Administración de Proyectos

**Proyecto:** Son tareas interrelacionadas basadas en el esfuerzo y los recursos disponibles durante la duración del proyecto. Están orientados a objetivos que permiten construir productos o servicios únicos, y que a la vez guían al proyecto. Los objetivos deben ser claros, no ambiguos, y alcanzables. Tienen una duración limitada con un fin que no es claro en el desarrollo de software, pero debería serlo. Los proyectos son importantes porque se trabaja en equipo, y revolucionarios porque surgen de un problema u oportunidad.

**Ciclo de Vida:** El ciclo de vida del producto es mayor al del proyecto ya que para hacer un cambio al producto se inicia un nuevo proyecto. Algunas características comunes a los ciclos de vida son:

* Los costos y nivel de personal son bajos al inicio y altos al final.
* El riesgo e incertidumbre son altos al inicio del proyecto, por lo que la probabilidad de éxito es baja.
* La capacidad de los involucrados para influir en las características finales del producto es más alta al comienzo.

**Administración de Proyectos:** Es la aplicación de conocimientos, habilidades, herramientas y técnicas en las actividades del proyecto para satisfacer los requerimientos. Incluye identificar los requerimientos, establecer objetivos claros y alcanzables y adaptar las especificaciones, planes y el enfoque a los diferentes intereses de los stakeholders. El objetivo es tener el trabajo hecho en tiempo, con el presupuesto acordado y habiendo satisfecho el alcance.

**Triple Restricción:** Consiste en saber balancear los costos, el tiempo y el alcance del proyecto, lo que afecta directamente a la calidad. El líder de proyecto es el responsable de balancear estos tres objetivos. Se suele negociar el tiempo y el costo y no conviene tomar proyectos que fijen las tres variables. Una forma de reducir el costo es sacrificando testing y calidad.

**Involucrados:** Los stakeholders son los interesados en el proyecto, el equipo de proyecto, el equipo de dirección, el líder de proyecto y el patrocinador. Este último debe tener jerarquía en la empresa y garantizar los recursos. En los métodos agiles el líder del proyecto es el scrum master.

**Equipo de Proyecto:** Es un grupo de personas comprometidas en alcanzar un conjunto de objetivos de los cuales se sienten responsables. Cuentan con diversos conocimientos y habilidades, desarrollan sinergia, y suelen ser grupos pequeños.

**Líder del Proyecto:** Para ser un líder uno debe sentirse cómodo con los cambios y entender a la organización. El líder debe ubicar a la gente en el rol que mejor sepan cumplir y guiarla en el. El líder debe tener los Hard Skills, que son los conocimientos del producto, herramientas y técnicas, y los Soft Skills, que es la capacidad de trabajar con gente y de los más difíciles de conseguir (comunicación, liderazgo, creatividad).

El líder debe definir el alcance del proyecto, estimar tiempos y requerimientos, identificar los recursos requeridos, identificar y evaluar riesgos, sus planes de contingencia, etc.

**Inicio del Proyecto:**

1. Identificación del cliente o del mercado.
2. Estudio de factibilidad.
3. Propuesta de proyecto: Establece que, como, cuando se hará y cuánto costará.
4. Negociación: Se definen las características del producto, el cronograma, y los hitos. Se define el presupuesto, la estructura del equipo, etc.
5. Inicio del proyecto:Se realiza el kick-off (inicio oficial del proyecto).

**Planificación del Proyecto y Armado del Equipo:**

1. Definición del proyecto: Se define el problema u oportunidad, las condiciones de aceptación, el objetivo, la misión, y las asunciones y riesgos (probabilidad e impacto). No se debe asumir sin considerar los riesgos propios ya que de seguro se fallará. A la vez, los objetivos se deben definir asumiendo ciertas situaciones por parte del cliente para evitar problemas a futuro.
2. Creación del plan: El plan provee una herramienta de comunicación estándar a través del ciclo de vida de un proyecto. Establece las responsabilidades y sirve como documento de gestión básico, a diferencia de la ERS que es un documento técnico y se complementan.
3. Formación del equipo.

Los documentos principales son el libro de requerimientos (ERS o Statement Of Work), el plan de proyecto, el plan de calidad y el plan de administración de la configuración.

No es lo mismo Alcance del proyecto que alcance del producto, ya que el primero es todo el trabajo y solo el trabajo que debe hacerse para entregar el producto con todas las características especificadas, y el segundo son todas las características que pueden incluirse en un producto. El cumplimiento del alcance del proyecto se mide contra el plan de proyecto y el cumplimiento del alcance del producto contra la especificación de requerimientos.

También se considera el tamaño (cantidad y complejidad de los requerimientos) y el esfuerzo (horas hombre). Finalmente se define el proceso y ciclo de vida, los controles y las métricas.

**Organización de las Actividades:** Las actividades en el proyecto deben producir salidas tangibles, para poder medir el progreso del proyecto. Los hitos son puntos finales de las actividades del proceso y los entregables son resultados del proceso entregados al cliente. Los entregables suelen ser hitos aunque no todos los hitos son entregables.

**Work Breakdown Structure:** Es un arreglo jerárquico de los elementos de una actividad que se puede estimar. Permite planificar, administrar y realizar los reportes de un proyecto. La descomposición puede ser por productos, por trabajo, por fases u organizacional.

El líder de proyecto decide el enfoque de arquitectura y el nivel de detalle del WBS. El WBS se considera terminado cuando cada actividad tiene un entregable, los tiempos y costos pueden ser estimados, las actividades tienen una duración aceptable y esta definido el comienzo y fin.

**Programación:** Determina que tareas se pueden hacer en paralelo y cuales deben hacerse en serie. Indica el tiempo estimado para las tareas.

**Entregables y Soporte:** Se definen en el libro de requerimientos o en el plan. La entrega no es necesariamente el fin del proyecto, ya que puede iniciar la fase de soporte que generará nuevas entregas con agregados de funcionalidad.

**Éxitos y Fracasos:** El éxito depende del monitoreo, la comunicación y tener un objetivo claro. El fracaso surge de fallas al definir el problema, no saber quien estaba a cargo, falta de datos y seguimiento y fallas en la estimación.

Proceso de Desarrollo y Ciclo de Vida

**Proceso de Desarrollo:** Es un conjunto estructurado de actividades para desarrollar un sistema de software. Estas actividades varían dependiendo la organización y el tipo de sistema a desarrollar. Debe ser modelado si va a ser administrado. Un proceso de desarrollo se puede aplicar a varios ciclos de vida. La elección del ciclo de vida depende de la estrategia. Las personas utilizan herramientas y equipos para llevar a cabo los procedimientos que componen el proceso de desarrollo. Debe permitir, seleccionar los artefactos a producir, definir actividades y roles, y modelar conceptos.

**Ciclos de Vida:** Es la serie de pasos a través de los cuales el producto progresa. Es una representación de un proceso desde una perspectiva particular. Los modelos especifican las fases del proceso y el orden en que se llevan a cabo. Todos los modelos atraviesan las mismas etapas variando la cantidad de trabajo en cada una. Es una guía para la administración del proyecto ya que indica el progreso a través de hitos. Los modelos de ciclos de vida se han vuelto necesarios debido a que los sistemas son más complejos por el aumento de funcionalidad y la mayor variedad de usuarios.

Los modelos de ciclo de vida son independientes de los procedimientos de cada actividad del ciclo de vida. La actividad más costosa es el mantenimiento. Los ciclos de vida se clasifican en:

* Secuenciales: donde una actividad no inicia hasta que termina la anterior.
* Iterativos: consisten en hacer algo una y otra vez, agregando cosas. Son la tendencia y es conveniente usarlos cuando no se sabe que modelo elegir.
* Recursivos: Se inicia con algo en forma completa, como una subrutina que se llama a si misma e inicia nuevamente. Se presenta un prototipo que va mejorando con cada vuelta.

**Code and Fix:** Se desarrolla sin especificaciones o diseño y se lo modifica hasta que el cliente este satisfecho. Los cambios se realizan durante el mantenimiento, lo que es muy caro.

