mantenimiento de sistemas y el soporte técnico.

3. Dominio Complejo: Dentro del Modelo Cynefin de Dave Snowden encontramos el Ámbito Complejo. Nos hallamos en este dominio cuando nos enfrentamos a desafíos complejos, cuyos resultados se tornan más impredecibles. Para las situaciones que abarca este ámbito, no existen ni mejores ni buenas prácticas predefinidas.

Dentro del Ámbito Complejo, no hay ni mejores ni buenas prácticas establecidas. Simplemente, no se puede prever de antemano si una solución específica será efectiva. Solo es posible evaluar los resultados y ajustarse en consecuencia. Este es el terreno de las prácticas emergentes. Las soluciones que se descubren rara vez pueden aplicarse de manera idéntica a otros problemas similares, con resultados idénticos. Para operar en la complejidad, es necesario crear entornos que permitan la experimentación y minimicen el impacto del fracaso. Se requiere una alta dosis de creatividad, innovación, interacción y comunicación. El desarrollo de nuevos productos o la incorporación de funciones adicionales en productos existentes representa un ámbito complejo en el que Scrum se emplea con frecuencia para actuar, evaluar y ajustar las prácticas emergentes de un equipo de trabajo.

4.- Dominio Caótico: Dentro del Marco Cynefin, en el Ámbito Caótico nos encontramos con situaciones caóticas que exigen una respuesta inmediata. Nos hallamos en una crisis y es necesario tomar medidas inmediatas para restablecer algún grado de orden. Pongamos como ejemplo un escenario en el que el siste-

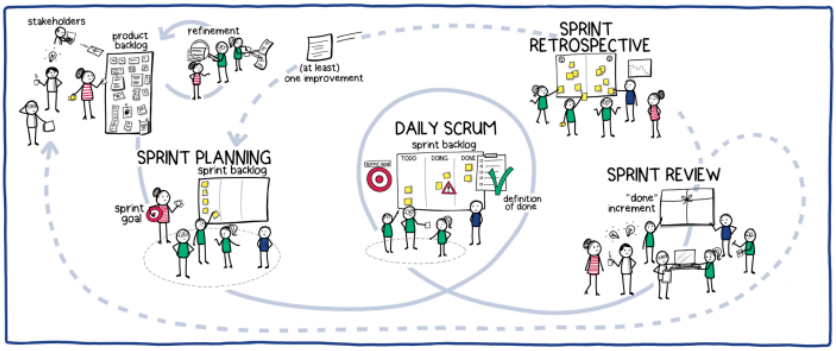
ma de programación de vuelos en un aeropuerto muy concurrido deja de funcionar. En este contexto, no sería apropiado aplicar Scrum; en su lugar, se requiere una acción inmediata, alguien debe asumir el control y sacar la situación del caos. Por ejemplo, resolver el problema de inmediato (sin importar el enfoque técnico), y luego, una vez fuera del caos, evaluar y aplicar una solución más sólida si es necesario. Este ámbito es donde la improvisación impera.

5. Dominio Desordenado: Según Dave Snowden, ingresamos al territorio del desorden cuando carecemos de claridad sobre en qué dominio nos encontramos. En el Modelo Cynefin, esta región se cataloga como un espacio arriesgado debido a la dificultad para evaluar las situaciones y establecer un enfoque de actuación. En tales circunstancias, es común que las personas interpreten las situaciones y tomen acciones basadas en preferencias personales. El mayor riesgo en el ámbito del desorden radica en actuar de manera incongruente con las necesidades para resolver problemas específicos. Un ejemplo de esto se observa en el campo del desarrollo de software, donde a menudo se emplea un enfoque secuencial y una planificación detallada basada en las mejores prácticas de la industria, lo cual es apropiado para situaciones simples. Sin embargo, cuando se aplica de manera incorrecta en situaciones complejas, como las que caen en el ámbito del desorden, puede resultar ineficaz. En estas circunstancias, la principal prioridad debería ser salir de la zona de desorden hacia un dominio más definido, y luego ajustar nuestras acciones de acuerdo con las demandas de ese nuevo dominio.

4.3. Elementos del marco de trabajo Scrum

Una cosa es expresar la necesidad de tener claridad en el trabajo que llevas a cabo para un producto, de revisar dicho trabajo con regularidad y de basar las modificaciones en los hallazgos resultantes de esta revisión. Otra cuestión es implementar todos estos aspectos de manera tangible y concreta. Esto es precisamente lo que convierte a Scrum en un marco de trabajo; presenta cinco eventos que se repiten para trabajar en tres artefactos, establece tres roles para brindar respaldo y abarca diversos principios y reglas que amalgaman todo en un conjunto coherente y cohesionado.

