



Técnico en
< DESARROLLO DE SOFTWARE >

***Fundamentos de Construcción
de Software***

(CC BY-NC-ND 4.0)
International

Attribution-NonCommercial-NoDerivatives 4.0



Atribución

Usted debe reconocer el crédito de una obra de manera adecuada, proporcionar un enlace a la licencia, e indicar si se han realizado cambios. Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que tiene el apoyo del licenciante o lo recibe por el uso que hace.



No Comercial

Usted no puede hacer uso del material con fines comerciales.



Sin obra derivada

Si usted mezcla, transforma o crea un nuevo material a partir de esta obra, no puede distribuir el material modificado.

No hay restricciones adicionales - Usted no puede aplicar términos legales ni medidas tecnológicas que restrinjan legalmente a otros hacer cualquier uso permitido por la licencia.

<http://creativecommons.org/licenses/by-nc-nd/4.0/>



Fundamentos de Construcción de Software

Unidad III

Gestión de Proyectos

1. Gestión del proyecto

La gestión del proyecto es una parte muy importante del mismo esto se debe a que los proyectos siempre están asociados a diversos factores, tales como tiempo de entrega y un presupuesto. El trabajo del administrador del proyecto es asegurar software de alta calidad y que se adecúe a las restricciones impuestas para asegurar que el proyecto sea exitoso.

Los objetivos primordiales de la gestión del proyecto son los siguientes:

- Entregar el proyecto al cliente en el tiempo acordado.
- Mantener todos los precios dentro del presupuesto.
- Entregar software que cumple las expectativas del cliente.
- Mantener un equipo de desarrollo en óptimas condiciones.

Estos objetivos pueden ser aplicados a cualquier proyecto pero sin embargo, el software es diferente que otros proyectos por las razones que se mencionan a continuación. Estos factores hacen que el desarrollo de software sea particularmente difícil.

- El software no es tangible. Esto puede incrementar la dificultad de la gestión del proyecto ya que no es algo que pueda verse a simple vista, es necesario reunir evidencia para medir el progreso.
- Muchos proyectos son empezados desde cero y se construyen con diferencias en la arquitectura, esto los hace difíciles de mantener.
- Los procesos son variables y se adecúan a una organización. El software no puede construirse como con una receta donde las especificaciones y pasos están definidos con claridad. Además, el software debe ser hecho a medida para poder cumplir el criterio de aceptación.

El trabajo de un administrador de proyectos es muy complejo debido a que involucra varias tareas. Un administrador de proyectos debe de estimar los tiempos, asignar tareas y recursos así como también encargarse de la planeación. También son los responsables de reportar los avances del proyecto. Una característica muy importante que debe tener un administrador de proyectos es tener la capacidad de hablar y comunicarse para poder compartir información crítica de los avances del proyecto. Los administradores de proyecto deben de evaluar los riesgos involucrados y los efectos que éstos puedan llegar a causar al progreso del proyecto y tomar las acciones necesarias para mitigar los efectos adversos. Adicionalmente a esto, los administradores deben de dirigir a los equipos de trabajo y deben de tratar de maximizar la productividad.

2. Gestión de riesgos

La gestión de riesgos es una de las funciones principales de un administrador de proyectos e involucra anticiparse a los posibles riesgos que puedan afectar al horario o

la calidad del software que se desarrolla o a la empresa en general. Existen tres categorías en las que se pueden clasificar los riesgos:

- **Riesgos de proyecto:** Estos riesgos afectan directamente con el horario o los recursos. Un ejemplo de esto es cuando el proyecto pierde a un miembro experimentado del grupo de trabajo, ya sea por una reasignación de recursos o algún otro imprevisto. Esto hace que los miembros restantes deban adquirir las habilidades necesarias del miembro faltante y esto puede tomar mucho tiempo, retrasando el proyecto e incrementando el costo.
- **Riesgos de producto:** Son aquellos ligados a la calidad o al rendimiento del software desarrollado. Un ejemplo de esto puede ser cuando se necesita utilizar librerías externas pagadas pero que no pueden probarse con anterioridad y que al momento de comprar la licencia e integrar la librería, esta no se comporte con el rendimiento esperado.
- **Riesgos del negocio:** Son los riesgos que afectan a la empresa como tal. Un ejemplo de riesgo es cuando sale al mercado otra empresa con un producto similar y que sea una opción que el cliente considere. Otro ejemplo que aplica a esta categoría son cambios de legislaciones que modifiquen en gran manera el sistema desarrollado.

La gestión de riesgos comprende varias etapas:

1. **Identificación de riesgos:** Identificar los posibles riesgos para el proyecto, el producto y los negocios. Comprende el descubrimiento de los posibles riesgos del proyecto. En principio, no hay que valorarlos o darles prioridad en esta etapa

aunque, en la práctica, por lo general no se consideran los riesgos con consecuencias menores o con baja probabilidad.

Hay al menos seis tipos de riesgos que pueden aparecer:

- Riesgos de tecnología: Se derivan de las tecnologías de software o de hardware utilizadas en el sistema que se está desarrollando.
- Riesgos de personal: Riesgos asociados con las personas del equipo de desarrollo.
- Riesgos organizacionales: Se derivan del entorno organizacional donde el software se está desarrollando.
- Riesgos de herramientas: Se deriva de herramientas CASE y de otro software de apoyo utilizado para desarrollar el sistema.
- Riesgos de requerimientos: Se derivan de los cambios de los requerimientos del cliente y el proceso de gestionar dicho cambio.
- Riesgos de estimación: Se derivan de los estimados administrativos de las características del sistema y los recursos requeridos para construir dicho sistema.

2. **Análisis de riesgos:** Valorar las probabilidades y consecuencias de estos riesgos. Durante este proceso, se considera por separado cada riesgo identificado y se decide acerca de la probabilidad y la seriedad del mismo. No existe una forma fácil de hacer esto -recae en la opinión y experiencia del gestor del proyecto-. No se hace una valoración con números precisos sino en intervalos:

- La probabilidad del riesgo se puede valorar como muy bajo ($< 10\%$), bajo ($10-25\%$), moderado ($25-50\%$), alto ($50-75\%$) o muy alto ($>75\%$).
- Los efectos del riesgo pueden ser valorados como catastrófico, serio, tolerable o insignificante.

Una vez que los riesgos se hayan analizado y clasificado, se debe discernir cuáles son los más importantes que se deben considerar durante el proyecto. Este discernimiento debe depender de una combinación de la probabilidad de aparición del riesgo y de los efectos del mismo. En general, siempre se deben tener en cuenta todos los riesgos catastróficos, así como todos los riesgos serios que tienen más que una moderada probabilidad de ocurrir.

3. **Planificación de riesgos:** Crear planes para abordar los riesgos. ya sea para evitarlos o minimizar sus efectos en el proyecto. El proceso de planificación de riesgos considera cada uno de los riesgos clave que han sido identificados, así como las estrategias para gestionarlos. Otra vez, no existe un proceso sencillo que nos permita establecer los planes de gestión de riesgos. Depende del juicio y de la experiencia del gestor del proyecto.

Estas estrategias seguidas pueden dividirse en tres categorías.

- **Estrategias de prevención:** Siguiendo estas estrategias, la probabilidad de que el riesgo aparezca se reduce. Un ejemplo de este tipo de estrategias es la estrategia para evitar de defectos en componentes, tener componentes de respaldo.
- **Estrategias de minimización:** Siguiendo estas estrategias se reducirá el impacto del riesgo. Un ejemplo de esto es la estrategia frente a enfermedad

del personal, planificar quién puede cubrir a cada uno de los miembros del equipo.

- **Planes de contingencia:** Seguir estas estrategias es estar preparado para lo peor y tener una estrategia para cada caso. Un ejemplo de este tipo de estrategia es preparar un documento breve para el gestor principal que muestre que el proyecto hace contribuciones muy importantes a la empresa.

4. **Supervisión de riesgos:** Valorar los riesgos de forma constante y revisar los planes para la mitigación de riesgos tan pronto como la información de los riesgos esté disponible.

3. Gestión de personal

Las personas que trabajan en una organización de software son los activos más importantes. Cuesta mucho dinero reclutar y retener al buen personal, así que depende de los administradores de software garantizar que la organización obtenga el mejor aprovechamiento posible por su inversión. En las compañías y economías exitosas, esto se logra cuando la organización respeta a las personas y les asigna responsabilidades que reflejan sus habilidades y experiencia.

