RESUMEN INGENIERIA DE SOFTWARE

-2019-

INDICE

UNIDAD 1: Ingeniería de Software en contexto

Introducción

¿Qué es el Software?

Se define como un conjunto de programas con la documentación que lo describe. Es decir que esta conformado por:

1. Diversos programas independientes.
2. Archivos de configuración utilizados para la ejecución de los programas.
3. Documentación que describe la estructura del sistema.
4. Documentación para el usuario que explica como usar el sistema.

En la actualidad, es imposible operar sin software. Lo encontramos en las telecomunicaciones, en la industria, en el entretenimiento, en los servicios públicos.

Clasificación de Productos de Software

1. Productos genéricos: sistemas independientes producidos por una organización de desarrollo y vendidos en el mercado abierto a cualquier cliente que desee comprarlo. Procesadores de texto, motores de base de datos, paquetes de dibujo son ejemplos de estos. Usualmente los requerimientos son especificados por la organización que lo desarrolla.
2. Productos personalizados: sistemas destinados a un cliente en particular. Un contratista de software desarrolla el producto para ese cliente. Sistema para dar soporte a una unidad de negocio de una empresa es un ejemplo. Los requerimientos son especificados por el cliente que lo solicita.

En contraste con la manufactura

1. Es menos predecible: es mas difícil de adivinar que producto de software es realmente útil y necesario.
2. No existe la producción en masa: ningún producto de software es igual a otro.
3. No todas las fallas son errores
4. El software no se gasta.
5. El software no esta gobernado por las leyes de la física.

¿Qué es la Ingeniería de Software?

Es una disciplina de ingeniería que se encarga de todos los aspectos de la producción del software, desde la primera etapa de especificación hasta el mantenimiento del sistema luego de que se pone en operación.

Disciplina de ingeniería

Los ingenieros buscan soluciones delimitadas por restricciones organizacionales y financieras, con la utilización de técnicas, herramientas y teorías adecuadas para que las cosas funcionen correctamente.

Aspectos de la producción del software

Incluye:

1. Desarrollo de software.
2. Administración de proyectos de software.
3. Desarrollo de herramientas, métodos y teorías.

Disciplinas que conforman a la Ingeniería de Software

1. Técnicas: ayudan a construir el producto. Técnicas para relevar requerimientos, para hacer análisis y diseño, para implementar lo diseñado y para probar lo implementado.
2. De gestión: incluye las actividades de planificación de proyectos, monitoreo y control.

3. De soporte: gestión de configuración, métricas y aseguramiento de calidad.

Retos que afronta la Ingeniería de Software

1. Heterogeneidad: desarrollar técnicas para la construcción de software que pueda operar como un sistema distribuido en redes con distintos tipos de computadoras, con sistemas en otro lenguaje, etc.
2. Entrega: reducir los tiempos de entrega de sistemas grandes y complejos sin afectar la calidad del sistema.
3. Confianza: desarrollar técnicas que demuestren que los usuarios confían en el sistema.

La crisis del software

Este termino hace referencia a un conjunto de hechos relativos al software, planteados en la conferencia de OTAN en 1968 por Friedrich Bauer. Sin embargo, este termino fue utilizado anteriormente por Dijkstra en El Humilde Programador.

Noción del termino

La crisis del software refiere a un conjunto de dificultades o errores ocurridos en la planificación, estimaciones, productividad y calidad de sistemas de software grandes y complejos.

Causas

La evolución de la tecnología del hardware mediante los circuitos integrados permitió la posibilidad del desarrollo de sistemas mas grandes y complejos. El salto en el hardware no fue acompañado por un salto en el desarrollo del software, esto en gran parte se debe a que el software es una actividad humana cuyo producto es abstracto e intangible.

En combinación con lo anterior, la demanda creciente de estos sistemas complejos, la subestimación de la complejidad que supone el desarrollo de software, los cambios a los que tienen que ser sometidos los productos para ser adaptados a las necesidades del cliente y la falta de una disciplina que intervenga en todos los aspectos de la producción del software fueron las causas de esta crisis.

Síntomas y consecuencias

Las causas mencionadas se materializaban en:

1. Atraso en la finalización del los proyectos.
2. Se sobrepasaban los presupuestos planificados.
3. Muchos proyectos se cancelaban.
4. Se comenzaba a desarrollar el software sin especificaciones.

Esto implico software de mala calidad y de desempeño pobre, lo cual derivaba en la realización de intensas actividades de mantenimiento que degradan aun mas al software.

De esta manera, se hizo evidente que la construcción de software de gran magnitud mediante un enfoque informal no era lo adecuado, sino que se necesitaban nuevas técnicas y métodos para lidiar con la complejidad inherente de los sistemas grandes.

Sin embargo, la realidad actual es que cuanto mas crezca la capacidad de producir software, también lo hará la complejidad de los sistemas demandados.

Ciclo de vida (Modelo de proceso)

Es una representación simplificada de un proceso, el cual define elementos del proceso (actividades estructurales, productos del trabajo, tareas, etc.) y el flujo del proceso (o flujo de trabajo), que especifica la relación y el orden de dichos elementos.
Los modelos de proceso pueden incluir las mismas actividades, pero cada uno, desde una perspectiva particular, pondrá mas énfasis en algunas respecto a otras y definirán flujos de procesos diferentes.

Modelo secuencial

Este modelo dispone a las actividades de forma lineal. Una actividad no puede comenzar sin que la precedente haya sido finalizada. El valor agregado se obtiene al finalizar la ultima fase del proceso.
Suele utilizarse en aquellos sistemas donde los requerimientos se comprender bien y son exhaustivos.

Dificultades

1. Los proyectos reales raramente siguen un flujo secuencial.
2. El modelo secuencia exige que los requerimientos sean completamente explícitos desde un principio y esto en la realidad no suele suceder. No puede lidiar con la incertidumbre.
3. El cliente obtiene una versión funcional del producto en etapas muy avanzadas del proyecto.
4. Encontrar un defecto implica un gran rediseño de la solución, ya que esta prácticamente todo hecho.
5. Dependencia entre los equipos de trabajo. Un equipo no puede comenzar hasta que el otro no haya finalizado.

Modelo iterativo-incremental

El modelo iterativo-incremental aplica sucesivas iteraciones en forma escalonada a medida que avanza el calendario de actividades. Cada iteración produce un incremento de software funcional potencialmente entregable. Usualmente los primeros incrementos incluyen las funciones básicas o criticas que mas requiere el cliente.

Ventajas frente al Modelo secuencial

1. El retrabajo en requerimientos y análisis es menor, ya que en cada incremento se obtiene una retroalimentación que permite adaptar el modelo a las necesidades del cliente.
2. El cliente es parte del proceso de desarrollo. En el modelo secuencial, el cliente debía esperar hasta la ultima instancia para recibir el producto y no era capaz de juzgar el avance del producto a partir de documentos de diseño del software.
3. La entrega de software funcional y de valor es mas rápida, por lo cual el cliente puede obtener los beneficios de utilizar el sistema antes con respecto al modelo secuencial.

Dificultades desde una perspectiva administrativa

1. Se invisibiliza el proceso.
2. Alto costo de documentar cada incremento.
3. La estructura del software tiende a degradarse frente a una cantidad considerable de incrementos.

Modelo recursivo

El modelo recursivo es utilizado para gestionar los riesgos del desarrollo de sistemas complejos a gran escala. Requiere de la intervención del cliente. El modelo es espiral es un modelo recursivo, que consta de 4 etapas recursivas:

1. Determinar objetivos.

2. Identificar y resolver los riesgos.
3. Desarrollar y probar.
4. Planificar la próxima iteración.

Para llevarlo adelante, se subdivide el proyecto en varios mini proyectos, intentando resolver en cada uno los riesgos mas relevantes hasta que no quede ninguno.

Criterio para la selección de un modelo de proceso

Es necesario evaluar los niveles de riesgo, la claridad y cantidad de requerimientos, los conocimientos técnicos con los que cuenta el equipo de desarrollo, los plazos que se tienen para ejecutar el proyecto y la posibilidad de poder validar los resultados del trabajo con el cliente.

1. Si los requerimientos son claros y exhaustivos, en un contexto de baja incertidumbre en el que el cliente no requiere obtener el producto rápidamente, es posible aplicar el modelo secuencial.
2. Si la situación anterior no se presenta y el cliente desea obtener resultados lo antes posible, con la funciones principales del sistema, entonces seria conveniente la elección del modelo iterativo-incremental.
3. Si el objetivo es disminuir los riesgos presentes en el desarrollo de sistemas grandes y complejos, entonces el modelo recursivo puede ser una buena opción.

Proceso de desarrollo de software

Las actividades del proceso de desarrollo son la especificación de requerimientos, en análisis, el diseño, la implementación y la prueba. El orden y la relación entre las diferentes actividades esta determinada por la elección del ciclo de vida.

Procesos definidos

Los procesos definidos son considerados deterministas: ante la misma entrada obtendremos la misma salida. Asumen que, si se aplican una y otra vez, se obtendrán siempre los mismos resultados.

1. Están inspirados en las líneas de producción.
2. La administración y el control provienen de la predictibilidad del proceso.

Procesos empíricos

Los procesos empíricos se conforman en base a la experiencia de las personas que intervienen en un contexto particular. No es considerado determinista, ya que no se asume que ante la aplicación del mismo proceso se obtendrán los mismos resultados. Dado que el factor diferencial es la experiencia sensible, en estos procesos se pueden obtener diferentes resultados dependientes del contexto en el cual se apliquen.

1. Se ajustan de mejor forma en aquellos dominios complejos donde prima la creatividad.
2. La administración y el control es por medio de inspecciones frecuentes y adaptaciones.

Proyectos de Software

Definición de un Proyecto de Software

Un proyecto de software es un esfuerzo temporal que requiere del acuerdo de un conjunto de especialidades y recursos para la creación de un producto, servicio o resultado único. Dentro de sus características, nos encontramos que son:

1. Únicos: todos los proyectos por mas similares que sean, tienen características propias que los hacen únicos como por ejemplo el calendario.
2. Orientado a objetivos:
    a. No ambiguos: deben ser lo suficientemente claros para guiar el desarrollo del proyecto.
    b. Medibles: es necesario medir el objetivo para determinar el avance sobre el proyecto.
    c. Alcanzable: deben ser factibles de realizarse por el equipo.
3. Duración limitada de tiempo: esto implica que un proyecto tiene un principio y un fin. Cuando se alcanzan los objetivos del proyecto, este llega a su fin. Un ejemplo claro de lo que no es un proyecto es una línea de producción, ya que esta se ejecuta indefinidamente en el tiempo.
4. Conjunto de tareas interrelacionadas basadas en esfuerzo y recursos: esto se debe a la complejidad sistemática de los problemas.

Administración de un Proyecto de Software

Consiste en planificar y seguir continuamente las actividades del proyecto aplicando los conocimientos , habilidades, herramientas y técnicas necesarios para satisfacer los requerimientos del proyecto: cumplir con sus objetivos con el presupuesto y en el tiempo acordado.
La administración incluye:

1. Identificar los requerimientos.
2. Establecer objetivos claros y alcanzables.
3. Adaptar las especificaciones, planes y el enfoque a los diferentes intereses de los involucrados (stakeholders)

Una buena gestión no puede garantizar el éxito del proyecto. Sin embargo, una mala gestión usualmente implica una falla en el proyecto: el software puede entregarse tarde, costar más de lo estimado originalmente o no cumplir las expectativas de los clientes.

Metas mas importantes de la Administración de Proyecto

1. Entregar el software al cliente en el tiempo acordado.
2. Mantener los costos dentro del presupuesto general.
3. Entregar software que cumpla con las expectativas del cliente.
4. Mantener un equipo de desarrollo optimo y con buen funcionamiento.

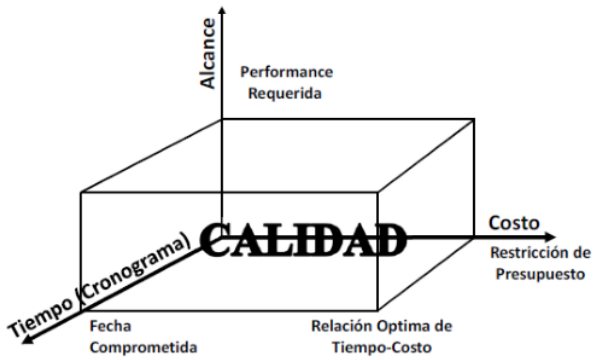
Atributos particulares de la Administración de Proyectos de Software

1. El producto es intangible: no poder ver ni tocar el producto que se obtiene al ejecutar el proyecto dificulta la medición del progreso del proyecto. Una construcción civil es visible y se puede evidenciar el progreso en función de las partes construidas.
2. Los proyectos de software suelen ser excepcionales: los conocimientos aprendidos en un proyecto de software son difícilmente extrapolables a otro proyecto. El factor tecnológico es una de las causas.
3. Los procesos de desarrollo son variables y específicos de la organización: a pesar del intento de estandarizar los procesos de desarrollo, la realidad es que no es posible predecir de manera confiable que proceso es el adecuado para adoptar en el proyecto.

Triple restricción

Las variables de restricción de un proyecto son 3: tiempo, costo y alcance. El concepto de la triple restricción hace referencia a que no es posible fijar de forma arbitraria a las 3 variables, sino que será posible fijar 2 y la tercera se ajustará a lo definido.

El balance de estos tres factores afecta directamente la calidad del proyecto, ya que un proyecto de alta calidad es aquel que entrega el producto, servicio o resultado requerido, satisfaciendo los objetivos en el tiempo estipulado y con el presupuesto planificado. Es responsabilidad del Líder de proyecto balancear estas variables.



Alcance

Son los requerimientos que se incluyen en el proyecto y definen la funcionalidad que tendrá el producto a entregar al cliente. En las metodologías tradicionales el alcance es lo primero que se determina y se deja fijo en función de las necesidades del cliente.

Tiempo

Define la fecha de calendario en la cual se compromete a finalizar el proyecto y entregarle el producto resultante al cliente. En las metodologías agiles, el tiempo es fijo.

Costo

Define el costo de todos los recursos implicados en el desarrollo del proyecto.

Proceso, Proyecto y Producto

La administración efectiva de un proyecto de software se enfoca en las cuatro P: personal, producto, proceso y proyecto.

Decimos que el proceso, automatizado por herramientas, se adapta al proyecto, al cual se incorporan personas, y de este se obtiene como resultado un producto.

Producto

En el enfoque tradicional, previo a la definición del proyecto, es necesario determinar los objetivos y el ámbito del producto para de esta manera poder realizar las estimaciones pertinentes. Los objetivos identifican las metas globales del producto sin especificar como se van a lograr. El ámbito del producto define las funciones y comportamientos que caracterizan al producto que utilizara cliente.

Proceso

El proceso proporciona un marco conceptual en donde se establece un plan completo para el desarrollo del software. Define como se va a desarrollar el software, estableciendo un conjunto de actividades.

Las disciplinas de gestión permiten que las actividades del marco conceptual se adapten a las características del proyecto de software y a los requerimientos del equipo.

Las disciplinas de soporte son independientes del marco conceptual y recubren al modelo del proceso, es decir que atraviesan todas las actividades del proceso de desarrollo.

Se debe definir el modelo de proceso a utilizar: secuencial, iterativo o recursivo. Luego, el marco conceptual que incluye las actividades fundamentales del desarrollo del software, se adapta al modelo elegido.

Proyecto

Una vez definido el modelo de proceso en el contexto del proyecto, es posible realizar la planificación del proyecto.

Personal

El factor mas importante que determina el éxito de un proyecto de software, es el personal.

Participantes

Se pueden clasificar en 5 categorías

1. Gestores ejecutivos: definen aspectos del negocio y tienen influencia significativa en los proyectos.
2. Gestores de proyectos (técnico): planifican, motivan, organizan y controlan a los profesionales que realizan el trabajo del software.
3. Profesionales: quienes proporcionan las habilidades técnicas necesarias para la elaboración de un producto de software.
4. Clientes: quienes especifican los requerimientos.
5. Usuarios finales: quienes interactúan con el software una vez que se libera el producto.

Líder de proyecto

Usualmente los profesionales técnicos no disponen de las habilidades necesarias para tratar con la gente, por lo que es necesaria la figura de un líder de proyecto que tenga las siguientes aptitudes (Soft skills)

1. Motivación: habilidad para alentar al personal técnico a producir a su máxima capacidad.
2. Organización: habilidad para adaptar el proceso existente (o inventar nuevos), para lograr que el concepto inicial se transforme en el producto final.
3. Innovación: habilidad para alentara a las personas a crear y sentirse creativas.

También cuenta con las Hard skills, las cuales están asociadas a los conocimientos del producto, de técnicas y de herramientas.

Dado que el líder de proyecto tiene un enfoque basado en la resolución de problemas, también son características propias las siguientes:

1. Resolución de problemas: identifica conflictos técnicos y organizativos, estructura la solución o motiva a otros profesionales para que la desarrollen, aplica lecciones aprendidas en otros proyectos y adapta el proceso de resolución ante inconvenientes.
2. Identidad administrativa: tiene la confianza de asumir el control cuando es necesario.
3. Logro: recompensa las iniciativas y los logros, incentivando al equipo a correr los riesgos de manera controlada.
4. Influencia: define la estructura del equipo en función de la retroalimentación que el mismo suministra mediante palabras y gestos.

Entre las responsabilidades del líder de proyecto se encuentran:

1. Definir el alcance del proyecto.
2. Identificar a los involucrados, recursos y presupuesto.
3. Identificar y evaluar riesgos.
4. Preparar planes de contingencia.
5. Controlar hitos y participar en las revisiones de las fases del proyecto.

Equipo de proyecto

Es un grupo de personas comprometidas con alcanzar un conjunto de objetivos de los cuales se sienten mutuamente responsables.

Entre sus características se encuentran:

1. Cuentan con conocimientos, habilidades técnicas, experiencias y diversas personalidades.
2. Suelen ser grupos pequeños de no mas de 10 personas.
3. Trabajan en forma conjunto con espíritu de grupo, convirtiéndose en un equipo cohesivo donde prima la sinergia del mismo.
4. Tienen el sentido de responsabilidad como una unidad y se focalizan en el cumplimiento de las metas grupales.

En equipos cohesivos:

1. El grupo establece sus propios estándares de calidad.
2. Los individuos aprenden de los demás y se apoyan mutuamente.
3. El conocimiento se comparte.
4. Prevalece el mejoramiento continuo.

Etapas del Proyecto de Software

Antes del proyecto

Se identifica al cliente y al mercado, realizando un estudio del dominio en el cual se implantará el sistema. Luego se realiza un estudio de factibilidad técnica, económica y operativa, para determinar si el sistema es posible de realizarse y si otorgara beneficios.

Inicio del proyecto

Se elige un nombre para el proyecto y se define:

1. Estructura del equipo.
2. Necesidades de soporte.
3. Necesidades financieras.
4. Objetivos de calidad.

Planificación del proyecto y armado del equipo

1. Definición del proyecto: determinar los problemas y los objetivos, los cuales deben ser claros, distinguibles y medibles.
2. Creación del plan del proyecto: permite conducir el desarrollo del proyecto. Puede revisarse y adaptarse en el ciclo de vida del proyecto.
3. Definición la infraestructura del proyecto. La administración de configuración, la preparación de los diferentes entornos, herramientas de comunicación, etc.
4. Formación del equipo: determinar los miembros que componen al proyecto y los roles que tendrán.
5. Producción de documentos claves:
    a. Libro de requerimientos.
    b. Plan de gestión de configuración.
    c. Plan de aseguramiento de calidad.
    d. Directorio o repositorio del proyecto.

Monitoreo y Control

El control del proyectos se refiere a comparar el progreso con respecto al plan, con el objetivo de determinar si estamos bien encaminados. De no ser así, tomar medidas preventivas para evitar desviaciones futuras. En caso de que las desviaciones sean manifiestas, entonces tomar medidas correctivas que permitan reencausar el plan.

El objetivo del monitoreo es determinar el estado actual del proyecto:

1. Proyecto bajo control: se alcanzan los hitos a tiempo con los recursos estimados y con el nivel de calidad esperado.
2. Proyecto fuera de control: se debe renegociar y adaptar el plan del proyecto.

Para realizar seguimiento se realizan reuniones semanales que pueden incluir encuentros con el cliente. Se pueden incluir check list de fin de fase, revisiones del plan y auditorias.
La comparación de lo real contra lo planificado puede realizarse sobre el diagrama de Gantt, el cual permite estimar el porcentaje de tareas no finalizadas y que aun no han comenzado, cuando debieron haberlo hecho.
Para el seguimiento del proceso se realiza un control sobre hitos o tareas, en cambio para el seguimiento del producto se realiza un control sobre entregables.

Entregables y soporte

1. Hitos son puntos finales de las actividades del proceso (no necesariamente entregas), representan el fin de una etapa lógica en el proyecto y para cada uno de ellos debe existir una salida formal. No necesariamente todas las actividades deben finalizar en un hito.
2. Entregables: son hitos, resultado del proyecto entregado al cliente al final de una fase principal del proyecto, estos están definidos en el libro de requerimientos o de en plan. La entrega no tiene que ser necesariamente la culminación del proyecto.

Las entregas son hitos, pero los hitos no necesariamente son entregas

Plan de Proyecto

Es un documento que especifica:

1. ¿Qué hacemos? Refiere al alcance del proyecto que guiara su desarrollo.
2. ¿Cuándo lo hacemos? Refiere al calendario, las fechas estipuladas para cubrir el alcance del proyecto.
3. ¿Como lo hacemos? Refiere a todas las actividades o tareas que se deben llevar a cabo para cubrir el alcance.
4. ¿Quién lo va a hacer? Refiere a los responsables de llevar a cabo cada actividad o tarea especificada anteriormente.

En definitiva, en el plan se establecen el alcance, los recursos disponibles para el proyecto, la división del trabajo y el calendario para realizarlo. Se especifica el proceso y el ciclo de vida. También debe identificar los riesgos para el proyecto y para el software en desarrollo, especificando la gestión de dichos riesgos.
Es de suma importancia que este plan sea correctamente mantenido y actualizado, ya que es el que conduce el desarrollo del proyecto y si no se encuentra al día, es probable que el proyecto fracase.

Gestión de riesgos

La gestión de riesgos implica identificar riesgos que pudieran alterar el calendario del proyecto, el presupuesto inicial o la calidad de software a entregar, y posteriormente tomar acciones para evitar la materialización de los mismos.

Riesgo

Se define como la probabilidad de ocurrencia de un evento no deseado y se clasifican en:

1. Riesgos del proyecto: aquellos que alteran el calendario o los recursos del proyecto. Un ejemplo de riesgo de proyecto es la renuncia de un diseñador experimentado.

2. Riesgos del producto: aquellos que afectan la calidad o el rendimiento del software a desarrollar. Un ejemplo de riesgo de producto es la falla que presenta un componente que se adquirió al no desempeñarse como se esperaba.
3. Riesgos organizacionales: aquellos que afectan a la organización que desarrolla o que adquiere el software. Por ejemplo, un competidor que introduce un nuevo producto.

Para medir los riesgos se utiliza lo que se conoce como exposición al riesgo, la cual se obtiene como el producto entre la probabilidad de ocurrencia de la amenaza por el impacto que generaría si realmente ocurriera. Este valor permite organizar y gestionar los riesgos, y se debe hacer foco en reducir la exposición del riesgo.
Entro los riesgos mas comunes se encuentran:

1. Cronogramas y presupuestos irreales.
2. Desarrollo de funciones e interfaces de usuario erróneas.
3. Cambio de requerimientos.

Fases de la Gestión de riesgos

1. Identificación de riesgos: se identifican los riesgos que suponen una mayor amenaza al proceso de ingeniería de software, al software a desarrollar o a la organización que lo desarrolla.
2. Análisis de riesgos: para cada riesgo identificado se realiza un juicio acerca de la probabilidad y del impacto. No existe una forma certera de realizar esto. Se utilizan intervalos de probabilidad y clasificaciones de gravedad. El criterio dependerá de la experiencia de quien realice el análisis de riesgos.
3. Planeación del riesgo: para cada riesgo analizado, se definen estrategias para manejarlos. Estas estrategias consisten en considerar acciones a tomar para minimizar o evitar el impacto sobre el proyecto como consecuencia de la ocurrencia de la amenaza que representa el riesgo. Otras estrategias consisten en desarrollar planes de contingencia, el cual define un plan para enfrentar una situación controversial.
4. Monitorización del riesgo: proceso que permite determinar que las suposiciones acerca de los riesgos de producto, proceso y organización no han cambiado. Esteo permite revalorar al riesgo en términos de posibles variaciones de su probabilidad e impacto. Este proceso se aplica en todas las etapas del proyecto.