Figura 20. Marco de Trabajo Scrum.



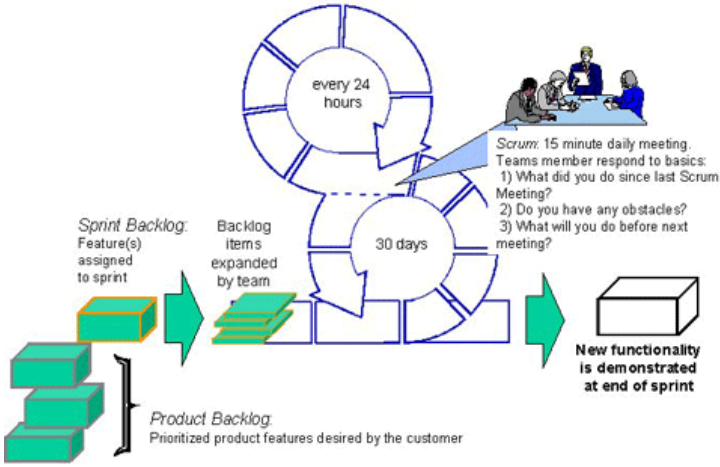
Nota. Scrum es un marco de trabajo para el Control del Proceso Empírico.

4.3.1 Justificación de Negocio

Es esencial para una organización comprender el concepto y el propósito de la Justificación de Negocio en el contexto de los proyectos Scrum. Antes de embarcarse en cualquier proyecto, resulta crucial que la organización realice una adecuada Justificación de Negocio y desarrolle una Declaración de la Visión del Proyecto que sea factible. Esto brinda la oportunidad a las partes clave o los responsables de tomar decisiones de evaluar si hay una necesidad real de que la empresa introduzca un cambio o un nuevo producto o servicio. Además, proporciona el razonamiento necesario para avanzar con el proyecto. Esta práctica también facilita al Product Owner la creación de un Backlog de Producto Priorizado, que tenga en cuenta las expectativas empresariales de la Alta Dirección y las partes interesadas relevantes (Overeem & Verwijs, 2020).

4.3.2 Sprint Backlogs

Figura 21. Sprint Backlogs.



Nota. El Sprint Backlog es la suma de el Objetivo del Sprint, los elementos del Product Backlog elegidos para el Sprint, más un plan de acción de cómo crear el Incremento de Producto.

Es uno de los tres artefactos esenciales de Scrum y se crea durante la reunión de Sprint Planning. Se trata de un plan elaborado por los Desarrolladores, diseñado para guiar su trabajo durante el Sprint.

Usualmente, el equipo descompone el trabajo en elementos conocidos como Elementos del Backlog del Sprint (SBI). Estos elementos pueden representar tareas específicas que el equipo debe llevar a cabo, componentes intermedios que se combinan para formar una entrega o cualquier otra unidad de trabajo que ayu-

de al equipo a comprender cómo alcanzar el Objetivo del Sprint dentro del propio Sprint (Garcia, 2020).

El compromiso del Sprint Backlog: El Objetivo del Sprint

Como sucede con todos los elementos en Scrum, el Sprint Backlog también incluye un compromiso específico. El compromiso asociado al Sprint Backlog es conocido como el Objetivo del Sprint, y se establece durante la reunión de Sprint Planning.

El Objetivo del Sprint representa el propósito central del Sprint. Aunque este compromiso es responsabilidad de los Desarrolladores, ofrece una cierta flexibilidad en cuanto a la naturaleza exacta del trabajo que se necesita para alcanzarlo.

Además, el Objetivo del Sprint contribuye a generar coherencia y concentración, fomentando así que el Equipo Scrum colabore en lugar de emprender iniciativas de manera aislada.

Ejemplo de Sprint Backlog

Aunque la guía Scrum no especifica un método específico para implementar este artefacto, considero que un enfoque recomendado podría ser la implementación de un tablero Kanban.

El tablero Kanban es una herramienta compuesta por columnas que representan el estado actual de una tarea y filas que reflejan diferentes categorías de actividades (por ejemplo, tareas desglosadas de las Historias de Usuario).

Cada tablero Kanban generalmente incluye al menos tres columnas con estados básicos:

"To Do" / Por hacer (donde se inicia una tarea).