Es importante que los administradores de proyecto de software comprendan los conflictos técnicos que influyen en el trabajo del desarrollo de software. El administrador de proyecto, deberá estar al tanto de los problemas potenciales de administrar personal y debe tratar de desarrollar habilidades de gestión de recursos humanos. Existen cuatro factores críticos en la gestión de personal:

1. **Consistencia:** Todas las personas en un equipo de proyecto deben recibir un trato similar. Nadie espera que todas las distinciones sean idénticas, pero las personas podrían sentir que sus aportaciones a la organización se menosprecian.
2. **Respeto:** Las personas tienen distintas habilidades y los administradores deben respetar esas diferencias. Todos los miembros del equipo deben recibir una oportunidad para aportar. Desde luego, en algunos casos, usted encontrará que las personas simplemente no se ajustan al equipo y no pueden continuar, pero es importante no adelantar conclusiones sobre esto en una etapa temprana del proyecto.
3. **Inclusión:** Las personas contribuyen efectivamente cuando sienten que otros las escuchan y que sus propuestas se toman en cuenta. Es importante desarrollar un ambiente laboral donde se consideren todas las visiones, incluso las del personal más joven.
4. **Honestidad:** Como administrador, siempre debe ser honesto acerca de lo que está bien y lo que está mal en el equipo. También debe ser honesto respecto a su nivel de conocimiento técnico y voluntad para comunicar al personal más conocimiento cuando sea necesario. Si trata de encubrir la ignorancia o los problemas, con el tiempo, éstos saldrán a la luz y perderá el respeto del grupo

El administrador de proyecto, necesitará motivar a las personas con quienes trabaja, de manera que éstas contribuyan con lo mejor de sus habilidades. Motivación significa organizar el trabajo y el ambiente laboral para alentar a los individuos a desempeñarse tan efectivamente como sea posible. Si las personas no están motivadas, no estarán interesadas en la actividad que realizan. Así que trabajarán con lentitud, y será más

probable que cometan errores y que no contribuyan con las metas más amplias del equipo o la organización.

Para fomentar este ánimo, hay que saber un poco acerca de qué motiva a la gente. Maslow (1954) sugiere que las personas se sienten motivadas para cubrir sus necesidades, las cuales se ordenan en una serie de niveles.

Los niveles más bajos de esta jerarquía representan necesidades fundamentales de alimentación, sueño, etcétera, y la necesidad de sentirse seguro en un ambiente. Las necesidades sociales se relacionan con el hecho de sentirse parte de un grupo social. Las necesidades de estima representan la necesidad de sentirse respetado por otros, y las necesidades de autorrealización tienen que ver con el desarrollo personal. Las personas requieren cubrir las necesidades de nivel inferior, como el hambre, antes de las necesidades de nivel superior, que son más abstractas.

Las personas que trabajan en organizaciones de desarrollo de software, por lo general, no están hambrientas ni sedientas ni físicamente amenazadas por su ambiente. Por lo tanto, asegurarse de que se cubren las necesidades sociales, de estima y autorrealización de las personas es más importante desde un punto de vista administrativo.

Trabajo en equipo

La mayor parte del software profesional se desarrolla mediante equipos de proyecto, cuyo número de miembros varía entre dos y varios cientos de personas. Sin embargo, como es imposible que todos los integrantes de un grupo grande trabajen en conjunto en un solo problema, los equipos grandes habitualmente se dividen en grupos más

pequeños. Cada grupo es responsable de desarrollar parte del sistema global. Como regla general, los grupos del proyecto de ingeniería de software no deben tener más de 10 miembros. Cuando se usan grupos pequeños se reducen los problemas de comunicación. Todos conocen a todos los demás, y el grupo en su conjunto puede reunirse en torno a una mesa para estudiar el proyecto y el software que desarrollan.

Conformar un grupo que tiene el equilibrio justo de habilidades técnicas, experiencia y personalidades es una tarea administrativa fundamental. Sin embargo, los grupos exitosos son mucho más que una colección de individuos con el equilibrio justo de habilidades. Un buen equipo es cohesivo y tiene espíritu de grupo. Las personas que participan están motivadas tanto por el éxito del grupo como por sus metas personales.

En un grupo cohesivo, los miembros piensan que el equipo es más importante que los individuos que lo integran. Los miembros de un grupo cohesivo bien liderado son leales al equipo. Se identifican con las metas del grupo y con los demás miembros. Tratan de proteger al grupo, como entidad, de cualquier interferencia externa. Esto hace que el grupo sea sólido y pueda enfrentar problemas y situaciones inesperadas.

4. Planificación del proyecto

La gestión efectiva de un proyecto de software depende de planificar completamente el progreso del proyecto. El gestor del proyecto debe anticiparse a los problemas que puedan surgir. Así como preparar soluciones a esos problemas. Un plan, preparado al inicio de un proyecto, debe utilizarse como un conductor para el proyecto. Este plan inicial debe ser el mejor posible de acuerdo con la información disponible. Éste evolucionará conforme el proyecto progrese y la información sea mejor

Para poder realizar la planificación se pueden seguir los siguientes pasos:

Establecer las restricciones del proyecto

Hacer la valoración inicial de los parámetros del proyecto

Definir los hitos del proyecto y productos a entregar

Mientras el proyecto no se haya completado o cancelado

Diseñar o ajustar la planificación en el tiempo del proyecto

Iniciar actividades acordes con la programación

Esperar (por un momento)

Revisar el progreso del proyecto

Revisar las estimaciones de los parámetros del proyecto

Actualizar la programación del proyecto

Renegociar las restricciones del proyecto y los productos a entregar

Si (surgen problemas) entonces

Iniciar la revisión técnica y la posible solución

fin de si

fin de repetir

La planificación es un proceso iterativo que solamente se completa cuando el proyecto mismo se termina. Conforme la información se hace disponible el plan debe revisarse regularmente. Las metas globales del negocio son un factor importante que debe considerarse cuando se formula el plan del proyecto. Conforme éstas cambien, serán necesarios cambios en el proyecto.

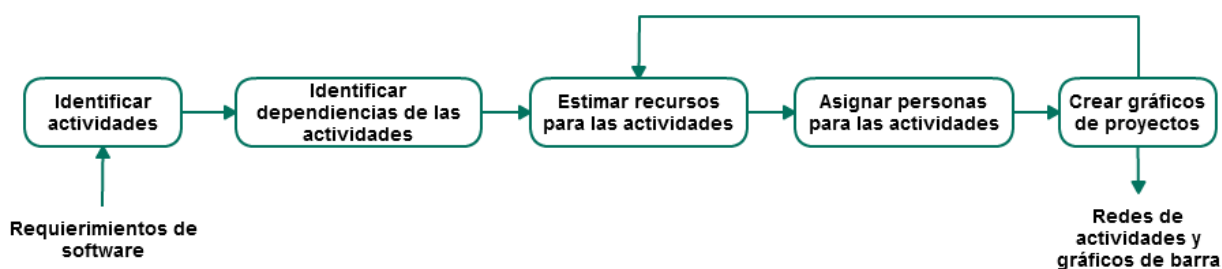
El proceso de planificación se inicia con una valoración de las restricciones que afectan al proyecto (fecha de entrega requerida, personal disponible, presupuesto global, etcétera). Ésta se lleva a cabo en conjunto con una estimación de los parámetros del proyecto, como su estructura, tamaño y distribución de funciones. Entonces se definen los hitos de progreso y productos a entregar, un hito de un proyecto es el resultado predecible de una actividad en el que se debe presentar un informe del progreso a la gestión. Los hitos ocurren de forma frecuente en un proyecto de software. Una entrega es un hito que se entrega al cliente del proyecto. En ese momento, el proceso entra en un ciclo. Se prepara un calendario para el proyecto y las actividades definidas en el calendario se inician o se continúan. Después de algún tiempo (por lo general 2 o 3 semanas) se revisa el proyecto y se señalan las discrepancias. Debido a que las estimaciones iniciales de los parámetros del proyecto son aproximaciones, el plan siempre deberá actualizarse.

Cuando se dispone de más información, los gestores del proyecto revisan las suposiciones del proyecto y la agenda. Si el proyecto se retrasa, tienen que renegociar con el cliente las restricciones del mismo y las entregas. Si esta renegociación no tiene éxito y no se puede cumplir el calendario, se debe llevar a cabo una revisión técnica. El objetivo de esta revisión es encontrar un enfoque alternativo que se ajuste a las

restricciones del proyecto y cumpla con las metas del calendario. Por supuesto, los gestores de proyectos inteligentes no suponen que todo irá bien. Durante el proyecto siempre surgen problemas en algunas descripciones. Las suposiciones iniciales y el calendario deben ser más bien pesimistas que optimistas. Debe haber suficiente holgura para que las contingencias en el plan, las restricciones del proyecto y los hitos no se tengan que negociar cada vez que se efectúa un ciclo en el plan.

Calendarización

La calendarización es una de las tareas más difíciles para los gestores de proyectos. Los gestores estiman el tiempo y los recursos requeridos para completar las actividades y organizarlas en una sucesión coherente. A menos que el proyecto a calendarizar sea similar a otro anterior, las estimaciones previas son una base incierta para la calendarización del nuevo proyecto. La estimación del calendario se complica más por el hecho de que proyectos diferentes pueden utilizar métodos de diseño y lenguajes de implementación diferentes.



Si el proyecto es técnicamente complejo, las estimaciones iniciales casi siempre son optimistas aun cuando los gestores traten de considerar las eventualidades. A este respecto, la calendarización del tiempo para la creación del software no es diferente a la de cualquier otro tipo de proyecto grande y complejo. Los nuevos aeroplanos, los puentes

e incluso los nuevos modelos de automóviles se retrasan debido a problemas no anticipados. Por lo tanto, los calendarios se deben actualizar continuamente en la medida que se disponga de mejor información acerca del progreso.