Éxitos y fracasos en Proyectos

Éxito

Un proyecto es exitoso cuando se cumplen los siguientes puntos mas importantes :

1. Monitoreo y Feedback: Realizar verificaciones constantes para comprobar que la implementación avanza como se planificó y feedback de tu plan a tu equipo, cliente, etc. Así te aseguras de que es viable y que todo el mundo está de acuerdo.
2. Tener una misión/objetivo claro : Si bien muchas veces se da por sentado que todo el mundo entiende las razones de por qué se está llevando a cabo un determinado proyecto, es necesario ser claro y explícito al declarar cuál es el objetivo general del proyecto, y qué objetivos específicos se han determinado que darán por resultado entonces este objetivo general.
3. Comunicación: la forma en que se comunique al personal lo que se va a realizar es fundamental, porque habitualmente los cambios generan dudas, temor y ansiedad. Por otro lado, nunca hay un exceso de información. Lo ideal es que haya informes frecuentes sobre la marcha del proyecto, y que se habiliten varios canales para hacer llegar las dudas

Fracaso

1. Fallas al definir el problema
2. Planificación basada en datos insuficientes
3. Planificación realizada por grupo de planificaciones
4. No hay seguimiento del plan de proyecto
5. Mala planificación de recursos
6. Estimaciones basadas en supuestos sin consultar datos históricos
7. Nadie estaba a cargo

UNIDAD 2: Gestión Lean-Ágil de Productos de Software

Metodologías agiles

El objetivo primordial es entregar software de forma frecuente en un entorno cambiante. Estas metodologías se utilizan en entornos con gran variabilidad de requerimientos, involucrando al cliente en el proceso de desarrollo para conseguir una rápida retroalimentación.
El desarrollo ágil adopta el ciclo de vida iterativo-incremental, donde el concepto del empirismo subyace al proceso. Esta es una de las diferencias con el PUD o el RUP, que, si bien utilizan un modelo de proceso iterativo-incremental, son procesos definidos.
Usualmente los incrementos se crean cada 2 o 3 semanas y se pone a disposición del cliente. El cliente se encuentra inmerso en el proceso, ofreciendo una rápida retroalimentación sobre los requerimientos cambiantes. Por otro lado, se reduce la cantidad de documentación con el uso de comunicaciones frecuentes e informales con el cliente en lugar de reuniones formales con documentos escritos.
Fowler define al enfoque ágil como un compromiso entre nada de proceso y demasiado proceso.
Los métodos agiles son adaptables en lugar de predictivos y están orientadas a la gente en lugar del proceso.

Manifiesto Ágil

Es un documento redactado en 2001 por 17 expertos de la ingeniería en software, el cual promueve la filosofía del desarrollo ágil del software.

Valores del Manifiesto Ágil

1. Individuos e interacciones por sobre procesos y herramientas: este valor se refiere a enfocarnos en que las personas estén motivadas en hacer su trabajo. Para enfocarnos en esto se debe generar un ambiente de respeto, comunicación abierta, transparencia y compromiso, donde los procesos y las herramientas sean un complemento que mejoran la eficiencia, pero sin personas con conocimiento técnico y actitud adecuada, no se producen resultados.
2. Software funcionando por sobre documentación detallada: poder anticipar cómo será el funcionamiento del producto final, observando incrementos previos provee un "feedback" estimulante, que genera ideas imposibles de concebir en un primer momento, y difícilmente se podrían incluir al redactar un documento de requisitos detallado en el comienzo del proyecto.
3. Colaboración por sobre negociación con el cliente: comprometer al cliente como parte importante del producto que se está construyendo. No se trata de tener a un cliente lejano que no sabe lo que sucede durante el desarrollo, si no tener a uno que este comprometido con cada entrega, sobre todo al momento de probarlo y dar el feedback. Resulta por tanto más adecuada una relación de implicación y colaboración continua con el cliente, más que una contractual de delimitación de responsabilidades
4. Responder a cambios por sobre seguir un plan: para desarrollar productos de requisitos inestables, que tienen como factor inherente el cambio y la evolución rápida y continua, resulta mucho más valiosa la capacidad de respuesta que la de seguimiento y aseguramiento de planes. Los principales valores de la gestión ágil son la inspección y la adaptación, diferentes a los de la gestión de proyectos tradicional: planificación y control que evite desviaciones del plan.

Principios agiles

1. La prioridad es satisfacer al cliente a través de entregas tempranas y frecuentes de software valioso. V2.
2. Recibir cambios de requerimientos, aun en etapas finales. V3, V4.

3. Entregas frecuentes (2 semanas a un mes). V2.
4. Técnicos y no técnicos trabajando juntos durante todo el proyecto. V1.
5. Hacer proyectos con individuos motivados. V1.
6. El medio de comunicación por excelencia es cara a cara. V1.
7. La mejor métrica de progreso es la cantidad de software funcionando. V2.
8. El ritmo de desarrollo es sostenible en el tiempo. V2, V4.
9. Atención continua a la excelencia técnica. V1.
10. Simplicidad – Maximizar el trabajo no hecho. V2, V3.
11. Las mejores arquitecturas, diseños y requerimientos surgen de equipos auto organizados. V1.
12. A intervalos regulares, el equipo evalúa su desempeño y ajusta la manera de trabajar. V1, V4.

¿Cuándo es aplicable Agile?

En función del grafico de áreas acerca del entendimiento y certeza que se tiene sobre los problemas, decimos que las metodologías agiles son aplicables en el contexto de lo complejo, en el cual se ubican los nuevos productos sobre los cuales el entendimiento y la certeza que se tiene es parcial.
La investigación actúa en el campo de la "anarquía", donde existe una gran incertidumbre.
Usualmente y no es una regla general, el mantenimiento se encuentra en el campo de lo relativamente simple.

Filosofía Lean

Lean es una filosofía de operación que se basa en el mejoramiento continuo a través de la eliminación de desperdicios en todos los procesos en una empresa; básicamente busca reducir el tiempo entre la demanda del cliente y su satisfacción eliminando desperdicios dentro del sistema.
La filosofía "Lean", conduce a una visión integrada de la cultura y la estrategia para atender al cliente final con alta calidad, bajo costo y tiempo de entrega, produciendo exactamente lo que el cliente final quiere, cuando lo quiere, donde lo quiere, a un costo mínimo y precio justo. Siendo el cliente final quien determina si el servicio o producto que la empresa entrega tiene valor o no.
Para esto utiliza el método Just in Time, el cual permite el intercambio del producto entre fases del proceso de desarrollo o a la entrega del producto final al cliente justo a tiempo, esto es poco antes de que lo usen y en la cantidad justa y necesaria.

Origen de la filosofía

Lean surge en la industria manufacturera, específicamente en la empresa automotriz Toyota. Su origen fue concebido para lidiar contra el desperdicio generado por la producción en masa, siendo un desperdicio cualquier actividad que consume recursos y no agrega valor.
La forma para eliminar el desperdicio consistía en prever la demanda, producir en función de esta y vender según se vaya construyendo.

Principios Lean

1. Eliminar desperdicio: eliminar del proceso y del producto todo aquello que consume recursos y no aporta valor al cliente. En la industria manufacturera un desperdicio sería el inventario, dado el peso que tiene el costo del capital inmovilizado sobre el costo total. En el software son las características extras, el trabajo parcial, el switcheo de tareas, etc. Se debe tener en cuenta que en el software aplica el principio de Pareto, el 20% del producto contiene el 80% del valor para el cliente. Se relaciona con los principios agiles de satisfacer al cliente con entregas tempranas y frecuentes, y el de simplicidad, maximizando el trabajo no hecho.

2. Amplificar el aprendizaje: crear y mantener una cultura de mejoramiento continuo, donde los individuos intercambien sus experiencias y conocimientos para contribuir con el aprendizaje colectivo. Crear conocimiento implica esperar que el diseño evolucione con la implementación del software y no definirlo previamente de forma exhaustiva. Por otro lado, todo conocimiento que se genere debe codificarse para que sea fácilmente accedido por toda la organización. Se relaciona con los principios agiles de recibir requerimientos aun en etapas finales y de técnicos y no técnicos trabajando juntos todo el proyecto.
3. Embeber la integridad conceptual: el cliente debe percibir el producto como algo coherente, donde los requerimientos que cubre la solución se observen como un todo cohesionado armónicamente. Hace falta de integridad técnica: una arquitectura coherente, usable, que responde a su cometido y a la que se puede dar mantenimiento, adaptar y ampliar. La integración de las personas hace al producto mas integro. La integridad percibida hace referencia a un balance entre requerimientos funcionales y no funcionales que agraden al cliente final. El objetivo es construir un producto de calidad desde el principio. Se relaciona con los principios agiles de las mejores arquitecturas y de atención a la excelencia técnica.
4. Diferir compromisos: tomar las decisiones en el ultimo momento responsable. Las decisiones tempranas son tomadas en un contexto de mayor incertidumbre. Esta asociado al principio de amplificar el aprendizaje, mientras mas tarde decidamos (en un marco de responsabilidad), mas conocimiento tenemos. Se debe tratar de tomar decisiones reversibles, de forma tal que pueda ser fácilmente modificable. Las mejores estrategias de diseño de software están basadas en dejar abiertas opciones de forma tal que las decisiones irreversibles se tomen lo más tarde posible. Se relaciona con los principios agiles de maximizar el trabajo no hecho y de recibir cambios de requerimientos aun en etapas finales.
5. Dar poder al equipo: dotar a aquellos en los que reside el conocimiento y realizan el trabajo del liderazgo suficiente para tomar decisiones y realizarlo, evitando pasos adicionales de aceptación a otras instancias que obstaculizan el flujo normal de actividad. Fomentar la buena ética laboral y delegar decisiones y responsabilidades del producto al nivel mas bajo posible. Se relaciona con los principios agiles de las mejores arquitecturas y de mantener a los individuos motivados.
6. Ver el todo: tener un visión holística, que permita asociar y comprender el todo, el producto, el valor agregado que hay detrás, el servicio que brinda el complemento de los productos.
7. Entregar lo antes posible:
    a. Entregar rápido: estabilizar ambientes de trabajo a su capacidad mas eficiente y acotar ciclos de desarrollo
    b. Entregar rápidamente: el producto se va transformando a través de incrementos pequeños de valor. Salir pronto al mercado al llegar a un producto mínimo que sea valioso.

   Se relaciona con el principio ágil de entregas tempranas y frecuentes de software de valor para el cliente.

Relación Lean-Ágil

Lean es previo al Manifiesto Ágil. Algunos principios de Agile heredan de los fundamentos de Lean. Ambos están orientados al cliente, a proveerle el máximo valor posible. Para esto, entienden que los individuos de los equipos deben estar motivados, de forma tal que la sinergia del mismo facilite el desarrollo del proyecto. Esta sinergia, se interpreta como un valor agregado para el cliente. La flexibilidad para adaptarse a los cambios en pos de ofrecerle valor al cliente, es también un denominador común de ambos enfoques. Otro aspecto que tienen en común es generar productos de calidad y de mejora continua.

Requerimientos en ambientes agiles

Introducción

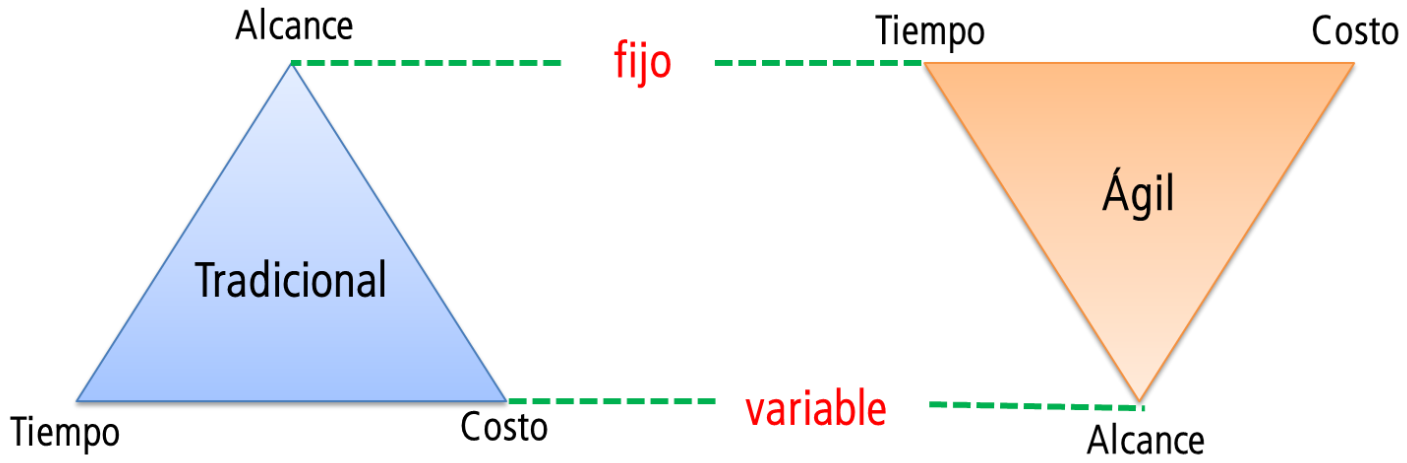
Lo mas difícil del software es decidir que software queremos construir, ya que las cuestiones tecnológicas no son el impedimento, sino que los proyectos de software fracasan debido a que no se cumplen con las expectativas del cliente: lo que se construye no es lo que el usuario quería o necesitaba. Este problema se advierte en la crisis del software. Probablemente una de estas razones es no asimilar la complejidad que supone hacer software y que la etapa de relevar requerimientos (la cual es social y no técnica) suele ser subestimada.

Fundamentación de los requerimientos agiles

El fondo de la cuestión esta en los principios y valores explicitados en el manifiesto ágil, fundamentalmente la entrega frecuente y temprana de software valioso y la posibilidad de recibir cambios de requerimientos aun en etapas finales. Esto ultimo implica una predisposición, una actitud del equipo frente a la posibilidad de cambios. La base de los requerimientos en ambientes agiles tiene que ver con la posibilidad de comenzar a construir el producto cuando el producto no esta completamente definido.

Cambio de requerimientos

En el enfoque tradicional, se intenta de que los cambios no ocurran.
Los requerimientos definen que es lo que hace el sistema, conforman el alcance del producto de software y el alcance es una de las dimensiones de cualquier proyecto de software.



En los enfoque tradicionales, el alcance se fija y en función de eso derivan los recursos (costo) y el tiempo.
El cliente desea controlar el alcance, ya que este comprende lo que el realmente necesita. Luego el cliente pregunta por el tiempo y el costo, los cuales son derivados por el equipo. Por esta razón es tan costosa incorporar cambios de requerimientos, ya que se estaría modificando el alcance y como consecuencia los recursos y el tiempo.
El enfoque ágil acepta que los requerimientos cambian y deciden fijar el tiempo y los recursos. De esta manera, el alcance es una dimensión variable.

Requerimientos Just in Time

La gestión ágil de requerimientos sostiene los requerimientos justo a tiempo, asociados a los principios de eliminar el desperdicio y diferir compromisos, dentro de la filosofía Lean.
Se empieza con una visión del producto y con algunos requerimientos identificados. A partir de esta base, se comienza a construir el producto y se incorpora al cliente para obtener retroalimentación.

En el enfoque tradicional se aplica el up front specification, mediante el cual se especifica todos los requerimientos antes de comenzar el desarrollo. Esto implica que un error en la captura de requerimientos tiene un alto impacto. En cambio, los requerimientos en ambientes agiles al ser consensuado continuamente con el cliente, el impacto que se tiene es menor.

Fuente de requerimientos

Los requerimientos agiles, basados en el principio de que el mejor medio de comunicación es el cara a cara, promueven que los requerimientos del software se obtienen conversando cara a cara con el cliente.
En los ambientes tradicionales, los requerimientos se encuentran especificados en un documento de especificación de requerimientos y el cliente luego lo lee para validarlo.
Los agilistas no escriben una ERS, pero lo compensan con la disponibilidad del cliente para resolver las dudas. Esto permite reducir la cantidad de documentación formal. Elicitar los requerimientos es un proceso que requiere de conversar de forma personal, analizando lo que los cliente dicen y los gestos que manifiestan.
El desarrollo ágil no implica la no documentación o la no especificación de requerimientos, sino que prioriza documentar y especificar aquellos que sea de valor para el cliente.

Momento de la captura de requerimientos

En el enfoque tradicional, los requerimientos son definidos al principio del proyecto. El caso particular del PUD, que es un proceso definido con un modelo de proceso iterativo e incremental, el flujo de trabajo de requerimientos tiene un gran peso en la fase inicial y de elaboración.
En el enfoque ágil, los requerimientos son relevadas continuamente durante todo el proyecto. El inicio de la construcción del software se da en un contexto en el cual los requerimientos no están completamente definidos, sin embargo, se tiene una visión.

Contenedor y persistencia de los requerimientos

El enfoque tradicional, la ERS contiene los requerimientos a lo largo del ciclo de vida del producto todo el tiempo. En caso de que surjan cambios, es necesario actualizar la ERS. En definitiva, es el documento de mas alto nivel que define las características del sistema.
El enfoque ágil, se utiliza al Product Backlog como un contenedor transitorio de los requerimientos, ya que una vez implementados estos se “persisten” en el producto funcionando. El detalle de valor queda persistente en los casos de prueba que se construyen en el testing.

User Stories

Definición

Son una técnica para trabajar requerimientos en ambientes agiles. La US contienen una descripción corta de una funcionalidad que se espera del producto valuada por un usuario del sistema.
Funciona como un recordatorio para el equipo de desarrollo y el Product Owner de que tienen que conversar acerca de la necesidad en términos de negocio para la construcción de una característica del producto que aporte valor. En otras palabras, es un mecanismo para diferir una conversación.
Es un requerimiento a nivel de usuario, no al nivel de sistema, por lo que se encuentra a un nivel de abstracción mas elevado. Es un ítem de planificación que no acompaña al producto durante todo el ciclo de vida. Para eso esta el manual de usuario.
Las US son escritas por el Product Owner.

Partes de la User

Tarjeta

Es una descripción de la historia, utilizada para planificar y como recordatorio. Tiene una sintaxis particular que es la siguiente:

Como <<**rol de usuario**>>
yo puedo <<**actividad**>>
de forma tal que <<**valor de negocio**>>

1. Rol de usuario: representa quien esta realizando la acción o quien recibe el valor de la actividad.
2. Actividad: representa la acción que realizara el sistema.
3. Valor de negocio: es la mas importante y representa el por que es necesaria la actividad. Esta justifica la razón de que el Product Owner escriba las US. Permite priorizar el desarrollo.

Conversación

Es la parte mas importante de la US, ya que explicita la comunicación entre el Product Owner y el Equipo de desarrollo que permite compensar la falta de especificación y detalle. El acuerdo por sobre la negociación del contrato y el principio de técnicos y no técnicos trabajando juntos durante todo el proyecto están asociadas a esta parte de la US. El Product Owner es parte del equipo, por lo cual se espera responsabilidad, compromiso y confianza de su parte para hacerse cargo y llegar a un acuerdo común con el equipo sobre lo definido acerca de la funcionalidad.

También es posible dejar la conversación persistente en grabaciones, minutas o softwares de gestión que posibiliten la conversación. De todas maneras, la conversación se materializa de forma distribuida en la tarjeta y en las pruebas de aceptación.

Confirmación

Definición de un acuerdo entre el Product Owner y el Equipo para demostrar que la característica del producto realmente se implemento. Conforma a la definición de las pruebas de aceptación. El PO decide si las pruebas de aceptación son validas.

Características

Relación con la arquitectura en capas

La User es una porción vertical, la cual abarca todas las capas de arquitectura: capa de datos, lógica de negocio y capa de presentación. Esto permite entregar una característica terminada al cliente.

Niveles de granularidad

1. User Story: cumple con el criterio de Ready para ingresar a una iteración.
2. Épica: es una User muy grande. El tamaño es relativo a si entra en una iteración o no, ya que las User se empiezan y se terminan en la iteración. Si la User no entra en la iteración es considerado una épica. Tienen la misma sintaxis que la User.
3. Temas: agrupación de varias Users y/o épicas. Es un agrupador de características de producto, como si fuese un modulo funcional o no funcional como la seguridad. Gestión de alumnos es un ejemplo.

Modelado de roles

El tema critico de la funcionalidad esta relacionado con quien va a utilizar dicha funcionalidad. Por esta razón existe una tendencia por ocuparse de quien va a usar el sistema, especificando los roles de usuario donde definimos las características del usuario respecto al producto y en relación con su perfil genérico.

Técnicas adicionales

1. Personas: esta técnica busca ser mas especifico acerca de las descripción de la persona concreta que va a utilizar el sistema. Este análisis es exhaustivo y por lo tanto complejo. Si los roles son clases, las personas serian los objetos.
2. Personajes extremos: que personas en términos extremos podrían llegar a utilizar el software.

Criterios de aceptación

Es una nota no obligatoria que define un acuerdo respecto a como se debe comportar el software para que el PO acepte la implementación. Lo fundamental es que este escrito en términos objetivos y concretos.

Pruebas de aceptación

Son declaraciones de intención de que hay que probar. El conocimiento del negocio está inmerso en las pruebas de aceptación. Se prueban escenarios exitosos y escenarios que fallen. En las pruebas de aceptación se encuentran las mas importantes, ya que no es posible escribir todas las posibles pruebas de aceptación.
El Product Owner acepta la User como implementada si todas las pruebas de aceptación se cumplen.

Definición de Ready

Determina si la User Story esta en condiciones de entrar a una iteración de desarrollo. Si cumple entonces puede, de lo contrario se debe trabajar para cumplir con el criterio. Mínimamente debe satisface el INVEST Model.

INVEST Model

Es un modelo que nos ayuda a verificar si la User cuenta con las características de la calidad mínima para satisfacer la definición de Ready y poder ingresar en una iteración de desarrollo.

1. Independiente: la user es calendarizable e implementable en cualquier orden. Esto implica que no existe una dependencia con otra user para desarrollarla o para mostrarla al Product Owner.
2. Negociable: significa que el cliente debe expresar lo que necesita, no el como. El como es decisión del equipo y debe ser negociable con el Product Owner.
3. Valuable: la user debe contener el valor de negocio para el cliente.
4. Estimable: la user debe contener la cantidad de información suficiente para estimar el tamaño, la complejidad y el esfuerzo necesario para realizarlo. Sin poder estimarla no es posible determinar si se puede finalizar en una iteración por lo tanto no puede ser construida.
5. Pequeña: la user debe ser lo sufrientemente pequeña como para ser finalizada en una iteración.
6. Testeable: la user debe contener las pruebas de aceptación para poder probar si la misma fue o no implementada

Definición de Done

Determina si la User esta terminada. Esto significa que la User esta en condiciones de mostrarse al Product Owner, es decir que esta implementada, testeada y documentada. Es una construcción colectiva que hace el equipo.

Spike

Son un tipo especial de Users que se producen por la incertidumbre que la misma presenta, la cual imposibilita que pueda ser estimada y por lo tanto no cumple con la definición de Ready. Es una característica mas del producto a la cual el equipo le tiene que dedicar tiempo para:

1. Familiarizarse con una nueva tecnología o dominio.

2. Investigar y prototipar para ganar confianza frente a:
    a. Riesgos tecnológicos.
    b. Riesgos funcionales, donde no esta claro como debe reaccionar el sistema para satisfacer las necesidades del usuario.

Una vez resuelta la incertidumbre, la spike se convierte en una o mas Users.

Spikes técnicas

Utilizadas para evaluar diferentes enfoques técnicos y tecnológicos en el dominio de la solución. Esta asociado a la implementación de la funcionalidad. Nueva tecnología.

Spikes funcionales

Utilizada cuando la incertidumbre es sobre la información del negocio o sobre como interactuara el usuario con el sistema. AFIP.

Product Backlog

Lista priorizada de las características del producto. Dentro del Product Backlog se encuentran los Ítems de Product Backlog. El dueño es el Product Owner y el decide como priorizar, sabiendo que mas alta implica mas prioridad.

Los ítems pueden tener distintos niveles de granularidad, el cual es inversamente proporcional al detalle. La granularidad fina es mucho detalle y la gruesa es poco detalle. La altura de un ítem en el Product Backlog es proporcional al detalle que se tenga sobre el mismo, por lo tanto, los ítems de granularidad fina se encuentran en la cima y están en condiciones de ser construido.

Los ítems de granularidad gruesa se encuentran abajo en la pila, se deben dividir en piezas mas pequeñas.

Estimaciones

Definición

Estimar es el proceso de obtener una aproximación sobre una medida con el objetivo predecir la completitud y gestionar los riesgos en el contexto de un proyecto.