**Modelo en Cascada Puro:** Se sigue una secuencia de pasos ordenada y se realiza revisión al final de cada fase. Es conducida por la documentación con fases que no se superponen. Permite encontrar errores en etapas tempranas. Hay exceso de documentación y falta de resultados hasta el final. Se utiliza cuando la definición del producto es estable o los requerimientos de calidad dominan a los de costo y tiempo.

**Modelo en Cascada con Fases Solapadas:** Hay un fuerte grado de solapamiento. La documentación es menor pero es más difícil hacer el seguimiento de progreso.

**Modelo de Entrega por Etapas:** Se ven signos tangibles de progreso. Es útil cuando el cliente necesita funcionalidad de inmediato y las necesidades se modifican. Debe haber una planificación rigurosa para que funcione.

**Modelo en Cascada con Retroalimentación:** Es una variación del modelo clásico en que es posible volver a una etapa anterior antes de llegar al final aunque es muy caro hacerlo.

**Modelo en Cascada con Subproyectos:** Es un modelo secuencial. Se avanza en cascada hasta el diseño arquitectónico, de ahí en mas se avanza en partes dividiendo en Subproyectos.

**Modelo en espiral:** Busca minimizar los riesgos haciendo un análisis de riesgos y buscando alternativas al iniciar cada fase. Al final de cada fase se planifica la siguiente y se evalúa si se sigue adelante. Permite saber si un nuevo producto puede funcionar o no. En proyectos grandes la identificación de los riesgos puede costar más que el desarrollo. Es un modelo adecuado para proyectos con ciclos de vida largos, que puede utilizar equipos diferentes en cada ciclo y muestra el progreso al fin de cada ciclo.

**Modelo Evolucionario:** Se fija el tiempo o el alcance mientras el otro se va modificando. Es útil cuando los requerimientos no son claros y se realizan entregas tempranas al cliente. El problema es que es un modelo recursivo que repite todas las tareas y depende de que los diseñadores desarrollen un sistema fácil de modificar.

**Diseño para Cronograma:** No se asegura alcanzar la versión final, aunque si una entrega del producto para la fecha definida con los aspectos más importantes. Es útil cuando no hay seguridad sobre las capacidades de programación.

**Diseño para Herramientas:** Consiste en incluir solo la funcionalidad soportada por las herramientas de software existentes. Se usa en proyectos muy sensibles al tiempo. Puede combinarse con otros ciclos de vida, pero no se podrán implementar todos los requerimientos.

**Prototipación Evolucionaria:** Se da cuando hay cambios rápidos de requerimientos y los clientes no se comprometen con los requisitos. Requiere menos documentación pero no se pueden programar los release. Se desarrolla el concepto del sistema a medida que se avanza.

Monitoreo y Control de Proyectos

**Control:** Se refiere a comparar el progreso con respecto al plan para tomar acciones correctivas cuando ocurran desviaciones significativas respecto al plan.

**Monitoreo:** Su objetivo es determinar si el proyecto esta bajo control. Si esta fuera de control se puede re planificar manteniendo o cambiando la fecha limite. Si se están alcanzando los hitos del proyecto a tiempo, con los recursos estimados, la calidad deseada y sigue siendo aceptable económicamente, entonces el proyecto está bajo control.

**Seguimiento y Control:** Un proyecto se retrasa de a un día por vez. El seguimiento y control se puede hacer de varias formas: revisiones mensuales del proyecto, auditorias, reuniones de Postmortem donde se da más información porque se hacen luego de cada fase, o el más común que son las reuniones semanales de seguimiento.

**Esperado vs Real:** Lo esperado se basa en los estándares de productividad y en los compromisos de planificación. Si no se cumple el estándar se debe estudiar la desviación, y hacer los ajustes necesarios. Si no se cumple lo planificado se debe re planificar y crear medidas correctivas para que el calendario se ajuste a la realidad. Si no se cumple ninguno de los dos, se deben aplicar ambas medidas.

Usando Gantt se puede ver en que situación estamos. Si una línea vertical representa lo planificado, podemos ir recorriendo esa línea saltando hacia adelante o atrás en cada actividad según el porcentaje de avance. Lo normal es que estemos sobre la línea pero no a la perfección. Muy detrás sería una subestimación y por delante sobrestimación.

**Planificado vs Real:** Para esto se propone un sistema binario (realizado o no realizado) en lugar de usar porcentajes, ya que lleva al síndrome del 90% eterno. Para esto las tareas no tengan una duración excesiva.

**Seguimiento:** El seguimiento permite minimizar los riesgos, tomar acciones correctivas, mejorar la calidad y evitar errores. Los proyectos cortos deben aumentar el seguimiento. El seguimiento puede ser por proceso (siguiendo hitos y tareas) o por productos (controlando los entregables). Estos no son independientes y se puede trabajar con ambos al mismo tiempo.

**Control de Cambios:** Los requerimientos siempre cambian, y por ellos es esencial administrar los cambios de un proyecto. El Comité de Control de Cambios debe estar formado por los roles claves del proyecto y gerenciar los mayores cambios.

Scrum

**Manifiesto Ágil:** Los cuatro valores son:

* Individuos e interacciones por sobre procesos y herramientas
* Software funcionando por sobre documentación detallada: Esto no significa que no se documente, sino que se hace foco en la implementación por sobre el modelado.
* Colaboración por sobre negociación con el cliente: Hay que estar dispuesto al cambio y cerca del cliente para predecirlo.
* Responder a cambios por sobre seguir un plan: Los nuevos requerimientos pueden tener un mayor valor que los iniciales.

Se deben descartar los modelos temporales, centrarse en desarrollar código antes que mantener un diagrama. Debemos modelar con otros y mostrar los modelos públicamente. Se deben usar herramientas simples y mostrar los modelos de manera simple. Se deben crear varios modelos en paralelo.

**Principios del Manifiesto:**

1. La prioridad es satisfacer al cliente a través de releases tempranos y frecuentes.
2. Recibir cambios de requerimientos, aun en etapas finales.
3. Releases frecuentes: Permite una evolución del modelo en espiral y un mejor manejo de los requerimientos. Deben hacerse de 2 semanas a un mes.
4. Técnicos y no técnicos trabajando juntos en todo el proyecto.
5. Hacer proyectos con individuos motivados.
6. El medio de comunicación por excelencia es cara a cara: El espacio físico debe favorecer la comunicación. Esto no significa reducir la documentación.
7. La mejor métrica de progreso es la cantidad de software funcionando, sí bien se colectan otras métricas.
8. El ritmo de desarrollo es sostenible en el tiempo: Mas de 40 horas por semana no es sostenible. Debe hacerse con responsabilidad social y efectividad monetaria.
9. Atención continúa a la excelencia técnica: mediante mejora continua del producto.
10. Maximización del trabajo no hecho: No implementar más de lo acordado. Tampoco debe confundirse con saltar directo a la codificación.
11. Las mejores arquitecturas, diseños y requerimientos emergen de equipos auto organizados: Se gana sinergia, se deben preservar los equipos de un proyecto a otro.
12. A intervalos regulares, el equipo evalúa su desempeño y ajusta la manera de trabajar.

**Empírico vs Definido:** Hay dos alternativas para controlar cualquier proceso:

* Definido: Dado un conjunto de entradas bien definido, se generan siempre las mismas salidas. La administración y control provienen de la predictibilidad del proceso definido.
* Empírico: Se provee control a través de inspecciones frecuentes y adaptación de procesos definidos imperfectamente, generando salidas no repetibles.

**Métodos Ágiles:** Los métodos ágiles son adaptables en lugar de predictivos, y orientados a la gente. Son un subconjunto de los métodos iterativos. El modelado ágil no es un proceso completo, sino un conjunto de principios y prácticas para modelado y análisis de requerimientos, que complementa a la mayoría de los ciclos de vida iterativos e incrementales. Las metodologías agiles son aplicables cuando los problemas a ser resueltos son complejos, es decir, que se esta relativamente alejado del acuerdo y la certeza pero no tanto como para caer en una “zona de anarquía”.

**Scrum:** Es un framework que permite crear su propio proceso para crear nuevos productos. Es simple y puede implementarse en pocos días pero perfeccionarlo lleva tiempo. El control se alcanza con inspecciones frecuentes y los ajustes correspondientes. Se debe planear, ejecutar, reflexionar y volver a iniciar. Permite trabajar con sistemas funcionando, con tecnologías inestables y el surgimiento de requerimientos.