La calendarización del proyecto implica separar todo el trabajo de un proyecto en actividades complementarias y considerar el tiempo requerido para completar dichas actividades. Por lo general, algunas de éstas se llevan a cabo en paralelo. Debemos coordinar estas actividades paralelas y organizar el trabajo para que la mano de obra se utilice de forma óptima. Deben evitarse situaciones en que el proyecto entero se retrase debido a que no se ha terminado una actividad crítica.

Normalmente, las actividades del proyecto deben durar por lo menos una semana. Hacer subdivisiones más finas significa invertir una cantidad desproporcionada de tiempo en la estimación y revisión de tablas. También es útil asignar una cantidad de tiempo máxima de 8 a 10 semanas para realizar cualquier actividad. Si lleva más tiempo, se deben hacer subdivisiones.

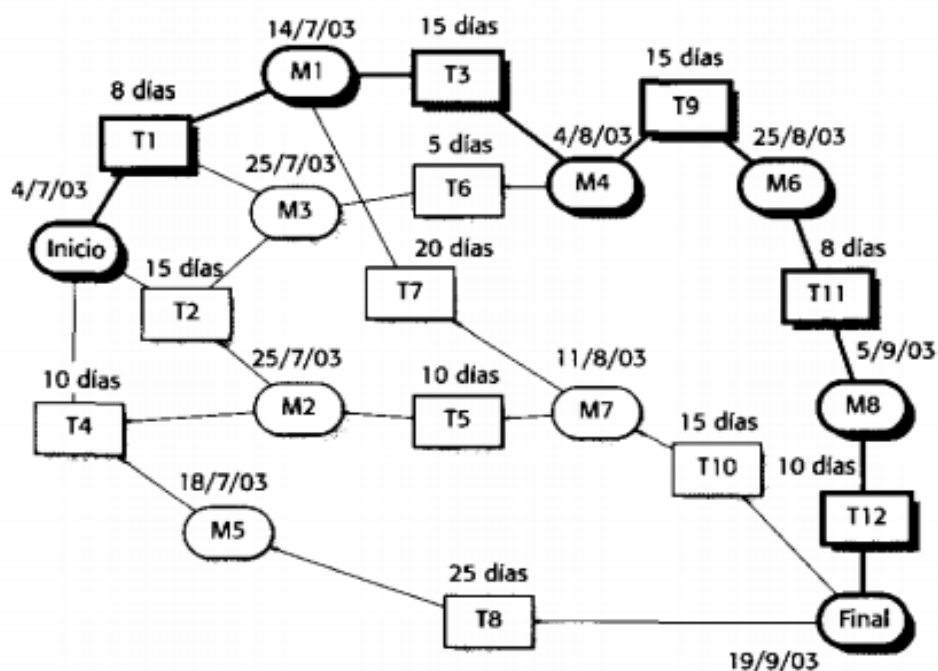
Al estimar la calendarización, los gestores no deben suponer que cada etapa del proyecto estará libre de problemas. Las personas que trabajan en él pueden enfermarse o renunciar, el hardware puede fallar y el software o hardware de soporte puede llegar tarde. Si el proyecto es nuevo y técnicamente complejo, ciertas partes podrían ser más complicadas y llevarían más tiempo del que se estimó originalmente.

Como en los calendarios, los gestores deben estimar los recursos necesarios para completar cada tarea. El recurso principal es el esfuerzo humano que se requiere. Otros recursos pueden ser el espacio en disco requerido en un servidor, el tiempo requerido

de hardware especializado, un simulador o el presupuesto para viajes del personal del proyecto.

Una buena regla práctica es estimar como si nada fuera a salir mal, y entonces incrementar la estimación para abarcar los problemas previstos. Con este mismo fin, a la estimación se le debe agregar un factor de contingencia adicional. Este factor extra de contingencia depende del tipo de proyecto, de los parámetros del proceso (fecha de entrega, estándares, etcétera) y de la calidad y experiencia de los desarrolladores de software que trabajen en el proyecto. Como regla, para los problemas previstos siempre debe agregarse un 30 % a la estimación original y otro 20 % para cubrir algunas cosas no previstas.

Por lo general, el calendario del proyecto se representa como un conjunto de gráficos que muestran la división del trabajo, las dependencias de las actividades y la asignación del personal.



Costos

Cuando estimamos el costo de una actividad, debemos responder a las siguientes preguntas:

1. ¿Cuánto esfuerzo se requiere para completar una actividad?
2. ¿Cuánto tiempo, de calendario, se necesita para completar una actividad?
3. ¿Cuál es el coste total de una actividad?

La estimación y la creación del calendario del proyecto se llevan a cabo de forma conjunta. Sin embargo, en las primeras etapas del proyecto se requieren algunas estimaciones de costos, antes de que se haga la planificación detallada.

Estas estimaciones son necesarias para establecer un presupuesto para el proyecto o para asignar un precio para el software de un cliente. Existen tres parámetros involucrados en el cálculo del coste total de un proyecto de desarrollo de software:

- Los costos hardware y software, incluyendo el mantenimiento.
- Los costos de viajes y capacitación.
- Los costos de esfuerzo (los costos correspondientes al pago de los desarrolladores).

Para muchos proyectos, los costos dominantes son los costos de esfuerzo. Las computadoras con potencia suficiente para desarrollar software son relativamente baratas. Aunque los costos de viajes pueden ser importantes si un proyecto se desarrolla en sitios distintos, son una pequeña parte comparados con los costos de esfuerzo. Además, el uso de correo electrónico, sitios web compartidos y videoconferencias reducen el coste de los viajes y del tiempo hasta en un 50%.

Los costos de esfuerzo no sólo son los salarios de los desarrolladores que intervienen en el proyecto. Las organizaciones calculan los costos de esfuerzo en función de los costos totales, donde se tiene en cuenta el coste total para hacer funcionar la organización y dividen éste entre el número de personas productivas. Por lo tanto, los siguientes costos son parte de los costos totales:

1. Costos de proveer, aclimatar e iluminar las oficinas.
2. Los costos del personal de apoyo como administrativos, secretarias, limpiadores y técnicos.
3. 3. Los costos de redes y las comunicaciones.
4. Los costos de los recursos centralizados como los recursos recreativos, lugares para comer, etc.
5. Los costos de seguridad social. pensiones. seguros privados, etc.

Este factor de sobrecarga normalmente es el doble del salario de un desarrollador, dependiendo del tamaño de la organización y sus sobrecargas asociadas. Por lo tanto, si a un desarrollador se le pagan Q.90,000 al año, el coste total de la organización es de 180,000 por año o Q.15.000 por mes. Una vez que se inicia el proyecto, los gestores deben actualizar regularmente sus estimaciones de tiempo y de coste. Esto ayuda en el proceso de planificación y en el uso efectivo de los recursos. Si los gastos actuales son significativamente mayores que los estimados, el gestor del proyecto deberá tomar alguna decisión. Esto implica que puede incorporar recursos adicionales al proyecto o modificar los trabajos a realizar.

5. Gestión de calidad

La gestión de la calidad del software permite señalar si éste tiene un escaso número de defectos y si alcanza los estándares requeridos de mantenibilidad, fiabilidad, portabilidad, etc. La gestión formal de la calidad es particularmente importante para equipos que desarrollan sistemas grandes y complejos. La documentación de la calidad es un registro de que es hecho por cada subgrupo en el proyecto. Esto ayuda a la gente a ver qué tareas importantes no deben ser olvidadas o que una parte del equipo no haga suposiciones incorrectas acerca de lo que otros miembros han hecho. La documentación de calidad es también un medio de comunicación sobre el ciclo de vida de un sistema. Ésta permite al grupo responsabilizarse de la evolución del sistema para saber qué ha hecho el equipo de desarrollo.

Para sistemas pequeños, la gestión de calidad es importante todavía, pero se debe adoptar una aproximación más informal. No son tan necesarios los documentos porque el grupo puede comunicarse informalmente. La clave de la calidad en el desarrollo de sistemas pequeños es el establecimiento de cultura de calidad y asegurarse de que todos los miembros del equipo hacen una aproximación positiva a la calidad del software.

La gestión de calidad del software se estructura en tres actividades principales:

1. **Garantía de la calidad:** El establecimiento de un marco de trabajo de procedimientos y estándares organizacionales que conduce a software de alta calidad.

2. **Planificación de la calidad:** La selección de procedimientos y estándares adecuados a partir de este marco de trabajo y la adaptación de éstos para un proyecto software específico.
3. **Control de la calidad:** La definición y fomento de los procesos que garanticen que los procedimientos y estándares para la calidad del proyecto son seguidos por el equipo de desarrollo de software.