Según McConell una estimación en el contexto del software es una predicción de cuanto tiempo durara o costara un proyecto. Es considerada una de las actividades mas complejas en el desarrollo de software.

Las estimaciones no son planes ni compromisos, sino que sirven de base para planificar, pero el plan final no tiene por que coincidir con lo estimado.

El concepto de estimación proviene del campo de probabilidad y estadística, siendo formalmente el proceso de obtener un estadígrafo de una muestra que estima el parámetro poblacional correspondiente. En este orden de ideas, a mayor diferencia entre lo estimado y lo planeado, mayor es el riesgo.

Proceso de estimación

Inicio con la especificación de requerimientos

En primer lugar, es necesario tener claro el alcance del proyecto, el cual contiene las funcionalidades que el cliente espera del producto resultante.

Estimar el tamaño

Una vez definido el alcance, lo primero que se estima es el tamaño, el cual define que tan grande y complejo es el producto a desarrollar. Suele ser uno de los factores mas difíciles de estimar. Usualmente en este caso se estima la cantidad de líneas de código, de clases, de paginas web, de casos de uso, etc. Responde el que.

Estimar el esfuerzo

Una vez estimado el tamaño, se estiman las horas personas lineales que se necesitan para construir el producto con la funcionalidad esperada por el cliente. Cuantas horas necesita una persona para construir una característica de producto de tal tamaño, asumiendo que esta solo y que no hay solapamiento de tareas. De alguna manera, responde el como.

Estimar el tiempo (Calendario)

Estimado el esfuerzo, se estima el tiempo. Consiste en estimar cuanto tiempo de calendario se necesita para construir el producto. Se establecen fechas y se determina que tareas pueden hacerse en paralelo para reducir los tiempos. Esta estimación esta relacionada a cuantas personas van a hacer, cuanto es el tiempo efectivo de trabajo por día, que cosas se pueden hacer en paralelo y que otras son dependientes. Se tienen en cuenta los trabajos adicionales, para estimar con un delta que permita solventar esas tareas adicionales. Responde el cuando.

Estimación de recursos y costo

Se estiman los recursos técnicos, logísticos y de infraestructura. Finalmente se estiman los costos, que se corresponde con el monto total de dinero que se requiere para llevar a cabo el proyecto. El costo mas significativo es el esfuerzo de los recursos humanos. Responde el cuanto

Errores frecuentes

Suelen confundir el esfuerzo con el tamaño. El problema, es el que el esfuerzo es muy personal. Otro error frecuente es confundir el esfuerzo con el tiempo.

¿Cómo mejorar las estimaciones?

Demorar la estimación hasta mas tarde en el proyecto

Esta opción en la mayoría de los casos no es práctica, porque el cliente requiere en el inicio del proyecto un conocimiento del costo y el tiempo que llevará la realización proyecto.

Basar las estimaciones en proyecto similares que hayan sido completados

Esta alternativa es poco viable, dado que los proyectos de software son únicos y por mas similares que sean dos proyectos siempre habrá diferencias respecto al equipo de personas que lo integra y al cliente final.

Emplear técnicas de descomposición o modelos empíricos para la estimación de esfuerzo y costo

Estas son opciones viables.

Contar como técnica fundamental para estimar

Se cuenta aquello que este relacionado con el tamaño del software a desarrollar. Lo que se cuenta debe requerir poco esfuerzo.

1. En etapas tempranas, contamos requerimientos, características, casos de uso, user stories, etc.
2. En la mitad del proyecto, contamos pedidos de cambios, cantidad de pantallas, tablas de base de datos, etc.
3. Mas avanzado en el proyecto, contamos clases, defectos, tareas, etc.

Métodos utilizados para estimar

Se recomienda utilizar diferentes método de estimación y luego contrastar.

Basados en la experiencia

1. Datos históricos: consiste en recolectar datos básicos de otros proyectos para ir generando una base de conocimientos que sea de utilidad para futuras estimaciones.

2. Juicio experto: es uno de los mas utilizados y existen dos técnicas para aplicarlo:
    a. Juicio experto puro: un experto estudia las especificaciones y realiza una estimación. Esta estimación se basa fundamentalmente en la experiencia del experto.
    b. Juicio wideband Delphi: es similar al anterior pero grupal, un grupo de personas se reúne con el objetivo de converger a una estimación común.

Analogía

Se comparan los factores a estimar de forma relativa con respecto a las estimaciones de otros proyectos similares. Es un método que tiene mucho error dado que el conocimiento en un proyecto no es extrapolable a otro.

Otros

Existen otro métodos como:

1. Basados exclusivamente en los recursos: consiste en analizar el personal que se tiene y de cuanto tiempo se dispone del mismo, haciendo las estimaciones exclusivamente en función de este factor.
2. Basados en la capacidad del cliente o exclusivamente en el mercado: ponen el foco en el cliente y realizan las estimaciones en función de su capacidad de pago.
3. Basados en los componentes del producto: se pueden aplicar dos enfoques
    a. Bottom-up: se descompone el producto en unidades lo mas pequeñas posible. Se estima cada unidad y luego se hace la estimación total
    b. Top-down: se descompone el producto en grandes bloques y luego se estima cada uno.
4. Métodos algorítmicos: se ejecutan algoritmos que tienen como parámetros de entrada medidas cuantitativas de tamaño, complejidad y conocimiento de dominio. Cada variable tendrá asociada un factor multiplicativo y en función de esto se obtiene como resultado un valor de esfuerzo.

Estimaciones en ambientes agiles

Diferencias con las estimaciones en ambientes tradicionales

Respecto a las tres dimensiones

En Agile se tiene fijo el tiempo (lo que dura la iteración) y los recursos afectados para trabajar, dejando variable el alcance. Por lo tanto, al estimar en estos ambientes se responde a cuanto se puede comprometer el equipo, que cantidad de las características de producto se puede construir en la iteración. Se entiende, además, que en estos ambientes el producto no esta completamente definido, sino que se tiene una visión de mismo.

En los ambientes tradicionales, el alcance es lo que esta fijo y el producto se encuentra definido. A partir de esto, se deriva el resto de las variables-. Por esta razón en los procesos definidos se estima primero el tamaño, ya que es la base para estimar el resto.

Respecto a la precisión

Las estimaciones en los ambientes tradicionales buscan ser mas precisas, debido a que son las bases para luego planificar.

Dado que la precisión es realmente cara, la precisión en los ambientes agiles y en correspondencia con la filosofía Lean, es considerada un desperdicio.

Principal diferencia

Las estimaciones agiles son relativas, a diferencia de las estimaciones en ambientes tradicionales que son absolutas.

En Agile, las características del producto son estimadas utilizando una medida de tamaño relativa dado que las personas no somos buenas realizando estimaciones absoluta, sino que es mas fácil y rápido comparar medidas. Por otra parte, la practica de estimar mediante comparaciones permite obtener una mejor dinámica grupal y pensamiento de equipo por sobre el individuo.

Quien estima

En las metodologías de estimación tradicional, las estimaciones las realiza el líder del proyecto o en ocasiones las realiza un experto, por lo que su estimación no es representativa de la cantidad de trabajo que requiere una persona del equipo para terminar una funcionalidad del producto. Esta es una de las razones principales de los desvíos en el proyectos de software tradicional.
En los ambientes agiles y en concordancia con los valores y principios explicitados en el manifiesto ágil, es el equipo el que realiza la estimación. Lo optimo es que cada uno estime su propio trabajo y de esta manera, es posible disminuir la probabilidad de desviaciones.

Confusión con la planificación

En los ambientes tradicionales existe una gran error que es asumir las estimaciones como compromisos, como si fueran planificaciones. Esto implica que las estimaciones se transforman en promesas.
Mientras que en agile, asumen que las estimaciones están inmersas en el campo de las probabilidades y por lo tanto no implican un compromiso. Una falla en la estimación no implica extender el tiempo, ya que en agiles trabajamos con el tiempo fijo, por lo tanto, lo que se ajusta es el alcance.

Momentos para estimar

En el enfoque tradicional se estima al comienzo del proyecto, luego de la especificación de requerimientos y luego del diseño. Realmente se estima cada vez que se modifica el alcance, ya que es la variable que se mantiene fija y de la cual se deriva el resto.
Las estimaciones a lo largo del proyecto se van haciendo cada vez mas certeras. Al comienzo, pueden diferir hasta un 400%, esto se debe a:

1. Al comienzo tenemos poca información y alta incertidumbre, por lo que aumenta el riesgo de estimar erróneamente.
2. Las estimaciones son demasiado optimistas.
3. Decimos que aquel que no saber estimar implica estimar muchas veces. Lo mismo ocurre con planear.

En agiles se estima al comienzo de cada sprint.

Tamaño

Dado que el tamaño es lo que se compara, lo definimos como la medida de cuan compleja y grande es una user. También el tamaño es función de la incertidumbre de dicha user, definida la incertidumbre como la falta de información.
El tamaño no es lo mismo que el esfuerzo, ya que este ultimo indica las horas humano lineal requeridas para el desarrollo de la user. Por lo tanto, una user de mismo tamaño puede implicar esfuerzos diferentes en función de la persona que se este evaluando, debido a que el esfuerzo varia dependiendo del habilidades, conocimientos, experiencia, familiaridad con el dominio, etc.

Formas de estimar el tamaño

Existen diferentes escalas para estimar el tamaño y siempre son por comparación, entre ellas:

1. Numero del 1 al 10.

2. Talles de remeras: S, M, L: suele ser utilizado como escala de estimación para los ítems del Product Backlog.
3. Serie exponencial de base 2.
4. Serie de Fibonacci

Una vez definida la escala, no se cambia. En caso de cambiarla, también se debe cambiar el patrón de medición.

Story Point

Es una unidad de estimación que especifica la complejidad, incertidumbre y esfuerzo propio del equipo respecto a una user story en términos relativos respecto a otra user story.

Complejidad

Esta relacionado a la cantidad de partes y de relaciones entre la mismas que posee la user.

Esfuerzo

Horas persona lineales que se requieren para terminar la user.

Incertidumbre

Grado de duda o falta de información para el desarrollo de la user. Puede ser técnica o funcional.

¿Para que estimamos?

1. En un primer momento para determinar cuantas Users se puede comprometer el equipo de trabajo a terminar en una iteración.
2. Al finalizar la iteración, contamos los puntos de historia correspondiente a las Users aceptadas por el Product Owner y obtenemos la velocidad del sprint, la cual es una de las métricas fundamentales en ambientes agiles. Sin embargo, es necesario aclarar que la velocidad no es una estimación sino una medición propiamente dicha. Si un equipo alcanza una velocidad sostenible en el tiempo, es decir que mantiene la velocidad en las diferentes iteraciones, permite que el equipo obtenga previsibilidad, la cual ayuda a estimar en las siguientes iteraciones.

Estimaciones a nivel del release

1. En primer lugar, se estiman las Users, asignado los story points correspondientes.
2. Luego, se estima la cantidad de Sprint necesarios para obtener el release de producto que agrupa determinadas características del producto a implementar. Si los sprint tienen la misma duración, entonces la release tendrá una duración estimada igual al producto de la duración del sprint por la cantidad de Sprints estimada. De esta manera podemos apreciar que existe un determinado nivel de certeza y no de precisión.

Poker estimation

Es una técnica de estimación en ambientes agiles publicada por Mike Cohn. Resulta de la conjunción de diferentes métodos como el de juicio experto, analogía y desagregación, basado en que 4 ojos ven mas que dos. En contraposición al método de juicio experto tradicional, asume que las personas que desarrollan el producto deben ser aquellas que estimen. En otras palabras, el equipo estima su propio trabajo y los miembros opinan sobre lo mismo, donde se mezcla la experiencia y el conocimiento de cada uno con la posibilidad de compartir con el resto del grupo.

Escala

Dado que la complejidad en el software incrementa de forma exponencial, se decide adoptar la serie de Fibonacci como escala para la determinación de los puntos de historia. Cada valor de la serie se calcula como la

sumatoria de los 2 valores de la serie que lo preceden, siendo 0 y 1 los valores de la serie para 0 y 1 respectivamente.

1. Valor 1 o 1/2: son los valores mas bajos y se aplican a una funcionalidad pequeña.
2. Valor 2- 3: funcionalidad pequeña a mediana.
3. Valor 5: funcionalidad media.
4. Valor 8: funcionalidad grande, donde nos preguntamos si es posible dividir.
5. Valor 13 o mas: necesariamente hay que dividir ya que no ingresa en el sprint.

Procedimiento

1. Se define una lista de Users stories.
2. Se acuerda cual de las anteriores es la que tiene menor grado de incertidumbre y se la define como canónica asignándole 1 o 2 story point. Esta User es utilizada como referencia para comparar. Los equipos avanzados podrían elegir canónicas de 3 SP, con el objetivo de tener un margen para hacia arriba y hacia abajo. Por esta razón, también el ½ se incluye en la serie.
3. Se ejecuta en wideband Delphi y se realiza la estimación relativa del resto de Users, comparándola con la user canónica elegida anteriormente. Los valores de story points deben pertenecer a la serie de Fibonacci, teniendo en cuenta que un valor superior a 8 implica que la user debe ser dividida en unidades mas pequeñas.
4. Utilizando juicio experto, cada integrante del equipo de desarrollo expone su valor de estimación de forma simultanea. En caso de que la estimación difiera, el mayor y el menor estimador justifican sus estimaciones para conocer sus opiniones. Luego el resto del medio aportan sus opiniones. Siempre la escucha es activa, para poder aprender.
5. En caso de que la primera vuelta existiese diferencias considerables, se realiza nuevamente la ejecución del wideband Delphi.
6. Si las estimaciones no convergen, entonces se llega a una estimación intermedia o promedio.

Consideraciones

1. Para realizar una estimación se debe tener en cuenta el criterio de Done y no solo el la tarea de programación, es decir todo lo que hay que hacer para terminar una user y poder lidiar con el problema de las actividades omitidas.
2. El tiempo para realizar las estimaciones, como cualquier otra actividad en los ambientes agiles, es time boxing.
3. Las user a estimar son las menos posibles, en orden con el principio de diferir compromisos.

Estimaciones en Lean

En Lean, el cual es mas extremo que las metodologías agiles, asume que las estimaciones son opcionales y en algunas ocasiones pueden convertirse en un desperdicio dado que tienen una alta probabilidad de no coincidir con la realidad, dado que cualquier variable que se modifica, modificara a la estimación también.

Framework Scrum

Visión general de Scrum

Scrum es un marco de trabajo, constituido por el equipo Scrum y sus roles, los eventos, los artefactos y las reglas asociadas que permiten relacionarlos, dentro del cual se pueden emplear diferentes procesos y técnicas. Scrum no es un proceso ni una técnica.

Scrum permite gestionar productos complejos, lo cual implica desarrollarlos, entregarlos y mantenerlos. Posibilita la mejora continua del producto, del equipo y del entorno de trabajo.

Usos de Scrum

1. Investigar e identificar mercados viables, tecnologías y capacidades de los productos
2. Desarrollar productos y mejoras
3. Liberar productos y mejoras los mas frecuente posible
4. Desarrollar y mantener productos en la nube
5. Mantener y renovar productos

Pilares

Scrum esta basado en la teoría de control de procesos empíricos, donde el conocimiento y las decisiones emergen de la experiencia y de lo conocido. Para el control de procesos empíricos, se sustenta en 3 pilares:

Transparencia

Los aspectos relevantes del proceso deben ser visibles para aquellos que son responsables del resultado. Se debe acordar un estándar para la definición de estos aspectos, lo cual posibilita compartir un entendimiento común. Por ejemplo: concepto de “Terminado”.

Inspección

Los usuarios de Scrum deben inspeccionar frecuentemente, sin interferir con el trabajo, los artefactos de Scrum y el progreso hacia el objetivo, con el fin de detectar eventuales desviaciones.

Adaptación

Si en la inspección se detecta que al menos un aspecto de un proceso se desvía de los limites aceptables, resultando un producto inaceptable, se debe adaptar o reajustar el proceso para evitar mayores desviaciones.
Las ceremonias dentro del sprint, permiten la inspección y adaptación.

Valores

El éxito de Scrum depende de que las personas asimilen los siguientes valores:

1. Compromiso: las personas se comprometen a alcanzar las metas del Equipo Scrum.
2. Coraje: los miembros del Equipo Scrum tienen coraje para hacer bien las cosas y para trabajar en problemas complejos.
3. Foco: todos se enfocan en el sprint y el objetivo del Equipo Scrum.
4. Apertura: el equipo y los interesados están abiertos a afrontar los desafíos que supone sus trabajos.
5. Respeto: Los miembros del Equipo Scrum se respetan entre si para ser capaces e independientes.

Equipo Scrum

Los equipos de Scrum son:

1. Autoorganizados: eligen la mejor forma de llevar a cabo su trabajo y no son dirigidos por personas externas al equipo.
2. Multifuncionales: tienen todas las competencias necesarias para llevar a cabo su trabajo sin depender de personas externas al equipo.

Los equipos de Scrum entregan productos iterativos e incrementales, aprovechando la retroalimentación del cliente. Las entregas incrementales de producto “Terminado” asegura que siempre existe una versión de producto útil y funcional, potencialmente entregable.

Product Owner

Es una única persona responsable de maximizar el valor del producto resultante del trabajo del Equipo de Desarrollo y de la gestión del Product Backlog, la cual incluye las siguientes tareas:

1. Definir claramente los elementos del Product Backlog.
2. Ordenar dicho elementos para alcanzar mejor los objetivos .
3. Optimizar el valor de trabajo que realiza el Equipo de Desarrollo.
4. Asegurar que el Product Backlog es visible, transparente y claro para todos, mostrando en que se trabajara a continuación.
5. Asegurar que el Equipo de Desarrollo entiende los elementos del Product Backlog en el nivel necesario.

El PO puede delegar estas tareas al Equipo de Desarrollo, pero el sigue siendo el responsable.
Debe ser un persona cuyas decisiones son respetadas dentro de la organización, la cual pueda modificar la prioridad de los elementos del Product Backlog.

Equipo de desarrollo

Conjunto de profesionales que tienen por trabajo la entrega de producto "Terminado", el cual puede ponerse en producción al final del sprint. Únicamente los miembros de este equipo participan de la creación del Incremento.
Sus características son:

1. Son autoorganizados: nadie le indica al Equipo de Desarrollo como convertir los elementos del Product Backlog en funcionalidad del producto.
2. Son multifuncionales: cuentan con todas las habilidades necesarias para crear un Incremento del producto.
3. No se reconocen títulos para sus miembros, independientemente del trabajo que realicen.
4. No se reconocen subequipos, independientemente de los dominios (Análisis, Diseño, Testing, etc.)
5. Los miembros pueden tener especialidades y áreas en las que estén mas enfocados, pero la responsabilidad recae en el equipo como un todo

Su tamaño es lo suficientemente pequeño para permanecer ágil y lo suficientemente grande para completar una cantidad de trabajo significativa. El numero debería variar entre 3 y 9, ya que:

- Tener menos de tres miembros reduce la interacción y se podrían encontrar limitaciones en cuanto a las habilidades necesarias durante el sprint.
- Tener mas de nueve miembros requiere demasiada coordinación, ya que supone una gran complejidad para la aplicación de un proceso empírico.

El PO y el Scrum Master no cuentan en el calculo a menos que realicen trabajo del Sprint Backlog.

Scrum Master

Es la persona responsable de promover y apoyar Scrum, ayudando a todos a entender la teoría, practicas, reglas y valores de Scrum. Es un proveedor de servicios para el Equipo Scrum y ayuda a externos del mismo a entender que interacciones con el Equipo pueden ser útiles y cuales no.

1. Servicios al Product Owner:
    a. Asegurar que los objetivos, el alcance y el dominio del producto sean comprendidos por todos.
    b. Encontrar técnicas para gestionar de forma efectiva el Product Backlog.
    c. Ayudar al Equipo Scrum a entender la necesidad de tener claro y conciso los elementos del Product Backlog.
    d. Entender la planificación del producto en un entorno empírico.

e. Asegurar que el Product Owner sepa como priorizar el Product Backlog para maximizar el valor.
f. Entender y practicar la agilidad.
g. Facilitar los eventos de Scrum según se requieran.

2. Servicios al Equipo de Desarrollo:
   a. Guiarlo a ser autoorganizado y multifuncional.
   b. Ayudarlo a crear producto de alto valor.
   c. Eliminar impedimentos para que el Equipo progrese.
   d. Facilitar los eventos de Scrum según se requieran.
   e. Guiarlo en entornos organizaciones donde aun Scrum no se haya adoptado o entendido.
3. Servicios a la Organización:
   a. Liderar y guiar en la adopción de Scrum.
   b. Planificar la implementación de Scrum en la Organización.
   c. Ayudar a los empleados e interesados en entender y aplicar Scrum y desarrollo empírico.
   d. Motivar cambios que incrementen la productividad del Equipo Scrum.
   e. Trabajar con otros Scrum Masters para mejorar la efectividad Scrum en la Organización.

Eventos

Son bloques de tiempo con duración máxima, que proveen transparencia y posibilitan la inspección y adaptación, además de minimizar reuniones no definidas en Scrum. El Sprint es el contenedor del resto de eventos, el cual no puede acortarse ni alargarse. El resto de los eventos pueden durar menos de lo definido.

Sprint

Es un bloque de tiempo de un mes o menos, en el cual se crea un Incremento del producto "Terminado" utilizable y potencialmente despegable. Es conveniente que la duración de los Sprint sea la misma a lo largo del desarrollo. Cada Sprint comienza inmediatamente luego de la finalización del Sprint anterior.
Puede considerarse como un proyecto con duración menos a un mes, el cual tiene una meta a alcanzar, un diseño y un plan flexible que guiara su construcción, el trabajo del equipo y el incremento resultante.
Durante el Sprint:

1. No se realizan cambios que puedan afectar al Sprint Goal.
2. Los objetivos de calidad no disminuyen.
3. El alcance puede clasificarse y negociarse con el Product Owner y con el Equipo de Desarrollo a medida que se va aprendiendo.

El Product Owner puede cancelar el Sprint si detecta que el Sprint Goal quedo obsoleto. Los elementos del Product Backlog que se hayan "Terminado", se revisan y el PO decide si al menos una parte es potencialmente entregable. Los elemento que no se hayan no completados se deben volver a estimar e introducir en el Product Backlog.
La cancelación de los Sprint consume muchos recursos, no es común y suele ser caótica.

Sprint Planning

Tiene una duración máxima de 8 horas donde el Equipo Scrum de forma colaborativa planifica el trabajo a realizar en el Sprint. La entrada a este evento esta constituida por el Product Backlog, el ultimo Incremento, la capacidad proyectada y la velocidad del ultimo sprint. Responde las siguientes preguntas:

1. ¿Qué puede hacerse en este Sprint? El Equipo Scrum define el Sprint Goal. El PO define que elementos del Product Backlog se deben implementar para cumplir con el Sprint Goal. El Equipo de Desarrollo define la cantidad de elementos a implementar a lo largo del Sprint.

2. ¿Cómo se realizará el trabajo seleccionado? El Equipo de Desarrollo decide como implementara los elementos seleccionados en el punto anterior para crear un Incremento de producto "Terminado" durante el Sprint. El conjunto de elementos a implementar y el plan para terminarlos recibe el nombre de Sprint Backlog. El trabajo a realizar es estimado por el Equipo de Desarrollo, mientras que el PO ayuda a clarificar los elementos seleccionados y puede hacer concesiones.

Al final de esta ceremonia, el Equipo de Desarrollo tiene que poder explicarle al Product Owner y al Scrum Master como pretende trabajar como equipo autoorganizado para cumplir con el Sprint Goal y construir un incremento de producto "Terminado".

Sprint Goal

Es la meta u objetivo a alcanzar durante el Sprint, el cual se logra con la terminación de los elementos del Sprint Backlog y proporciona una guía para el Equipo de Desarrollo durante el Sprint.

Sprint Daily

Tiene una duración máxima de 15 minutos, donde el Equipo de desarrollo todos los días, a la misma hora y en el mismo lugar se reúne para planificar como trabajara las siguientes 24, evaluando el progreso hacia el Sprint Goal y la tendencia de este progreso hacia la finalización del trabajo contenido en el Sprint Backlog.

El Equipo de Desarrollo dirige y define la estructura de la reunión, tratando temas como los siguientes:

1. ¿Qué hice ayer que ayudo al Equipo de Desarrollo a lograr el Sprint Goal?
2. ¿Qué hare hoy para ayudar al Equipo de Desarrollo a lograr el Sprint Goal?
3. ¿Existe algún impedimento que evite que el Equipo de Desarrollo o yo logremos el Sprint Goal?