Se trabaja con equipos pequeños y una serie de Sprints de los cuales se obtienen incrementos visibles y usables. El manejo del tiempo es inflexible, y lo que quede es para el próximo Sprint. Los valores de Scrum son el compromiso, el foco, ser abierto a ideas, el respeto y la valentía.

Se dedica menos tiempo a planear y definir tareas, crear y leer reportes, y más a investigar situaciones con el equipo. Los roles en Scrum son los del Scrum Master, Equipo de Trabajo, Upper Management, Cliente y el Dueño del Producto.

**Cimientos de Scrum:**

* Empirismo: Los procesos definidos y la planificación detallada en las primeras fases son remplazadas por ciclos de inspección y revisión just in time, y ciclos adaptativos.
* Auto Organización: En grupos de trabajo que manejan sus propias tareas y objetivos.
* Colaboración: Los lideres de Scrum, diseñadores de productos y clientes colaboran con los desarrolladores. Ellos no los gerencian o dirigen.
* Priorización: Dejar de lado el trabajo que no agrega valor.
* Time Boxing: Es una técnica de planificación que crea el ritmo que guía el desarrollo.

**Scrum Master:** Es el responsable de que las prácticas, valores y reglas se realicen, y el nexo entre la gerencia y el equipo. Dirige los Scrum diarios comparando el progreso planeado con el real. Produce el Backlog del producto junto al dueño del producto y el equipo. Debe tomar las decisiones, resolver los problemas, mantener al equipo enfocado. Es responsable de realizar reuniones diarias cortas, donde los gerentes pueden asistir pero no participar.

**Dueño del Producto:** Es una persona que controla y gestiona el Backlog, con conocimiento del negocio y capacidad para escribir un requerimiento.

**Equipo:** Son alrededor de 7 personas, grupos mas grandes se deben dividir en varios equipos. Son autónomos y auto organizados que se comprometen a entregar un conjunto de ítems del Backlog al final del Sprint. No hay roles y todos codifican. A veces se alcanza el objetivo reduciendo funcionalidad.

**Ciclo de Vida de Scrum:** Se toman del Product Backlog las tareas que forman el Sprint Backlog, el cual indica qué se debe hacer para cumplir el objetivo del Sprint actual. Los Sprints tienen una duración de dos a cuatro semanas. Día a día se realizan los Daily Scrum para conocer el avance del proyecto. Al finalizar el Sprint se obtiene el incremento del producto.

**Backlog de Producto:** Contiene la lista de requerimientos. Se listan características, funciones y mejoras que se aplicaran al producto. Al inicio no esta completo, pero solo se necesita lo suficiente para el primer Sprint. Debe estar priorizado por el product owner. Las estimaciones son un esfuerzo de las partes e iterativas. Si un ítem no puede ser debidamente estimado, se debe dividir en el backlog

**Sprint:** Período fijo de tiempo, sugerido en 30 días. Durante el Sprint no se puede cambiar el alcance, agregar funcionalidad ni modificar las reglas del equipo. Los Sprints producen un incremento usable basado en el producto anterior. No deben existir interrupciones y debe hacerse foco en las tareas.

Durante las reuniones de planeamiento, se decide que funcionalidad se va a implementar, y el equipo decide cómo se hará. En los Scrum diarios, de aproximadamente media hora el Scrum Master pregunta a los asistentes que se completó desde la última reunión, que obstáculos se presentaron y que se hará hasta la próxima reunión. Los Scrum diarios mejoran la comunicación y el conocimiento de todos, elimina otras reuniones, promueve decisiones rápidas y remueve obstáculos.

Durante los Sprint se congelan tres de las cuatro variables del proyecto, que son el tiempo, los costos y la calidad, pero puede variar la funcionalidad siempre y cuando se cumpla con el objetivo del Sprint.

Un Sprint puede cancelarse si el objetivo se vuelve obsoleto, si las condiciones técnicas o del mercado cambian, o si el equipo lo decide ya que no se puede alcanzar el objetivo o se encuentra un problema muy grande. Cancelar un Sprint es un costo y esta mal organizacionalmente.

En las revisiones del Sprint se presenta el incremento desarrollado a gerentes, clientes y usuarios, se reportan los problemas encontrados y puede agregarse, eliminarse y re priorizarse cualquier ítem. Se estima el nuevo Sprint y asignan las tareas. Tienen una duración de 4 horas. Son reuniones informales donde todo el equipo participa.

La retrospectiva del Sprint se realiza luego de cada Sprint con una duración de 15 a 30 minutos donde participa todo el equipo y se analiza lo que funciona y lo que habría que mejorar.

**Herramientas de Scrum:** El taskboard es un tablero dividido en las secciones Story (lo que el usuario quiere), To Do, In Process, To Verify y Done. En cada sección se ubican las tarjetas con las distintas tareas a realizar. Estas contienen entre otras cosas el número del Sprint, el código de ítem Backlog, el nombre de la actividad el tiempo estimado y las iniciales de quien desarrolle la actividad. Es importante que haya una buena granularidad, es decir muchas tareas cortas.

Los gráficos de Backlog brindan datos sobre el progreso del Sprint, Release y producto a la gerencia, indicando la cantidad de trabajo que queda por hacer en el tiempo.

Las firmas de Sprint son los patrones que reflejan las actividades del equipo en cada Sprint contra el tiempo. Ayudan a la predicción y a los gerentes a determinar si se planeo correctamente o están haciendo horas extra.

**Beneficios de Scrum:** Se gestionan los cambios de requerimientos, se incorpora la visión de mercado, los clientes ven incrementos que refinan los requerimientos en tiempos razonables y mejora la relación con el cliente.

Gestión De Configuraciones

**Integridad del Producto:** Un producto tiene integridad si:

* Satisface las necesidades del usuario: Considera los requerimientos funcionales.
* Puede ser fácilmente rastreado durante su ciclo de vida: sé cuando aparece algo, por qué y dónde lo hace.
* Satisface criterios de performance: Tiene requerimientos no funcionales asociados.
* Cumple expectativas de costo: existe algo de planificación

**Problemas en el Manejo de Componentes:** La administración de configuraciones permite solucionar los siguientes problemas:

* Pérdida de un componente.
* Pérdida de cambios o superposición de cambios.
* Doble mantenimiento
* Cambios no validados.

**Administración de Configuraciones:** Tiene el propósito de establecer y mantener la integridad de los productos del proyecto de software a lo largo del ciclo de vida del mismo. Involucra identificar la configuración en un momento dado, controlar sistemáticamente los cambios y mantener su integridad y origen. Es parte del aseguramiento de la calidad.

Una configuración es el conjunto de todos los componentes fuentes que son compilados, sus documentos e información de la estructura que definen una versión del producto a entregar.

**Planificación de la Gestión de Configuraciones:** Un plan describe los estándares y procedimientos utilizados para la gestión de la configuración. Contiene la estructura del repositorio, la identificación de los ítems de configuración, la planificación y nombres de los baselines, como se hará el control de cambios, quien formara parte del comité de control de cambios, como se hará el registro y quien hará la auditoria.

**Identificación de los elementos de configuración:** Cuando se desea controlar los cambios de un elemento es necesario identificarlo, para lo que se declara un ítem de configuración. Para que un ítem de configuración se vuelva tal debe estar versionado. Un ítem de configuración inicia en un estado borrador donde se van haciendo los cambios, luego pasa a una etapa de revisión y finalmente queda aceptado. Aquí queda en estado “baseline”, es decir, que no se puede cambiar más.

Una línea base es una instantánea, una configuración que se ha revisado formalmente y que sirve como base para un desarrollo posterior, que solo puede cambiarse a través de una petición formal al comité de control de cambios. Todos los ítems que la forman están en estado “baseline”. Da repetitividad, permite entregar siempre lo mismo. Los baseline no se modifican ni eliminan ya que se pierde trazabilidad. Las baselines se identifican con etiquetas.

**Control de Cambios:** El comité de control de cambios analiza la propuesta de cambio teniendo en cuenta el impacto, para determinar si es un cambio mayor o menor. Los cambios de tipo mayor puede que no se implementen en la versión actual. Además, indica en que baseline hay que basarse para efectuar un cambio.