La gestión de la calidad provee una comprobación independiente de los procesos de desarrollo software. Los procesos de gestión de la calidad comprueban las entregas del proyecto para asegurarse que concuerdan con los estándares y metas organizacionales. El equipo de garantía de calidad debe ser independiente del equipo de desarrollo para que puedan tener una visión objetiva del software. Ellos transmitirán los problemas y las dificultades al gestor principal de la organización.

Un equipo independiente debe ser responsable de la gestión de la calidad y debe informar al gestor del proyecto. El equipo de calidad no está asociado con ningún grupo de desarrollo, sino que tiene la responsabilidad de la gestión de la calidad en toda la organización. La razón de esto es que los gestores del proyecto deben mantener el presupuesto y la agenda. Si aparecen problemas, éstos pueden verse tentados de comprometer la calidad del producto para mantener su agenda. Un equipo independiente de calidad garantiza que los objetivos organizacionales y la calidad no sean comprometidos por consideraciones de presupuesto o agenda.

6. Estándares de calidad

Los estándares de software tienen una función muy importante en la gestión de calidad del software. Como se indicó, un aspecto importante del aseguramiento de calidad es la definición o selección de estándares que deben aplicarse al proceso de desarrollo de software o al producto de software. Como parte de este proceso QA, también pueden elegirse herramientas y métodos para apoyar el uso de dichos estándares. Una vez seleccionados éstos para su uso, deben definirse procesos específicos de proyecto para monitorizar el uso de los estándares y comprobar que éstos se siguieron.

Los estándares de software son importantes por tres razones:

1. Los estándares reflejan la sabiduría que es de valor para la organización. Se basan en conocimiento sobre la mejor o más adecuada práctica para la compañía. Con frecuencia, este conocimiento se adquiere sólo después de gran cantidad de ensayo y error. Configurarla dentro de un estándar, ayuda a la compañía a reutilizar esta experiencia y a evitar errores del pasado.
2. Los estándares proporcionan un marco para definir, en un escenario particular, lo que significa el término “calidad”. Como se dijo, la calidad del software es subjetiva, y al usar estándares se establece una base para decidir si se logró un nivel de calidad requerido. Desde luego, esto depende del establecimiento de estándares que reflejen las expectativas del usuario para la confiabilidad, la usabilidad y el rendimiento del software.

3. Los estándares auxilian la continuidad cuando una persona retoma el trabajo iniciado por alguien más. Los estándares aseguran que todos los ingenieros dentro de una organización adopten las mismas prácticas. En consecuencia, se reduce el esfuerzo de aprendizaje requerido al iniciarse un nuevo trabajo.

Existen dos tipos de estándares de ingeniería de software relacionados que pueden definirse y usarse en la gestión de calidad del software:

1. Estándares del 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 una definición de clase de objeto) 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.

Los estándares de proceso pueden incluir definiciones de especificación, procesos de diseño y validación, herramientas de soporte de proceso y una descripción de los documentos que deben escribirse durante dichos procesos.

Ejemplos:

Estándares de producto	Estándares de proceso
Formato de revisión de diseño	Realizar revisión de diseño
Estructura de documento de requerimientos	Enviar nuevo código para construcción de sistema
Formato de encabezado por método	Proceso de liberación de versión
Estilo de programación Java	Proceso de revisión de codificación
Formato de solicitud de cambio	Proceso de registro de prueba y aplicación del cambio

Los estándares deben entregar valor, en la forma de calidad aumentada del producto. No hay razón para definir estándares que sean costosos en términos de tiempo y esfuerzo, pues aplicarlos sólo conduce a mejoras secundarias en la calidad. Los estándares de producto deben diseñarse de forma que puedan aplicarse y comprobarse de manera efectiva en cuanto a costos, y los estándares de proceso deben incluir la definición de procesos que comprueben que se siguieron dichos estándares.

Estándares ISO 9001

Existe un conjunto internacional de estándares que pueden utilizarse en el desarrollo de los sistemas de administración de calidad en todas las industrias, llamado ISO 9000. Los estándares ISO 9000 pueden aplicarse a varias organizaciones, desde las industrias manufactureras hasta las de servicios. ISO 9001, el más general de dichos estándares, se aplica a organizaciones que diseñan, desarrollan y mantienen productos, incluido

software. El estándar ISO 9001 se desarrolló originalmente en 1987, y su revisión más reciente fue en 2008.

El estándar ISO 9001 no es en sí mismo un estándar para el desarrollo de software, sino un marco para elaborar estándares de software. Establece principios de calidad total, describe en general el proceso de calidad, y explica los estándares y procedimientos organizacionales que deben determinarse. Éstos tienen que documentarse en un manual de calidad de la organización.

7. Métricas

La medición del software se ocupa de derivar un valor numérico o perfil para un atributo de un componente, sistema o proceso de software. Al comparar dichos valores unos con otros, y con los estándares que se aplican a través de una organización, es posible extraer conclusiones sobre la calidad del software, o valorar la efectividad de los procesos, las herramientas y los métodos de software.

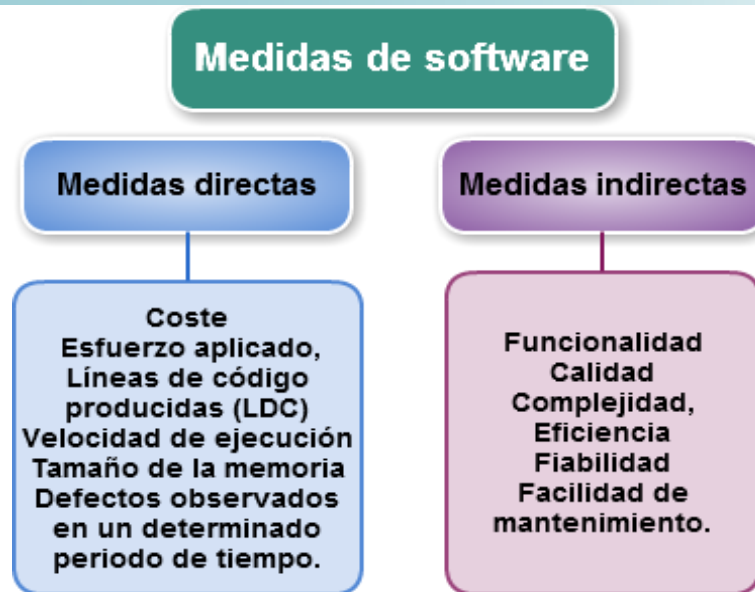
Por ejemplo, suponga que una organización pretende introducir una nueva herramienta de prueba de software. Antes de introducir la herramienta, hay que registrar el número de defectos descubiertos de software en un tiempo determinado. Ésta es una línea de referencia para valorar la efectividad de la herramienta. Después de usar la herramienta durante algún tiempo, se repite este proceso. Si se descubren más defectos en el mismo lapso, después de introducida la herramienta, usted tal vez determine que ofrece apoyo útil para el proceso de validación del software.

La meta a largo plazo de la medición del software es usar la medición en lugar de revisiones para realizar juicios de la calidad del software. Al usar medición de software, un sistema podría valorarse preferentemente mediante un rango de métricas y, a partir de dichas mediciones, se podría inferir un valor de calidad del sistema. Si el software alcanzó un umbral de calidad requerido, entonces podría aprobarse sin revisión. Cuando es adecuado, las herramientas de medición pueden destacar también áreas del software susceptibles de mejora. Sin embargo, aún se está lejos de esta situación ideal y no hay señales de que la valoración automatizada de calidad será en el futuro una realidad previsible.

Una métrica de software es una característica de un sistema de software, documentación de sistema o proceso de desarrollo que puede medirse de manera objetiva. Los ejemplos de métricas incluyen el tamaño de un producto en líneas de código, que es una medida de la legibilidad de un pasaje de texto escrito; el número de fallas reportadas en un producto de software entregado, y el número de días/hombre requerido para desarrollar un componente de sistema.

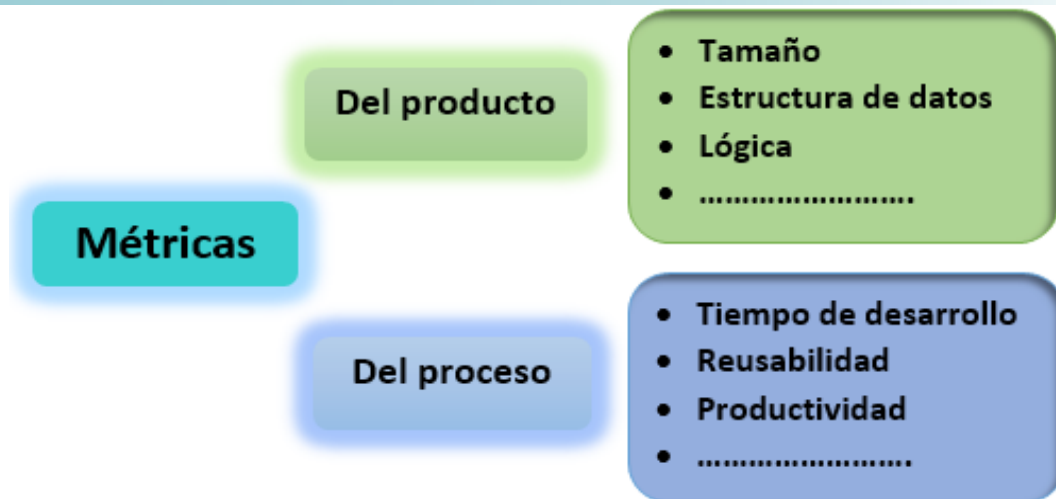
Las métricas de software se clasifican en dos tipos: medidas directas y medidas indirectas.