Sprint Review

Tiene una duración máxima de 4 horas, donde el Equipo Scrum inspecciona el incremento y de ser necesario adapta el Product Backlog. Incluye los siguientes elementos:

1. Los asistentes son el Equipo Scrum y los interesados considerados por el Product Owner.
2. El Product Owner determina que elementos del Product Backlog se han "Terminado" y cuales no.
3. El Product Owner habla sobre el estado actual del Product Backlog. Proyecta objetivos probables y fechas de entrega basándose en el progreso obtenido (de ser necesario).
4. El Equipo de Desarrollo hace una demostración del trabajo "Terminado" y responde preguntas sobre el incremento.
5. El grupo completo colabora de que hacer a continuación y que sirva como entrada a la siguiente Sprint Planning.
6. Revisar si el mercado o el uso potencial del producto modifico aquello de mas valor para hacer a continuación.

La salida de esta reunión en un Product Backlog revisada y consistente para el siguiente Sprint.

Sprint Retrospective

Tiene una duración máxima de 3 horas y cuyo propósito es:

1. Inspeccionar como fue el ultimo Sprint en cuanto a personas, relaciones, procesos y herramientas.
2. Identificar y ordenar los elementos mas importantes que salieron bien y las posibles mejoras.
3. Crear un plan para implementar las mejoras que sea compatible con la forma en la que el Equipo Scrum trabaja.

Para el final de la Retrospectiva de Sprint el Equipo Scrum debería haber identificado mejoras que implementará en el próximo Sprint. El hecho de implementar estas mejoras en el siguiente Sprint constituye la adaptación subsecuente a la inspección del Equipo mismo. Aunque las mejoras pueden implementarse en cualquier

momento, la Retrospectiva de Sprint ofrece un evento dedicado para este fin, enfocado en la inspección y la adaptación.

Artefactos

Los artefactos de Scrum representan trabajo o valor en diversas formas que son útiles para proporcionar transparencia y oportunidades para la inspección y adaptación.

Product Backlog

Lista ordenada de todo lo que se conoce que es necesario en el producto, siendo la única fuente de requisitos para cambios en el producto. Siempre esta incompleta y es dinámica, identificando lo que el producto necesita para ser adecuado, competitivo y útil. Los cambios en los requisitos de negocio, las condiciones del mercado o la tecnología podrían causar cambios en el Product Backlog.

El Product Backlog enumera todas las características, funcionalidades, requisitos, mejoras y correcciones que constituyen cambios a realizarse sobre el producto para entregas futuras. Los elementos de esta lista tienen como atributos la descripción, el orden, la estimación y el valor. Usualmente incluyen las pruebas para demostrar que los elementos están "Terminados".

El refinamiento del Product Backlog es un acto opcional en Scrum, en el cual el Product Owner y el Equipo de Desarrollo agregan detalle, ajustan las estimaciones y orden a los elementos que lo constituyen.

La posición vertical de un elemento del Product Backlog es proporcional a la claridad y al detalle, e inversamente proporcional a la granularidad que se tiene del mismo. Los elementos de granularidad mas final que pueden ser "Terminados" en un Sprint, son considerados como "Listos" para entrar en el siguiente Sprint.

Sprint Backlog

Lista de elementos del Product Backlog seleccionados para el Sprint junto con el plan para cumplir con el Sprint Goal y entregar un incremento "Terminado" del producto.

El seguimiento del progreso del sprint se realiza en la Daily Scrum y consiste en sumar el trabajo restante total del Sprint Backlog para proyectar la posibilidad de cumplir con el Sprint Goal.

Incremento

Es la suma de todos los elementos del Product Backlog completados durante un Sprint y el valor de los incrementos de todos los Sprints anteriores.

Definition of "Done"

El Equipo Scrum define el concepto de "Terminado" para garantizar la transparencia. El criterio de "Terminado" permite determinar si un trabajo ha sido o no completado y es un estándar en el equipo. Si existen varios Equipos Scrum, entonces todos los Equipos de Desarrollo deben tener el mismo concepto de "Terminado".

Framework Nexus

Necesidad de escalar

Dado que Scrum impone un limite en la cantidad máxima de integrantes del Equipo de Desarrollo, surge la necesidad de escalar este Framework para poder gestionar productos de mayor envergadura, ya que las practicas agiles se utilizan en ambientes complejos para reducir tal complejidad.

Existen diferentes frameworks para escalar como Nexus, scale scrum, less, SAFe.

Visión de Nexus

Nexus es un Framework que consiste en roles, eventos, artefactos y técnicas que vinculan el trabajo de aproximadamente tres a nueve equipos Scrum que trabaja sobre un Product Backlog común para la construcción de un incremento integrado de producto "Terminado".

Nexus surge para lidiar con la complejidad que supone varios Equipos Scrum trabajando sobre un mismo Product Backlog. Esta complejidad esta dada por las siguientes dependencias entre equipos:

1. Requerimientos: el alcance de los mismos puede superponerse. La forma en que se implementan también puede afectar a los demás.
2. Conocimiento del dominio: el conocimiento del sistema de negocio debería mapearse a los Equipos Scrum para minimizar las interrupciones entre los mismos durante el Sprint.

Equipo de Integración Nexus

Es responsable de asegurar que se produzca al menos un Incremento Integrado "Terminado" de producto en cada Sprint.

Los miembros del Equipo de Integración Nexus también pueden ser miembros de algún Equipo Scrum de ese Nexus. La membrecía al Equipo de Integración Nexus tiene precedencia sobre la membrecía a Equipos Scrum individuales.

Product Owner en el Equipo de Integración Nexus

Como existe una único Product Backlog, el Product Owner es único y es responsable de maximizar el valor del producto y del trabajo ejecutado e integrado por los Equipos Scrum.

Miembros del Equipo de Integración Nexus

Estos miembros son profesionales de software con habilidades en el uso de herramientas y practicas en el campo de la ingeniería de sistemas. Son responsables de que los Equipos Scrum las entiendan e implementen para detectar dependencias e integrar frecuentemente los artefactos hasta una definición de "Terminado".

Enseñan a los Equipos Scrum sobre los estándar de desarrollo, infraestructura y arquitectura, para el desarrollo de productos de calidad.

Si la responsabilidad primaria se satisface, pueden ser miembros del Equipo de Desarrollo de uno o mas Equipos Scrum.

Scrum Master en el Equipo de Integración Nexus

El Scrum Master en el Equipo de Integración Nexus tiene la responsabilidad general de asegurar que el framework Nexus se entienda y se divulgue. Este Scrum Master también puede ser un Scrum Master en uno o más de los Equipos Scrum en ese Nexus.

Eventos de Nexus

Son bloques de tiempo adicionales a sus eventos correspondientes en Scrum.

Refinamiento del Product Backlog

Este evento se realiza si es necesario para detallar un Product Backlog con dependencias e incertidumbre. Este se realiza hasta que los elementos del Product Backlog sean lo suficientemente independientes para que un Equipo Scrum pueda trabajar con ellos sin generar conflicto.

Nexus Sprint Planning

El propósito es coordinar todas las actividades de todos los Equipos Scrum de un Nexus durante un Sprint. El Product Owner ayuda en el conocimiento del dominio y guía las decisiones de selección y priorización de los elementos del Product Backlog. El PO también discute el Nexus Sprint Goal.

Una vez entendido el trabajo del Nexus, esta planificación continua en el Sprint Planning de cada Equipo Scrum. La Nexus Sprint Planning finaliza cuando finalizan todos los Sprint Planning de cada Equipo Scrum.
La salida consiste en el Nexus Sprint Goal, al cual están alineados cada uno de los Sprint Goal de los Equipos Scrum, y el Nexus Sprint Backlog que es una composición de los Sprint Backlog de cada Equipo Scrum.

Nexus Sprint Goal

Es la suma del trabajo y de los Sprint Goal de todos los Equipos Scrum del Nexus. Este se logra desarrollando todos los elementos del Nexus Sprint Backlog.

Nexus Sprint Daily

Evento en el cual representantes apropiados de cada Equipo de Desarrollo Scrum inspeccionan el incremento integrado y el avance hacia el Nexus Sprint Goal, e identifica problemas de integración o dependencias entre equipos. Se responden preguntas como:

1. ¿El trabajo del día anterior se integro correctamente? Si no es así ¿Por qué no?
2. ¿Cuales nuevas dependencias han sido identificadas?
3. ¿Qué información nueva debe compartirse entre los Equipos Scrum del Nexus?

Luego, los Equipos de Desarrollo toman la salida de esta reunión para realizar su propio Sprint Daily.

Nexus Sprint Review

Se realiza al final del Sprint y proporciona retroalimentación sobre el Incremento Integrado construido y para adaptar el Product Backlog si es necesario.
La Nexus Sprint Review reemplaza las Sprint Review de cada Scrum, ya que el objetivo es revisar el incremento Integrado y no individual.

Nexus Sprint Retrospective

Este evento es una oportunidad formal para que el Nexus se inspeccione, adapte y cree un plan de mejoramiento que se ejecute a partir del próximo Sprint para asegurar así el mejoramiento continuo. Tiene 3 partes:

1. Identificación de problemas que hayan afectado a mas de un Equipo Scrum.
2. Cada Equipo Scrum realiza la Sprint Retrospective.
3. Representantes apropiados de cada Equipo Scrum se reúnen nuevamente y acuerdan como adaptar la forma de trabajo y aplicar las mejoras a partir de los problemas identificados.

Artefactos de Nexus

Los artefactos de Nexus representan trabajo o valor en diversas formas que son útiles para proporcionar transparencia y oportunidades para la inspección y adaptación.

Product Backlog

El Product Backlog es único. Sin embargo, en Nexus debe considerarse las dependencias entre los elementos que pueden generar dependencias entre los diferentes Equipos Scrum.

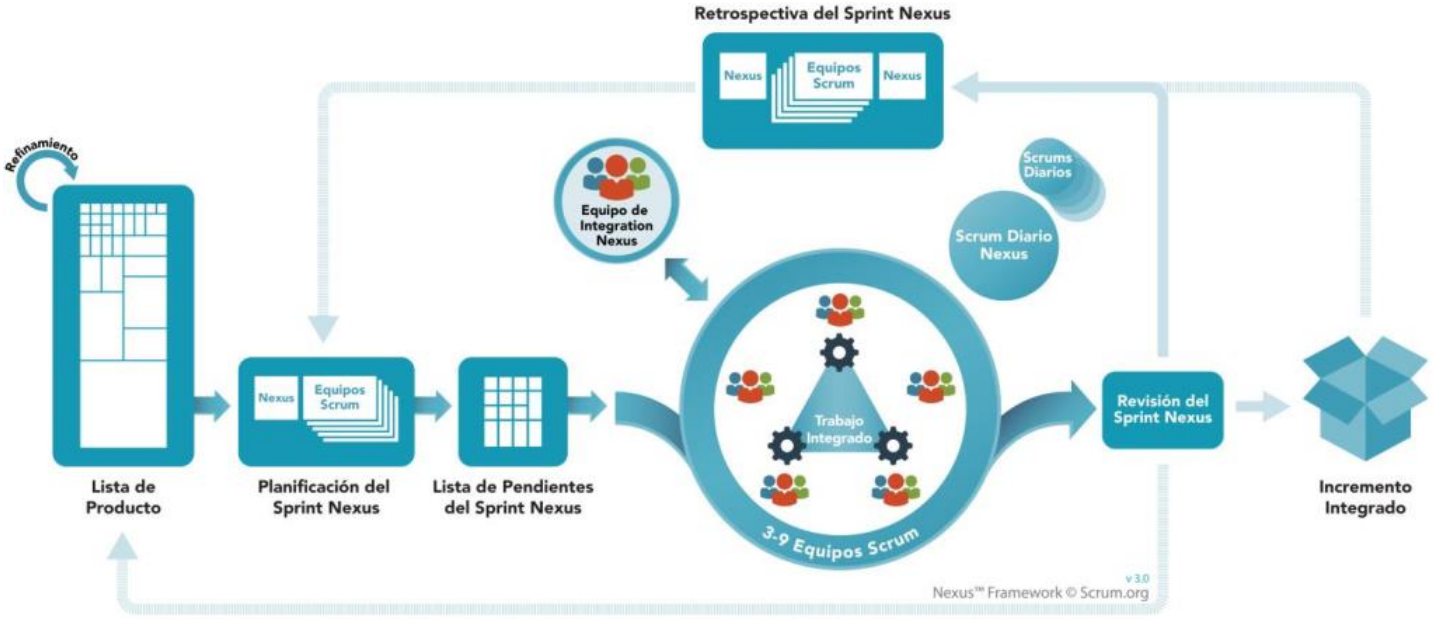
Nexus Sprint Backlog

Es la unión de todos los elementos de los Sprint Backlog de todos los Equipos Scrum. Permite visualizar las dependencias y el flujo de trabajo en el Sprint. Se actualiza al menos una vez al día.

Incremento Integrado

Es la suma de todos los elementos del Product Backlog completados e integrados durante un Sprint y el valor de los incrementos integrados de todos los Sprints anteriores. Debe cumplir con la definición de “Terminado”, siendo usable y potencialmente distribuible.

Flujo de Proceso Nexus



1. **Refinamiento del Product Backlog**
2. **Nexus Sprint Planning.**
3. **Trabajo de desarrollo:** todos los equipos desarrollan software, integrando frecuentemente su trabajo en un ambiente común que puede probarse para asegurar que la integración se haya realizado.
4. **Nexus Sprint Daily.**
5. **Nexus Sprint Review.**
6. **Nexus Sprint Retrospective.**

Definition of "Done"

El Equipo de Integración Nexus es responsable de una definición de "Terminado" que pueda aplicarse al Incremento Integrado desarrollado en cada Sprint. Todos los Equipos Scrum de un Nexus adhieren a esta definición de "Terminado". El Incremento está "Terminado" solamente cuando esté integrado, sea usable y potencialmente distribuible por el Dueño de Producto.

Métricas

Se utilizan para informar, comparar, predecir, mejorar, etc. Proveen información para la toma de decisiones. No son el problema, sino que lo visualizan y no son la solución, sino que las personas son las que solucionan los problemas.

Definiciones

Medida

Proporciona una indicación cuantitativa de la cantidad, dimensiones o tamaño de alguno atributo de un proceso o producto.

Medición

Acto de determinar una medida.

Métrica

Es la medida del grado en que un sistema, componente o proceso posee un atributo dado

Indicador

Es una métrica o un conjunto de métricas que proporcionan una visión profunda del proceso, proyecto o producto de software.

Métricas directas

Una métrica es directa si se puede medir directamente del atributo, donde su valor no depende de la medida de otros atributos y cuya forma de medir es un método de medición. Esta métrica puede ser utilizada en una función de calculo.

Ejemplos de métricas directas: cantidad de líneas de código, horas-programador diarias, costo por hora-programador.

Métricas indirectas

Una métrica es indirecta si en la medición de dicha métrica se utilizan medidas obtenidas en mediciones de otras métricas directas o indirectas. Esta métrica puede ser utilizada en una función de calculo y su forma de medir es una función de calculo.

Ejemplo de métricas indirectas: líneas de código fuente por hora de programador, costo total actual del proyecto, costo por línea de código fuente.

Escala

Conjunto de valor con propiedades definidas.

Confiabilidad y Validez

Estos parámetros visualizan la calidad de una métrica.

La validez hace referencia al grado en el cual la métrica refleja el significado real de lo que se pretende medir. La validez esta asociado a la exactitud y el valor que le otorga la decisor.

La confiabilidad hace referencia a métricas sin errores y esta asociado a la precisión de una métrica.

Métricas confiables (sin errores) no son necesariamente validas (miden lo que se pretende medir) y viceversa.

Métricas en el Software

La métricas en el software consisten en la aplicación continua de mediciones basadas en técnicas para el proceso de desarrollo de software y sus productos para suministrar información relevante a tiempo, así el líder de proyecto junto con el empleo de estás técnicas mejorará el proceso y sus productos.

Métricas de Proceso

Son métricas estratégicas a nivel organizacional, orientadas a mejorar el proceso. Son de publico conocimiento y se forman a partir de la despersonalización de métricas privadas. Un promedio de métricas privadas podría conformar a una métrica publica.

Ejemplos:

- Desviación organizacional de estimaciones.
- Defectos por severidad en productos de la organización.

Los indicadores de proceso permiten tener una visión profunda de la eficacia del proceso y permiten que los gestores evalúen que funciona y que no.

Métricas de Proyecto

Son métricas tácticas y locales, las cuales se toman en el contexto de un proyecto. Son privadas, es decir, son visibles solo para los involucrados en el proyecto.

Ejemplos:

- Eficiencia de estimación del proyecto.
- Costos estimados vs costos reales.
- Esfuerzo por etapa.

Los indicadores de proyecto permiten identificar riesgos potenciales y detectar áreas de problemas antes de que se conviertan en criticas, de forma tal de ajustar el flujo y la manera de trabajo.

Métricas de Producto

Miden características del producto, como tamaño, defectos, calidad. Son privadas porque están asociadas a un proyecto.

Ejemplos:

- Líneas de código,
- Defectos por severidad de un producto.
- Cantidad de casos de uso por complejidad.

Métricas básicas

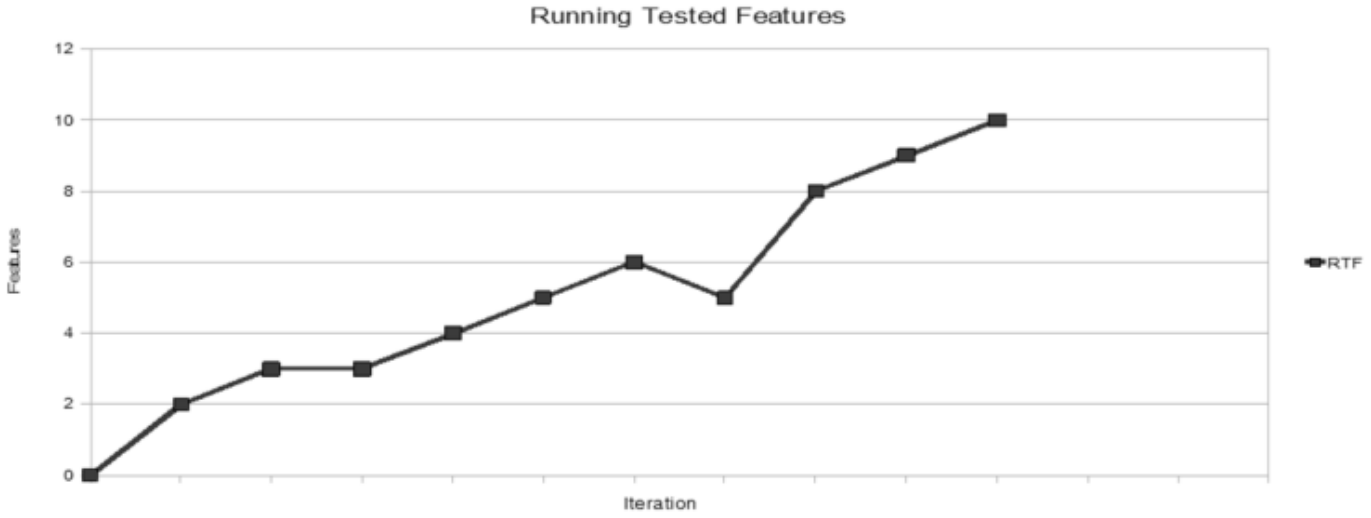
1. De proyecto: esfuerzo y tiempo
2. De producto: tamaño y defectos

Métricas en ambientes agiles

En enfoques ágiles nuestra mayor prioridad es satisfacer al cliente por medio de entregas tempranas y frecuentes de software valioso, es decir el software funcionando es la principal medida de progreso, por lo cual no se debe tomar a las métricas como una actividad, sino como una salida. Esto quiere decir que se debe medir lo que sea necesario y nada más, es decir lo que agregue valor para el cliente.

Running Tested Features (RTF)

Esta métrica esta basada en el principio de que la mejor medida de progreso es el software funcionando. Permite visualizar la evolución en el tiempo de los incrementos de producto entregados al cliente. Estos incrementos incluyen un conjunto de características que fueron probadas y aceptadas por el cliente.



Si la curva es constante o tiene pendiente negativa, entonces indica la existencia de un problema, ya que las características no incrementan o disminuyen en el tiempo. Esta ultima situación se presenta cuando una funcionalidad del sistema deja de ser valida.

Capacidad

Es una estimación de cuanto trabajo puede completarse en un periodo de tiempo basado en la cantidad de tiempo ideal disponible del equipo.

Características

1. Se puede estimar en horas ideales o en puntos de historia.
2. Se calcula como el producto entre la cantidad de días disponibles de iteración y la sumatoria de horas ideales disponibles de los miembros del equipo.
3. La estimación que realizan los miembros del equipo debe ser sincera.
4. Se debe considerar una capacidad máxima no mayor al 70%.

Velocidad

Es una observación empírica de la capacidad del equipo para completar el trabajo por iteración. Únicamente se contabiliza el trabajo finalizado. Esta basado en el tamaño de los ítems de trabajo propio de cada equipo, por lo tanto, es comparable para iteraciones de un equipo dado de un proyecto dado.
Lo ideal en un equipo ágil es mantener la velocidad constante, cumpliendo el principio del ritmo de desarrollo sostenido en el tiempo.
Se puede medir en cantidad de historias, puntos de historias u horas ideales.

Herramientas de Scrum

Tarjeta de tarea

Es una tarjeta que cuenta con 5 partes:

1. Código del Ítem Backlog: es un identificador del ítem en el Product Backlog.
2. Nombre de la actividad: suele ser una frase verbal que define que es lo que hay que hacer.
3. Iniciales del responsable de realizar la tarea.
4. Numero de día del sprint.
5. Estimación del tiempo para finalizar la tarea.

Tablero

Tablero utilizado por el Equipo de Desarrollo en el cual se colocan las tarjetas de tareas a realizar durante el sprint. Se encuentra dividido en 5 secciones:

1. To Do: aquí se encuentran las tareas por hacer en el Sprint.
2. Work In Process: aquí se encuentran las tareas que se están realizando.
3. To Verify: aquí se encuentran las tareas finalizadas que deben verificarse.
4. Done: aquí se encuentran las tareas terminadas: finalizadas y verificadas.

Lo ideal es que el nivel de granularidad de las tareas sea lo mas fino posible, de tal forma que se detalle profundamente el avance del sprint, teniendo varias tareas en cada sección del tablero.

Gráficos del Backlog

Gráficos que brindan información acerca del progreso de un Sprint, de una reléase o del producto. El backlog de trabajo es la cantidad de trabajo que queda por ser realizado, mientras que la tendencia del backlog compara esta cantidad con el tiempo.

Burn-down Chart

Visualiza con una curva descendente cuanto trabajo queda para terminar. En el eje de las abscisas se coloca el tiempo que va desde el inicio del sprint. En el eje de las ordenadas se coloca los puntos de historia a realizar en el sprint.

Burn-up Chart

Visualiza con una curva ascendente cuanto trabajo se ha completado a lo largo de la release, es decir que visualiza los puntos de historia terminados en cada sprint.

Combined Burn Chart

Visualiza cuanto trabajo queda para terminar y cuanto trabajo se ha realizado.

Beneficios de Scrum

1. Se gestionan los cambios de requerimientos.
2. Se incorpora la visión de mercado.
3. Los clientes ven incrementos que refinan los requerimientos en un tiempo razonable.
4. Mejores relaciones con el cliente.

Gestión del producto

Cuando creamos un producto debemos enfocarnos en aquellos que es de valor para el cliente, sabiendo que el 7% de las características del software son siempre usadas. Esto implica, que la mayor parte del valor agregado esta concentrado en ese 7%.

Las organizaciones que usan el software se enfocan en la parte superficial del software, aquello que consideran significativo y placentero. Usualmente no se fijan si el producto es funcional o confiable. Esa debe ser la base, pero el producto debe proveer una experiencia que cumpla con todas las expectativas del usuario.

Planificación del producto en agile

En las metodologías agiles, se tiene una visión del producto materializado en el Product backlog, que contiene todas las características de las funcionalidad que requiere producto que han sido identificadas hasta el momento. Esta visión se descompone en releases, las cuales son las entregas frecuentes que se hará al cliente. Cada release se construye a partir de sucesivas iteraciones.

Valor vs desperdicio

La productividad de un Startup no puede medirse en términos de cuánto se construye cada día, por el contrario, se debe medir en términos de construir lo correcto cada día y lo correcto es aquello que agrega valor. Si no construimos valor, entonces estamos invirtiendo esfuerzo en desperdicio.