Para la gestión de cambios y líneas bases existe herramientas automáticas que brindan un fácil acceso a los componentes, permiten reconstruir cualquier versión, registran el historial de cambios y brindan información resumida. Estas herramientas realizan versionado y trabajan sobre un repositorio. Los repositorios contienen los ítems de configuración y mantienen una estructura de directorios. Por detrás, implementan una base de datos. Un Checkout permite extraer los datos al “Working Area” donde se realizan los cambios, al finalizar se hace un Checkin de los mismos al repositorio.

Lo más importante de versionar son los fuentes, los ejecutables, los scripts de creación y parámetros de la base de datos, los scripts de procedimientos almacenados y scripts de instalación, seguido de la documentación.

**Gestión de versiones y entregas:** Es el proceso de identificar y mantener registros de las versiones y entregas de un sistema. Una versión es una instancia de un sistema que difiere de otras instancias. Si sólo hay pequeñas diferencias entre las versiones, éstas se denominan variantes. Una entrega es una versión del sistema que se distribuye a los clientes. Las nuevas entregas no pueden depender de la existencia de las entregas previas. La distribución de nuevas entregas tiene altos costos por lo que los problemas se resuelven con parches.

**Identificación de Versiones:** Existen tres técnicas básicas utilizadas para la identificación de componentes en una versión particular del sistema.

* Numeración de las versiones: Al componente se le asigna un número de versión explícito y único. Un modelo es el de Version.CambioMayor.CambioMenor.Build.
* Identificación basada en atributos.
* Identificación orientada al cambio.

**Construcción del Sistema:** Es el proceso de compilar y vincular los componentes del software en un programa que se ejecuta en una configuración particular.

**Reportes del Estado de la Configuración:** Se ocupa de mantener los registros de la evolución del sistema. Permite rastrear los cambios realizados a las líneas base. Maneja mucha información y salidas por lo que se suele implementar dentro de procesos automáticos.

**Auditoria de Gestión de Configuración:** Consiste en determinar la semejanza entre el estado actual del sistema y el establecido como línea base. Provee el mecanismo para establecer una línea base. Busca integridad en el repositorio, asegurando que este disponible y en donde corresponda.

**Validación:** Que el problema este resuelto de forma apropiada para que el usuario obtenga el producto correcto.

**Verificación:** Se controla que lo que se hizo funcione correctamente y que cumpla los objetivos establecidos en la documentación de líneas base.

**Auditoria física de Configuración:** Asegura que lo que está indicado para cada Ítem de Configuración de Software en la línea base o actualización se ha alcanzado realmente y que el software y la documentación son internamente consistentes para entregarlos al cliente.

**Auditoría funcional de Configuración:** Es la evaluación independiente de los productos de software, verificando que la funcionalidad y rendimiento reales de cada ítem sean consistentes con la especificación del requerimiento. Se busca una matriz de rastreabilidad para saber qué caso de prueba corresponde a cada requerimiento.

Revisiones e Inspecciones

**Falla:** Es un error en un producto de trabajo. El producto de trabajo es la salida de una actividad del ciclo de vida de desarrollo. Las fallas se pueden dar por ruido en la comunicación, ya que mientras más personas participen en el proyecto más caminos puede tomar la comunicación; o por limitaciones de memoria, que se van presentando a medida que el proyecto se vuelve más complejo.

Las fallas presentes en las primeras etapas del proyecto son las más caras de corregir, ya que atraviesan más etapas y el trabajo realizado en estas se basa en errores.

Se las clasifica según la ubicación, la descripción, la severidad, la fase en que se detectó y si es error o defecto.

Una falla puede considerarse como defecto, cuando se originó en el producto de trabajo predecesor o en producción; o como error, cuando se origina en el producto de trabajo que se está inspeccionando. Los errores se pueden clasificar como mayor, menor o cosmético:

* Mayor: Puede causar una falla operacional o producir un resultado inesperado durante la ejecución de la operación especificada. Por ejemplo un error lógico de código que produzca una falla operacional; un diseño que ocasione una falla si se implementa tal cual está especificado; una mala elicitación de requerimientos que no cumpla la necesidad del cliente o un plan de pruebas que pase algunas fallas por alto.
* Menor: Es una condición no deseable pero que no puede ocasionar una falla operacional. Por ejemplo en el código o diseño la violación de estándares, requerimientos que no se puedan probar y en el plan de prueba información poco clara.
* Cosmético: Errores de tipeo, ortografía, etc.

**Disposición del producto de trabajo:** El equipo de inspección determina la disposición basado en el tipo y cantidad de fallas encontradas. Puede ser aceptado, donde solo se encontraron fallas menores; aceptado condicionalmente, cuando se puede verificar la corrección de las fallas y estas correcciones son triviales; o re inspección, si las correcciones de las fallas deben ser re inspeccionadas por el equipo.

**Verificación y Validación:** Es un proceso que se realiza a lo largo de todo el ciclo de vida. La validación consiste en definir si se está haciendo el producto correcto, por ejemplo a través de prototipos o con el cliente. La verificación define si se está construyendo el producto correctamente, por ejemplo con casos de prueba.

**Principios:**

* Es mejor prevenir que curar.
* Evitar es más efectivo que eliminar.
* La retroalimentación enseña efectivamente.
* Priorizar lo rentable.
* Olvidarse de la perfección.
* Enseñar a pescar en lugar de dar el pescado.

**Revisiones del Software:** Su objetivo es detectar defectos y corregirlos en las etapas iniciales del desarrollo. Se aplican a productos en distintas fases del desarrollo para detectar errores que se le hayan pasado por alto a quien los originó. Motiva a realizar un mejor trabajo y no requiere la ejecución del programa. Se hace foco en encontrar el defecto, y no la solución. Se toman pequeñas porciones completas de trabajo y se corrigen todos los defectos mayores. Durante el proceso se registran los defectos, el esfuerzo y la duración medidos para mejora, se previenen defectos sistemáticos, etc. Los inspectores examinan el producto individualmente antes de reunirse, y las correcciones realizadas se examinan antes de terminar el proceso.

Las revisiones presentan mayor efectividad para descubrir defectos que las pruebas ya que se detectan varios defectos en una sola sesión de inspección, y se re utiliza el conocimiento del dominio y del lenguaje de programación.

**Costos de las revisiones:**

* Infraestructura: Son los costos relacionados al entrenamiento, las herramientas de soporte y el desarrollo de guías de lectura e informes.
* Operacionales: Son los costos por cada persona involucrada en la revisión, los tiempos en completar las tareas.
* Adicionales: Corresponde a preparar el material, recolectar datos, mejorar la calidad, etc.

**Métricas Sugeridas:** Algunas métricas que se pueden utilizar en el proceso de revisión son: densidad de defectos (defectos encontrados/ tamaño) y esfuerzo por defecto (esfuerzo inspección / defectos encontrados).

**Revisión Técnica Formal / Inspección:** Las inspecciones son la forma más barata y efectiva e encontrar fallas. Proveen métricas al proyecto y conocimiento cruzado. Promueven el trabajo en grupo y mejoran la calidad del producto. Sin embargo, no se encargan de solucionar las fallas, ni evalúan el desempeño de las personas. Son efectivas en un 75%. Se llevan a cabo mediante una reunión y el éxito depende de su planificación. De los roles en la inspección el autor, moderador e inspector no se comparten:

* Autor: Es el encargado del producto a inspeccionar. Inicia el proceso al asignar un moderador darle el producto. Con el moderador define el resto de los roles.
* Moderador: Planifica y lidera la revisión. Entrega el producto a los inspectores con tiempo antes de la reunión. Coordina las reuniones.
* Lector: Lee el producto a inspeccionar.
* Anotador: Registra los hallazgos de la revisión.
* Inspector: Examina el producto antes de la reunión para encontrar defectos.