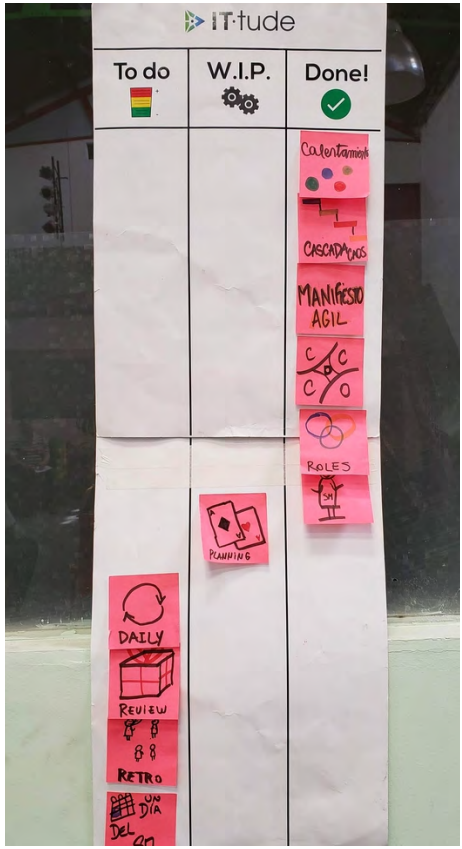
"W.I.P" / Trabajo en proceso (donde se trabaja activamente en la tarea).

"Done" / Terminado (donde se completa la tarea).

Aunque personalmente favorezco la opción de tener una representación física de este artefacto para fomentar la comunicación cara a cara, en la actualidad, existen soluciones digitales de tableros Kanban muy efectivas, como Trello, que son especialmente útiles para equipos que trabajan de forma remota.

Además, es posible agregar columnas adicionales a este tablero, como, por ejemplo "QA" (En etapa de pruebas).

Figura 22. Sprint Backlogs.



Nota. Ejemplo de implementación en un tablero Kanban.

Autoevaluación 14

1. **¿Cuál es el objetivo principal de la estrategia ágil en el desarrollo de software?**
    1. Desarrollar software con documentación detallada.
    2. Realizar proyectos de desarrollo a largo plazo.
    3. Entregar sistemas de software operativos de forma continua mediante iteraciones rápidas.
    4. Cumplir rigurosamente un conjunto de reglas predefinidas.
2. **¿Qué representa principalmente la metodología ágil en el desarrollo de software?**
    a. Un conjunto rígido de reglas para ejecutar proyectos.
    b. Una aproximación simplificada a la documentación de software.
    c. La resistencia a los cambios en el ciclo de vida del desarrollo.
    d. La entrega de software en grandes entregas finales.
3. **¿Cuáles son los tres objetivos esenciales que fomentan las metodologías ágiles en proyectos?**
    1. Establecer un cronograma estricto, minimizar costos y documentar exhaustivamente.
    2. Proporcionar valor, minimizar desperdicios y mejorar continuamente.
    3. Cumplir con los requisitos del cliente, evitar cambios y mantener una documentación detallada.
    4. Maximizar el alcance, minimizar los recursos y acelerar el proceso.

4. **¿Qué representa el Triángulo de Hierro en las metodologías ágiles?**
    a. Un conjunto de reglas estrictas para la gestión de proyectos.
    b. La priorización de la documentación sobre otros aspectos.
    c. Costo, Tiempo, Alcance y Calidad como elementos constantes.
    d. Un enfoque rígido en la entrega de proyectos.
5. **¿Cómo se relaciona el «Cono de Incertidumbre» con las metodologías ágiles?**
    1. Indica que la incertidumbre aumenta a medida que avanza el proyecto.
    2. Se utiliza para fijar objetivos claros en la gestión de proyectos.
    3. Sugiere que la documentación detallada es esencial para reducir la incertidumbre.
    4. Muestra que la gestión ágil minimiza la incertidumbre en el desarrollo de software.
6. **¿Cuál es el primer principio del Manifiesto Ágil?**
    a. Aprovechar el cambio como ventaja competitiva.
    b. Satisfacer al cliente mediante la entrega temprana y continua.
    c. Construir proyectos en torno a individuos motivados.
    d. Utilizar la comunicación cara a cara.
7. **¿Qué representa el segundo principio del Manifiesto Ágil?**
    1. Aprovechar el cambio como ventaja competitiva.
    2. Satisfacer al cliente mediante la entrega temprana y continua.
    3. Entregar valor frecuentemente.
    4. Cooperación negocio-desarrolladores durante todo el proyecto.

