Tecnólogo en Informática – San José
Ingeniería de Software

ISO 9126 – Calidad de software

Estándar internacional para la evaluación de la Calidad del Software.

Está dividido en cuatro partes:

1. Modelo de calidad
2. Métricas externas
3. Métricas internas
4. Calidad en las métricas de uso.

El modelo de calidad establecido en la primera parte del estándar ISO 9126-1, clasifica la calidad del software en un conjunto estructurado de características y subcaracterísticas de la siguiente manera:

- **Funcionalidad**- Un conjunto de atributos que se relacionan con la existencia de un conjunto de funciones y sus propiedades específicas. Las funciones son aquellas que satisfacen las necesidades implícitas o explícitas.
    1. Adecuación - Atributos del software relacionados con la presencia y aptitud de un conjunto de funciones para tareas especificadas.
    2. Exactitud - Atributos del software relacionados con la disposición de resultados o efectos correctos o acordados.
    3. Interoperabilidad - Atributos del software que se relacionan con su habilidad para la interacción con sistemas especificados.
    4. Seguridad - Atributos del software relacionados con su habilidad para prevenir acceso no autorizado ya sea accidental o deliberado, a programas y datos.
    5. Cumplimiento funcional.

- **Fiabilidad**- Un conjunto de atributos relacionados con la capacidad del software de mantener su nivel de prestación bajo condiciones establecidas durante un período establecido.
    1. Madurez - Atributos del software que se relacionan con la frecuencia de falla por fallas en el software.
    2. Recuperabilidad - Atributos del software que se relacionan con la capacidad para restablecer su nivel de desempeño y recuperar los datos directamente afectos en caso de falla y en el tiempo y esfuerzo relacionado para ello.
    3. Tolerancia a fallos - Atributos del software que se relacionan con su habilidad para mantener un nivel especificado de desempeño en casos de fallas de software o de una infracción a su interfaz especificada.
    4. Cumplimiento de Fiabilidad - La capacidad del producto software para adherirse a normas, convenciones o legislación relacionadas con la fiabilidad.

- **Usabilidad**- Un conjunto de atributos relacionados con el esfuerzo necesario para su uso, y en la valoración individual de tal uso, por un establecido o implicado conjunto de usuarios.
    1. Aprendizaje- Atributos del software que se relacionan al esfuerzo de los usuarios para reconocer el concepto lógico y sus aplicaciones.
    2. Comprensión - Atributos del software que se relacionan al esfuerzo de los usuarios para reconocer el concepto lógico y sus aplicaciones.

3. Operatividad - Atributos del software que se relacionan con el esfuerzo de los usuario para la operación y control del software.
4. Atractividad

- **Eficiencia**- Conjunto de atributos relacionados con la relación entre el nivel de desempeño del software y la cantidad de recursos necesitados bajo condiciones establecidas.
  1. Comportamiento en el tiempo - Atributos del software que se relacionan con los tiempos de respuesta y procesamiento y en las tasas de rendimientos en desempeñar su función.
  2. Comportamiento de recursos - Usar las cantidades y tipos de recursos adecuados cuando el software lleva a cabo su función bajo condiciones determinadas.

- **Mantenibilidad**- Conjunto de atributos relacionados con la facilidad de extender, modificar o corregir errores en un sistema software.
  1. Estabilidad - Atributos del software relacionados con el riesgo de efectos inesperados por modificaciones.
  2. Facilidad de análisis - Atributos del software relacionados con el esfuerzo necesario para el diagnóstico de deficiencias o causas de fallos, o identificaciones de partes a modificar.
  3. Facilidad de cambio - Atributos del software relacionados con el esfuerzo necesario para la modificación, corrección de falla, o cambio de ambiente.
  4. Facilidad de pruebas - Atributos del software relacionados con el esfuerzo necesario para validar el software modificado.