**Proceso de Inspección:**

1. Planificación: Se obtiene el paquete de revisión (producto, checklist, referencias), se forma el equipo de inspección y determinan las fechas de reunión.
2. Resumen: El autor da una introducción, se establecen las metas y los revisores obtienen el paquete a revisar.
3. Preparación: Se encuentra la mayor cantidad de errores no triviales.
4. Reunión de revisión: Se crea una lista de errores, se busca obtener sinergia y mejorar las cualidades de revisión.
5. Re trabajo: Se eliminan los defectos del producto, determinan las acciones realizadas por cada error y los defectos triviales a medida que sea necesario.
6. Seguimiento: Se asegura la calidad del producto, el proceso de inspección y se acepta o rechaza el producto.

**Registro e informe de la Revisión:** Se registran todos los problemas o errores, un resumen de los sucesos, la decisión de aprobar, aprobar provisoriamente o rechazar el producto y el procedimiento de seguimiento que asegure que los puntos de la lista son corregidos.

**Inspecciones por Fases:** El proceso de revisión puede dividirse en fases lo que permite concentrarse en aspectos puntuales del software. Cada fase chequea una parte importante del software (constantes, variables, comentarios, etc), y se construye en base a la anterior.

**Recorrida:** El Walktrough es una 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 y los participantes realizan preguntas y comentarios de posibles errores y problemas. Se toman pocas métricas y no hay control del proceso.

Estimaciones de Software

**Estimación:** Es una predicción del tiempo o costo que tomara un proyecto. Se relaciona con los objetivos del negocio, compromisos y control. La estimación se hace para predecir la completitud y administrar riesgos. Los errores de estimación pueden venir de información imprecisa acerca del software a estimar, acerca de la capacidad para realizar el proyecto, demasiado caos en el proyecto e imprecisión generada por el proceso de estimación.

Una de las fuentes de error más común es omitir actividades necesarias para la estimación del proyecto tales como, requerimientos faltantes, licencias, reuniones, revisiones, etc.

**Relación entre Estimaciones y Planes:** El objetivo de la estimación es la precisión y de planes es obtener un resultado en particular. Las estimaciones son la base de los planes, pero los planes no tienen que ser lo mismo que lo estimado. A mayor diferencia entre lo estimado y lo planeado mayor riesgo.

**Técnicas de Estimación:** La técnica fundamental es contar. Hay que contar lo que esté muy relacionado al tamaño del software a estimar. Hay que entender lo que se cuenta y debe requerir poco esfuerzo.

Los métodos utilizados pueden ser basados en la experiencia, en los recursos, en el mercado, en los componentes del producto o proceso de desarrollo, o pueden ser algorítmicos. Aquellos basados en la experiencia son: Datos Históricos, Juicio Experto (Puro o Delphi), Analogía y Distribución del uso de recursos en el ciclo de vida.

**Datos Históricos:** La productividad de una organización es un atributo que no tiene gran variación entre los proyectos. Los datos históricos se deben usar para evitar las discusiones entre desarrolladores, clientes y ejecutivos de la empresa. Se deben recolectar los datos básicos del proyecto, como el tamaño, esfuerzo, tiempo y defecto, y asegurarse que el proceso se realiza de la misma manera en todos los proyectos.

**Juicio Experto:** Es la técnica más utilizado en la práctica. Se deben tener tareas de una granularidad aceptable, usar un checklist y un criterio definido para asegurar cobertura, y el método de tiempo optimista, pesimista, habitual ((O + 4H + P) /6). Este último punto debe usarse con cuidado ya que provee un buen comienzo pero un pésimo final.

En el Juicio Experto Puro, un experto estudia las especificaciones y hace su estimación en base a sus conocimientos.

En Wideband Delphi, un grupo de personas son informadas y tratan de adivinar lo que costará el desarrollo en esfuerzo y duración. Las estimaciones en grupo suelen ser mejores que las individuales. El proceso inicia cuando se le entregan las especificaciones al grupo de expertos, estos se reúnen para discutir el producto y la estimación. Luego cada estimador le entrega su estimación al coordinador. Los estimadores reciben información sobre su estimación y las ajenas de forma anónima. Luego el proceso inicia de vuelta y se repite hasta que llegan a una estimación unificada razonablemente.

**Analogía:** Consiste en comparar las especificaciones de un proyecto con las de otros proyectos. Algunas especificaciones que se comparan son, el tamaño, complejidad, usuarios, entornos, personal del proyecto, etc.

**Estimación de Tamaño:** Lo más buscado es el tamaño del software a ser construido. Este se puede expresar en Líneas de Código, Puntos de Función, Nro. de Requerimientos, etc. Otros factores pueden ser la complejidad, la madurez de la tecnología y la gente.

Las Líneas de Código son fáciles de obtener de proyectos anteriores y existe mucha información de otras empresas disponible. Se encontró que el esfuerzo por línea de código se mantiene constante entre los lenguajes.

**Estimación del Esfuerzo:** Muchos proyectos estiman el esfuerzo con una lista detallada de tareas. Para estimar inicialmente el esfuerzo se usa como base el tamaño. La mayor influencia en el esfuerzo son el tamaño del software y la productividad del equipo. Cuando no se dispone de datos históricos pueden utilizarse tablas de productividad por tipos de software.

**Estimaciones Agile:** En los equipos agile, las stories son estimadas usando una medida de tamaño relativo conocida como story points. Las personas estiman mejor comparando que en términos absolutos. Comparar es más rápido, por lo que se obtiene una mejor dinámica grupal, pensamiento de equipo, y se emplea mejor el tiempo de análisis de las stories.

En Agile, el tamaño es la cantidad de trabajo necesaria para producir una story. Indica cuan compleja y grande es.

Un story Point es una unidad de medida específica de complejidad, riesgo y esfuerzo. Indica el “peso” del story y que tan grande es. La complejidad de una story crece exponencialmente.

**Poker Estimation:** Es popular en la aplicación de agile. Combina el uso de experto, analogía y desegregación. Los participantes en póker planning son los desarrolladores, ya que la filosofía es que deben estimar las personas más competentes en resolver una tarea.

En póker planning se necesita la lista de stories a estimar y un mazo de cartas por estimador. Primero se elige el base story para comparar. Luego una a una se va estimando las stories. Por cada una se discute y cada estimador toma una carta. Sí todos eligieron el mismo valor, ese es el estimado. Caso contrario se discute y vuelve a elegir. Los valores menores indican una funcionalidad muy simple y los mayores demasiado compleja.

**Estimación en Proyectos Pequeños:** Los datos industriales de estimación no están calibrados para estos proyectos, Lo mejor es usar los datos históricos de la organización. Las estimaciones en proyectos pequeños dependen en gran medida de las capacidades de los individuos que hacen el trabajo. El PSP (Personal Software Process) es de gran ayuda en estos casos.

**Proyectos de Mantenimiento:** Cuando se estima el tamaño de un proyecto hay que tener en cuenta que insertar una nueva funcionalidad será factible si la arquitectura existente del producto puede acomodarse a dicha función. Puede existir una sobreestimación si se utilizan modelos de estimación calibrados para nuevos proyectos.

Métricas

**Métrica:** Es una medida del grado en que un sistema, componente o proceso posee un atributo dado.

**Indicador:** Es una métrica o combinación de métricas que proporcionan una visión profunda del proceso del software, del proyecto de software o del producto en sí.

**Métrica Directa:** Es una métrica de la cual se pueden realizar mediciones sin depender de ninguna otra métrica. Es un método de medición.

**Métrica Indirecta:** Es una métrica cuya forma de medir es una función de cálculo, es decir, las mediciones de dicha métrica utilizan las medidas obtenidas de otras métricas.

**Confiabilidad y Validez:** Muestran la calidad de la métrica. La validez es el grado en el cual la métrica refleja el significado real del concepto que se observa. La confiabilidad es la consistencia de un número de mediciones usando el mismo método sobre el mismo fenómeno. Puede representarse con la desviación estándar, a menor variación más confiable es la métrica. Las métricas confiables no necesariamente son válidas y viceversa.

**Métricas de Software:** Es la aplicación continua de mediciones basadas en técnicas para el proceso de desarrollo de software y sus productos para suministrar información relevante a tiempo. De esta forma el líder de proyecto puede mejorar el proceso y sus productos.

Las métricas de software se dividen en métricas de proceso, de proyecto y de producto. Las métricas del proyecto se consolidan para crear métricas de proceso que sean públicas para toda la organización del software.