Mínimo Producto Viable

Es un concepto de Lean Startup que enfatiza el impacto del aprendizaje en el desarrollo de nuevos productos.
Se define como la versión de un nuevo producto que permite a un equipo recopilar la cantidad máxima de aprendizaje validado sobre los clientes con el menor esfuerzo.
A este concepto le subyace la idea de que ver lo que la gente realmente hace con el producto es mucho más confiable que preguntarle a la gente qué harían.
Entre sus características se destaca que:

1. Tiene el valor suficiente para que las personas estén dispuestas a usarlo o comprarlo.
2. Demuestra beneficios a futuro para retener a los usuarios.
3. Proporciona un ciclo de retroalimentación para guiar el desarrollo a futuro.

Los errores comunes son:

1. Confundirlo con MMF o MMP. La MMF (Característica Comerciable Mínima) o el MMP (Producto Comerciable Mínimo) a diferencia del MVP se enfocan en ganar.
2. Enfatizar la parte de mínima y dejar de lado la parte de viable. Si el producto no tiene la calidad suficiente, no será posible obtener la retroalimentación del cliente.
3. Hacer del MVP un producto final.

Un MVP permite validar diferentes hipótesis:

1. Hipótesis del valor: prueba si el producto realmente está entregando valor a los clientes después de que comienzan a usarlo. Una métrica de prueba: tasa de retención
2. Hipótesis de crecimiento: prueba cómo nuevos clientes descubrirán el Producto. Una métrica de prueba: tasa de referencia o Net Promoter Score (NPS)

Design Thinking

Es una metodología para la creación de servicios digitales centrada en el cumplimiento de las expectativas del usuario mediante un proceso de mejora continúa que advierte las verdaderas necesidades del mismo.

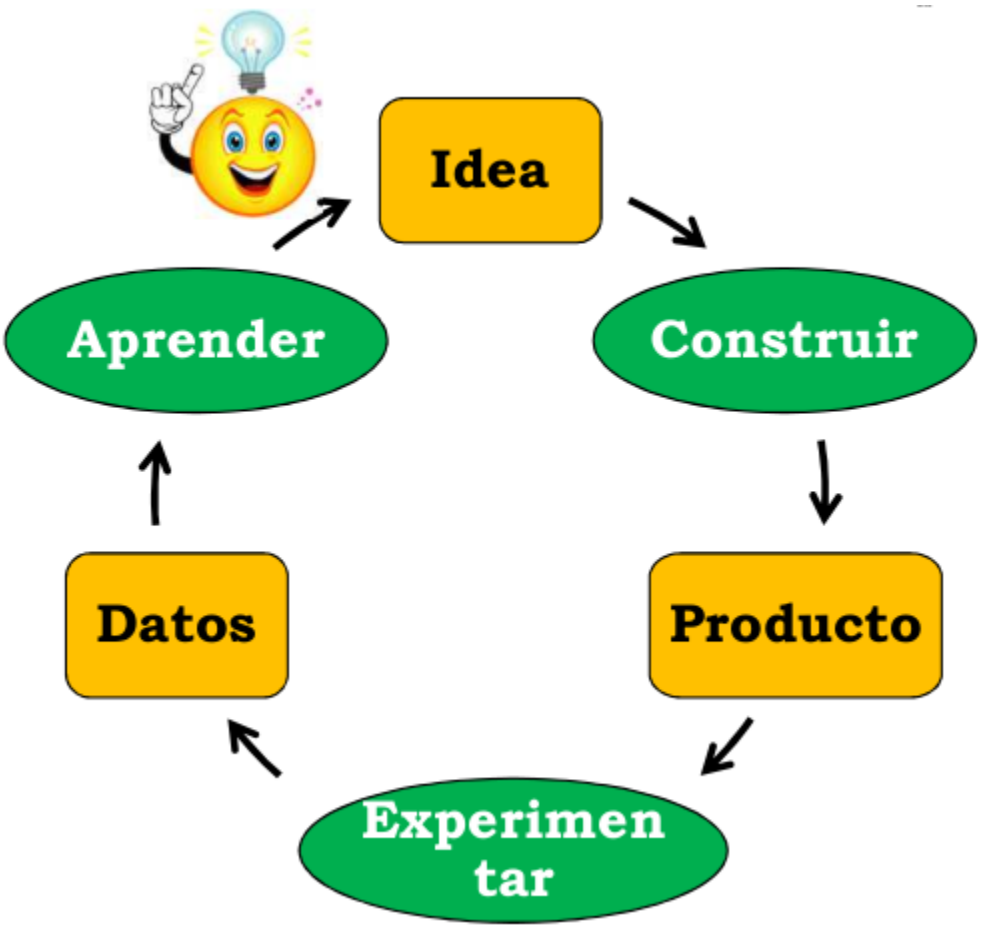
Para llevar a cabo su proceso, utiliza un esquema de pensamiento divergente y convergente. Con el primero se busca tener una visión amplia del espectro de soluciones que responden al problema inicial. El segundo consiste en reducir a la mejor idea el conjunto de posibles soluciones, con el objetivo de brindar aquella que es mas adecuada para las necesidades del cliente.

El proceso cuenta con 5 pasos y es iterativo:

1. Empatizar: implica comprender las necesidades del cliente mediante la utilización del razonamiento intuitivo.
2. Definir: determinar lo valioso para el cliente a partir de lo recopilado durante la primer etapa, lo cual permitirá guiar el trabajo.
3. Idear: articular muchas soluciones posibles. Esto se puede lograr en los equipos multifuncionales, donde cada miembro aporta una mirada diferente para generar muchas ideas para luego filtrar las mas viables.
4. Prototipar: consiste en materializar las ideas. Las ideas seleccionadas se transforman en prototipos que permitan visualizar la solución propuesta.

5. Probar: es una de las etapas mas importantes del proceso, ya que permite validar los prototipos con los usuarios implicados en la solución, obteniendo retroalimentación y permitiendo iterar.

Lean UX



UNIDAD 3: Gestión del software como Producto

Gestión de Configuración

Introducción

Las sistemas de software siempre cambian durante su desarrollo y uso. Conforme se hacen cambios al software, se crean nuevas versiones del sistema. Los sistemas pueden considerarse como un conjunto de versiones donde cada una de ellas debe mantenerse y gestionarse. Esto es necesario para no perder la trazabilidad de los cambios que se incorporan a cada versión.

Definición de Gestión de configuración

Es una disciplina de soporte transversal al proyecto que aplica dirección y monitoreo administrativo y técnico a: identificar y documentar las características funcionales y técnicas de los ítems de configuración, controlar los cambios de esas características, registrar y reportar los cambios y su estado de implementación y verificar correspondencia con los requerimientos.

En esta definición dada por la IEEE se mencionan las actividades fundamentales de la Gestión de Configuración. Referentes en el software la definen como el arte de coordinar el desarrollo de software para minimizar la confusión y los errores, identificando, organizando y controlando las modificaciones que se realizan en el software que construye el equipo.

Todos los involucrados en el desarrollo del producto de software son responsables de realizar la gestión de configuración.

Razón para gestionar la configuración

Su propósito es establecer y mantener la integridad de los productos de software a lo largo de su ciclo de vida. Esto implica identificar la configuración en un momento dado, controlar sistemáticamente sus cambios y mantener su integridad y origen.

Integridad del producto de software

La integridad es el medio por el cual podemos garantizar que el producto a entregar tiene la calidad correspondiente.

El problema de la calidad es que es subjetiva y se suma a que el software es intangible, por lo tanto, es muy difícil medir si la calidad que responde al cumplimiento de las expectativas del cliente realmente se verifica.

La idea de la integridad del producto es hacer explicita las características o expectativas que tiene el cliente sobre el producto de software.

Decimos que un producto es integro si se verifica que:

1. Satisface las expectativas del usuario: tanto en lo funcional como en lo no funcional.
2. Puede ser fácilmente rastreado durante su ciclo de vida: implica tener una rastreabilidad de cada pieza de software, pudiendo determinar como llego cada una al estado en el que se encuentra. Permite establecer una correspondencia entre los diferentes niveles de abstracción del software: requerimiento, modelos de análisis y diseño, código, casos de prueba.
3. Satisface criterios de performance
4. Cumple expectativas de costo: esta apartado incluye la satisfacción del equipo de proyecto. No solo el cliente debe estar satisfecho, sino también el desarrollo del producto debe ser redituable.

Conceptos generales

Ítem de configuración

Es cualquier aspecto asociado al producto de software (requerimientos, diseño, código, datos de prueba, documentos, etc.) que es necesario mantener.

Plan de Gestión de Configuración

Es un plan de soporte donde se definen los acuerdo y las políticas de trabajo respecto a la gestión de configuración.

Definen la convención de nombres para los ítems de configuración que garantice la unicidad de los mismos. Para esto primero se identifica el nivel del ítem respecto al ciclo de vida, luego se identifica si es un ítem único o múltiple y también se define el nombre físico donde se encuentra almacenado en el repositorio.

Versión

Instancia de un ítem de configuración que difiere de otras instancias del mismo ítem. Las versiones se identifican unívocamente. Controlar una versión refiere a la evolución de un único ítem de configuración.

Variante

Versión de un ítem de configuración (o de la configuración) que evoluciona por separado. Las variantes representan configuraciones alternativas. Las variantes de un ítem de configuración permiten satisfacer requerimientos en conflicto, como por ejemplo la compatibilidad del producto con diferente hardware o sistemas operativos.

Configuración

Conjunto de todos los ítems de configuración con su versión especifica. Define una foto de los ítems de configuración en un momento de tiempo determinado.

Línea base

Es un ítem de configuración o una configuración de un momento de tiempo determinado que ha sido revisada, evaluada y aprobada, y que para ser modificada tiene que pasar por un proceso formal de control de cambios.
Se considera que contiene un estado estable del producto y sirve como base para desarrollos posteriores.
Para definir una línea base, se “taguea” o etiqueta a los ítems de configuración que componen a la línea base.
Existen dos tipos de línea base:

1. Operacionales: contiene una versión de producto cuyo código es ejecutable. La primera línea base operacional corresponde con la primera release.
2. De especificación o documentación: son las primeras línea base, dado que no cuentan con código. Podría contener el documento de especificación de requerimiento.

Gestión de versiones

Es el proceso de realizar un seguimiento de las diferentes versiones de los ítems de configuración, asegurando que los cambios realizados por los trabajadores del software se integren correctamente y no interfieran unos con otros.
Un repositorio es el encargado de contener los ítems de configuración.
Los sistemas de gestión de versiones ofrecen diferentes características como:

1. Identificación de versión y entrega: cada versión gestionada se identifica al realizar la release.
2. Gestión de almacenamiento: en vez de conservar una copia completa de cada versión, el sistema almacena una lista de diferencias (deltas) entre una versión y otra. Esto permite reducir el espacio de almacenamiento requerido por múltiples versiones con pequeñas variaciones entre unas y otras.

3. Registro historial de cambios: todos los cambios realizados sobre el repositorio son registrados y enumerados. Mantiene la historia de cada ítem de configuración: sus atributos y sus relaciones.
4. Desarrollo simultaneo: es posible que diferentes desarrolladores trabajen sobre el mismo componente de software al mismo tiempo. El sistema se encarga de gestionar los cambios realizados sobre el mismo componente para que no haya interferencia.

Funcionamiento del repositorio

Para trabajar sobre ese repositorio, se utiliza lo que se conoce como modelo checkIn/checkOut.
Checkout permite extraer los datos del Area de trabajo (esto es la primera vez, luego se hace update), acá es donde se trabaja y se hacen los cambios. Una vez que se finaliza, se hace un Checkin de los Cambios realizados al repositorio. Se puede utilizar un esquema de bloqueo completo para evitar inconsistencias, o bien permite la modificación simultanea.

Tipos de repositorios

Los sistemas de control de versiones se clasifican en dos grupos:

1. Centralizados: todos los ítems de configuración y sus respectivas versiones se encuentran almacenados en el mismo directorio (denominado repositorio de fuentes) en un mismo servidor. Los desarrolladores deben solicitar al sistema de control de versiones una copia del repositorio de forma local. Cada vez que realizan un cambio el sistema lo almacena como una nueva versión. Tiene como ventaja que facilita el control al administrador. Sin embargo, la desventaja es que al ser centralizado ante una falla del servidor se pierde la disponibilidad del repositorio.
2. Distribuidos: cada desarrollador tiene una copia de todo el repositorio, con todas las versiones y todo el historial. El sistema de control de versiones en este caso permite que en cualquier momento dos desarrolladores puedan sincronizar sus repositorios. Si un servidor falla, se tiene al resto funcionando, por lo tanto, solo es cuestión de copiar y pegar.

Actividades

Identificación de los ítems de configuración

Para controlar y gestionar los ítems de configuración primero deben identificarse unívocamente a cada uno. El enfoque es orientado a objetos, existiendo dos tipos de objetos:

1. Básico: es una unidad de información que se genera en cualquier etapa del proceso de desarrollo. Ejemplo: una especificación de requerimiento, un caso de uso, un caso de prueba.
2. Agregado: es una colección de objetos básicos y/o agregados. Ejemplo: modelo de dominio.

Cada ítem tiene un nombre conformado por una cadena de caracteres que identifica el ítem sin ambigüedades. Esta se obtiene a partir de la convención de nombres.

Clasificación de ítems según su ciclo de vida

Define la clasificación en función de cuanto tiempo es necesario mantener un determinado ítem.

1. Ítems de producto: documento de arquitectura, casos de uso, código, manual de usuario y de parametrización, casos de prueba.
2. Ítems de proyecto: plan de proyecto,
3. Ítems de iteración: planes de iteración, cronograma de iteración.

Conocer el ciclo de vida de un ítem permite establecer la nomenclatura del ítem.
Luego de identificar los ítems se debe armar la estructura del repositorio con nombres representativos. Luego se vinculan los ítems de configuración con el lugar físico del repositorio donde se van a almacenar.

Proceso de Control de Cambios

Una vez definida la línea base, no es posible cambiarla sin antes pasar por un proceso formal de control de cambios. Es decir que el control se hace sobre los ítems de configuración que pertenecen a la línea base, ya que todos los trabajadores del software tienen a dichos ítems como referencia.

Comité de control de cambios

Este proceso es llevado a cabo por el comité de control de cambios, el cual se reúne para autorizar la creación y cambios sobre la línea base. Forman parte de dicho comité los interesados en evaluar y en enterarse del cambio, decidiendo si lo aceptan o no. El líder del proyecto, el analista, el arquitecto e incluso el cliente pueden constituir a dicho comité. Realmente la integración del comité depende de la propuesta del cambio.

Etapas del proceso

Si el comité fijo una línea base luego de la revisión de lo ítems de configuración que la conforman y llegada a la conclusión de que los mismos se encuentran en estado estable, en caso de recibir una propuesta de cambio, los pasos a seguir son los siguientes:

1. Se recibe una propuesta de cambio sobre una línea base determinada.
2. El comité de control de cambios realiza un análisis de impacto del cambio para evaluar el esfuerzo técnico, el impacto en la gestión de los recursos, los efectos secundarios y el impacto global sobre la funcionalidad y la arquitectura del producto.
3. En caso de que se autorice la propuesta de cambio, se genera una orden de cambio que define lo que se va a realizar, las restricciones a tener en cuenta y los criterios para revisar y auditar.
4. Luego de realizado el cambio, el comité vuelve a intervenir para aprobar la modificación de la línea base y marcarla como línea base nuevamente.
5. Finalmente se notifica a los interesados los cambios realizados sobre la línea base.

Informes de estado

Los informes nos dicen el estado actual de la configuración del software. Cada herramienta de gestión de configuración tiene un conjunto de informes.

El reporte mas conocido es el de inventario, el cual provee una copia del contenido del repositorio con la estructura de directorios.

Objetivos

1. Mantener los registros de la evolución del sistema, incluyendo reportes de rastreabilidad de todos los cambios realizados a las líneas base durante el ciclo de vida, es decir nos dice cuáles son las líneas base, cuando se modificó, quien modifico cada ítem, etc.
2. Brindar información para el proceso de auditoria.
3. Permitir responder a preguntas como:
    a. ¿Cuál es el estado del ítem?
    b. ¿La propuesta de cambio fue aceptada o rechazada por le comité?
    c. ¿Qué versión de ítem implementa un cambio de una propuesta de cambio aceptada?
    d. ¿Cuál es la diferencia entre dos versiones de un mismo ítem?

Registros e informes

1. Momento de incorporación o actualización de una línea base.
2. Estado de cada cambio propuesto.
3. Momento en que se incorpora un cambio de configuración de software.

4. Estado de la documentación administrativa o técnica.
5. Deficiencia detectadas durante la auditoria de configuración.

Auditorias de configuración

Auditoria

Actividad de control correspondiente a una revisión objetiva e independiente.

1. Es objetiva cuando hay independencia de criterio, por lo que el auditor no tiene ningún condicionamiento respecto de lo auditado.
2. Es independiente cuando no depende jerárquica, funcional ni salarialmente del equipo o elemento a auditar.

La auditoria de configuración se hace mientras el producto se esta construyendo y su objetivo es que el producto tenga calidad e integridad. Se entiende que lo mas barato es prevenir.
La auditoria de configuración complementa a la revisión técnica y consiste en revisar si se están realizando las tareas tales como se planificaron y especificaron en el plan de gestión de configuración. Las auditorias se realizan sobre una línea base. Es decir, la auditoria requiere la existencia de un plan de gestión de configuración y de la línea base a auditar.

Trazabilidad o rastreabilidad de requerimientos

Establecer una correspondencia de relación entre ítems de configuración de diferentes niveles de abstracción en el proceso de desarrollo de software. Una traza puede ser constituida por el requerimiento, sus modelos de análisis y de diseño, la implementación en código y los casos de prueba para validar y verificar el requerimiento. La trazabilidad permite realizar una análisis del impacto sobre el sistema a partir de una solicitud de cambio en el requerimiento.
Los modelos de calidad exigen la trazabilidad bidireccional. Desde el requerimiento hacia el componente de código y viceversa.

Tipos de auditorias

1. Auditoria funcional de configuración: audita la trazabilidad de requerimientos, controlando que la funcionalidad y performance real de cada ítem de configuración sean consistentes con la especificación de requerimientos. El propósito final es validar que el código implementa solo y todos los requerimientos de la ERS.
2. Auditoria física o de contingencia: se realiza primero. Se validan los ítems de configuración de la línea base contra el plan de gestión de configuración, asegurando que la documentación que se entragara es consistente con el código desarrollado. Si esta no pasa, la funcional no se realiza.

Gestión de configuración en ambientes agiles

Introducción

En las metodologías agiles la gestión de configuración cambia el enfoque, ya que la misma es de utilidad para los miembros del equipo de desarrollo y no viceversa.
En orden con la caracterización de los equipos agiles, decimos que la gestión de configuración posibilita el seguimiento y la coordinación del desarrollo en lugar de controlar a los desarrolladores.
Los equipos agiles son autoorganizados, por lo que las Auditorias de configuración no son una actividad propia de la gestión de configuración en los ambientes agiles. Todos los procesos definidos establecidos en la gestión

de configuración del enfoque tradicional son relativizados en agile. Por ejemplo, no existe un comité para el proceso forma de control de cambios.

Características

1. Dada la naturaleza de los requerimientos agiles, la SCM responde a los cambios en lugar de evitarlos.
2. La automatización debe ser aplicada donde sea posible. Esto colabora con la entrega frecuente y rápida software.
3. Provee retroalimentación continua sobre calidad, estabilidad e integridad.
4. Las actividades de SCM se encuentran embebidas en las demás tareas para alcanzar el objetivo del sprint.
5. Es responsabilidad de todo el equipo, por lo tanto, requiere capacitación.
6. Adopción de las practicas continuas.

Integración Continua

Es una práctica de desarrollo que promueve que los desarrolladores adopten la costumbre de integrar su código a un repositorio compartido varias veces al día. Cada integración de código es luego verificada por una serie de pruebas automatizadas, permitiéndole al equipo a detectar problemas de manera temprana. Ya que al integrar el código al repositorio de manera frecuente resulta más fácil detectar y corregir los errores.

Entrega Continua

Es una disciplina de desarrollo software en la que el software se construye de tal manera que puede ser liberado en producción en cualquier momento.
No implica necesariamente que se libere cada vez que hay un cambio, solo ocurre si el responsable de negocio, el Product Owner de turno, decide pasar a producción. Es decir que hay un componente «humano» a la hora de tomar la decisión. Pero, en cualquier caso, la versión está lista de inmediato. Esto implica, entre otros, que se prioriza que la versión esté en un estado en el que puede ser puesta en producción.

Despliegue Continuo

A diferencia de la entrega continua, en el despliegue continuo no existe la intervención humana para desplegar el producto en producción. Para esto, se utilizan pipelines que contienen una serie de pasos que deben ejecutarse en un orden determinado para que la instalación sea satisfactoria.

UNIDAD 4: Gestión Lean-Ágil de Productos de Software

Calidad

Son todos los aspectos y características de un producto o servicio que se relacionan con su habilidad de alcanzar las necesidades manifiestas.
Formalmente se puede definir como el cumplimiento de los requerimientos funcionales y de performance explícitamente definidos, de los estándares de desarrollo explícitamente documentados y de las características implícitas esperadas del desarrollo de software profesional.

1. La calidad no se inyecta, ni se compra, debe ser embebida, las personas son la clave para lograrlo (capacitación).
2. Se necesita soporte gerencial
3. El aseguramiento de calidad debe planificarse.
4. El aumento de las pruebas no aumenta la calidad.

Factores de calidad

En el software, la calidad es subjetiva y circunstancial: subjetiva porque depende de lo atributos para medirla y circunstancial porque el conjunto de atributos elegidos puede variar en situaciones diferentes.
Los factores de calidad suelen ser difíciles o prácticamente imposibles de medir y están asociados a los requerimientos del sistema.

Importancia del desarrollo con calidad

En la actualidad, el trabajo para y con calidad es fundamental en la producción del software ya que:

1. Es un aspecto competitivo, que permite sobrevivir en el mercado internacional. Las empresas certifican estándares de calidad para poder competir en el mismo. En otras palabras, la calidad es el sello de clase en el mundo de los negocios.
2. Retiene a los clientes e incrementa los beneficios. Siempre que hablamos de calidad, en el fondo nos referimos al cumplimiento de las expectativas del cliente en todos los sentidos.
3. Determina un equilibrio del costo-efectividad. Trabajar con calidad reduce los costos y el retrabajo, ya que se asume que las fallas serán menores.

Es indudable que la calidad inmersa en el proceso y en el producto resultante es realmente beneficioso. Sin embargo, existen algunas consideraciones a tener en cuenta.

Dilema de la calidad

Si se produce un sistema de mala calidad, entonces nadie lo querrá comprar. Por otro lado, si se dedica un gran esfuerzo de tiempo y un alto presupuesto al desarrollo del software perfecto, entonces demorará tanto tiempo en desarrollarlo y costará tanto producirlo que de igual forma estará fuera de mercado. La industria del software trata de situarse en un punto intermedio, donde el producto final sea lo sufrientemente bueno para no ser rechazado, sin ser un producto perfeccionista que lo convierta en algo que requiera demasiado tiempo o dinero para terminarlo.
El software suficientemente bueno contiene las funciones y características de alta calidad que desean los usuarios, pero al mismo tiempo tiene otras particulares que contienen errores conocidos. El vendedor de software espera que la gran mayoría de usuarios finales valore las características de calidad por sobre los errores conocidos, con la esperanza de que en futuras versiones estos serán corregidos.

Costo de la calidad

La alta calidad y la mala calidad tienen ambas un costo asociado. El costo de la calidad incluye todos los costos en los que se incurre al buscar la calidad o al realizar actividades relacionadas con ella y los costos posteriores de la falta de calidad. Este costo esta determinado por los costos de prevención, evaluación y falla.

1. Costo de prevención: comprende 4 costos
    a. Costo de las actividades de administración requeridas para planear y coordinar todas las actividades de control y aseguramiento de la calidad.
    b. Costo de las actividades técnicas agregadas para desarrollar modelos completos de los requerimientos y del diseño.
    c. Costos de planear las pruebas.
    d. Costo de la capacitación asociada a cada una de estas actividades.
2. Costo de evaluación: conformado por costos de revisiones técnicas, de hacer y depurar pruebas, de recabar datos y mediciones para la evaluación.
3. Costo de falla: son aquellos que se eliminarían si no hubiera errores antes o después de enviar el producto a los consumidores. Se dividen en internos y externos.
    a. Internos: falla detectada antes de entregar el producto al cliente.
    b. Externo: falla detectada luego de que el producto se entrega al cliente.

La conclusión es que los costos de corregir un error manifiesto en producción son mucho mas elevados que la prevención y corrección de fallas previo a entregar el producto al cliente.