- **Portabilidad**- Conjunto de atributos relacionados con la capacidad de un sistema software para ser transferido desde una plataforma a otra.
  1. Capacidad de instalación - Atributos del software relacionados con el esfuerzo necesario para instalar el software en un ambiente especificado.
  2. Capacidad de reemplazamiento - Atributos del software relacionados con la oportunidad y esfuerzo de usar el software en lugar de otro software especificado en el ambiente de dicho software especificado.
  3. Adaptabilidad - Atributos del software relacionados con la oportunidad para su adaptación a diferentes ambientes especificados sin aplicar otras acciones o medios que los proporcionados para este propósito por el software considerado.
  4. Co-Existencia - Coexistir con otro software independiente, en un entorno común, compartiendo recursos comunes.

La subcaracterística Conformidad no está listada arriba ya que se aplica a todas las características. Ejemplos son conformidad a la legislación referente a usabilidad y fiabilidad.

Cada subcaracterística (como adaptabilidad) está dividida en atributos. Un atributo es una entidad la cual puede ser verificada o medida en el producto software. Los atributos no están definidos en el estándar, ya que varían entre diferentes productos software.

Métricas internas son aquellas que no dependen de la ejecución del software (medidas estáticas).

Métricas externas son aquellas aplicables al software en ejecución.

La calidad en las métricas de uso están sólo disponibles cuando el producto final es usado en condiciones reales.

Idealmente, la calidad interna no necesariamente implica calidad externa y esta a su vez la calidad en el uso.

Calidad en Uso:

- **Efectividad**: Capacidad de un producto de software para permitir a los usuarios alcanzar objetivos especificados con exactitud y completitud, en un contexto de uso especifico.
- **Productividad**: Capacidad de un producto de software para permitir a los usuarios gastar una cantidad adecuada de recursos con relación a la efectividad alcanzada, en un contexto de uso específico.
- **Seguridad**: Capacidad de un producto de software para alcanzar niveles aceptables de riesgo de hacer daño a personas, al negocio, al software, a las propiedades o al medio ambiente, en un contexto de uso específico.
- **Satisfacción**: Capacidad de un producto de software para satisfacer a los usuarios en un contexto de uso específico.

Cualidades del software

Correctitud (Correctness):

Un programa es funcionalmente correcto si se comporta de acuerdo a la especificación de las funciones (especificación de requerimientos funcionales) que debería proveer. Esta definición de correctitud asume que existe una especificación de requerimientos funcionales del sistema y que es posible determinar en forma no ambigua si las cumple o no. Se presentan diversas dificultades cuando no existe dicha especificación, o si existe pero está escrita informalmente utilizando, por ejemplo, lenguaje natural por lo que es posible que contenga ambiguedades.

La correctitud es una propiedad matemática que establece la equivalencia entre el software y su especificación, por lo que cuanto más riguroso se haya sido en la especificación, más precisa y sistemática podrá ser su evaluación.

Posteriormente se verá que la correctitud puede ser evaluada mediante diversos métodos, algunos de enfoque experimental como las pruebas, otros de enfoque analítico como verificación formal de la correctitud.

Es de notar que esta definición de correctitud no toma en consideración el que la especificación en sí misma puede ser incorrecta por contener inconsistencias internas, o no corresponder de forma adecuada a las necesidades para las que fue concebido el programa.

Confiabilidad (Reliability):

Informalmente el software es confiable si el usuario puede tenerle confianza. Formalmente la confiabilidad se define en términos del comportamiento estadístico: la probabilidad de que el software opere como es esperado en un intervalo de tiempo especificado. Contrariamente a la correctitud que es una cualidad absoluta, la confiabilidad es relativa. Cualquier desviación de los requerimientos hace que el sistema sea incorrecto, por otro lado, si la consecuencia de un error en el software no es seria, el software incorrecto aún puede ser confiable.

En el caso ideal en el que la especificación de requerimientos funcionales captura todas las propiedades deseables de la aplicación y no hay propiedades indeseables erróneamente especificadas, el conjunto de todos los programas confiables incluye el conjunto de programas correctos, pero no a la inversa. En la práctica, como la especificación es un modelo de lo que quiere el usuario que puede ser o no adecuado para sus necesidades y requerimientos reales, lo máximo que puede hacer el software es cumplir los requerimientos especificados del

modelo, sin asegurar la adecuación del mismo. Pueden tenerse aplicaciones correctas diseñadas para requerimientos “incorrectos”, por lo que la correctitud del software puede no ser suficiente para garantizar al usuario que el software se comporta como “es esperado”.