Las métricas para un desarrollador pueden ser esfuerzo, duración estimada y actual de una tarea, etc. Para una organización pueden ser el tiempo calendario, los defectos del reléase, etc. Para un equipo de desarrollo puede ser el tamaño del producto, el número de tareas planificadas y completadas, etc.

Las métricas dan visibilidad a los problemas y proveen información para que las personas tomen mejores decisiones.

**Métricas Agiles:** Se debe medir solo lo necesario. La información para cada involucrado debe ser solo la necesaria para tomar decisiones, en un nivel de detalle que puedan usar, con el alcance y en el tiempo que les interesa.

El estilo de los procesos ágiles, las características de la organización, y el tipo de trabajo que se hace, son algunos de los factores que influyen en la elección de las métricas. Los dos principios ágiles que guían la elección de las métricas son: “La prioridad es satisfacer al cliente con entregas tempranas y frecuentes” y “El software trabajando es la principal medida de progreso”.

Para las metodologías agiles no son relevantes el diagrama de Gantt, el porcentaje completado, o el tiempo real vs. el estimado.

**Clases de Métricas:** La métrica informacional informa lo que está pasando, el diagnóstico indica las áreas de mejora y la motivacional influye en el comportamiento. Una métrica puede funcionar en más de una categoría.

**Running Tested Features:** Responde a las características probadas entregadas al cliente. Es informacional porque es una medida directa de los resultados entregados, diagnostico porque si por ejemplo es llano o disminuye en el tiempo indica que hay un problema, y motivacional porque los miembros del equipo quieren ver un incremento en el RTF.

**Capacidad:** Es una estimación de cuanto trabajo puede completarse en un período dado basado en la cantidad de tiempo ideal disponible del equipo. Se puede medir en esfuerzo o story points. Se deben aplicar estimaciones honestas a las tareas para calcular capacidades realistas. Es informacional porque muestra la disponibilidad del equipo en lo que resta del sprint, es diagnóstico porque la tendencia de la capacidad indica si el equipo podrá cumplir con los compromisos, y motivacional porque el equipo se enorgullece al alcanzar los compromisos.

**Velocidad:** Es una observación empírica de la capacidad del equipo para completar el trabajo por iteración. No es una estimación u objetivo a alcanzar. Está basada en el tamaño de los ítems de trabajo propio de cada equipo. Es comparable entre iteraciones para un equipo y proyecto dado, pero no así entre equipos y proyectos. Es informacional porque muestra la capacidad de trabajo del equipo, es diagnostico porque las tendencias de la velocidad indican problemas diferentes, y motivacional porque el equipo se enorgullece de lograr una alta velocidad.

**Burndown Chart:** Es una representación gráfica del trabajo por hacer en un proyecto en el tiempo. Es informacional porque es una medida directa del trabajo restante, diagnostico porque indica si el cambio de alcance o rendimiento del equipo es la causa de la variación del cronograma, y motivacional porque el equipo está motivado para ver cuando se termina el proyecto y ver la reducción de trabajo pendiente.

Prueba de Software

**Prueba de Software:** Es someter un software a ciertas condiciones para demostrar si es o no válido a los requerimientos planteados. Además valida que las funciones se implementen correctamente. Agrega valor al producto y al proceso de desarrollo solo sí se consideran los resultados generados.

Es un proceso destructivo que trata de encontrar defectos en el código. Se debe tener una actitud negativa para demostrar que algo es incorrecto. El test exitoso es aquel que encuentra muchos defectos, por lo que un desarrollo exitoso puede llevar a un test no exitoso. Se alcanza el éxito si nosotros encontramos el error y no el cliente.

No hay testing confiable si no se clasifican los errores. Una clasificación posible es: invalidante, severo, leve, mejora y cosmético.

**Principios del Testing:**

* Un programador debería evitar probar su propio código.
* El testing es una disciplina profesional que requiere gente entrenada y competente.
* Se debe cultivar una actitud de equipo positiva para la destrucción creativa.
* Sí el testing no es formal, intercambiar código entre desarrolladores.

**Cuando Parar:** Es imposible probar el 100% del proyecto, por lo que es necesario definir una estrategia para saber cuándo dejar de probar. Algunas de estas son:

* Pasa exitosamente el conjunto de pruebas diseñado y la cobertura estructural.
* Good Enough: Cierta cantidad de fallas no críticas es aceptable.
* Defectos detectados es similar a la cantidad de defectos estimados.

**Planificación de Pruebas:** Las pruebas se planifican al inicio. Una vez que los requerimientos son estables se diseñan casos de prueba. Es posible planear el testing sin haber codificado nada ya que se puede planear a partir de los Casos de Uso. Se deben definir estándares y plantillas para los documentos.

**Etapas de Testing:** El plan de tareas inicia con el Startup donde se obtiene el “plan de testing”. Luego se prepara el ambiente y genera el “documento de casos de prueba”. Aquí inicia un ciclo repetitivo donde se ejecuta el plan, se genera el “reporte de incidentes”, se hace un seguimiento de los incidentes y se vuelve a ejecutar, hasta que se obtiene el “Informe final”.

En el plan de tareas se pueden identificar los roles:

* Líder de testing: Es el responsable de la planificación, seguimiento y control del testing.
* Diseñador de la prueba: Define la estrategia de prueba y las condiciones de test.
* Tester: Es el responsable de elaborar el documento de requerimientos de datos para la generación de los casos de test y de ejecutar las pruebas.
* Responsable del ambiente: Administra el ambiente de pruebas y controla el pasaje del ambiente de desarrollo al de testing.
* Proveedor de datos: Brinda los datos a partir del documento de requerimientos asociado.
* Referente funcional: Es con quien se validan las condiciones de test especificadas y los resultados esperados.
* Líder de desarrollo: Es con quien se monitorea el estado del producto a testear, a quien se reporta el resultado, y con quien se validan los incidentes encontrados.

**Niveles de Prueba – Unitaria:** La primera etapa de la prueba está enfocada a los componentes más pequeños del software que se pueden probar. En general es hecha por los desarrolladores. Algunos aspectos que se verifican son: valores de inicialización incorrectos, precisión en las funciones de cálculo, manejo de errores inadecuado, etc.

**Niveles de Prueba – Integración:** Es un test orientado a verificar que las partes de un sistema que funcionan bien aisladamente, también lo hagan en conjunto. No hay que confundirlo con testear un sistema integrado. Los módulos críticos deben ser probados lo más temprano posible. Los puntos clave del test de integración son: Conectar de a poco las partes más complejas; y minimizar la necesidad de programas auxiliares.

**Niveles de Prueba – Sistema:** Es un tipo de prueba de versión realizado para conocer la estabilidad de la aplicación antes de realizar una prueba en profundidad. Se realiza cuando la aplicación está funcionando como un todo. Es como un test de unidad, formada por otras unidades. La diferencia con el test de unidad es que puede ser encarado por QA y las funcionalidades tienen visibilidad para el cliente.

**Niveles de Prueba – Aceptación:** Es la prueba realizada por el usuario para ver si la aplicación se ajusta a sus necesidades. Está formado por la prueba realizada por el usuario en ambiente de laboratorio (alfa), como la prueba en ambientes reales (beta). Conviene realizarla ya que al hacerla el cliente, da valor agregado.

**Pruebas de Regresión:** La prueba de regresión implica repetir todos los casos de prueba de una aplicación a medida que se avanza en la prueba de nuevos módulos, lo que conlleva mucho esfuerzo. Al concluir un ciclo de pruebas y reemplazarse la versión del sistema sometido al mismo, debe realizarse una verificación total de la nueva versión, a fin de prevenir la introducción de nuevos defectos al intentar solucionar los detectados.

**Métodos de Caja Negra:** Son métodos dinámicos donde se prueba con distintas entradas y salidas, el sistema sin analizar su funcionalidad interna. Los métodos más utilizados son:

* Partición de equivalencias: Es un proceso sistemático que identifica un conjunto de clases de pruebas representativas de conjuntos más grandes.
* Análisis de valores límite: Es una variante de la partición de equivalencias, donde se seleccionan los bordes de una clase como representativos.
* Adivinanzas de defectos: Es un enfoque ad hoc, basado en intuición y experiencia, para identificar pruebas que probablemente expondrán defectos.