Aseguramiento de calidad

El aseguramiento de calidad es una actividad continua que se realiza acompañando a la construcción del producto. Esto quiere decir que la calidad no es una etapa puntual. El propósito del aseguramiento de calidad es que el producto llegue en las mejores condiciones posibles al testing y que este sea lo menos costoso.

Se aplica calidad en el producto y en el proceso. No se habla de aseguramiento de calidad en el proyecto, dado que esta se encuentra implícita por el hecho de que el proyecto implementa un proceso. Para garantizar la calidad se realizan auditorias.

La calidad del producto se realiza mediante actividades de revisiones técnicas y de auditorias. Estas son las herramientas principales para el seguimiento.

Objetivos

1. Realizar los controles apropiados del software y de su proceso de desarrollo.
2. Asegurar el cumplimiento de los estándares y procedimientos para el software y el proceso.
3. Asegurar que los defectos en el producto, proceso o estándares son informados a la gerencia para que puedan ser solucionados.

Procesos definidos

En los procesos definidos se considera que el proceso es el único factor controlable, por lo tanto, se asume que la mejora de la calidad continua se realiza sobre el proceso. Si el proceso cuenta con calidad, entonces el producto será de calidad.

Todos los modelos de calidad están basados en procesos definidos, por lo que asumen que la calidad del producto se obtiene si se tiene un proceso de calidad para construir el producto.

Procesos empíricos

Los procesos empíricos están basados en la experiencia, por lo que no consideran que la calidad en el proceso determine la calidad en el producto. Por otra parte, no están de acuerdo con las auditorias, ya que asumen que los equipos son autoorganizados. Sin embargo, la calidad en estos procesos se lleva a cabo mediante revisiones técnicas y la constante inspección y adaptación del trabajo.

Modelos de calidad

Introducción

Un aspecto importante del aseguramiento de calidad es la definición o selección de los estándares que deben aplicarse al proceso de desarrollo de software y al producto de software. También es posible elegir herramientas y métodos para apoyar el uso de dichos estándares.

Importancia de los modelos de calidad

Su importancia se puede resumir en 3 aspectos

1. Se basan en conocimiento sobre la mejor o más adecuada práctica para la organización.
2. Brindan el marco para la definición del termino calidad. Dado que la calidad es subjetiva, la utilización de estándares permite determinar si se logro el nivel de calidad requerido en términos de confiabilidad, usabilidad y rendimiento del software.
3. Auxilian la continuidad cuando una persona toma el trabajo iniciado por otro, dado que los estándares establecen practicas comunes dentro de la organización y facilitan la adaptación en términos del esfuerzo de aprendizaje del nuevo personal.

Tipos de modelos de calidad

Estándares de producto

Se aplican al producto de software a desarrollar. Incluyen estándares de documentos (como la estructura de los documentos de requerimientos), estándares de documentación (como el encabezado de un comentario estándar para la definición de una clase) y estándares de codificación, los cuales definen cómo debe usarse un lenguaje de programación.

Estándares de proceso

Establecen los procesos que deben seguirse durante el desarrollo del software. Deben especificar cómo es una buena práctica de desarrollo. Estándares de procesos pueden ser el proceso de liberación de una versión, el proceso de control de cambio, el proceso de compilación automática.

Lineamientos para la implementación de Modelos de calidad en las organizaciones

Aunque en general los ingenieros de software están de acuerdo con la necesidad de implementar modelos de calidad, a menudo encuentran razones para justificar que los mismos no son necesariamente adecuados para sus proyectos particulares.

Para que los gestores de calidad puedan implementar dichos modelos en las organizaciones se deben cumplir ciertos aspectos:

1. Lo principal es compromiso visible y continuo por parte de la alta dirección.
2. Capacitación y educación: se requiere que los empleados adquieran y desarrollen nuevos conocimientos y habilidades.
3. Elaborar y establecer en todos los niveles un programa de comunicación acerca del proceso de implementación del modelo.

4. Realizar una planeación de la implementación del modelo. Esto conlleva las actividades de:
    a. Evaluar el estado actual.
    b. Establecer la estructura del proyecto.
    c. Elaborar un plan de proyecto.
    d. Elaborar directrices del sistema de calidad.
    e. Elaborar directrices para la preparación de documentos.
    f. Seleccionar un organismo certificador.

Modelo de mejora de procesos

Definición

La mejora continua de procesos significa comprender los procesos existentes y cambiarlos para incrementar la calidad del producto o reducir los costos y el tiempo de desarrollo. Los procesos definidos están de acuerdo con esta definición, ya que consideran que la calidad del producto final depende de la calidad del proceso.

Principales Modelos de Calidad existentes

Estándar ISO 9001

Estos estándares se aplican a organizaciones que diseñan, desarrollan y mantienen productos, incluido software.

Características

1. No es un estándar para el desarrollo del software, sino que establece principios de calidad total y define que procesos y procedimientos organizacionales deben determinarse y documentarse.
2. Para certificar, una organización debe ajustar sus procesos particulares a los 9 procesos centrales descriptos por el estándar. Además, debe definir y mantener los registros que demuestran que se siguieron los procesos establecidos.
3. No define ni prescribe los procesos de calidad específicos que debe usar una organización. Esto permite la flexibilidad frente a diversos sectores y tamaños de organizaciones.
4. La certificación de la norma no implica que el producto resultante tenga calidad. Sin embargo, algunos clientes demandan que sus proveedores de software estén certificados en la norma.
5. Autoridades de acreditación independientes examinan los procesos de gestión de calidad y la documentación de los mismos, y deciden si la organización certifica o no la norma.
6. Las organizaciones agiles no tiene el foco en procesos formales de gestión de calidad, por lo cual no se preocupan por la certificación de esta norma.

CMMI

CMMI (Integración de modelos de madurez de capacidades, por sus siglas en inglés) es un modelo para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software, que constituye un conjunto de buenas prácticas que son publicadas en modelos.

Beneficios

1. Reducción de costos
2. Incremento en la calidad de los productos y servicios
3. Mejoras en la eficiencia y la productividad del personal
4. Métricas confiables y definidas

Areas de aplicabilidad

1. Desarrollo: aseguramiento de calidad, planificación y ejecución del trabajo, gestión de la fuerza de trabajo, desarrollo de productos, entre otros.
2. Servicios: orientado a mejorar la capacidad de la organización de brindar servicios de calidad a sus clientes, haciendo hincapié en las necesidades insatisfechas de los mismos y mejorando la experiencia de usuario.
3. Adquisición: tanto de bienes como de servicios; permite mejorar los procesos de selección de proveedores, reduciendo costos e incrementando la eficiencia de la cadena productiva.
4. Personal: modelo orientado a fortalecer y promover el desarrollo de habilidades del personal, brindando herramientas que permitan aprovechar al máximo la capacidad de los miembros de una organización.

Representación continua

Se centra en la mejora de un proceso o un conjunto de ellos relacionados a un área de proceso en que una organización desea mejorar, una organización puede obtener la certificación para un área de proceso en cierto nivel de capacidad.

1. Nivel de capacidad 0 - Incompleto: no se realiza o se realiza parcialmente. Al menos una de las metas específicas del área de proceso no se satisface y no existen metas genéricas para este nivel.
2. Nivel de capacidad 1 - Proceso realizado: el proceso se completa y genera producto de trabajo. En este caso se satisfacen las metas específicas del área de proceso.
3. Nivel de capacidad 2 - Proceso gestionado: proceso realizado que se planifica y ejecuta de acuerdo con la política; emplea personas con habilidades que tienen recursos adecuados para producir productos de salida controlados; es monitorizado, controlado y revisado;
4. Nivel de capacidad 3 - Proceso definido: proceso gestionado que se adapta a partir del conjunto de procesos estándar de la organización, de acuerdo con las guías de adaptación de la organización; tienen una descripción de proceso mantenida; y contribuye con productos de trabajo, mediciones y otra información de mejora de procesos a los activos de proceso de la organización.

Representación escalonado

Esta representación ofrece un método estructurado y sistemático de mejoramiento de procesos, implica mejorar por etapas o niveles. Las empresas que se certifican alcanzan un determinado nivel de madurez.

1. Nivel de madurez 1 – Inicial: los procesos son generalmente ad hoc y caóticos. La organización generalmente no proporciona un entorno estable para dar soporte a los procesos. Sin embargo, a menudo, producen productos y servicios que funcionan, pero exceden con frecuencia el presupuesto y los plazos planificados.
2. Nivel de madurez 2 – Gestionado: se garantiza que en los proyectos los procesos se planifican y ejecutan de acuerdo con las políticas; los proyectos emplean personal cualificado que dispone de recursos adecuados para producir resultados controlados; se monitorizan, controlan y revisan; y se evalúan en cuanto a la adherencia a sus descripciones de proceso.
3. Nivel de madurez 3 – Definido: procesos están bien caracterizados y comprendidos, y se describen en estándares, procedimientos, herramientas y métodos. Estos procesos estándar se utilizan para establecer la integridad en toda la organización. Los proyectos establecen sus procesos definidos adaptando el conjunto de procesos estándar de la organización de acuerdo a las guías de adaptación.
4. Nivel de madurez 4 - Gestionado cuantitativamente: la organización y los proyectos establecen objetivos cuantitativos para la calidad y el rendimiento del proceso, los cuales se basan en las necesidades del

cliente, usuarios finales, organización e implementadores del proceso. La calidad y el rendimiento del proceso se interpretan en términos estadísticos y se gestionan durante la vida de los proyectos.

5. Nivel de madurez 5 - En optimización: una organización mejora continuamente sus procesos basándose en una comprensión cuantitativa de sus objetivos de negocio y necesidades de rendimiento. La organización utiliza un enfoque cuantitativo para comprender la variación inherente en el proceso y las causas de los resultados del proceso.

SPICE

El ISO/IEC 15504, también conocido como Determinación de la capacidad de mejora del proceso de software abreviado SPICE es un modelo para la mejora y evaluación de los procesos de desarrollo, mantenimiento y operación de los productos de software.

Estructura de la norma

SPICE consta de 10 partes que se han ido publicando por separado desde 2003 a 2011:

1. Parte 1: Introducción y conceptos.
2. Parte 2: Un modelo para la gestión de procesos.
3. Parte 3: Procesos de evaluación.
4. Parte 4: Guía para realizar la evaluación.
5. Parte 5: Guía para la construcción, selección y uso de instrumentos y herramientas de evaluación.
6. Parte 6: Formación de los evaluadores.
7. Parte 7: Guía para la mejora de procesos.
8. Parte 8: Guía para la determinación de la capacidad del proceso del proveedor
9. Parte 9: Vocabulario

Niveles de capacidad

Hereda los niveles de CMMI.

1. Nivel 0: El proceso es incompleto. No está completamente implementado y no logra sus objetivos
2. Nivel 1: El proceso se realiza. Se implementa y logra sus objetivos
3. Nivel 2: El proceso se gestiona. Está controlado, su implementación está planificada, monitoreada y ajustada. Sus resultados (productos de trabajo) son establecidos, controlados y debidamente registrados y mantenidos;
4. Nivel 3: el proceso está establecido. Está documentado para garantizar su capacidad para cumplir sus objetivos;
5. Nivel 4: el proceso es predecible. Opera de acuerdo con los objetivos de calidad cuantificables.
6. Nivel 5: el proceso está en optimización (optimización). Mejora continuamente para ayudar a alcanzar los objetivos actuales y futuros.

Atributos del proceso

Para evaluar el alcance de un nivel de capacidad determinado para un proceso, el estándar especifica una serie de atributos del proceso que están ligados a cada nivel de capacidad.

1. Nivel 1: Atributos de rendimiento del proceso
2. Nivel 2:
    a. Atributo de gestión del rendimiento
    b. Atributos de la gestión de los productos de las actividades
3. Nivel 3:
    a. Atributos de definición de proceso

    b. Atributo de despliegue de proceso
4. Nivel 4:
    a. Atributos de medición del proceso
    b. Atributo de control de proceso
5. Nivel 5:
    a. Atributos de innovación de procesos
    b. Atributo de optimización del proceso

Atributos de clasificación

Finalmente se requiere el establecimiento de una escala de calificación cuyos valores se basan en el porcentaje de logro de los atributos:

1. N, no implementado (0-15%)
2. P, Parcialmente implementado (> 15-50%)
3. L, Ampliamente implementado (> 50-85%)
4. F, completamente implementado (> 85%)

El uso de la escala de calificación permitirá posicionar un proceso en su nivel de capacidad.

Auditoria

Definición

Es una actividad incluida dentro de las disciplinas de soporte, la cual implica una evaluación independiente de productos o procesos de software que permiten asegurar el cumplimiento con estándares, lineamientos, especificaciones y procedimientos basada en un criterio objetivo incluyendo documentación que especifique:

1. La forma o contenido de los productos a ser desarrollados.
2. El proceso por el cual los productos son desarrollados.
3. Como debería medirse el cumplimiento con estándares o lineamientos.

Beneficios de la auditoria

1. Permite evaluar el cumplimiento del proceso de desarrollo.
2. Permite determinar la implementación efectiva de:
    a. Proceso de desarrollo organizacional.
    b. Proceso de desarrollo del proyecto.
    c. Proceso de desarrollo de las actividades de soporte.
3. Proveen mayor visibilidad a la gerencia sobre los procesos de trabajo.
4. Los resultados de las auditorias que solicitan acciones correctivas llevan a la mejora del proceso y del producto, y por ende al crecimiento y a la satisfacción del cliente.

Auditoria de Proyecto

El objetivo de esta auditoria es verificar objetivamente la consistencia del producto a medida que evoluciona a lo largo del proceso de desarrollo, determinando que:

1. Las interfaces de hardware y software sean consistentes con los requerimientos de diseño en la ERS.
2. Los requerimientos funcionales de la ERS se validan en el Plan De Verificación y Validación de Software.
3. El diseño del producto, satisface los requerimientos funcionales de la ERS.
4. El código es consistente con el DDS.

Consideraciones importantes

1. Las auditorías de proyecto se llevan a cabo de acuerdo con lo establecido en el PACS (Plan de Aseguramiento de Calidad de Software), el cual debería indicar la persona responsable de realizar estas auditorías.
2. Las inspecciones de software y las revisiones de la documentación de diseño y prueba deberían incluirse en esta auditoría.

Roles

Si bien existen tres roles en la auditoría, el Gerente de SQA, el auditado y el auditor, los de mayor relevancia son estos últimos dos

Gerente de Aseguramiento de Calidad

El gerente de Aseguramiento de Calidad es el responsable de:

1. Prepara el plan de auditorías: todas las auditorias deben estar planificadas de antemano, y esto queda registrado en el Plan de Calidad del Proyecto, lo mínimo es una de cada tipo por proyecto antes del reléase del producto, pero puede haber más.
    a. El plan puede ser un documento aparte o puede ser una sección del Plan de Proyecto en la sección de Planes de Soporte.
    b. Las organizaciones más pequeñas tienen un plan de calidad general para toda la organización.
    c. Las inspecciones de software forman parte del plan de calidad.
2. Asigna recursos y calcula el costo de las auditorías: determinar los costos en los que se incurrirá para llevar a cabo la auditoria, esto incluye salarios del personal afectado, costos de materiales, incluye tiempos necesarios para llevar adelante las mismas.
3. Responsable de resolver las no-conformidades: es decir que actividades se llevarán adelante para resolver los desacuerdos que existen entre el auditor y el auditado durante el proceso de auditoría.

Auditor

No necesariamente debe ser una persona externa a la organización, sino que puede ser una persona de otra área o sector. No puede tener ningún grado de participación en el proceso auditado, para mantener la objetividad de este proceso. Sus responsabilidades principales son:

1. Acuerda la fecha de la auditoria junto con el auditado.
2. Comunica el alcance de la auditoria.
3. Recolecta y analiza la evidencia objetiva que es relevante y suficiente para tomar conclusiones acerca del proyecto auditado.
4. Realiza la auditoria y preparar el reporte.
5. Realiza el seguimiento de los planes de acción acordados con el auditado.

Durante la auditoria:

1. Escuchar y no interrumpir.
2. No hacer juicios de valor.
3. Observar el lenguaje corporal del auditado.
4. Manejar el propio lenguaje corporal.
5. Manejo del tiempo del auditado.
6. Evitar opiniones personales.
7. Tomar notas y preguntar.

Auditado

En la mayoría de las auditorías quien ocupa el rol de auditado es el Líder de Proyecto. Sus responsabilidades principales son:

1. Acuerda la fecha de la auditoria junto con el auditor.
2. Participa de la auditoria.
3. Proporciona evidencia al auditor.
4. Contesta al reporte de auditoria.
5. Propone el plan de acción para deficiencias citadas en el reporte.
6. Comunica el cumplimiento del plan de acción.

Proceso de auditoria

Preparación y planificación

El auditado solicita la auditoria y junto con el auditor definen la fecha, el alcance y objetivo, acordando los participantes de la misma.

El auditor puede solicitar la documentación preliminar y el auditado prepara la documentación que será requerida.

Ejecución

Existen dos instancias:

1. Reunión de apertura: se presentan los participantes de la auditoria y el equipo auditor. Se revisa lo acordado durante la planificación y se informa cómo será el proceso, acordando el lugar y la hora. En organizaciones más chicas el auditor directamente acuerda un día y pide artefactos definidos con un chek-list para su posterior revisión.
2. Ejecución de la auditoria: Luego se realiza la ejecución propiamente dicha, donde se responden las preguntas, se muestra la evidencia y se completa el checklist.

Análisis y reporte del resultado

1. Evaluación de los resultados:
    a. Se revisa el checklist, las anotaciones y la documentación solicitada.
    b. Se verifican las desviaciones del proceso.
    c. Se realiza un informe preliminar.
2. Reunión de cierre:
    a. Se presenta el reporte preliminar.
    b. Pueden ser solicitadas por parte del auditado, clarificación de resultados
    c. El auditor explica los resultados.
    d. Se llega a un acuerdo con los resultados encontrados.
    e. El auditor define fecha en la que se presentará el reporte final.
3. Entrega del reporte final:
    a. Se actualiza el informe preliminar con las observaciones realizadas en la reunión de cierre.
    b. Se prepara el informe final.
    c. Se genera y se guarda una copia del informe final.
    d. Se envía el resultado al auditado.

Seguimiento

1. Se analizan las desviaciones y prepara un plan de acción que se envía al auditor para que lo revise y apruebe.

2. Hecho esto el auditado aplica y realiza un seguimiento de las acciones e informa el cumplimiento de las mismas.
3. Una vez verificadas por el auditor, se cierran las desviaciones y generan los informes de cierre.

Checklist de auditoria

El checklist de auditoría es una de las herramientas que puede ser utilizada por parte del auditor para llevar adelante el relevamiento necesario de la auditoría, esta básicamente contendrá información del contexto y una lista de preguntas mediantes las cuales el auditor ira guiando la auditoría.

Componentes

1. Fecha de la auditaría
2. Lista de auditados (identificando el rol)
3. Nombre del auditor
4. Nombre del proyecto
5. Fase actual del proyecto (si aplica)
6. Objetivo y alcance de la auditaría
7. Lista de preguntas (cerradas: adecuado, no adecuado, necesita mejora, no aplica)

Preguntas frecuentes

Planificación de proyectos

1. ¿Existe un plan de proyecto y esta actualizado?
2. ¿Existe un responsable para cada actividad?
3. ¿Se han asignado recursos para las actividades de soporte?
4. ¿Están disponibles los planes para todos los involucrados?

El auditor deberá asegurarse:

1. Los planes estén basados en los requerimientos
2. Las actividades planificadas se hayan llevado a cabo
3. Todos los involucrados se han comprometido con la última versión de los planes
4. Los cambios a los planes se hayan aprobado por todos los involucrados
5. La decisión de esos cambios se haya documentado oportunamente
6. Se han identificado y comunicado los riesgos del proyecto

Fase de Requerimientos

1. ¿Existe un documento de especificación de requerimientos?
2. ¿Se han identificado unívocamente los requerimientos?
3. ¿Están descriptos cada uno de los requerimientos?

El auditor deberá asegurarse:

1. Se han revisado y aprobado los requerimientos por parte de todos los involucrados.
2. Si existen cambios a los requerimientos, los mismos han seguido el correspondiente proceso de cambios y se han revisado y actualizado los planes de proyecto

Técnicas utilizadas

Cuestionario

1. Comenzar con preguntas de final abierto (quién, cuándo, cómo, qué, dónde).
2. Realizar preguntas cortas y puntuales.
3. Finalizar con preguntas de final cerrado para clarificar conceptos.

Muestreo

No se puede revisar todo, se toman muestras por lo que es bueno conocer el proyecto para conocer lo esencial. La información obtenida de las entrevistas se contrasta contra las muestras.

Revisión de registros

Se analizan documentación y demás artefactos y se realiza una comparación entre lo visualizado (real) o lo esperado (ideal).

Análisis y reporte de resultados

Reporte de auditoria

El reporte de auditoría es el documento resultante del proceso de auditoría en el cual se registran todos los datos necesarios y recolectados a lo largo de dicho proceso, entre los cuales se pueden destacar:

1. Identificación de la auditaría
2. Fecha de la auditaría
3. Auditor
4. Auditados
5. Nombre del proyecto auditado
6. Fase actual del proyecto
7. Lista de resultados
8. Comentarios
9. Solicitud de planes de acción

Lista de resultados

Existen básicamente tres tipos de resultados en lo que respecta a lo auditado:

1. Buenas prácticas: práctica, procedimiento o instrucción que se ha desarrollado mucho mejor de lo esperado. Se deben reservar para cuando el auditado:
    a. Ha establecido un sistema efectivo
    b. Ha desarrollado un alto grado de conocimiento y cooperación interna
    c. Ha adoptado una práctica superior a cualquier otra que se haya visto
2. Desviaciones: se reportan a partir de la evidencia sin utilizar términos absolutos (ej. Nunca). Estas son las que disparan las solicitudes de acción correctiva o preventiva. Requieren un plan de acción por parte del auditado
    a. Cualquier desviación que resulta en la disconformidad de un producto respecto de sus requerimientos.
    b. Falta de control para satisfacer los requerimientos.
3. Observaciones: condiciones que deberían mejorarse, pero no requieren un plan de acción.
    a. Opinión acerca de una condición incumplida
    b. Práctica que debe mejorarse
    c. Condición que puede resultar en una futura desviación.

Métricas de auditoría

Dependen de las organizaciones, es decir las mismas determinarán que métricas son de más utilidad de acuerdo con sus objetivos, algunas de las métricas son:

1. Esfuerzo por auditaría
2. Cantidad de desviaciones

3. Cantidad de observaciones
4. Duración de auditaría
5. Cantidad de desviaciones clasificadas por PA de CMMI, para conocer las ares de proceso más riesgosas. (Estadísticamente hablando son planificación y gestión de configuración) También hay auditorías al área de calidad que constatan una vez al año que tengan plan de calidad, que las auditorias estén registradas, etc.

Revisiones técnicas

Introducción

Dada la complejidad inherente del software por ser intangible y por ser una actividad intelectual intensiva, se asume que la misma posee muchos errores. En gran parte esto esta dado por los problemas de comunicación al momento de Elicitar requerimientos, no es una cuestión técnica el problema principal.
La validación se ocupa de si hicimos el sistema que teníamos que hacer, o hicimos otro sistema que resuelve otro problema. La verificación mide si el sistema funciona correctamente.
En relación con las auditorias y al testing, cambia el quien lo hace: es un colega, un par.
Las revisiones técnicas suponen un costo por el esfuerzo del recurso afectado para realizar la actividad.

Definición

Es un proceso estático de validación y verificación, cuyo propósito es detectar lo mas pronto que se pueda las fallas, ya que es mas barato corregir mientras antes se detecte el problema.

Origen

Michael Fagan introdujo el concepto de inspecciones en el software basado en su experiencia en HW para detectar defectos lo más cerca posible de su generación. Sin embargo, la inspección es una técnica.

¿Qué se revisa?

Se revisa cualquier producto de trabajo que se genere en el proceso de desarrollo de software. Lo primero se revisaba técnicamente era el código.
Luego las revisiones se aplicaban sobre la arquitectura y a los requerimientos, dado que las fallas en esos productos son mucho mas caros. Las estimaciones también son susceptibles a revisiones, dado su importancia para que el proyecto de software prospere.
Revisar es necesario dado que las personas son las que hacen software y por regla general, las personas erramos.

Ventajas y desventajas

Ventajas

1. Permiten encontrar muchos errores.
2. Pueden inspeccionarse versiones completas.
3. Pueden considerarse otros atributos de calidad.
4. Brindan espacio para el aprendizaje colectivo: entre pares es posible aprender al momento de debatir positivamente del producto de trabajo.