8. ¿Cuál es el tercer principio del Manifiesto Ágil?

1. Aprovechar el cambio como ventaja competitiva.
2. Satisfacer al cliente mediante la entrega temprana y continua.
3. Entregar valor frecuentemente.
4. Construir proyectos en torno a individuos motivados.

9. ¿Qué enfatiza el cuarto principio del Manifiesto Ágil?

1. La comunicación cara a cara es esencial para el éxito.
2. La colaboración entre negocio y desarrollo.
3. La importancia de mantener un desarrollo sostenible.
4. La simplicidad en la toma de decisiones.

10. ¿Cuál es el quinto principio del Manifiesto Ágil?

a. Utilizar la comunicación cara a cara.
b. La comunicación efectiva es fundamental.
c. Software funcionando como medida de progreso.
d. Promover y mantener un desarrollo sostenible.

11. ¿Qué se destaca en el sexto principio del Manifiesto Ágil?

1. La importancia de la comunicación a través de herramientas digitales.
2. La necesidad de documentar detalladamente todos los procesos.
3. La excelencia técnica como base para la agilidad.
4. La simplicidad en la toma de decisiones.

12. ¿Cuál es el séptimo principio del Manifiesto Ágil?

a. La colaboración negocio-desarrolladores durante todo el proyecto.
b. Software funcionando como medida de progreso.
c. La excelencia técnica mejora la agilidad.
d. La simplicidad es fundamental.

13.¿Qué se resalta en el octavo principio del Manifiesto Ágil?

a. La necesidad de desarrollar rápidamente sin preocuparse por la calidad.
b. La importancia de mantener un desarrollo sostenible.
c. La comunicación efectiva es fundamental.
d. La simplificación de los procesos de desarrollo.

14.¿Cuál es el noveno principio del Manifiesto Ágil?

a. La comunicación cara a cara es esencial para el éxito.
b. La simplicidad en la toma de decisiones.
c. La excelencia técnica mejora la agilidad.
d. Equipos auto-organizados para generar más valor.

15.¿Qué se destaca en el décimo principio del Manifiesto Ágil?

a. La importancia de mantener la documentación detallada.
b. La necesidad de automatizar todas las tareas.
c. La simplicidad en la toma de decisiones.
d. La forma más simple de abordar una situación.

16.¿Cuál es uno de los eventos principales en Scrum?

1. Sprint Review
2. Justificación de Negocio
3. Documentación de Proyecto
4. Reunión de Estrategia

17.¿Qué es esencial para una organización antes de embarcarse en un proyecto Scrum?

1. Reunión de Estrategia
2. Sprint Backlog
3. Justificación de Negocio
4. Backlog de Producto Priorizado

18.¿Cuál es uno de los artefactos clave en Scrum?

a. Reunión de Estrategia
b. Declaración de la Visión del Proyecto

c. Sprint Backlog
d. Justificación de Negocio

19. ¿Cuál de los siguientes no es un evento en Scrum?

1. Sprint Review
2. Sprint Backlog
3. Daily Standup
4. Sprint Planning

20. ¿Qué rol es responsable de crear un Backlog de Producto Priorizado?

a. Scrum Master
b. Product Owner
c. Desarrollador
d. Alta Dirección

4.4 Eventos de Scrum

Scrum incluye cuatro reuniones distintivas, cada una con un objetivo específico: la planificación del sprint, la reunión diaria de Scrum, la revisión del sprint y la retrospectiva del sprint. La reunión diaria de Scrum se realiza a diario por la mañana, mientras que las otras reuniones tienen lugar una vez en cada iteración del ciclo.

La palabra "sprint", al igual que "scrum", proviene del ámbito deportivo, y es una metáfora sumamente adecuada. No se trata de carreras de larga distancia, sino de esfuerzos intensos con un objetivo claro durante un período breve.

La duración óptima de un sprint dependerá de las circunstancias. Aunque algunas opiniones favorezcan sprints muy cortos, también es posible establecerlos de un mes. De hecho, la duración puede variar, incluso en el mismo proyecto. Esto depende del propósito de la iteración. Por ejemplo, si es necesario presentar una nueva funcionalidad en un evento dentro de dos semanas, se podría ajustar la duración del sprint aunque el equipo generalmente opere con sprints de 3 semanas. La clave radica en comprender que estos sprints subdividen el desarrollo en pequeñas metas. Se trata de una técnica para gestionar el avance constante. El período de tiempo debe ser lo bastante corto como para permitir una evaluación frecuente de nuestro progreso en la dirección correcta y determinar si se requieren ajustes (Domínguez Coutiño, 2012).