Las medidas directas del proceso de ingeniería de software son el coste, el esfuerzo aplicado, las líneas de código producidas (LDC), la velocidad de ejecución, el tamaño de la memoria y los defectos observados en un determinado periodo de tiempo. Entre las medidas indirectas se encuentran: la funcionalidad, calidad, complejidad, eficiencia, fiabilidad, facilidad de mantenimiento.

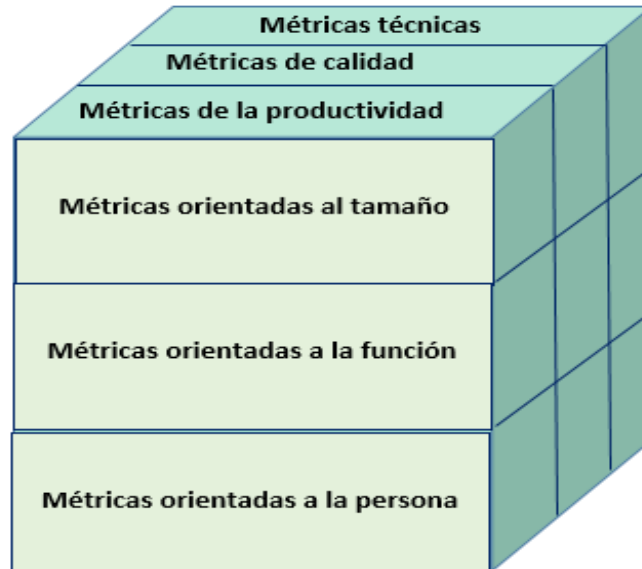


Las medidas directas tales como el coste o las líneas de código producidas son relativamente fáciles de obtener. Sin embargo la calidad, complejidad, eficiencia o fiabilidad son más difíciles de evaluar.

Otra posible clasificación de las métricas para valorar el software aparece en la siguiente figura y establece dos categorías principales: métricas del producto y métricas del proceso. El primer tipo se obtiene automáticamente tomando como entrada el código fuente: tamaño, estructuras de datos y lógica. Las métricas del proceso dependen del entorno de desarrollo: dificultad del problema, metodología empleada, capacidad del personal, etc.



Según Pressmann, el campo de las métricas de software también se puede clasificar de la siguiente manera. Las métricas de productividad se relacionan con el rendimiento del proceso de la ingeniería del software. Las métricas de calidad muestran cómo el software se ajusta a los requisitos del usuario. Las métricas técnicas se centran en las características del software. También se puede establecer las métricas orientadas al tamaño (se utilizan para obtener medidas directas del resultado de la calidad de la ingeniería del software), las métricas orientadas a la función (medidas indirectas) y las métricas orientadas a la persona (información sobre la forma en que la gente desarrollan el software).



Métricas orientadas al tamaño

Estas métricas proporcionan las medidas directas del software y consideran el “tamaño” del software que se ha producido. Por ejemplo, si una organización mantiene registros sencillos se puede crear una tabla:

Proyecto	LDC	Esfuerzo	Costo \$	Páginas de doc.	Errores	Personas
ABC-1	20,500	24	20.000	975	134	4

LDC - líneas de código

Esta tabla se refiere al proyecto ABC-1 en el cual se desarrollaron 20,500 líneas de código con un esfuerzo de 24 personas-mes y un coste de \$20,000. El esfuerzo y coste registrados en la tabla se refieren a todas las actividades de ingeniería del software y no solo a la codificación. Otra información que proporciona la tabla es que se desarrollaron

975 páginas de documentación, mientras se encontraron 134 errores. En el desarrollo del proyecto trabajaron 4 personas.

Tomando en cuenta estos datos se puede obtener otras métricas para comparar con otros proyectos como por ejemplo:

- Errores por KLDC (miles de líneas de código)
- Páginas de documentación por KLDC
- Errores por persona-mes
- LDC por persona-mes
- Costo por página de documentación

A partir de estos datos se puede determinar:

Productividad = KLDC / persona-mes

Calidad = errores / KLDC

Coste = dólares / KLDC

Documentación = páginas de documentación / KLDC


El método orientado al tamaño es bastante polémico por el uso de las líneas de código como medida principal. Esta medida es dependiente del lenguaje de programación lo que perjudica programas más cortos pero bien diseñados y que su utilización en la estimación requiere un nivel de detalle que puede ser difícil de conseguir.

Métricas orientadas a la función

Métricas orientadas a la función son medida indirectas y permiten medir la funcionalidad de un sistema. Estas métricas se basan en la funcionalidad o utilidad del software. La

métrica fue propuesta por Allan Albrecht de IBM quien sugirió el método de puntos de función. Los puntos de función se obtienen relacionando las medidas cuantitativas del dominio de la información del software con las valoraciones subjetivas de su complejidad.

Los puntos de función se calculan relleno la siguiente tabla:

Parámetro de medida	Cuenta	Factor de peso			
		Simple	Medio	Complejo	
Número de entradas de usuario		X 3	4	6 =	
Número de salidas de usuario		X 4	5	7 =	
Número de peticiones al usuario		X 3	4	6 =	
Número de archivos		X 7	10	15 =	
Número de interfaces externos		X 5	7	10 =	
Cuenta total 					

Se establecen 5 características del ámbito de la información:

- **Número de entradas de usuario:** se cuenta cada entrada de usuario que proporcione al software diferentes datos orientados a la aplicación.
- **Número de salidas de usuario:** se cuenta cada salida que proporciona al usuario información orientada a la aplicación. En este contexto las salidas se refieren a informes, pantallas, mensajes de error.

- **Número de peticiones de usuario:** cada petición es una entrada interactiva que resulta de la generación de algún tipo de respuesta en forma de salida interactiva. Se cuenta cada petición por separado.
- **Número de archivos:** se cuenta cada archivo maestro lógico.
- **Número de interfaces externas:** se cuentan todas las interfaces legibles por la máquina por ejemplo: archivos de datos, en cinta o discos que son utilizados para transmitir información a otro sistema.

Al recolectar todos los datos se asocia el valor de complejidad a cada cuenta y determina si una entrada es simple, media o compleja. Esta determinación es completamente subjetiva.

Para calcular los puntos de función se usa la fórmula:

$$PF = \text{cuenta-total} \times [0.65 + 0.01 \times \text{SUM} (F_i)]$$

Donde cuenta-total es la suma de todas las entradas PF obtenidas de la tabla. F_i ($i=1$ hasta 14) son los valores de ajuste de complejidad. Los valores constantes de la ecuación y los factores de peso fueron obtenidos empíricamente.

A partir de los puntos de función se puede obtener:

$$\text{Productividad} = PF / \text{personas-mes}$$

$$\text{Calidad} = \text{errores} / PF$$

$$\text{Coste} = \text{dólares} / PF$$

$$\text{Documentación} = \text{páginas de documentación} / PF$$

La métrica de puntos de función es independiente del lenguaje pero está basada en los cálculos muy subjetivos.

8. Gestión de configuración de software

El proceso de GCS da respuesta a las siguientes interrogantes:

- ¿De qué manera se puede identificar y gestionar varias versiones existentes de un software (y su documentación) de forma que se puedan introducir cambios eficientemente?
- ¿Cómo se puede controlar los cambios antes y después de que el software sea entregado al cliente?
- ¿Quién tiene la responsabilidad de aprobar y de asignar prioridades a los cambios?
- ¿Cómo podemos asegurar que los cambios se han llevado a cabo correctamente?
- ¿Qué mecanismos se usan para avisar a otros de los cambios realizados?

Estas interrogantes se resuelven en las cinco tareas de GCS:

Identificación - se establecen los estándares de documentación y un esquema de identificación de documentos.

Control de versiones - consiste en gestionar las versiones de los elementos de configuración durante el proceso de ingeniería del software.

Control de cambios - consiste en la evaluación y registro de todos los cambios que se hagan de la configuración software.

Auditorías de configuración - garantizan que el cambio se ha implementado correctamente.

Generación de informes - permite eliminar los problemas de comunicación entre las personas involucradas.

Identificación de objetos de GCS

La identificación de objetos de GCS permite definir una estructura de documentación organizada de un modo inteligible y predecible, o sea dar un formato. También proporciona los métodos para revisiones y añade los cambios conforme se producen y relaciona los cambios con “quién, qué, cuándo, por qué, cómo” para facilitar el control.