**Métodos de Caja Blanca:** Es una prueba estática donde se prueba un código determinado. Consiste en diseñar casos de prueba que garanticen que: todos los caminos dentro de un módulo han sido ejercitados por lo menos una vez, se prueben todos los ciclos dentro y sobre sus límites operacionales, y se asegure la validez de las estructuras internas. El testing de caja blanca detecta errores que pueden no ser detectados mediante el testing de caja negra; hay ciertos caminos lógicos que se cree que no serán ejecutados cuando en realidad se ejecutan en forma regular. Los métodos de caja blanca son:

* Cobertura de secuencias.
* Cobertura de decisión.
* Cobertura de condición.
* Cobertura de decisión/ condición.
* Cobertura de múltiple.
* Cobertura de enunciados.
* Cobertura de loop.

**Complejidad ciclomática:** Es una métrica que provee una medición cuantitativa de la complejidad lógica de un programa. En 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.

La complejidad ciclomática de McCabe se basa en la teoría y flujo de grafos, donde cada nodo representa una instrucción del programa y las vías de ejecución son las aristas. Sabiendo la complejidad ciclomática es posible determinar el riesgo.

**Diseño de Casos de Prueba:** Los casos de prueba que tienen mayor probabilidad de detectar el mayor número de errores son: particiones de equivalencias, análisis de valores límites y gráficos de causa – efecto (Caja Negra); y cobertura de sentencias, de condiciones y de bucles (Caja Blanca). En una prueba rigurosa se recomienda el uso de varios de los métodos ya que todos tienen sus fortalezas y debilidades.

Calidad y Modelos de Mejora

**Calidad:** Son los aspectos de un producto que se relacionan con su habilidad de alcanzar las necesidades requeridas. La calidad debe estar embebida, no se puede inyectar. Es un esfuerzo de todos donde la gente debe estar capacitada y se necesita soporte gerencial. El aseguramiento de la calidad debe planificarse, no se puede controlar lo que no se mide. El aumento de las pruebas no aumenta la calidad.

**Calidad y Proceso de Desarrollo:** El proceso es el único factor controlable al mejorar la calidad del software y su rendimiento como organización. Cuando el proceso no está definido, sus actividades no están definidas y por ende no se pueden planificar, controlar ni asegurar los resultados. Es importante usar el proceso adecuado y no solo usarlo por ser un estándar.

**Aseguramiento de Calidad de Software:** La administración de la calidad del software implica la definición de estándares y procesos de calidad apropiados a la, la planificación de calidad seleccionando los procedimientos y estándares para un proyecto, y el control de la calidad asegurando que los procedimientos sean respetados por el equipo de desarrollo. El Grupo de Aseguramiento de Calidad debería reportar a alguien interesado en la calidad del software, y ubicarse muy cerca de la gerencia de primer nivel.

Los estándares son la clave para la administración de calidad efectiva ya que encapsulan las mejores prácticas y evitan los problemas del pasado. El problema es que no se consideran como muy relevantes e implican tiempo a menudo considerado burocrático. Los estándares pueden ser internacionales, nacionales, organizacionales o del proyecto.

**Calidad de Proceso y Producto:** La calidad de un producto está influenciada por la calidad del proceso de producción. Esto es importante sobre todo cuando los atributos de calidad son difíciles de evaluar. La relación es más compleja en el software ya que las habilidades y experiencia de los individuos es importante en el desarrollo de software, y por los factores externos como el cronograma de desarrollo acelerado.

**Modelos de Mejora:** La mejora de procesos requiere un compromiso en todos los niveles de la organización, objetivos medibles y consensuados, confidencialidad y confianza. Algunos modelos para la mejora de procesos son SPICE e IDEAL.

La mejora de procesos permite reducir el re trabajo, los costos de desarrollo y mantenimiento. Aparte otorga un marco para mediciones y seguimiento. Reduce los ciclos de construcción y prueba y permite ciclos de diseño más largos. Se definen los roles y responsabilidades, mejora la comunicación y la disciplina. Se logra una reducción temprana de los defectos. La mejora ha demostrado retornos de inversión sustanciales.

CMMI 1.3

**Introducción:** Integración de Modelos de Madurez y Capacidad. No es una norma, no se certifica, sino que se acredita. CMMI-DEV brinda la guía para medir, monitorear y administrar los procesos de desarrollo. CMMI-SVC provee la guía para entregar servicios internos o externos. CMMI-ACQ provee la guía para permitir seleccionar y administrar productos y servicios.

**Representación:** Por etapas se definen los niveles de 1 a 5 en un conjunto de áreas de proceso. Cada uno indica la madures organizacional y provee una única clasificación que facilita comparaciones entre organizaciones.

Continua presenta los niveles 0 a 5 definidos por cada área de proceso. Cada uno indica la capacidad del área de proceso. Permite comparaciones sobre la base de cada área de proceso.

**Organizaciones Inmaduras:** Improvisan los procesos de software durante el curso del proyecto; y si el proceso existiera no es rigurosamente aplicado. Son reaccionarias y se concentran en apagar incendios. No cuentan con bases para juzgar la calidad y las planificaciones y presupuestos se exceden por no basarse en estimaciones realistas.

**Organizaciones Maduras:** Poseen la habilidad de administrar los procesos de desarrollo y mantenimiento de software, los cuales se comunican a todo el personal. Los administradores monitorean la calidad de los productos y la satisfacción del cliente. Con planificación y presupuestos realistas logra alcanzarse los resultados esperados. Los roles y responsabilidades se encuentran definidos.

**Niveles de Representación por Etapas:** Los niveles van de 1 a 5, siendo en los niveles más altos donde se reduce el riesgo y aumenta la productividad y calidad:

1. Inicial: Los procesos son impredecibles, están pobremente controlados y es reactivo.
2. Administrado: Los procesos están caracterizados por proyectos y a menudo es reactivo.
3. Definido: Los procesos están caracterizados por la organización y es proactivo.
4. Cuantitativamente Administrado: Los procesos están medidos y controlados.
5. Optimizado: Se hace foco en la mejora de procesos.

**Áreas de Proceso del Nivel Administrado:** Se realiza la administración de requerimientos, el planeamiento de proyectos, el monitoreo y control de proyectos, la medición y análisis, el aseguramiento de calidad del proceso y producto, la administración de configuración, y Administración de Acuerdo con el Proveedor (tercerización).

**Estructura de CMMI por Etapas:** Cada nivel de CMMI tiene dos o más áreas de proceso. Por cada área de proceso se define su propósito, áreas de proceso relacionadas, sus metas específicas, y metas genéricas. Para las metas específicas se esperan varias prácticas específicas de las que se especifican ejemplos de productos de trabajo, y sub prácticas. Para las metas genéricas se esperan varias prácticas genéricas de las que se especifican sub prácticas, y elaboraciones de prácticas genéricas.

**Nivel 2 – Administración de Requerimientos (REQM):** El propósito es administrar los requerimientos de los productos y componentes de producto del proyecto para asegurar alineación entre estos requerimientos y los planes y productos de trabajo.

**Nivel 2 – Planificación de Proyectos (PP):** Consiste en establecer y mantener los planes que definen las actividades del proyecto.

**Nivel 2 – Monitoreo y Control de Proyecto (PMC):** Provee la comprensión del progreso del proyecto para que se puedan tomar las acciones correctivas apropiadas cuando la performance del proyecto se desvía significativamente de los planes.

**Nivel 2 – Administración de Acuerdo con el Proveedor (SAM):** Consiste en administrar la adquisición de productos y servicios de proveedores. Aplica solo cuando se terceriza.

**Nivel 2 – Medición y Análisis (MA):** El propósito es desarrollar y sostener una capacidad de medición que sea utilizada para soportar las necesidades de información de la administración.

**Nivel 2 – Aseguramiento de Calidad de Proceso y de Producto (PPQA):** Consiste en proveer a la administración y al personal la comprensión objetiva respecto de los procesos y los productos de trabajo asociados.

**Nivel 2 (PA) – Administración de la Configuración (CM):** Consiste en establecer y mantener la integridad de los productos de trabajo utilizando identificación de configuración, control de configuración, reporte de estado de configuración y auditorías de configuración.