Los productos de la ingeniería son confiables, pero desafortunadamente los productos del software son en general liberados conjuntamente con una lista de “bugs” conocidos. Este es uno de los síntomas de la inmadurez de la Ingeniería de Software como disciplina ingenieril y sólo podrá alcanzar ese estatus cuando se logre que la confiabilidad en el software sea comparable a la confiabilidad en otros productos como por ejemplo los automóviles.

Robustez (Robustness):

Un progama es robusto si se comporta en forma razonable aún en circunstancias que no fueron anticipadas en la especificación de requerimientos; por ejemplo cuando encuentra datos de entrada incorrectos o algún malfuncionamiento del hardware como rotura de disco. Un programa que genere un error no recuperable en tiempo de ejecución tan pronto como el usuario ingrese inadvertidamente un comando incorrecto no será robusto, aunque podría ser correcto si en la especificación de requerimientos no se establece la acción a tomar si se ingresa un comando incorrecto.

La robustez es una cualidad difícil de definir, ya que si se pudiera establecer en forma precisa lo que se debiera hacer para obtener una aplicación robusta, se podría especificar completamente el comportamiento “razonable”, con lo cual sería equivalente a la correctitud, o a la confiabilidad en el caso ideal mencionado en la definición anterior.

Se puede observar que la robustez y la correctitud están fuertemente relacionadas: si se incluye un requerimiento en la especificación será un tema de correctitud, si no se incluye podría ser un tema de robustez. La línea divisoria entre ambos es la especificación del sistema. La relación con la confiabilidad surge del hecho de que no todos los comportamientos incorrectos significan problemas igualmente serios, algunos comportamientos incorrectos pueden ser tolerados.

Observación:

La correctitud, confiabilidad y robustez también se aplican al proceso de producción del software, por ejemplo un proceso es robusto si puede adaptarse a cambios no previstos en el entorno como ser una nueva liberación del sistema operativo o una transferencia repentina de la mitad de los empleados a otra sección, un proceso es confiable si lleva consistentemente a la producción de productos de alta calidad.

Performance (también Eficciency):

En la Ingeniería de Software generalmente performance equivale a eficiencia. Un sistema de software es eficiente si utiliza los recursos computacionales en forma económica. La performance de un sistema es importante porque afecta su usabilidad, por ejemplo, si es muy lento reduce la productividad de los usuarios, si usa demasiado espacio de disco puede ser muy caro de ejecutar, si utiliza demasiada memoria puede afectar al resto de las aplicaciones que se están ejecutando o ejecutarse demasiado lentamente mientras el sistema operativo intenta balancear el uso de la memoria por parte de las distintas aplicaciones. Detrás de estos problemas están los límites cambiantes de la eficiencia según cambia la tecnología, ya que la visión de “demasiado caro” cambia constantemente según los avances tecnológicos, actualmente una computadora cuesta bastante menos que hace unos años y son bastante más poderosas. La performance de un sistema también afecta su escalabilidad: un algoritmo de orden al cuadrado puede funcionar bien con entradas pequeñas y no funcionar para nada con entradas muy grandes.

Amigabilidad (Friendliness):

Un sistema de software es amigable si un usuario humano lo encuentra fácil de utilizar. Esta definición refleja la naturaleza subjetiva de la amigabilidad: una aplicación utilizada por usuarios no experientes califica como amigable por varias propiedades distintas a las de una aplicación utilizada por programadores expertos, por ejemplo, los primeros apreciarían el uso de menúes mientras los segundos se sentirían más cómodos ingresando comandos.

La interfaz de usuario es un componente importante de la amigabilidad al usuario, siguiendo el ejemplo, un sistema con una interfaz de ventana y un mouse es más amigable para un usuario no experiente, mientras que un usuario más avanzado podría preferir utilizar un conjunto de comandos. La amigabilidad es más que la interfaz de usuario, por ejemplo, un sistema embebido no tiene interfaz humana, ya que sólo interactúa con hardware y quizás con otros sistemas. En este caso la amigabilidad está dada por la facilidad con que el sistema puede configurarse y adaptarse al ambiente de hardware.