El proceso de identificación de la configuración inicia con la definición de los elementos de la configuración software representativos de los productos en cada línea base establecida. Se definen el formato, los contenidos y los mecanismos de control para toda la documentación para enlazar la información cuando la jerarquía de la configuración se despliega. Luego se asignan identificadores apropiados a todos los programas, documentos y periféricos, usando un esquema numerado que proporciona información sobre el elemento de la configuración software. Finalmente, la identificación debe facilitar el control de cambios, para acomodar actualizaciones y modificaciones.

Control de versiones

El control de versiones es el seguimiento de las versiones en que se encuentra cada elemento que da la posibilidad de observar el historial de modificaciones que se realizó sobre los mismos.

El control de versiones consiste en los procedimientos y herramientas para gestionar los cambios en los elementos creados durante el ciclo de vida del software. Se inicia por la asignación de un número de versión para la configuración de la línea de base y se lleva un registro de las revisiones corrientes e históricas de cada elemento y del proyecto en su conjunto. Comúnmente las revisiones de cada objeto se relacionan formando un grafo de dependencias representando la evolución del mismo. Todo esto debe organizarse de manera controlada, estableciendo procedimientos y permisos de acceso para realización de las modificaciones.

En la siguiente figura se muestra el ejemplo de un grafo de evolución de revisión del software.



Para controlar los cambios que se realizan en cada versión se usan los sistemas de control de versiones. El sistema de control de versiones registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante. Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas. Un sistema de control de versiones debe proporcionar la posibilidad de almacenamiento de los elementos que deba gestionar (archivos de texto, imágenes, etc.), debe realizar los cambios sobre los elementos almacenados y debe contener el registro histórico de las acciones realizadas sobre cada elemento.

Algunos ejemplos de estos sistemas son: CVS, GIT y subversiones.

CVS (Concurrent Version System) es una aplicación de software libre que mantiene el registro de todo el trabajo y los cambios en los códigos fuente y permite que distintos desarrolladores de software colaboren.

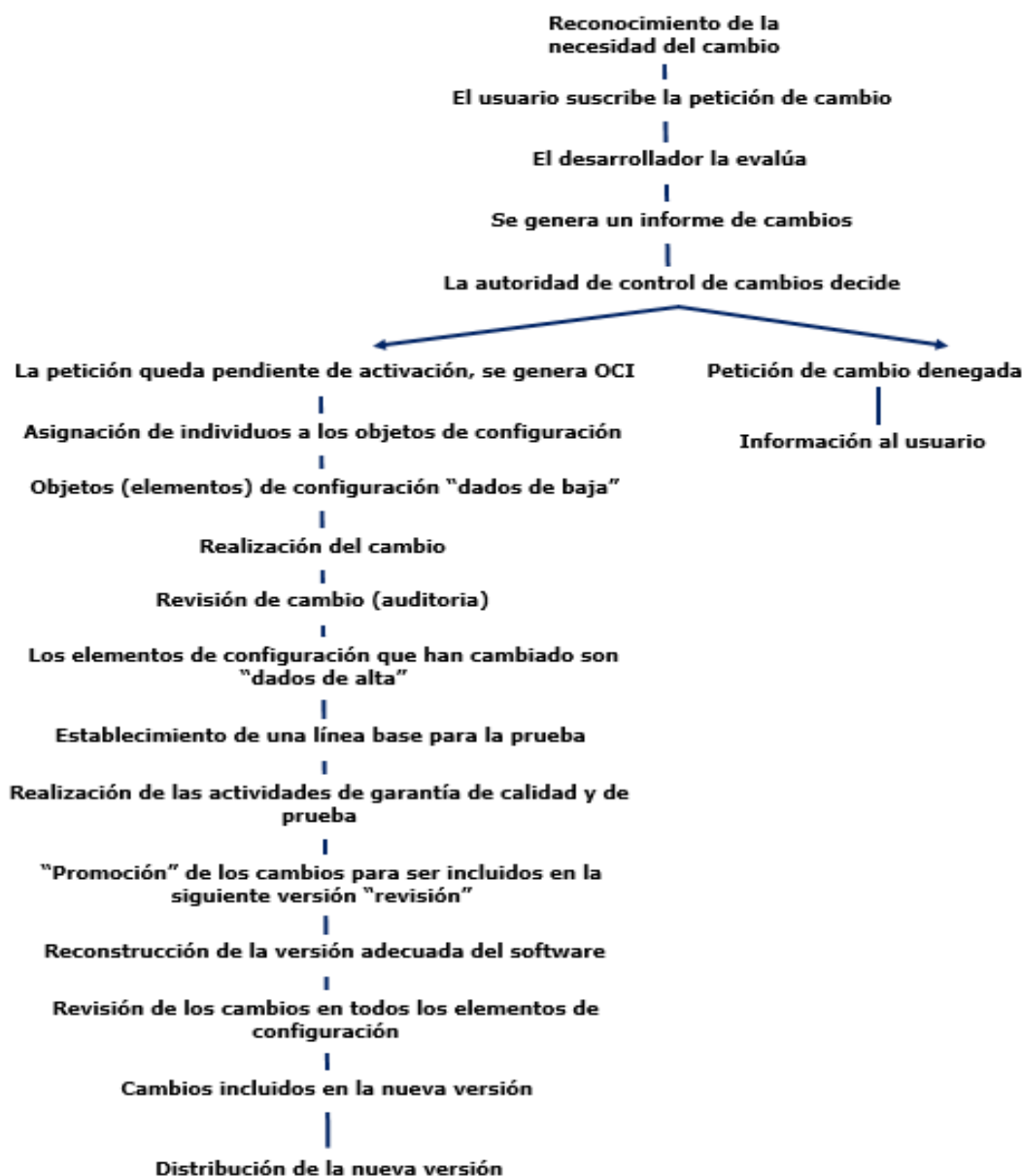
Subversion (SVN) es otra aplicación de software libre para el control de versiones diseñado especialmente para reemplazar al CVS. Una característica importante de Subversion es que todo el repositorio de las versiones tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en un instante determinado del repositorio que se está trabajando.

Git es un software libre de control de versiones diseñado por Linus Torvalds. Se ha diseñado haciendo énfasis en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.

Control de cambios

La actividad de control de cambios tiene como objetivo el seguimiento del ciclo de vida de un cambio, que comprende: la solicitud del cambio, la evaluación, la aprobación o rechazo del mismo y la implementación.

El proceso de control de cambio se muestra en la siguiente figura:



Se realiza una petición de cambio y se evalúa para calcular el esfuerzo técnico, los efectos secundarios, el impacto global, coste y otras funciones. Como resultado de evaluación se obtiene un informe de cambios y se presenta a la autoridad de control de cambios (una persona o grupo de personas que toman decisión final sobre el cambio).

Para cada cambio que se apruebe se genera una orden de cambio de ingeniería OCI que detalla el cambio que se va realizar. Luego se implementa el cambio y se usan los mecanismos de control de versiones apropiados para crear la siguiente versión del software.

El proceso de control de cambio se aplica tanto al proceso de desarrollo del software como al proceso de mantenimiento con el objetivo que los cambios son controlados por un método que garantiza el desempeño de los procesos.

9. Caso práctico planificación con historias de usuario

El cliente, el señor X, actualmente tiene una tienda de historietas en un local en la zona 1, él decide que es tiempo de tener su página web que le permita estar en contacto con sus clientes. Hay una convención dentro de 2 meses y él quiere presentar su página web a sus clientes durante el evento.

1. Recolectando Información

Como se puede notar el señor X quiere aventurarse en el mundo web, pero es su primera página web por lo que en realidad no tiene idea de lo que debería ser su página. Nuestro trabajo como desarrolladores es tener claro lo que él quiere y si él no tiene la menor idea, tenemos herramientas que nos ayudarán a llevar a cabo la tarea.

Como primera opción optamos por reunirnos con él y hablar de lo que su página web debe ser y hacer. Así que recurrimos a las siguientes preguntas y sus respectivas respuestas.

¿Por qué quiere tener una página web para su tienda de comics?	Porque quiero tener un lugar en donde mostrar mis productos e información sobre mi negocio.
¿Qué tipo de servicios quiere ofrecer a través de ella?	Pues la verdad no lo había pensado, solo tuve la idea de hacer una.
Había mencionado que quiere mantenerse en contacto con sus clientes. ¿Qué tipo de contacto espera?	Pues me gustaría que me pudieran enviar mensajes a mi correo a través de la página y yo les pueda responder desde mi correo.

Lamentablemente el señor X no sabe muy bien lo que quiere, pero a pesar de eso obtuvimos dos historias de usuario útiles.

Historia:	Como cliente puedo ver los productos de la tienda, en línea
Descripción:	Los clientes pueden ver los productos con los que dispone la tienda. Tiempo: 18 días

Título:	Como cliente puede enviar contactar a la tienda enviando un correo electrónico
Descripción:	Los clientes pueden contactar al correo del señor X a través de la página. Tiempo: 10 días

Decidimos reunirnos con el equipo de trabajo del señor X, el cual incluye a sus vendedores y mensajeros, y les pedimos que nos dijeran que cosas les gustaría que la página haga para ayudarles con su trabajo, así mismo hicimos algunas sugerencias al cliente.