Figura 23. Eventos de Scrum.

4.4.1 Scrum Diario

El Daily Scrum, que es uno de los cinco eventos esenciales en Scrum, tiene una duración de 15 minutos y está diseñado para que los Desarrolladores se sincronicen entre sí.

Esta reunión diaria se efectúa en un horario y locación constantes, con el propósito de simplificar la situación. Durante ella, se busca lograr transparencia e inspeccionar el trabajo realizado, a fin de crear una oportunidad para ajustarse de cara al día siguiente.

¿Cuál es el objetivo de la Daily Scrum?

El propósito de la Daily Scrum es examinar el avance en dirección al Objetivo del Sprint y ajustar el Sprint Backlog según sea necesario. Su enfoque está en la sincronización de los Desarrolladores, planificando las labores para las próximas 24 horas.

Mediante esto, se optimiza la colaboración y el rendimiento del equipo al inspeccionar el progreso desde la última Reunión Daily Scrum y al proyectar el trabajo que resta en el Sprint.

Los Desarrolladores utilizan la Daily Scrum para evaluar su avance en relación con el Sprint Goal, y para evaluar cómo se desarrolla esa tendencia en términos de culminar las tareas del Sprint actual.

Dado que esta reunión posee un tiempo limitado de 15 minutos, cada individuo debe resumir de manera concisa lo que considera más relevante para lograr la sincronización con sus colegas.

¿Quién debe asistir al Daily Scrum?

La Daily Scrum es un evento destinado a los Desarrolladores. En el caso de que el Product Owner o el Scrum Master estén involucrados en la ejecución de elementos del Sprint Backlog, se integran como Desarrolladores. Si hay otras personas presentes, el Scrum Master se asegura de que no interfieran con el propósito de la reunión

.

Individuos externos al equipo de Desarrolladores pueden asistir a esta reunión diaria de sincronización si son invitados por ellos. Los Desarrolladores podrían considerar que en Scrum existe un principio de transparencia. Sin embargo, la presencia de personas ajenas puede influir en la dinámica de la reunión. De todas formas, solamente los Desarrolladores participan activamente en la sesión. Esto abarca tanto al Product Owner como a cualquier otra persona que no forme parte del equipo de Desarrolladores.

Beneficios del Daily Scrum

Optimización: La reunión diaria del Daily Scrum maximiza las probabilidades de que los Desarrolladores alcancen el Sprint Goal.

Reducción de tiempo perdido: Los Desarrolladores exponen los obstáculos diariamente, minimizando la pérdida de tiempo.

Mejora de la coordinación: Facilita la identificación de oportunidades para coordinarse, ya que todos están al tanto de las tareas de los demás.

Fomento del intercambio de conocimientos: El Daily Scrum señala áreas de desconocimiento y posibilita el enriquecimiento del equipo.

Consolidación de la cultura: Esto se logra mediante rituales compartidos y la participación activa.

Incremento de la productividad: Proporciona eficiencia al proceso, resultando en un aumento de la productividad.

Buenas prácticas del DAILY SCRUM

Los equipos Scrum altamente efectivos utilizan esta ocasión para actualizar su pizarra Kanban, también reconocida como "pizarra Scrum".

El equipo actualizará el estado más reciente de las tareas en la pizarra, con el propósito de hacer visibles los avances y obstáculos, y así determinar cómo pueden cooperar entre sí para concluir PRIORITARIAMENTE los elementos que contribuyen en mayor medida al Sprint Goal.

También pueden emplear parte del tiempo durante esta reunión de sincronización para actualizar su gráfico de Burndown, con el propósito de evaluar qué tan cerca o lejos están de alcanzar el Sprint Goal y poder ajustar su plan con base en esa información.

En cada iteración, se generará un nuevo gráfico, y a lo largo del proyecto, se pueden identificar tendencias y los principales obstáculos que contribuyen a la demora del proyecto.

Es esencial considerar que cuando los miembros del equipo exponen sus impedimentos, es natural que surja el deseo de profundizar en ellos y buscar posibles soluciones.

Recuerda, el Daily Scrum es una reunión destinada a replanificar y tomar decisiones, no para resolver problemas directa-

mente. Para abordar esos problemas, los involucrados pueden acordar reunirse después de la reunión diaria o en un horario específico del día para encontrar soluciones, sin consumir el tiempo de los demás. En la siguiente reunión diaria, podrán informar sobre la decisión que tomaron o cómo resolvieron el problema (Garcia, 2020).