Las cualidades del software vistas previamente también afectan a la amigabilidad, por ejemplo un sistema que produce respuestas erróneas no es amigable sin importar lo “linda” que sea la interfaz de usuario, del mismo modo que un sistema que produce respuestas más lentas de lo que requiere el usuario no es amigable aunque estas respuestas sean desplegadas en colores.

Verificabilidad (Verifiability):

Un sistema de software es verificable si sus propiedades pueden ser verificadas fácilmente. Por ejemplo, la correctitud o la performance de un sistema son propiedades que interesa verificar. El diseño modular, prácticas de codificación disciplinadas, y la utilización de lenguajes de programación adecuados contribuyen a la verificabilidad de un sistema.
La verificabilidad es en general una cualidad interna pero a veces también puede ser externa, por ejemplo, en muchas aplicaciones de seguridad crítica, el cliente requiere la verificación de ciertas propiedades.

Mantenibilidad (Maintainability):

El término mantenimiento del software es utilizado generalmente para referirse a las modificaciones que se realizan a un sistema de software luego de su liberación inicial, siendo visto simplemente como “corrección de bugs”. Algunos estudios han mostrado sin embargo, que la mayor parte del tiempo utilizado en mantenimiento es para agregarle al producto características que no estaban en las especificaciones originales o estaban definidas incorrectamente.

En realidad la palabra “mantenimiento” no es apropiada para el software ya que cubre un amplio rango de actividades que tienen que ver con la modificación de un software existente para lograr una mejora, un término que se aplica mejor a este proceso es “evolución del software”. Además en otros productos de ingeniería como hardware de computadoras, automóviles o máquinas de lavar, el mantenimiento se refiere al costo de reparación del producto en respuesta al deterioro gradual de sus partes debido a su uso, sin embargo, se seguirá utilizando este término también para el software. Hay evidencia de que los costos de mantenimiento exceden el 60% del total de los costos del software. Para analizar los factores que afectan estos costos, es usual dividir el mantenimiento del software en tres categorías: correctivo, adaptativo y perfectivo.

El mantenimiento correctivo tiene que ver con la eliminación de errores residuales presentes en el producto al ser liberado así como errores introducidos al software durante su mantenimiento. Este tipo de mantenimiento corresponde aproximadamente al 20% de los costos de mantenimiento.

El mantenimiento adaptativo involucra el ajuste de la aplicación a cambios en el entorno, por

ejemplo una nueva liberación de hardware o del sistema operativo, o un nuevo sistema de bases de datos. En este caso la necesidad de cambios al software no puede ser atribuida a una característica del software en sí mismo como la inhabilidad de proporcionar determinada funcionalidad requerida por el usuario o errores residuales, sino que se originan debido a cambios en su entorno. Este tipo de mantenimiento corresponde aproximadamente a otro 20% de los costos de mantenimiento.

El mantenimiento perfectivo involucra cambios en el software para mejorar algunas de sus cualidades, los cuales se deben a la necesidad de modificar las funciones ofrecidas por la aplicación, agregar nuevas funcionalidades, mejorar la performance, facilitar su utilización, entre otras. Estos cambios pueden ser originados tanto por el ingeniero de software para mejorar el estatus del producto en el mercado, como por el cliente debido a nuevos requerimientos. Este tipo de mantenimiento corresponde a más del 50% de los costos de mantenimiento.

La mantenibilidad del software se verá a continuación como dos cualidades separadas: reparabilidad y evolucionabilidad.

Reparabilidad (Reparability):

Un sistema de software es reparable si permite la corrección de sus defectos con una carga limitada de trabajo. En otros campos de la ingeniería puede ser más barato cambiar un producto entero o una buena parte del mismo que repararlo, por ejemplo televisores, y una técnica muy utilizada para lograr reparabilidad es usar partes estándares que puedan ser cambiadas fácilmente. Sin embargo, en el software las partes no se deterioran, y aunque el uso de partes estándares puede reducir el costo de producción del software, el concepto de partes reemplazables pareciera no aplicar a la reparabilidad del software. Otra diferencia es que el costo del software está determinado, no por partes tangibles sino por actividades humanas de diseño.