Luego de una sesión de lluvia de ideas obtuvimos las siguientes historias de usuario. Los vendedores argumentan que a los clientes les cuesta llegar a la tienda por varias circunstancias por lo que creen que vender en línea es una buena opción, y sugieren hacer el cobro contra entrega.

Título:	Como cliente, puedo comprar los productos de la tienda en línea
Descripción:	Los clientes deben poder comprar productos de la tienda a través de la página. Tiempo: 20 días

Los mensajeros dicen que pueden entregar los productos vendidos pero necesitan tener un listado de entregas para organizarse.

Título:	Como mensajero, puedo ver la lista de entregas diarias
Descripción:	La página debe alimentar una lista de entregas para que los mensajeros lleven los productos a los clientes. Tiempo: 10 días

El señor X sabe que algunos clientes prefieren no pagar con efectivo por varias razones, por lo que pide que se permita que puedan cancelar con su tarjeta de crédito también, pero no quiere que este trabajo lo realicen los mensajeros por lo que pide que lo hagan desde la página.

Título:	Como cliente, puedo pagar mis compras en línea con tarjeta de crédito
Descripción:	Los clientes pueden realizar el pago con su tarjeta de crédito a través de la página. Tiempo: 20 días

El proyecto va tomando forma, y estando en la oficina del señor X me muestra un correo y me cuenta que le gusta escribirle a sus clientes para mantener contacto con ellos, generalmente son historias sobre comics y cosas así, y a veces son ofertas de la tienda, a mí me parece mejor tener un lugar donde publicar que enviar correos masivos por lo que le sugiero un blog para mantener este contacto de una mejor manera, lo cual le parece muy buena idea.

Título:	Como administrador, puedo crear publicaciones en el blog de la página web
Descripción:	El señor X quiere poder crear entradas en el blog de la tienda para compartir información y publicar ofertas. Tiempo: 10 días

El señor X quiere saber en cuánto tiempo le entregaremos su página web, con nuestras historias de usuario obtuvimos que sumando los tiempos estos nos dan el total de 88 días!

Etapas de planeación

El cliente quiere su software en el momento en que lo necesita, esto a veces trae dificultades pues nuestras estimaciones pueden sobrepasar por mucho el tiempo deseado, ¿qué hacer en este caso?

1. Escoger qué historias de usuario vamos a desarrollar

El primer paso es seleccionar lo que nos parezca prioritario y que esté dentro del rango de tiempo dado por el cliente.

2. Consultar con el cliente las historias de usuario seleccionadas

A pesar de que nosotros escogemos lo que creemos es prioritario en el software el cliente tiene la última palabra sobre lo que él necesita y lo que puede esperar. Como mencionamos antes, sería un error tomar nosotros la decisión y entregar algo que él no quiere.

3. Priorizar con el cliente

Muy probablemente los dos tendrán puntos de vista diferentes sobre lo que es necesario, hablar es la solución en éste caso, cada uno debe exponer sus puntos de vista y al final llegar a la conclusión de lo que el software va a hacer y lo que puede esperar.

Pero a veces re-factorizar los requerimientos no es suficiente, por lo que como desarrollador a veces es necesario en pensar en el equipo de trabajo, contratar más personas para el proyecto podría ayudar a reducir el tiempo, pero esto no pasa necesariamente, en realidad hay que evaluar para qué, si se tienen que añadir personas que este sea el número correcto y que tengan la preparación necesaria.

Cuando no se estudia la optimización del recurso humano, o solo se agregan más y más personas al proyecto esto en vez de favorecer disminuye la productividad del grupo.

Después de evaluar y quizás contratar más gente para el proyecto, se debe realizar una nueva estimación del tiempo. A veces incluso añadiendo gente el tiempo solicitado por el cliente puede no ser suficiente, entonces se procede a evaluar nuevamente los requerimientos para ver cuales pueden quedar en segundo plano.

Al comenzar a trabajar en el proyecto recurrimos a la iteración para mantener al corriente al cliente y no perder la comunicación sobre el proyecto.

Expectativa vs realidad:

Otro factor importante a tomar en cuenta es que aunque tengamos un plan, hay situaciones fuera de nuestro control que pueden alterar el proceso del proyecto cosas como fallas en el equipo, días de descanso, enfermedades, etc.

El desarrollador de software debe planear, no solo sentarse a escribir código, de esa manera él cuenta con la ventaja de tomar en cuenta todos los aspectos del proyecto no solo la parte de codificar.

Planificación del proyecto

Con anterioridad obtuvimos las historias de usuario para la página web del señor X y estimamos la flamante cantidad de 85 días de desarrollo, el problema es que el señor X quiere su página web dentro de 2 meses, al menos 60 días, estamos sobrepasando el tiempo 25 días después de la fecha de entrega.

Nosotros seleccionamos las siguientes historias de usuario como importantes:

- Mostrar productos
- Comprar en línea
- Permitir pagar con tarjeta de crédito
- Crear una lista de entregas

Las cuales hacen un tiempo estimado total de 68 días.

Se la mostramos al señor X, pero él opina diferente, dice que prefiere poder comunicarse con sus clientes a través del blog y que la lista de entregas se maneje como lo han hecho hasta ahora (el cliente manda), de igual manera el total es 68 días.

Luego de acordar con el señor X que estas historias de usuario serían las que se desarrollarían, le asignamos una prioridad a cada una de ellas, siendo 1 la prioridad más alta y 10 la más baja.

- | | |
|---|-------------|
| - Mostrar productos | Prioridad 1 |
| - Comprar en línea | Prioridad 2 |
| - Permitir pagar con tarjeta de crédito | Prioridad 5 |

- Crear blog

Prioridad 8

Ya tenemos nuestros requerimientos listos y el orden en el que se van a desarrollar, pero de igual manera nos estamos pasando del tiempo de entrega límite, y eso que no estamos tomando en cuenta las eventualidades que sabemos que se nos van a presentar.

Luego de hablar con el equipo de trabajo decidimos contratar a otro desarrollador, pues el tiempo no excede demasiado a la entrega del proyecto y una persona nueva no afectaría demasiado en la productividad del equipo completo.

Luego decidimos que las iteraciones la haríamos cada 20 días como máximo, pues ese es el tiempo estimado para algunas historias de usuario y nos permitiría iterar al menos 3 veces con el cliente.

De tal manera que nuestras iteraciones están programadas de la siguiente manera:

Haciendo cuentas determinamos que el desarrollador extra nos ayuda a reducir el tiempo estimado de las historias de usuario en una sexta parte del tiempo, aproximadamente. Esto nos ayuda a mantener las iteraciones por debajo del intervalo máximo de 20 días que fijamos.

Iteración 1:

- Mostrar productos - Tiempo estimado: 15 días

Iteración 2:

- Comprar en línea - Tiempo estimado: 17 días

Iteración 3:

- Permitir pagar con tarjeta de crédito - Tiempo estimado: 17 días
- Crear blog - Tiempo estimado: 8 días

Las iteraciones con las nuevas estimaciones nos dan la cantidad de 57 días, parece que si vamos a cumplir con el proyecto, incluso si tomamos en cuenta algunos posibles imprevistos.

Control del proyecto

Ya tenemos nuestras historias de usuario, pero como su nombre lo indica son para usuarios, nos ayudaron a describir lo que el software necesita hacer exactamente desde la perspectiva del cliente. Pero ahora que es momento de comenzar a trabajar en codificar hay que verlas de manera diferente.

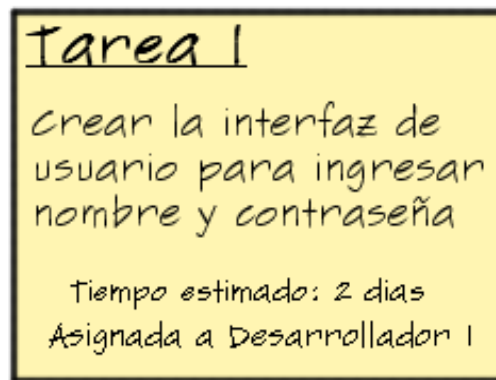
Cada historia de usuario es un conjunto de pequeñas tareas que se deben completar para llegar a lo que la historia de usuario requiere, cada tarea específica una parte de trabajo de desarrollo que debe ser realizada por un desarrollador, cada tarea tiene una descripción aproximada que contiene detalles de cómo se debe hacer el desarrollo de esa tarea y tiene una estimación.

Las tareas nos dan mejores estimaciones que las historias de usuario, por ello las mismas se deben crear lo antes posible, teniendo las tareas definidas podemos deshacernos de la descripción de la historia de usuario ya que la misma no nos sirve a los desarrolladores, las tareas sí.