**Objetivos Genéricos:** Son genéricos porque aparecen en múltiples áreas de proceso. Cada área de proceso tiene un solo objetivo genérico, y cumplirlo significa mejorar el control en la planificación e implementación de los procesos asociados con esa área de proceso.

El objetivo genérico de todas las áreas de proceso de nivel dos es que el proceso este institucionalizado como un proceso administrado, es decir, un proceso ejecutado que es planeado y realizado en concordancia con una política, usa personas capacitadas que tienen los recursos para producir salidas controladas, es controlado y evaluado por su adherencia a la descripción que se hizo de él.

El objetivo genérico de todas las áreas de proceso de nivel tres a cinco es que el proceso este institucionalizado como un proceso definido. Un proceso definido es un proceso administrado que es adoptado desde un conjunto de procesos estándares de la organización; tiene una descripción de proceso que es mantenida y genera productos de trabajo, mediciones y otra información de mejora de procesos.

**Prácticas Genéricas:** Son actividades que aseguran que los procesos asociados con el área de proceso serán efectivos, repetibles y duraderos. Contribuyen a cumplimiento del objetivo genérico cuando es aplicado a un área de proceso en particular. Las prácticas genéricas para el nivel dos son:

* Establecer una política organizacional: para la planificación y ejecución del proceso X.
* Planificar el proceso.
* Proveer recursos: adecuados para la ejecución del proceso X, desarrollando productos de trabajo y proveyendo servicios del proceso X.
* Asignar responsabilidad: y autoridad para ejecutar el proceso, desarrollando los productos de trabajo, y proveer los servicios del proceso X.
* Entrenar a las personas: para ejecutar o dar soporte al proceso X.
* Administrar configuraciones: ubicar los productos de trabajo diseñados del proceso X, bajo los niveles de gestión de configuración apropiados.
* Identificar e involucrar a las personas relevantes.
* Monitorear y controlar el proceso: contra el plan para realizarlo y tomar acciones correctivas.
* Evaluar objetivamente la adherencia: del proceso X contra su descripción, estándares y procedimientos y comunicar no conformidades.
* Revisar el estado con el nivel de administración más alto: las actividades, estados y resultados del proceso X.

Las prácticas genéricas del nivel tres son:

* Establecer un proceso definido.
* Recolectar información de mejora.

Auditoria de Calidad de Software

**Introducción:** La auditoría de calidad de software es la evaluación de los productos o procesos de software para asegurar el cumplimiento con estándares, especificaciones y procedimientos, basada en un criterio objetivo incluyendo documentación que especifique la forma o contenido de los productos a ser desarrollados, el proceso por el cual se desarrollan los productos y cómo debería medirse el cumplimiento con los estándares. Algunas técnicas comunes son el Checklist, el Muestreo y la Revisión de Registros.

**Beneficios:** Permite evaluar el cumplimiento del proceso de desarrollo y determinar la implementación efectiva del proceso de desarrollo organizacional, del proyecto, y de las actividades de soporte. Proveen mayor visibilidad a la gerencia sobre los procesos de trabajo. 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.

**Auditorias de Proyecto**: Se llevan a cabo de acuerdo a lo establecido en el Plan de Auditoria de Calidad de Software (PACS), el cual indica al responsable de realizar las auditorias. Las inspecciones de software y las revisiones de la documentación de diseño y prueba deberían incluirse en esta auditoría. El objetivo es verificar objetivamente la consistencia del producto a medida que evoluciona a lo largo del proceso de desarrollo verificando que:

* Las interfaces de HW y SW sean consistentes con los requerimientos de diseño en la ERS.
* Los requerimientos funcionales de la ERS se validen en el Plan de Verificación y Validación.
* El diseño del producto satisface los requerimientos funcionales de la ERS.
* El código es consistente con el DDS.

**Auditoria Funcional:** Compara el software que se ha construido con los requerimientos de software especificados de la ERS, buscando asegurar que el código implementa sólo los requerimientos y las capacidades descriptos en la ERS. El responsable de QA debe validar si la matriz de rastreabilidad está actualizada.

**Auditoria Física:** Compara el código con la documentación de soporte, buscando asegurar que la misma sea consistente y describa correctamente el código desarrollado. El software puede entregarse sólo cuando se han corregido las desviaciones encontradas.

**Roles – Gerente de SQA:** Es el encargado de preparar el plan de auditorías, calcular el costo de las mismas, asignar los recursos y resolver las no conformidades.

**Roles – Auditor:** Recolecta y analiza los datos relevantes para tomar conclusiones acerca del proyecto auditado, prepara el reporte y hace el seguimiento de los planes de acción acordados con el auditado.

Para lograr el éxito de la auditoria debe tener buena relación con el auditado, evitar culpar a la gente, actuar éticamente, etc.

**Roles – Auditado:** Participa en la auditoria proporcionándole los datos al auditor y respondiendo el reporte de auditoria. Propone el plan de acción para deficiencias citadas en el reporte y comunica el cumplimiento del plan de acción.

**Proceso de Auditoria:**

1. Planificación: El auditado solicita la auditoria y se acuerda lugar, fecha, alcance, y objetivos y participantes de la misma. El auditor puede solicitar la documentación preliminar y el auditado prepara la documentación que será requerida.
2. Ejecución: En la reunión de apertura se presentan los participantes de la auditoria y el equipo auditor. Se revisa lo acordado durante la planificación, se informa cómo será el proceso. Luego se realiza la ejecución propiamente dicha donde se responden las preguntas, se muestra la evidencia y se completa el checklist.
3. Análisis y reporte del resultado: Durante la evaluación de los resultados se revisa el checklist, las anotaciones, la documentación solicitada, se verifican las desviaciones del proceso y se realiza un informe preliminar. En la reunión preliminar se presentan y acuerdan los resultados y se define la entrega del reporte final.
4. Seguimiento: Se analizan las desviaciones y prepara un plan de acción que se envía al auditor para que lo revise y apruebe. Hecho esto el auditado realiza el seguimiento de las acciones e informa el cumplimiento de las mismas. Una vez verificadas por el auditor, se cierran las desviaciones y generan los informes de cierre.

**Checklist de Auditoria:** El checklist suele contener: fecha de la auditoria, lista de auditados y sus roles, nombre del auditor y del proyecto, fase actual del proyecto, objetivo y alcance de la auditoria y la lista de preguntas.

Con las preguntas sobre la planificación de proyectos, el auditor debe asegurarse que los mismos estén basados en los requerimientos, que se haya realizado lo planificado, que los cambios hayan sido aprobados y documentados, etc. Con las preguntas sobre la fase de requerimientos, el auditor debe asegurarse que se han revisado y aprobado los requerimientos por todos los involucrados, y que si estos presentan cambios, se hayan documentado.

**Técnica de Cuestionario:** Se debe iniciar con preguntas de final abierto (quien, como, qué), realizar preguntas cortas y puntuales, y finalizar con preguntas de final cerrado para aclarar conceptos.

**Reporte de Auditoria:** Los contenidos básicos de un reporte son: la identificación y fecha de la auditoria, el auditor y los auditados, nombre y fase del proyecto auditado, la lista de resultados, los comentarios y la solicitud de planes de acción.

En la lista de resultados se pueden presentar tres tipos de resultados:

* Buenas Practicas: práctica, procedimiento o instrucción que se ha desarrollado mucho mejor de lo esperado. Por ejemplo cuando ha adoptado una práctica superior a cualquier otra que se haya visto.
* Desviaciones: requieren un plan de acción por parte del auditado. Se presentan cuando hay disconformidad de un producto respecto a sus requerimientos, o incertidumbre sobre la calidad del producto o las actividades. Las desviaciones se registran con la forma de Solicitud de Acción Correctiva.
* Observaciones: Son condiciones que deberían mejorarse pero no requieren un plan de acción, por ejemplo, una condición que puede resultar en una futura desviación.

**Guía para Encontrar Desviaciones:** Para estar seguros de que se está frente a una desviación hay que asegurarse que se pueda probar la existencia de la desviación, si esta agrega valor al proyecto, y si se puede rastrear la corrección.

**Métricas de Auditoria:** Cada organización debe establecer las métricas más apropiadas, por ejemplo, esfuerzo por auditoria, cantidad de desviaciones, duración de auditoria, etc.