Desventajas

1. Es difícil introducir las inspecciones formales.
2. Sobrecargan al inicio los costos y conducen a un ahorro sólo después de que los equipos adquieran experiencia en su uso.
3. Requieren tiempo para organizarse y parecen ralentizar el proceso de desarrollo.

Tipos de revisiones

Dentro de los métodos de desarrollo de no sala limpia, nos encontramos con dos tipos de revisiones:

1. Revisiones técnicas informales: no existe un proceso de como realizarlo. Walkthrough.
2. Revisiones técnicas formales: tiene un proceso y define roles. Inspección de código de Fagan e inspección de Gilb.

Walkthrough

Técnica de análisis estático en la que un diseñador o programador dirige miembros del equipo de desarrollo y otras partes interesadas a través de un producto de software, donde los participantes formulan preguntas y realizan comentarios acerca de posibles errores, violación de estándares de desarrollo y otros problemas.
No existe un proceso formal, por lo que es mas difícil obtener métricas de este tipo de revisión. Consiste en reuniones informales de colegas donde se debaten las correcciones a aplicar al producto de trabajo.
Tiene los siguientes objetivos:

1. Mínima Sobrecarga
2. Capacitación de Desarrolladores
3. Rápido retorno

Esta técnica no obtiene métricas para aprender y dejar registros. Sin embargo, es una de las técnicas mas elegidas en los enfoques agiles.

Inspecciones

Tiene un proceso formal y cuenta con un conjunto de roles. Es necesario la utilización de un checklist, que ayuda a la memoria para saber que cosas controlar. Se toman métricas y finalmente se realiza un reporte de la revisión al final de la inspección para analizar los defectos encontrados.
Los siguientes son roles que debería revisar los documentos mencionados:

Tipo de documento	Revisores sugeridos
Arquitectura o Diseño de alto nivel	Arquitecto, analista de requerimientos, diseñador, lider de proyecto, testers.
Diseño detallado	Diseñador, arquitecto, programadores, testers
Planes de proyecto	Líder de proyecto, stakeholders, representante de ventas o marketing, líder técnico, representante del área de calidad,
Especificacion de requerimientos	Analista de requerimientos, líder de proyecto, arquitecto, diseñador, testers, representante de ventas y/o marketing
Codigo fuente	Programador, diseñador, testers, analista de requerimientos
Plan de testing	Tester, programador, arquitecto, diseñador, representante del área de calidad, analista de requerimientos

Métricas Sugeridas	Fórmula
Densidad de defectos	Total de defectos encontrados / tamaño actual
Total de defectos encontrados	Defectos.Mayor + Defectos.Menor
Esfuerzo de la inspección	Esfuerzo.Planning + Esfuerzo.Preparación + Esfuerzo.reunion + Esfuerzo.Retrabajo
Esfuerzo por defecto	Esfuerzo.Inspeccion / Total de def encontrados
Porcentaje de reinspecciones	Cantidad Reinspecciones / Cantidad Inspecciones
Defectos Corregidos sobre Total de Defectos.	Esfuerzo.Inspeccion / tamaño actual

Inspección

Es una actividad que garantiza la calidad del software, cuyo éxito depende de la planificación. Tiene como objetivos:

1. Descubrir errores.
2. Verificar que el software alcanza sus requerimientos.
3. Garantizar que el software fue construido de acuerdo con ciertos estándares.
4. Conseguir un software desarrollado de manera uniforme.
5. Hacer que los proyectos sean mas manejables.

SON

- **La forma más barata y efectiva de encontrar fallas**
- **Una forma de proveer métricas al proyecto**
- **Una buena forma de proveer conocimiento cruzado**
- **Una buena forma de promover el trabajo en grupo**
- **Un método probado para mejorar la calidad del producto**

NO SON

- **Utilizadas para encontrar soluciones a las fallas**
- **Usadas para obtener la aprobación de un producto de trabajo**
- **Usadas para evaluar el desempeño de las personas**

Son proceso time boxing y exigen una alto esfuerzo intelectual.

Proceso de inspección

Roles participantes

Al ser una técnica formal, si o si debe contar con los siguientes roles:

1. Autor: creador o encargado de mantener el producto a inspeccionar. Inicia el proceso seleccionado a un moderador y junto a este eligen al resto de los roles. Entrega el producto a ser inspeccionado al moderador.
2. Moderador: planifica y lidera la revisión. Trabaja junto al autor para elegir los demás roles. Entrega el producto a inspeccionar al inspector 2 días antes de la reunión. Coordina la reunión de forma tal que no ocurran conductas inapropiadas y realiza un seguimiento de los defectos encontrados.
3. Anotador: registra los hallazgos de la inspección. Usualmente termina confeccionado el reporte de la revisión.
4. Lector: lee el producto a ser inspeccionado. Este rol es necesario para que los participantes no se dispersen.
5. Inspector: Examina el producto antes de la reunión para encontrar defectos. Registra sus tiempos de preparación. todos pueden ser inspectores. Todos pueden inspeccionar.

Ciertos roles pueden ser asumidos por la misma persona.

Etapas del proceso

1. Planificación: el moderador, a pedido del autor, planifica la inspección definiendo el lugar, el tiempo de duración y los roles. La duración de las reuniones no debe superar las 2 horas, dado que la inspección es un proceso de alto esfuerzo intelectual.
2. Visión general: esta etapa es opcional. El autor realiza una descripción general del producto a inspeccionar.
3. Preparación: es la preparación de cada rol para la reunión. Cada rol adquiere una copia del producto de trabajo que deberá leer y analizar, con el fin de encontrar potenciales defectos. Esta preparación permite que la reunión de inspección sea mas productiva.
4. Reunión de inspección: el equipo realiza un análisis para recolectar los potenciales defectos previos y descartar falsos positivos. El lector lee el producto de trabajo y los inspectores comparten los defectos encontrados, los cuales son registrados por el anotador. La reunión finaliza con una conclusión acerca de si se acepta o no el producto de trabajo inspeccionado. Finalmente se realiza un informe detallando que se reviso, por quien, que se descubrió y que se concluyo.
5. Corrección: finalizada la reunión, el autor realiza las correcciones de los defectos encontradas.
6. Seguimiento: Dependiendo de la gravedad puede existir un proceso de re-inspeccion. En caso de que los defectos sean graves, se realiza nuevamente una inspección. De lo contrario, si el defecto era muy simple, el autor simplemente lo corrige. Otros defectos que no implican una re-inspeccion, pueden implicar que el autor se reuna con el moderador únicamente para tratar las correcciones realizadas

Puntos Claves

- Revisar al producto… no al productor
- Fijar una agenda y cumplirla
- Limitar el debate y las impugnaciones
- Enunciar las áreas de problemas, pero no tratar resolver cualquier problema que se manifieste
- Tomar notas escritas
- Limitar el nro. de participantes e insistir en la preparación por anticipado
- Desarrollar una lista de revisión
- Disponer recursos y una agenda
- Entrenamiento
- Repasar revisiones anteriores

Testing

Definición

Proceso mediante el cual se somete a un software o un componente del mismo a condiciones especificas con el fin de determinar y demostrar si el mismo es valido o no en función de los requerimientos especificados.
El proceso es destructivo, ya que tiene por objetivo encontrar defectos bajo la premisa de que en el software siempre existen defectos.

Características

1. No asegura la calidad del producto ni soluciona los defectos encontrados.
2. Verifica que el software este bien implementado y valida que sea correcto.
3. Puede empezar en cualquier momento desde que están definidos los requerimientos.
4. Representa entre un 30% y 50% del costo del desarrollo de software.

Principios del Testing

1. El testing muestra la presencia de defectos: las pruebas permiten demostrar que los problemas existen y no pueden demostrar que los problemas no existen, ya que en el software se asume de que siempre hay defectos.
2. El testing exhaustivo es imposibles: dada la complejidad del software, realizar pruebas que tengan una cobertura del 100% es imposible, ya que probar todas las combinaciones posibles suele ser un problema intratable. Por lo contrario, el testing se debe enfocar en las funcionalidades criticas del sistema y fijar un esquema de prioridades.
3. Testing temprano: los defectos se deben identificar lo mas rápido posible, de esta manera el costo para corregirlo es menor.
4. Agrupamiento de defectos: los defectos en el software suelen agruparse en un conjunto limitado de módulos o áreas (80/20). Bajo este principio, las pruebas deben priorizarse y enfocar el esfuerzo en ese conjunto acotado de funcionalidades.
5. Paradoja del Pesticida: con el tiempo el software se vuelve inmune a las pruebas repetitivas, esto invisibiliza los defectos. Refinar los casos de prueba podría ser una forma de reducir este efecto.

6. El testing es dependiente del contexto: las pruebas que se realizan para una funcionalidad de un producto son especificas para ese producto. Las pruebas no se pueden extrapolar por mas genérica que sea la funcionalidad a probar, como por ejemplo el login en una aplicación.
7. Falacia de la ausencia de errores: que la prueba no haya encontrado defectos no implica que el software este libre de ellos, mas aun teniendo en cuenta la premisa de que en el software siempre existen defectos.

Otros principios son:

1. Un programador debería evitar probar su propio código, dado que los humanos somos optimistas y no queremos equivocarnos, por ende, siempre vamos a probar nuestras implementaciones con sesgo.
2. Una unidad de programación no debería probar sus propios desarrollos.
3. Examinar el software para probar que no hace lo que debería hacer es la mitad, también se debe probar que hace lo que no debería hacer.
4. No se debe estimar el esfuerzo del testing sobre la suposición de que no se van a encontrar defectos.

¿Cuánto testing es suficiente?

Dado que el testing exhaustivo es imposible, se debe determinar cuando finalizar el proceso. Esto puede depender de una evaluación a nivel de riesgo o del costo asociado al proyecto. Los riesgos permiten definir prioridades de que testear primero y con que esfuerzo.

El criterio de aceptación se utiliza normalmente para decidir si una determinada fase de testing ha sido completada. Este puede ser definido en términos de:

1. Costos
2. % de tests corridos sin fallas
3. Inexistencia de defectos de una determinada severidad.
4. Pasa exitosamente el conjunto de pruebas diseñado y la cobertura estructural.
5. Good Enough: Cierta cantidad de fallas no críticas es aceptable.
6. Defectos detectados es similar a la cantidad de defectos estimados.

Error

Un error es una falla que se identifica en la misma etapa en la cual se origina, es decir originada en el producto de trabajo actual.

Defecto

Un defecto es una falla que se identifica en una etapa posterior a la etapa en la cual se origino, es decir originado en un producto de trabajo anterior.

Los defectos son mas costosos de corregir que los errores, ya que se deben corregir aspectos anteriores del software, que pueden afectar al desarrollo posterior de la falla.

Severidad del defecto

Grado de impacto que tiene la ocurrencia de un defecto sobre el desarrollo u operación de un componente de sistema. Esta asociado a aspectos técnicos y suele ser definida por el tester. Niveles de severidad:

1. Bloqueante: disrupción del servicio
2. Critico: defecto en una funcionalidad critica del sistema para el cual no existe una solución alternativa.
3. Mayor: defecto en una funcionalidad principal para el cual existe una solución alternativa simple o no deseable.
4. Menor: defecto en una funcionalidad no principal para el cual existe una solución alternativa simple.

5. Cosmético: defecto que no afecta a la funcionalidad y que no requiere de soluciones alternativas. Error ortográfico.

Prioridad del defecto

Grado de importancia que le asigna el cliente a la corrección del defecto.

1. Urgente: el defecto debe resolverse inmediatamente.
2. Alta: el defecto debe resolverse tan rápido como sea posible sin afectar al curso normal de desarrollo.
3. Media: el defecto puede resolverse, pero puede esperar a que se resuelvan otros de mayor prioridad.
4. Baja: el defecto podría o no resolverse.

Niveles de testing

Testing Unitario

Son pruebas que se realiza el programador sobre cada componente de código, las cuales se realizan de manera individual, es decir que son independientes del resto de componentes.

Se produce con acceso al código bajo pruebas y con el apoyo del entorno de desarrollo, tales como un framework de pruebas unitarias o herramientas de depuración.

Los errores se reparan tan pronto como se encuentran, sin dejar constancia oficial del incidente.

Una clase o un método pueden ser componentes a probar.

Testing de Integración

Son pruebas orientadas a verificar que las partes de un sistema que funcionan bien aisladamente, también lo hacen en conjunto. Suelen probarse las interfaces que relacionan a las partes.

Las estrategias de pruebas de integración son incrementales:

1. De lo mas general a lo mas particular (Top Down)
2. De lo mas especifico a lo mas general (Buttom up)

Lo que se busca es una combinación de ambos esquemas, y es importante tener en cuenta que:

1. Los módulos críticos deben ser probados lo más tempranamente posible.
2. Se deben conectar de a poco las partes más complejas
3. Minimizar la necesidad de programas auxiliares.

Testing de sistema

Son las pruebas que se aplican sobre la construcción final del sistema, donde la aplicación funciona como un todo.

Busca determinar si el sistema en su globalidad opera satisfactoriamente, es decir cumple con requerimientos funcionales y no funcionales.

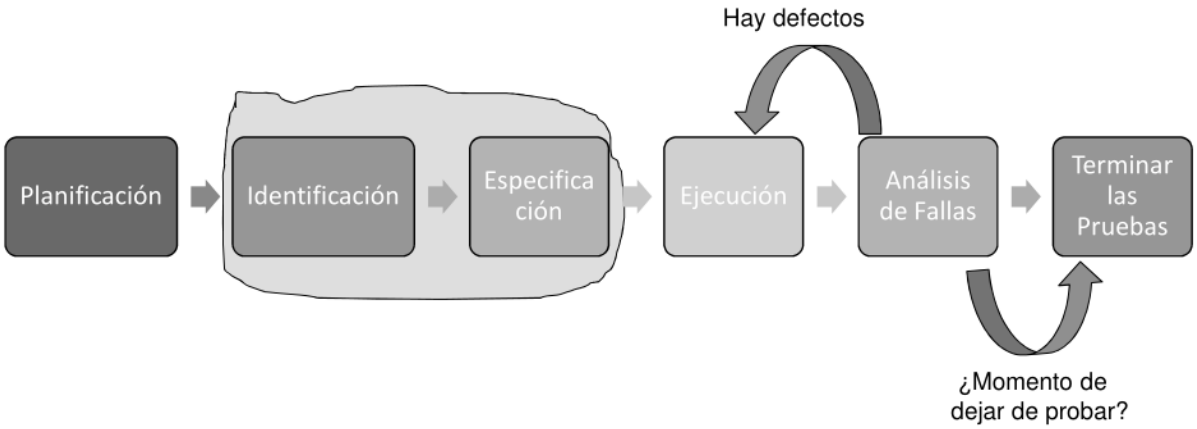
El entorno en que se prueba debe ser lo más similar posible al entorno de producción, para reducir al mínimo el riesgo de incidentes debido al ambiente.

Testing de aceptación

Son pruebas realizas por el usuario para validar si se ajusta a sus necesidades. El objetivo no es encontrar defectos, sino establecer confianza con características especificas y no funcionales del sistema.

Comprende tanto la prueba realizada por el usuario en ambiente de laboratorio (pruebas alfa), como la prueba en ambientes de trabajo reales (pruebas beta).

Proceso del testing



Planificación y control

La planificación consiste en verificar que se comprenden los objetivos del cliente, las partes interesadas, el proyecto y los riesgos de las pruebas que se pretenden ejecutar.

El resultado de la planificación es el Plan de Pruebas, que debe contener:

1. Riesgos y objetivos del Testing.
2. Estrategia de Testing.
3. Recursos.
4. Criterio de Aceptación.

En lo que respecta a control se debe:

1. Revisar los resultados del testing.
2. Controlar la cobertura del testing y el criterio de aceptación
3. Tomar decisiones en función de esto.

Identificación y especificación

Esta etapa incluye las siguientes actividades:

1. Verificación de las especificaciones del software bajo prueba.
2. Evaluación de la testeabilidad de los requerimientos.
3. Identificación de los datos necesarios.
4. Diseño y priorización de casos de prueba.
5. Diseño del entorno de pruebas.

Ejecución

Esta etapa incluye las siguientes actividades:

1. Creación de los datos necesarios para la prueba.
2. Automatizar lo que sea necesario.
3. Implementar y verificar el ambiente.
4. Ejecutar los casos de prueba.
5. Registrar los resultados de la ejecución.
6. Comparar los resultados reales con los esperados.

Evaluación y reporte

Esta etapa incluye las siguientes actividades:

1. Evaluar el criterio de aceptación.

2. Reporte del resultado de las pruebas a los interesados.
3. Verificar los entregables y que los defectos se hayan corregido.
4. Evaluación de cómo resultaron las actividades de testing y se analizan las lecciones aprendidas.

Casos de prueba

Es un conjunto de condiciones o variables bajo las cuales el tester determinara si el software funciona correctamente o no. Definen que se debe hacer, bajo este conjunto de condiciones, para obtener un resultado esperado.
Consta de 3 partes: objetivo, datos de prueba y resultado esperado.

Características

1. Es la unidad de la actividad de la prueba.
2. Debe estar definido de manera clara y comprensible, de tal forma que cualquier persona no experta pueda llevarlo a cabo.
3. La buena definición de los casos de prueba permite reproducir defectos
4. Se pueden derivar de:
    a. Documentos del cliente.
    b. Información de relevamiento.
    c. Requerimientos.
    d. Especificaciones de programación.
    e. Código.

Condición de prueba

Reacción esperada de un sistema frente a un estimulo en particular, el cual esta constituido por distintas entradas.
Una condición de prueba debe ser probada por al menos un caso de prueba.

Partes de un caso de prueba

1. Objetivo: la característica del sistema a probar.
2. Datos de entrada y de ambiente: datos a introducir en el sistema y aquellos que se encuentran en condiciones preestablecidas.
3. Comportamiento esperado: salida o acción esperada del sistema de acuerdo con los requerimientos del mismo.

Métodos de Testing

El objetivo por el cual se desarrollaron los métodos de testing fue para optimizar el tiempo aplicado a esta actividad. Bajo la premisa de que el testing exhaustivo es imposible, existen ciertos métodos que determinan que aspectos se deberían probar y en qué hacer foco, en definitiva, porque el tiempo y el presupuesto son limitados. Todo esto anteriormente mencionado se resume con la siguiente frase:

"Hay que maximizar la cobertura del testing, pasando por la mayor cantidad de funcionalidades del sistema, con la menor cantidad de casos de prueba".

Caja negra

Son métodos dinámicos en donde se le aplican al sistema una serie de entradas y en base a esto se analizan las respuestas que entrega el mismo, sin tener en cuenta la funcionalidad interna del sistema. Es decir, nos enfocamos en que hace sin tener en cuenta como lo hace.

Se clasifican en basados en especificaciones y basados en experiencia.

Métodos basados en especificaciones

Son aquellos que se ejecutan utilizando la documentación de especificaciones realizadas del producto.

Partición de equivalencias

Es un proceso sistemático que consiste en identificar clases de equivalencia, las cuales definen subconjuntos de datos que producen un resultado equivalente. Consta de dos pasos:

1. Identificación de clases de equivalencia validas y no validas: identificar para cada condición externa de entrada y salida, clases de equivalencias validas y no validas.
2. Identificación de casos de prueba:
    a. Hasta cubrir todas las clases validas, diseñar un caso de prueba que cubra la mayor cantidad de clases validas aun no cubiertas.
    b. Hasta cubrir todas las clases invalidas, diseñar un caso de prueba que cubra la mayor cantidad de clases invalidas aun no cubiertas.

Análisis de los valores limites

Cuando se define un caso de prueba es necesario especificar los datos particulares que se utilizaran en su ejecución. El análisis de los valores limites consiste en utilizar los limites o valores de borde de las clases de equivalencia para la definición de los casos de prueba.

Métodos basados en experiencia

Son aquellos donde la experiencia y los conocimientos del tester son fundamentales para determinar las entradas del sistema y analizar los resultados.

Adivinanza de defectos

Enfoque basado en la intuición y experiencia para identificar pruebas que probablemente expongan defectos del software. Se elabora una lista de defectos posibles o situaciones propensas a error y luego se realizan pruebas a partir de esa lista. Suele ser útil disponer de un historial de defectos.

Testing exploratorio

Su principal característica es que el aprendizaje, el diseño y la ejecución de las pruebas se realizan de forma simultánea, es decir el tester mientras va probando el software, va aprendiendo a manejar el sistema y junto con su experiencia y creatividad, genera nuevas pruebas a ejecutar.

Caja blanca

Son métodos estáticos que se basan en analizar la estructura interna del software o de un componente de software. Con este método se puede garantizar el testing coverage, es decir que todos los caminos de un modulo han sido recorridos al menos una vez.

Complejidad ciclomática

Métrica de software que provee una medición cuantitativa de la complejidad de un programa. En el testing define el número de caminos independientes en el conjunto básico y entrega un límite inferior para el número de casos necesarios para ejecutar todas las instrucciones al menos una vez.
Para hallar la complejidad ciclomática, los programas se presentan como un grafo
en donde cada instrucción se convierte en un nodo del grafo.

$$M = G – N + 2*P$$

M = Complejidad ciclomática (N° de regiones cerradas + 1)

G = Número de aristas del grafo
N = Número de nodos del grafo
P = Número de componentes convexos, nodos de salida

Métodos

Cobertura de enunciados o caminos básicos

Se debe calcular la complejidad ciclomática y diseñar casos de prueba que permitan pasar por todos los caminos independientes que conformar el conjunto básico.

Cobertura de secuencia o sentencia

Se deben recorrer todas las sentencias de todos los caminos lógicos.

Cobertura de decisión

Recorrer cada decisión al menos una vez por ambas ramas.

Cobertura de condición

Verificar que cada condición en una decisión tenga todos los valores posibles.

Cobertura de decisión/condición

Verificar que cada condición en una decisión tenga todos los valores posibles y que se recorre cada decisión por ambas ramas.

Cobertura múltiple

Probar todas las combinaciones posibles de valores de las condiciones de una decisión.

Cobertura de bucles

Focalizada en la validez de los loops, sean simples, anidados, concatenados o no estructurados

Ciclos de prueba

Un ciclo de pruebas consiste en la ejecución de la totalidad de los casos de prueba establecidos aplicados a una misma versión del sistema a probar.
Los ciclos de test se pueden llevar a cabo con o sin regresión.

Regresión

Al trabajar con regresión, por cada defecto corregido, se deben volver a ejecutar todos los Casos de prueba, inclusive aquellos que habían sido exitoso, para evitar que se propaguen errores sin que nos demos cuenta.

Tipos de pruebas

Testing funcional

Son aquellas pruebas que se basan en funciones o características y prueba lo que el sistema hace. Pueden ser:

1. Basadas en Requerimientos u otras especificaciones: utilizan a los requisitos definidos en una ERS o los acuerdos que contienen las pruebas de usuario y los criterios de aceptación de una US para realizar las pruebas.
2. Basadas en los proceso de negocio: se realizan de acuerdo con el conocimiento basado en el uso comercial del producto de software

Testing no funcional

Estas pruebas buscan determinar “como” funciona el sistema, en lugar de comprobar la funcionalidad que brinda el mismo “el que”. Es decir, son aquellas que hacen foco en los requerimientos no funcionales. Es

importante no olvidarlas, dado que los requerimientos no funcionales son tan importantes como los funcionales.

Pruebas de Performance

Se busca determinar cuál es su capacidad de respuesta y la estabilidad del sistema.

1. Pruebas de Carga: se realizan generalmente para observar el comportamiento de una aplicación bajo una cantidad de peticiones esperadas.
2. Pruebas de Stress: se utiliza normalmente para romper la aplicación. Se duplica la cantidad de peticiones carga hasta que se rompe. Permite determinar la robustez

Pruebas de usabilidad

Son pruebas centradas en los usuarios mediante las cuales se evalúa el producto a través de pruebas con usuarios reales, permitiendo medir el grado de facilidad que tienen para usar el sistema.

Pruebas de mantenimiento

Estas pruebas están directamente orientadas a medir lo que se conoce como la mantenibilidad del sistema, es decir que tan bien el sistema se adaptaría a cambios necesarios.

Pruebas de fiabilidad

Son aquellas pruebas que buscan determinar que tan bien una aplicación proporciona, sin errores, los servicios que se establecieron en las especificaciones originales

Pruebas de portabilidad

Tipo de prueba en el cual se busca determinar la capacidad que tiene el producto de software de ejecutarse en diferentes plataformas.

Pruebas de interfaz de usuario (UI)

Tiene como objetivo garantizar que la interfaz gráfica de usuario de la aplicación cumpla con las especificaciones.