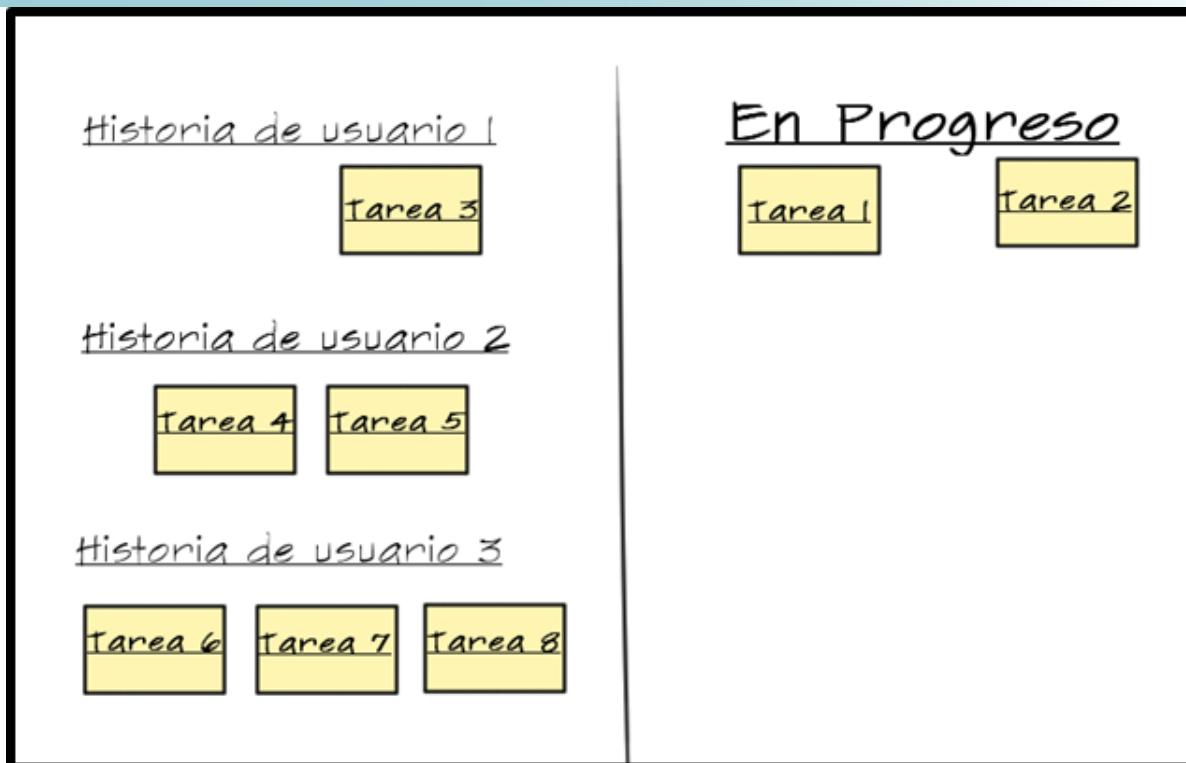
Una buena forma de manejar las tareas es a través de una pizarra y algunas notas en ella (sticky notes o post it como se conocen), no hablamos de una pizarra imaginaria, sino de una real, pues es importante que todo el equipo tome su tiempo para ver el progreso en la pizarra, a través de la misma vemos reflejada la realidad del proyecto.

Cada tarea debe tener asignado un desarrollador, en la pizarra debe haber espacio para las tareas que se están llevando a cabo, que una tarea tenga un desarrollador asignado no quiere decir que ya se comenzó a hacer.

Ejemplo de tarea:



Ejemplo de pizarra:

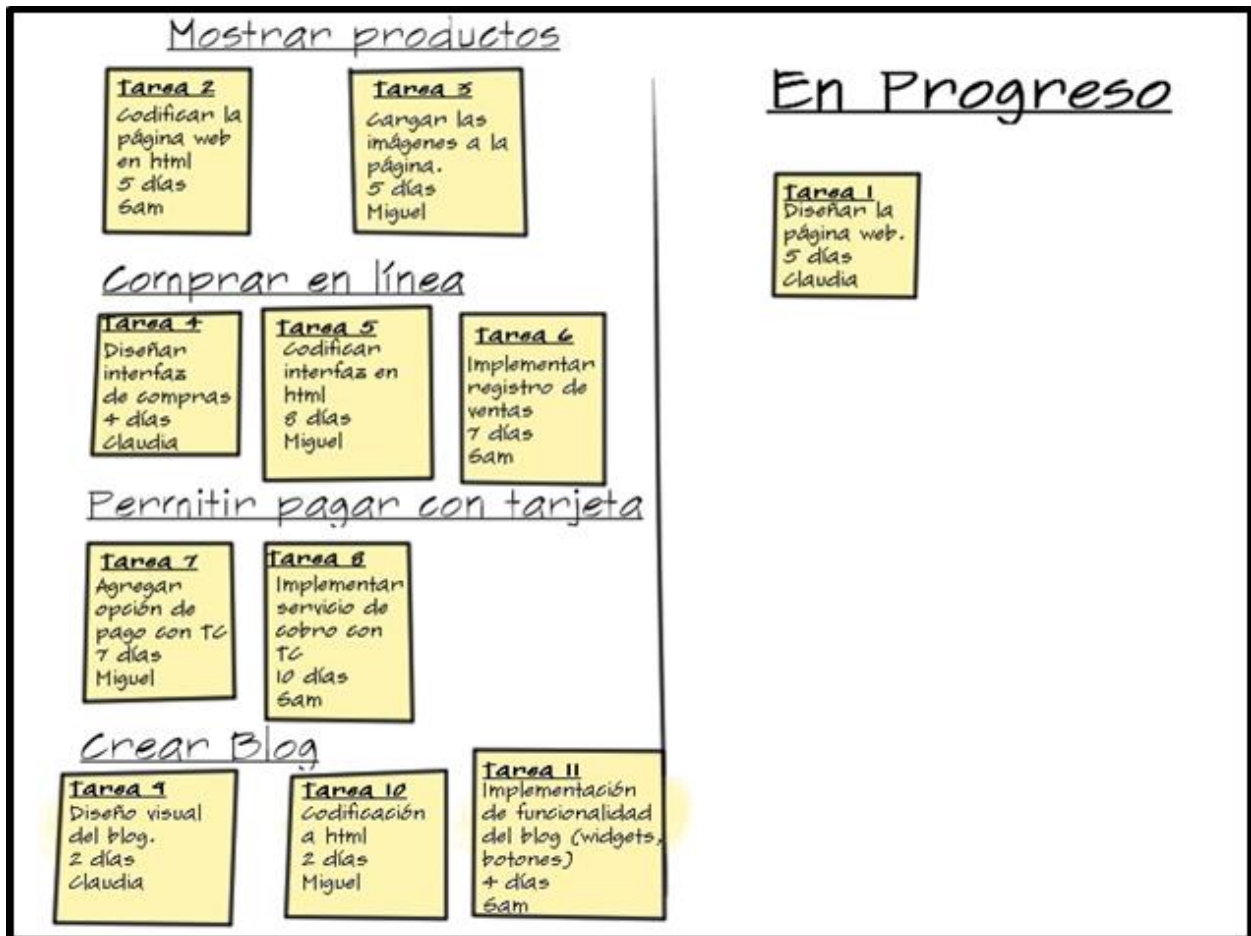


Para nuestro ejemplo práctico dividimos cada historia de usuario en tareas para asignarlas a distintos desarrolladores, las estimaciones de las tareas fueron hechas de manera que al sumarlas diera el tiempo total de la historia de usuario, hay muchos factores a tomar en cuenta para hacer una buena estimación. Para propósitos de este curso escriba las estimaciones conforme a su propio criterio (por ejemplo las tareas más complicadas mayor tiempo o por el tipo de tarea).

Contamos con 3 desarrolladores

- Miguel – El nuevo desarrollador
- Sam – El experimentado
- Claudia – La diseñadora web

Nuestra pizarra queda así:



Note que algunas tareas se pueden hacer aún más específicas, dependiendo de la cantidad de desarrolladores.

10. Referencias

- Somerville, 2010. Software engineering. 9ª Edición. Pearson.
- Pressman, 1993. Ingeniería del Software: Un Enfoque Práctico. 3ª Edición. McGraw-Hill, 1993. –Capítulo 1 y 2

- IEEE Computer Society, 2004. Guide to the Software Engineering Body of Knowledge (SWEBOK), 2004 versión. Disponible en <http://www.swebok.org>

Descargo de responsabilidad

La información contenida en este documento descargable en formato PDF o PPT es un reflejo del material virtual presentado en la versión online del curso. Por lo tanto, su contenido, gráficos, links de consulta, acotaciones y comentarios son responsabilidad exclusiva de su(s) respectivo(s) autor(es) por lo que su contenido no compromete al área de e-Learning del Departamento GES o al programa académico al que pertenece.

El área de e-Learning no asume ninguna responsabilidad por la actualidad, exactitud, obligaciones de derechos de autor, integridad o calidad de los contenidos proporcionados y se aclara que la utilización de este descargable se encuentra limitada de manera expresa para los propósitos educativos del curso.



Galileo
UNIVERSIDAD
La Revolución en la Educación

GES
Galileo Educational System