4.4.2 Revisión de Sprint

El término "sprint review" se refiere a la reunión llevada a cabo con la intención de examinar los logros alcanzados por el equipo Scrum después de un sprint. Esto posibilita, asimismo, la evaluación del avance del desarrollo en relación con el logro del objetivo establecido.

Aunque comúnmente se piensa que el sprint review se centra en la presentación del producto, esta representa únicamente una porción reducida del evento en sí. Exploraremos con mayor detalle este aspecto en futuras secciones al analizar cómo se desarrolla la reunión (Aguirre, 2022).

Sprint review: objetivos

Dentro del marco de Scrum, se realiza una evaluación del proyecto con respecto al propósito del sprint, el cual ha sido establecido durante la sesión de planificación del sprint.

Al concluir la revisión del sprint, los objetivos principales que se buscan son los siguientes:

- Examinar el avance alcanzado en el incremento.
- Ajustar el product backlog en caso de ser requerido.

Figura 24. Sprint review.

Nota. La figura muestra cuando el equipo revisa los elementos de trabajo que completaron durante el sprint o la iteración.

4.4.3 Refinamiento del Product Backlog

Dentro de Scrum, el proceso de mejora del Backlog es una actividad constante en la cual el Product Owner y el Equipo de Desarrollo trabajan juntos para garantizar que los elementos en el Product Backlog:

- Son comprensibles de manera uniforme por todos los participantes (comprensión colectiva),

- Cuentan con una evaluación de su nivel de complejidad y trabajo necesario (relativos) para su ejecución, y
- Se organizan en función de su prioridad en relación con el valor empresarial y el esfuerzo involucrado.

En resumen, el proceso de refinar el backlog implica establecer un conocimiento compartido acerca de las funcionalidades que el producto abarcará y aquellas que no, evaluar el grado de esfuerzo necesario para llevar a cabo su implementación, y determinar el orden secuencial en que se abordarán estos aspectos.

¿Por qué es importante el refinamiento en el backlog?

Examinemos nuevamente los propósitos del proceso de mejora del backlog expuestos en la sección previa.

Si no se establece un entendimiento común, existe la posibilidad de implementar algo erróneo, malgastar esfuerzos y verse obligado a retrabajar la implementación para ejecutarlo de manera adecuada.

La falta de estimación para cada elemento impide considerar el "costo" de los elementos, con el riesgo de sobrevalorar los elementos de alto valor y gran costo, mientras que se subestiman aquellos de menor valor y costo.

Si no se organiza el producto backlog en un orden descendente de prioridad, existe la posibilidad de enfocarse en elementos de menor importancia y pasar por alto aquellos que son esenciales.

Otras razones por las que es importante el refinamiento del backlog:

Incrementa la efectividad de la sesión de Planificación del Sprint debido a que gran parte de las incógnitas ya han sido resueltas.

Preserva la claridad, orden y pertinencia del Backlog del Producto, evitando que se abrume en una lista de tareas en constante crecimiento.

Capitaliza los beneficios de trabajar en equipo para elaborar con mayor profundidad las historias de usuario y los problemas identificados.

Fomenta la construcción de un conocimiento mutuo tanto en el Equipo Scrum como en los involucrados externos.

¿Cuánto tiempo debe durar el refinamiento del backlog?

La Guía de Scrum no establece una duración específica para el proceso de Refinamiento del Backlog. Simplemente menciona que generalmente no debe exceder el 10% de la capacidad del equipo de desarrollo. Dado que el Product Owner no forma parte del Equipo de Desarrollo, puede invertir el tiempo necesario y solicitar asistencia de otros miembros del Equipo Scrum. La conversión de una historia en un Spike es una estrategia para dejar esto explícito y prevenir que esa cuota del 10% se consuma (Work, 2023).

4.4.4 Conclusión

En este capítulo, hemos explorado una amplia gama de temas esenciales en el desarrollo de software y en la adopción de enfoques ágiles. Desde la Especificación del software hasta el Refinamiento del Product Backlog, hemos abordado áreas críticas como el Diseño e implementación del software, la Validación de Software y la Evolución de Software.

Hemos comprendido cómo la Especificación del software desempeña un papel fundamental al recopilar y analizar requisitos, y cómo el Diseño e implementación del software transforma estos requisitos en soluciones técnicas concretas. Exploramos los pilares de los enfoques ágiles, aprendiendo del Manifiesto y Principios Ágiles, y profundizamos en el marco de trabajo Scrum, donde se desglosaron los Valores, Pilares, Roles y Elementos esenciales.