Pruebas de seguridad

Tiene como objetivo probar que el sistema de información garantiza la confidencialidad, integridad y autenticidad. Las pruebas de vulnerabilidad y de penetración componen a estas pruebas.

Smoke test

Primer corrida de los test de sistema que provee cierto aseguramiento de que el software que está siendo probado no provoca una falla catastrófica.

Testing en ambientes agiles

Introducción

El testing ágil es una practica que se incluye en el conjunto de las pruebas del software en el contexto del desarrollo ágil de un producto.

De hecho, las pruebas ágiles surgieron abarcando los principios y valores explicitados en el manifiesto ágil, orientado a entregar frecuentemente software de calidad para la satisfacción de las necesidades del cliente.

Esta concepción, esta mentalidad, implica la utilización de técnicas y prácticas de testing adaptadas a la forma del desarrollo ágil.

Manifiesto de Testing

Valores del Testing Ágil

1. Testing durante sobre testing al final: la actividad del testing debe ser continua durante del desarrollo de una funcionalidad del producto y no debe realizarse únicamente al finalizar la iteración. Esto implica que el desarrollo y la prueba se hacen de manera simultanea.
2. Prevenir defectos sobre encontrar defectos: los defectos en el contexto de los procesos empíricos son un desperdicio, por lo tanto, el enfoque ágil prioriza la prevención de los defectos. Esto se logra mediante la colaboración y discusión positiva entre testers y desarrolladores, los cuales realizar revisiones frecuentes. Es necesario tener claro que hay que hacer, como y porque, con el fin de eliminar cualquier tipo de suposición.
3. Entender lo que se esta probando por sobre verificar funcionalidad: esto implica comprender la necesidad de lo que el usuario realmente quiere, para que lo quiere y como el lo va a usar, con el fin de realizar pruebas que den valor agregado, en lugar de solo validar y verificar que las especificaciones se cumplen en términos de cheklist de forma mecánica.
4. Construir el mejor sistema por sobre romper el sistema: invertir el esfuerzo del equipo en generar el ambiente adecuado para mejorar lo que se esta haciendo, en lugar de intentar romper el sistema.
5. El equipo es responsable de la calidad por sobre el tester es el único responsable: en agile la clave de entregar un producto con calidad es que el equipo se comprometa con todas las actividades y trabajen conjuntamente para generar valor, por eso es importante que la responsabilidad sobre la calidad sea del equipo por sobre que la calidad recaiga sobre un solo rol como es el del tester

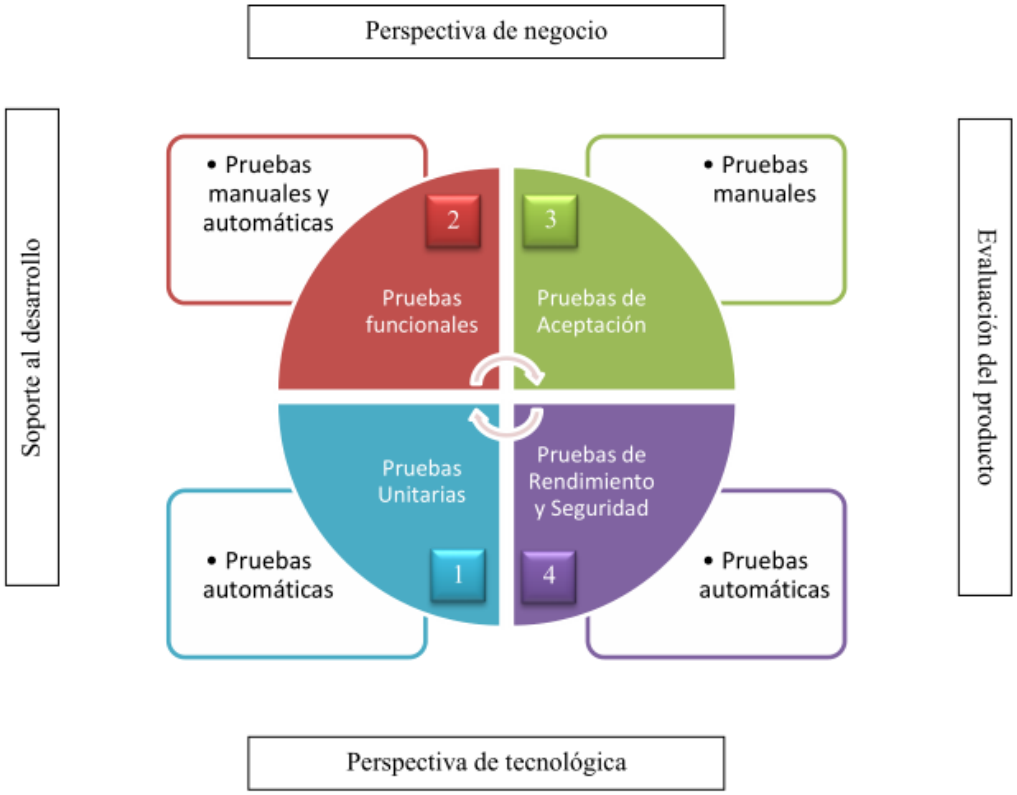
Principios del testing

1. Testing se mueve hacia adelante en el proyecto
2. Testing no es una fase: el testing continuo es la única forma de garantizar el avance continuo, por lo cual el testing se realiza de forma conjunta con el desarrollo del software y demás actividades. V1.
3. Todos hacen testing: la responsabilidad de probar al software recae sobre el equipo, tanto los analistas como los desarrolladores hacen testing, V5.
4. Reducir la latencia del feedback: el cliente esta involucrado durante todo el ciclo de iteración y no solo en la fase final para realizar las pruebas de aceptación. Esto permite reducir el tiempo de retroalimentación y como consecuencia es mas fácil prevenir los defectos y el costo de corrección de los mismos es menor. V2, V3.
5. Las pruebas representan expectativas: las pruebas que se realizan al software están basadas en la comprensión de la necesidad real del usuario. Existe un análisis previo de para que lo quiere y como lo va a usar. De esta manera, las pruebas cuentan con un valor agregado fundamental para el usuario, ya que incluye sus expectativas. V3.
6. Mantener el código limpio y corregir los defectos rápido: los defectos en el código se corrigen en la misma iteración. Esto permite mantener al código limpio. V4.
7. Reducir la sobrecarga de documentación de pruebas: el testing ágil se enfoca en lo esencial de las pruebas, utilizando listas de checkeo reutilizables en lugar de la documentación extensa y los detalles. Estas listas son el inicio y no el fin del testing, quien realiza as pruebas puede aportar el valor que considere necesario.
8. Las pruebas son parte del “done”: esto implica que el desarrollo de una funcionalidad de producto será considerado como terminada si y solo si la misma fue testeada.

9. Guiado por pruebas: las pruebas se hacen durante el desarrollo. Se aplica la técnica de desarrollo conducido por pruebas.

Cuadrantes del testing

Es una disposición grafica que visualiza diferentes pruebas a realizar en el entorno ágil en función del enfoque y propósito de las pruebas y del grado de automatización que se requiere para llevarlas a cabo.



Pruebas de soporte al desarrollo

Estas pruebas primero guían el desarrollo de la funcionalidad, esperando que el software hace lo que realmente tiene que hacer. Luego cuando se automatizan, sirven para apoyar la refactorización y la inclusión de nuevo código sin causar resultados inesperados en el comportamiento del sistema.

Pruebas de evaluación o critica al producto

En principio se tienen las pruebas de aceptación, a través de las cuales el cliente evalúa al producto validando si hace lo que realmente tiene que hacer. Permiten determinar el valor de negocio inmerso en el producto.
Únicamente pueden ser ejecutadas de forma manual, ya que se prueba el software desde el punto de vista del usuario final.
Luego se realizan pruebas para evaluar al software desde el punto de vista tecnológico. Las pruebas de carga, de estrés y de seguridad son ejemplos de tipos de pruebas que se realizan en ese cuadrante.

Practicas del testing ágil

Para materializar los principios agiles descriptos, se utilizan una serie de practicas que se aplican durante el desarrollo del producto.

Desarrollo Conducido por Pruebas (TDD)

Es una técnica de desarrollo del software que involucra dos practicas:

1. Test first development: en esta técnica primero se escriben las pruebas unitarias referentes a la característica de producto a implementar. Definido los test unitarios, se piensa en la codificación necesaria para que las pruebas se ejecuten con éxito. Dado que los test unitarios prueban unidades concretas de código, esta técnica obliga al desarrollador a modularizar su codificación haciendo que los métodos o clases a probar tengan una única responsabilidad. Además de pruebas unitarias, pueden incluirse en primer instancia pruebas de integración.
2. Refactoring: esta tecnica consiste en reestructurar el código existente sin modificar el comportamiento que el mismo provee. Implica la mejora de aspectos no funcionales del software, cuyo objetivo es mejorar la claridad del código y reducir su complejidad, con el fin de hacerlo mas mantenible.

Desarrollo Conducido por Pruebas de Aceptación (ATDD)

Es una practica en la que todo el equipo de trabajo analiza de forma conjunta los criterios y las pruebas de aceptación antes de comenzar la implementación de la funcionalidad de producto. Esta técnica la podemos resumir en dos etapas:

1. Antes de implementar un requerimiento, los miembros del equipo colaboran para diseñar escenarios de como se comportará el sistema frente a dicha necesidad.
2. Luego, el equipo de desarrollo convierte estos escenarios en pruebas de aceptación automatizadas mediante frameworks como Cucumber o Selenium en aplicaciones web.

Esta técnica permite que el equipo tenga un conocimiento común de que es lo que hay que hacer y el desarrollo en definitiva esta conducido por las necesidades del usuario, manifiestas en los criterios de aceptación.

Automatización de pruebas unitarias y de integración

Estas pruebas según la pirámide de Cohn, deben estar automatizadas, ya que el costo de hacerlo es relativamente bajo y sus beneficios son diferenciales. Los test unitarios permiten detectar defectos en porciones pequeñas de código y son la base del resto de pruebas. Si un defecto ocurre en este nivel, entonces se puede corregir rápidamente antes de pasar a los test de integración.

Una vez probado los componentes individuales de código, se deben probar la integración de las partes. Es decir, los componentes trabajando en conjunto.

Estos tests deberían ser incluidos en los servidores de automatización, de forma tal que al compilar el software se ejecuten y rápidamente poder obtener una retroalimentación de sus resultados.

Pruebas de regresión a nivel de sistema automatizadas

Las pruebas de regresión comprueban esencialmente si la funcionalidad anterior de la aplicación está trabajando coherentemente y que los nuevos cambios integrados no han introducido nuevos errores en la aplicación. En el desarrollo ágil, las pruebas de regresión se vuelven más relevantes ya que hay cambios continuos que se añaden a la aplicación. La automatización de estas pruebas, o al menos partes de estas pruebas, implican un gran ahorro en costo, ya que su ejecución manual para los sistemas que reciben muchos cambios, implican a un gran esfuerzo y por consiguiente un alto costo.

Pruebas exploratorias

Su principal característica es que el aprendizaje, el diseño y la ejecución de las pruebas se realizan de forma simultánea, es decir el tester mientras va probando el software, va aprendiendo a manejar el sistema y junto con su experiencia y creatividad, genera nuevas pruebas a ejecutar.

Depende en gran medida de las habilidades del tester y no se prueba por probar, sino que se fija un objetivo general antes de comenzar y se limita el tiempo de la actividad para mantener la productividad.

Repositorio que contenga el sistema y las pruebas

Se debe mantener un sistema de control de versiones que gestione el código y las pruebas automatizadas. La integración continua como practica ideal, promueve la utilización de un sistema de automatización que automatice las compilaciones frente a cada integración que realiza el equipo de desarrollo. Durante la compilación, el servidor ejecuta los test automatizados incluidos en el repositorio y en función de los resultados determina si la compilación pasa o falla.

Roles y competencias del tester

Mentalidad

Un Tester que se desenvuelve en un equipo de trabajo bajo metodologías ágiles, necesita trabajar en colaboración con otros Testers, desarrolladores de software e interesados del área de negocio del software que se está construyendo. De hecho, en los equipos agiles todos pueden hacer tarea de testers.

Mas haya de los conocimientos técnicos como los de automatizar pruebas y las habilidades para llevar a cabo pruebas exploratorias, se requiere un cambio de mentalidad. Los testers agiles deben entender el concepto de utilizar las pruebas para documentar los requerimientos y para conducir el desarrollo.

1. El tester debe velar por la construcción del mejor sistema posible. El éxito de los proyectos viene del resultado de buenas personas haciendo bien su trabajo.
2. El tester ágil no se ve así mismo como un garante de la calidad del producto, sino que esta listo para relacionarse con el cliente y ayudarlo a entender cuales son realmente sus necesidades para poder desarrollar las características que ellos necesitan.
3. El tester ágil tiene una mentalidad creativa y es un apasionado por agregar valor al producto de manera oportuna.
4. Ayuda al desarrollo y a entender cual es el objetivo del trabajo que satisface las necesidades del cliente.

Principios

1. Proveer retroalimentación continuamente: una de las contribuciones mas importantes del tester ágil es ayudar al dueño del producto a articular los requerimientos para cada historia en forma de ejemplos o pruebas.
2. Entregar valor a los clientes: el tester debe ser capaz de identificar aquello que realmente agrega valor al cliente. Elimina los desperdicios que suponen el agregado de funcionalidad no requerida que pueda comprometer el objetivo de la iteración.
3. Posibilitar la comunicación cara a cara: La comunicación cara a cara no tiene sustituto. El tester busca continuamente al cliente y a los miembros técnicos del equipo para discutir y colaborar.
4. Coraje: para fallar uno mismo, para permitir fallar al resto y para pedir ayuda. Esta es la única manera de aprender.
5. Simplicidad: El tester ágil y el equipo en sí tienen como desafío no solamente producir la implementación del software lo mas simple posible sino también tomar el objetivo más simple para asegurar que el software cumple con los requerimientos del cliente
6. Mejora continua: buscar las mejores formas para hacer el mejor trabajo.
7. Responder a los cambios: la predisposición y aceptación de que los requerimientos cambian, por lo que aquellos que fue testeado en esta iteración, puede ser testeado en la siguiente como consecuencia de un cambio de requerimiento
8. Autoorganizados: tienen las habilidades y conocen cuando pedir ayuda a un equipo de trabajado multifuncional.

9. Foco en la gente: los proyectos son exitosos cuando existe gente buena trabajando. La calidad del producto recae sobre todo el equipo
10. Disfrutan: trabajar en un grupo donde todos colaboran y se sienten identificados con que el producto a desarrollar tiene que ser de valor para el cliente, hace que el equipo disfrute de trabajar de esa manera.

Cambios culturales

La cultura de los equipos y organizaciones puede ser un impedimento para la aplicación de metodologías agiles.

Cultura organizacional

La cultura organizacional esta definida por los valores, las normas y los supuestos. Determina como se relacionan, comunican y toman decisiones las personas. Esto se pude analizar viendo el comportamiento de los empleados.

Usualmente esta cultura no es tenida en cuenta al momento de implementar un proceso ágil en la organización y como consecuencia los resultados no son los esperados. Los equipos acostumbrados a trabajar de una determinada manera, ofrecen resistencia al cambio ya que supone una amenaza a su trabajo y a la productividad del mismo.

La filosofía de la calidad

En algunas organizaciones prevalecen equipos de testing fuertes e independientes, que supuestamente son los encargados de asegurar la calidad del producto. Estos podrían considerar que la filosofía ágil sea lo contrario a la calidad, ya que modifican el esquema de trabajo y asumen que la calidad del producto debe recaer sobre el equipo completo.

Pensamiento del tester

Aquellos testers que vienen de empresas en las cuales prevalece la confrontación entre desarrolladores y testers, deben transitar un cambio de mentalidad. La actitud debe ser positiva para la construcción del mejor sistema.

Deben salir de su zona de confort y aprender nuevas habilidades como la automatización de las pruebas.

Tamaño de las organizaciones

Las organizaciones grandes suelen tener una gran cantidad de niveles jerárquicos, por lo que la comunicación se dificulta mas. Sobre todo en aquellas que tienen el área de desarrollo y de testing separada. Se deben generar mecanismos para que la comunicación fluya, dado que es una de las bases del agilismo: la interacción entre las personas.

Otro problema que se presenta en estas situaciones es que los clientes no suelen ser accesibles por el equipo durante todo el ciclo de vida de desarrollo. Por lo que dificulta el desarrollo ágil.

Filosofía Lean

Lean contribuye con una gestión eficiente de la producción y permite la reducción de los siguientes costos:

1. Producción en exceso y stock.
2. Pasos extra en los procesos.
3. Búsqueda de información.
4. Espera y transporte.

Desperdicios en el software

Previamente mencionamos que un desperdicio es todo aquello que no agrega valor al cliente, por lo tanto, Lean propone eliminarlos con su filosofía de gestión. Los desperdicios en el software son los siguientes:

1. Características extras: “ya que estamos”.
2. Trabajo a medias: al igual que en metodologías agiles, las tareas están completas o incompletas. No hay grises.
3. Proceso extra y movimiento.
4. Defectos: implican un retrabajo y perdida de tiempo.
5. Esperas: el equipo de testing espera a que el de desarrollo termine con su trabajo.
6. Cambio de tareas.

Kanban

Kanban es una palabra de origen japonés que significa tarjeta, su concepto ha evolucionado hasta convertirse en señal, y se puede definir como un sistema de flujo que permite, mediante el uso de señales, la movilización de unidades a través de una línea de producción mediante una estrategia pull.
Un sistema de flujo pull consiste en optimizar los inventarios y el flujo del producto de acuerdo con el comportamiento en tiempo real de la demanda.

Principios del Kanban

1. Visualizar el flujo: hacer que el trabajo sea visible para todos. Esto brinda transparencia al equipo de trabajo.
2. Limitar el Trabajo en progreso (WIP): restringir la cantidad máxima de elementos de trabajo en las diferentes etapas (columnas del tablero Kanban) del flujo de trabajo. Esto asegura que el equipo mantendrá un ritmo de trabajo óptimo sin exceder su capacidad de trabajo
3. Administrar el flujo: visualizar el trabajo por hacer y lo que actualmente se esta haciendo, permite, en combinación con el limite de WIP, gestionar el flujo de trabajo y hacer que el mismo sea continuo e ininterrumpido.
4. Hacer explicita las políticas: no puede mejorar algo que no se entiende. Esta es la razón por la cual el las políticas deben estar bien definidas, publicadas y promovidas.
5. Mejorar colaborativamente: la forma de lograr la mejora continua y el cambio sostenible dentro de una organización se consigue a través de la visión compartida y la comprensión colectiva de los problemas que deben superarse.

Kanban en el desarrollo de software

Kanban es un método para introducir cambios en un proceso de desarrollo de software o una metodología de administración de proyectos y no es ninguna de estas ultimas.
Aprovecha los conceptos Lean:

1. Especifica el valor desde la perspectiva del cliente.
2. Limita el trabajo en progreso
3. Identifica y elimina desperdicios y las barreras en el flujo
4. Promueve la cultura de mejoramiento continuo.

¿Cómo aplicar Kanban?

1. Modelar el proceso existente.
2. Decidir la unidad de trabajo y los tipos de trabajo, asignando la capacidad en función de la demanda.

3. Dividir el trabajo en piezas, por ejemplo, en User stories.
4. Definir un conjunto de columnas con nombres representativos para identificar donde esta cada ítem en el flujo de trabajo.
5. Asignar límites explícitos de cuántos ítems puede haber en progreso en cada estado (columna) del flujo de trabajo.
6. Definir políticas de calidad y clases de servicio. Una clase de servicio es una clasificación de tareas que deben ser gestionadas de una manera similar y cual tiene definidas un conjunto de políticas, por ejemplo la clase de servicio Urgente, la cual tiene las siguientes políticas asociadas:
    a. Color de tarjeta: Blanco.
    b. WIP = 1.
    c. Los demás trabajos se ponen en espera.
    d. Se puede exceder el limite de WIP para procesar este trabajo.
    e. Se puede instalar en producción de ser necesario.
7. Distribuir el trabajo en las columnas. El trabajo fluirá de izquierda a derecha.
8. Cuellos de botella y flujo: se debe asignar recursos a los cuellos de botella.

Métricas claves de Kanban

Cycle Time (Tiempo de ciclo)

Es una vista interna que mide el tiempo que transcurre desde el inicio hasta el fin del proceso, para un ítem de trabajo dado. Se suele medir en días de trabajo o esfuerzo.
Es una medición mecánica de la capacidad del proceso que define el ritmo de terminación.

Lead Time (Tiempo de entrega)

Es una vista externa que mide el tiempo que transcurre desde que se solicita un ítem de trabajo hasta el momento de su entrega (finalización del proceso). Se mide en días de trabajo.
Define el ritmo de entrega, el cual le interesa al cliente.

Touch Time (Tiempo de tocado)

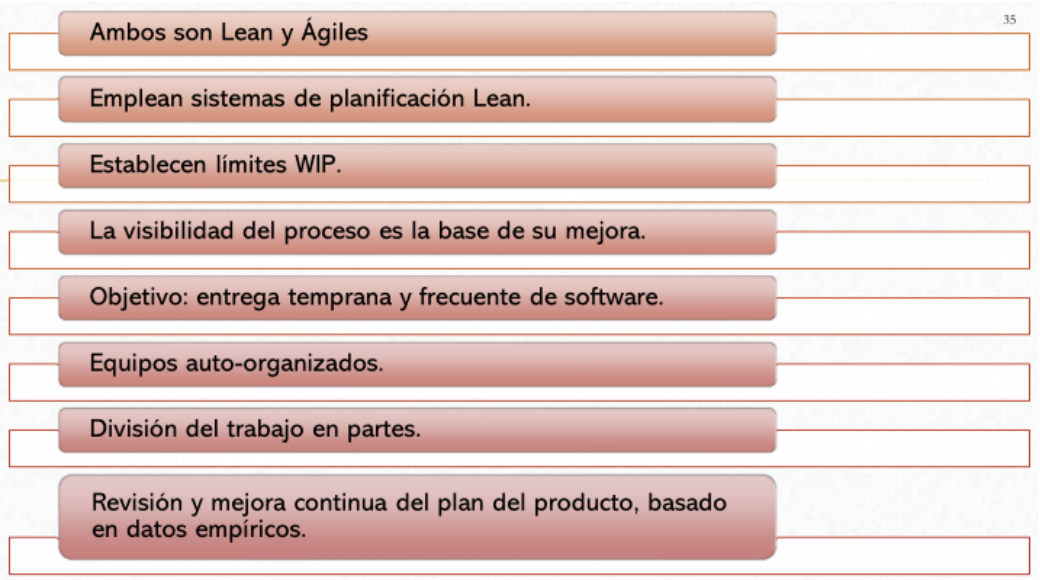
Es el tiempo durante el cual un ítem de trabajo fue realmente trabajado por el equipo. Unicamente se cuentan los días hábiles donde el ítem estuvo en las columnas de “trabajo en curso”, en oposición a columnas de tipo buffer y estado de bloqueado o sin trabajo del equipo sobre el mismo.

$$Touch\ Time \leq Cycle\ Time \leq Lead\ Time$$

Eficiencia del ciclo de proceso

Touch Time / Lead Time.

Similitudes de Scrum y Kanban



Diferencias entre Scrum y Kanban

Scrum	Kanban
Iteraciones de tiempo fijo.	**Tiempo fijo es opcional**. La cadencia puede variar. Pueden estar marcadas por la previsión de los eventos en lugar de tener un tiempo prefijado.
Equipo asume un compromiso de trabajo por iteración.	El **compromiso es opcional**.
Métrica para planificación y mejora: **Velocidad.**	Métrica por defecto es **Lead Time** (Tiempo de Entrega o tiempo medio)
Equipos Multifuncionales.	**Equipos Multifuncionales o especializados.**
Funcionalidad divididas para poder completarse en un Sprint.	No hay prescripción respecto del tamaño de la funcionalidad.
Deben emplearse gráficos **Burndown chart.**	No se prescriben diagramas de seguimiento.
Limitación **WIP indirecta** (por Sprint).	**Limitación WIP directa** (marcada por el estado del trabajo)
Se deben realizar estimaciones.	**Las estimaciones son opcionales.**
No se puede agregar alcance en medio de una iteración.	Siempre que haya capacidad disponible **se puede agregar trabajo.**
Sprint Backlog pertenece a un equipo determinado.	Varios equipos pueden compartir pizarra Kanban.
Se prescriben tres roles (PO / SM/ Equipo).	No hay roles prescritos.
En cada sprint **se limpia el tablero** de seguimiento.	El tablero Kanban **es persistente**.
Product Backlog Priorizado.	**La priorización es opcional.**