Además, se subrayó la importancia de la Justificación de Negocio y se exploraron los Artefactos y Eventos de Scrum, como los Sprint Backlogs y las Reuniones de Revisión, que permiten una adaptación constante. En resumen, este capítulo ofrece una visión completa para diseñar, desarrollar y evolucionar software de manera eficaz, resaltando la relevancia de los enfoques ágiles y Scrum en la consecución de la agilidad, colaboración y entrega continua de valor en el desarrollo de productos.

Autoevaluación 15

1. **¿Qué evento de Scrum se lleva a cabo al final de cada Sprint y se centra en mostrar lo que se ha logrado durante ese Sprint?**
    1. Sprint Planning
    2. Sprint Backlog
    3. Daily Standup
    4. Sprint Review
2. **¿Cuál de los siguientes eventos de Scrum es una reunión diaria de seguimiento de 15 minutos?**
    a. Sprint Planning
    b. Sprint Review
    c. Daily Standup
    d. Sprint Retrospective
3. **¿Cuál es el propósito principal de la reunión de Sprint Planning en Scrum?**
    a. Revisar el trabajo completado en el Sprint anterior.
    b. Definir los roles y responsabilidades del equipo.
    c. Seleccionar las tareas que se abordarán en el próximo Sprint.
    d. Evaluar el rendimiento del Scrum Master.
4. **¿Cuál de los siguientes eventos de Scrum se enfoca en la mejora continua y la adaptación de procesos?**
    1. Sprint Planning
    2. Sprint Review
    3. Daily Standup
    4. Sprint Retrospective

5. **¿Qué evento de Scrum se refiere a la reunión donde el equipo de Scrum analiza su trabajo y busca oportunidades para mejorar?**
    a. Sprint Planning
    b. Sprint Review
    c. Daily Standup
    d. Sprint Retrospective
6. **¿Cuál es el otro nombre comúnmente utilizado para la reunión diaria de Scrum?**
    1. Sprint Planning
    2. Daily Review
    3. Daily Standup
    4. Sprint Retrospective
7. **¿Cuál es el propósito principal de la reunión diaria de Scrum?**
    a. Tomar decisiones importantes sobre el proyecto.
    b. Revisar y actualizar el Backlog de Producto.
    c. Identificar problemas y coordinar el trabajo del equipo.
    d. Realizar una revisión detallada de los entregables del Sprint.
8. **¿Cuál es la duración recomendada para la reunión diaria de Scrum?**
    a. 1 hora
    b. 30 minutos
    c. 15 minutos
    d. 2 horas

9. ¿Qué pregunta típicamente se hace durante la reunión diaria de Scrum?

1. ¿Qué hizo usted durante el fin de semana?
2. ¿Qué obstáculos enfrentó ayer?
3. ¿Cuándo se completará el proyecto?
4. ¿Qué se espera del próximo Sprint?

10. ¿Cuál es el objetivo principal de limitar la duración de la reunión diaria de Scrum a 15 minutos?

1. Para que el equipo tenga tiempo para socializar.
2. Para evitar discusiones largas y centrarse en la coordinación.
3. Para cumplir con una regla arbitraria de tiempo.
4. Para dar tiempo a los gerentes para tomar decisiones.

11. ¿Qué es el «Refinamiento del Product Backlog" en Scrum?

a. Una reunión diaria para revisar el progreso del equipo.
b. Un evento en el que se crea el Product Backlog.
c. El proceso de mejorar y aclarar elementos del Product Backlog.
d. Una técnica para priorizar las historias de usuario.

12. ¿Quién es responsable de liderar el proceso de refinamiento del Product Backlog?

a. El Scrum Master.
b. El Product Owner.
c. El equipo de desarrollo.
d. La alta dirección de la empresa.

13. ¿Cuál es uno de los objetivos principales del refinamiento del Product Backlog?

1. Crear un plan detallado para el próximo Sprint.
2. Priorizar los elementos del Product Backlog en orden alfabético.
3. Asegurarse de que los elementos del Product Backlog sean comprensibles y estimables.
4. Evaluar el desempeño del Scrum Master.

14. ¿Cuándo suele llevarse a cabo el refinamiento del Product Backlog?

1. Al final de cada Sprint.
2. Durante la reunión de planificación del Sprint.
3. Durante la reunión diaria de Scrum.
4. De manera continua a lo largo del Sprint.

15. ¿Qué beneficio se obtiene al refinar el Product Backlog de forma continua durante el Sprint?

1. Se evitan las reuniones innecesarias.
2. Se facilita la identificación de nuevos elementos para el Sprint.
3. Se mantiene una visión clara de los elementos a trabajar en futuros Sprints.
4. Se reduce la necesidad de contar con un Scrum Master.

Referencias

Aguirre, M.F. (2022, 21 de 01). ¿Qué es y cómo llevar a cabo un sprint review? *Appvizer.* https://www.appvizer.es/revista/organizacion-planificacion/gestion-proyectos/sprint-review

Domínguez Coutiño, L.A. (2012). *Análisis de Sistemas de Información.* Red Tercer Milenio

Dremyn, P. (2020, 12 de octubre). El enfoque ágil y porqué debes usarlo. *Linkedin*. https://es.linkedin.com/pulse/el-enfoque-ágil-y-porqué-debes-usarlo-paul-dremyn

Garcia, M. (2020). ¿Qué es el sprint backlog? *IT.tude*. https://ittude-agile.com/b/scrum/que-es-el-sprint-backlog/

James, O. (2007). *Diseño de Sistemas de Información Gerencial*. McGraw-Hill

León Yacelga, A.R., Acosta Espinoza, J.L., & Díaz Vásquez, R.A. (2021). Aplicación de la metodología incremental en el desarrollo de sistemas de información. *Universidad Y Sociedad, 13*(5), 175-182. https://rus.ucf.edu.cu/index.php/rus/article/view/2223

Martins, J. (2023, 19 de junio). Scrum: conceptos clave y cómo se aplica en la gestión de proyectos. *Asana*. https://asana.com/es/resources/what-is-scrum

Overeem, B., & Verwijs, C. (2020). *Scrum: Un Marco de Trabajo para Reducir el Riesgo y Entregar Valor Antes.* The Liberators.

SENTRIO. (2021, 13 de octubre). *Metodologías Agile: los 4 valores y 12 principios del 'Manifiesto Ágil'*. https://acortar.link/Eko9Zt

Sommerville, I. (2011). *Ingeniería de Software*. Pearson Edcucación.

Sulbarán, H. (2014, 24 de septiembre). *Paradigmas en el desarrollo de software* [Blog]. https://acortar.link/VUTmOL

Sulbarán, I.. (2023, 27 de abril). Interfaz de usuario (ui): ejemplos y tipos. *Tiffin University*. https://global.tiffin.edu/noticias/interfaz-de-usuario-ui-ejemplos-y-tipos

§

Software Specification and Agile Approaches
Especificação de software e abordagens ágeis

Mónica Elizabeth Páez Padilla
Instituto de Educación Superior Nelson Torres | Cayambe | Ecuador
https://orcid.org/0009-0006-1030-1394
monica.paez@intsuperior.edu.ec
Diego Javier Portilla Martínez
Investigador independiente | Cayambe | Ecuador
https://orcid.org/0009-0008-7295-6227
dijapm@gmail.com

Abstract
In this chapter, fundamental issues ranging from software definition to the implementation of agile approaches are discussed, with a particular focus on the Scrum framework. It highlights the relevance of relying not only on methodologies and processes to produce high quality software, but also on accurate specification and the application of agile practices. The text focuses on the software specification stage, including design, implementation, validation and evolution. It also explores agile approaches to software development, detailing their suitability and the Agile Manifesto. Next, the key components of Scrum are discussed, including roles, responsibilities, interaction, business justification, and artifacts. Finally, it delves into the essential Scrum events, including the Daily Scrum, the retrospective and the refinement of the Product Backlog.
Keywords: Scrum, Product Backlog, Agile Manifesto, software.

Resumo
Este capítulo aborda questões fundamentais que vão desde a definição de software até a implementação de abordagens ágeis, com foco especial na estrutura Scrum. Ele destaca a relevância de contar não apenas com metodologias e processos para produzir software de alta qualidade, mas também com uma especificação precisa e a aplicação de práticas ágeis. O texto concentra-se no estágio de especificação do software, incluindo design, implementação, validação e evolução. Ele também explora abordagens ágeis para o desenvolvimento de software, detalhando sua adequação e o Manifesto Ágil. Em seguida, são discutidos os principais componentes do Scrum, incluindo funções, responsabilidades, interação, justificativa comercial e artefatos. Por fim, são abordados os eventos essenciais do Scrum, incluindo o Diário do Scrum, a retrospectiva e o refinamento do Backlog do Produto.
Palavras-chave: Scrum, Product Backlog, Manifesto Ágil, software.