Un producto de software consistente en módulos bien diseñados es más fácil de analizar y reparar que uno monolítico, sin embargo, el solo aumento del número de módulos no hace que un producto sea más reparable. Una modularización adecuada con definición adecuada de interfaces que reduzcan la necesidad de conexiones entre los módulos promueve la reparabilidad ya que permite que los errores estén ubicados en unos pocos módulos, facilitando la localización y eliminación de los mismos.

La reparabilidad de un producto afecta su confiabilidad, por otro lado la necesidad de reparabilidad decrece a medida que aumenta la confiabilidad.

Evolucionabilidad (Evolvability):

Un sistema es evolucionable si acepta cambios que le permitan satisfacer nuevos requerimientos. En otros productos de ingeniería las modificaciones van precedidas de actividades como estudios de factibilidad, diseño asociado, aprobaciones, evaluaciones y finalmente la introducción de la modificación. En el caso del software, en general la implementación del cambio se comienza sin realizar ningún estudio de factibilidad, dejando únicamente el diseño original y sin documentación a posteriori, esto es sin actualizar las especificaciones para reflejarlo, lo que hace que cambios futuros sean cada vez más difíciles de aplicar.

Por otro lado, los productos de software exitosos tienen larga duración, su primera liberación es el comienzo de un tiempo extenso de vida y cada liberación sucesiva es el próximo paso en la evolución del sistema. Si el software es diseñado cuidadosamente y cada modificación es realizada con cuidado puede evolucionar en buena forma. La mayoría de los sistemas de software comienzan siendo evolucionables pero luego de años de evolución alcanzan un estado en el cual cualquier modificación importante trae aparejado el riesgo de “dañar” características existentes. En general la aplicación de cambios sucesivos tiende a reducir la modularidad del

sistema original, que era lo que lo hacía evolucionable.

La evolucionabilidad es una cualidad tanto del producto como del proceso, aplicado al segundo éste debe ser capaz de adaptarse a nuevas técnicas de gestión y organización, cambios en la educación en ingeniería, etc. Es una de las cualidades más importantes del software, e involucra otros conceptos como familias de programas cuyo propósito es fomentar la evolucionabilidad.

Reusabilidad (Reusability):

La reusabilidad es similar a la evolucionabilidad: en la segunda se modifica un producto para construir una nueva versión del mismo producto, en la primera se utiliza un producto, posiblemente con modificaciones menores, para construir otro producto. Un ejemplo de un producto reusable es el shell de UNIX que además de aceptar comandos de usuario y ejecutarlos, puede ser iniciado mediante un archivo que contenga una lista de comandos del shell, lo que permite escribir programas (scripts) en el lenguaje de comandos del shell, por lo que puede verse el programa como un nuevo producto que utiliza el shell como componente. Puede parecer más apropiado aplicar este término a componentes del software que a productos completos, pero es ciertamente posible construir productos que sean reusables. Aunque es una herramienta importante para reducir los costos de producción del software, los ejemplos de reusabilidad son raros.

Es dificil lograr reusabilidad a posteriori, por el contrario se debe contemplar al momento de desarrollar los componentes del software; una técnica para lograr reusabilidad es la utilización de diseño orientado a objetos. Otro nivel de reusabilidad está dado en los requerimientos: al analizar una nueva aplicación se pueden identificar partes que son similares a otras utilizadas en una aplicación previa. También en el nivel del código, se pueden reutilizar componentes desarrollados en una aplicación anterior.

La reusabilidad puede afectar tanto al producto como al proceso, por ejemplo la reusabilidad de personas en el sentido de reutilizar sus conocimientos específicos en el dominio de una aplicación, entorno de desarrollo, etc, aunque también tiene sus problemas debido a que este conocimiento se va con las personas y nunca se vuelve un valor permanente independiente de éstas.

Es también una cualidad del proceso: varias metodologías de software pueden verse como intentos de reutilizar el mismo proceso para la construcción de productos distintos, y los modelos de ciclo de vida también son intentos de reutilizar procesos de alto nivel. En el enfoque “replay” de mantenimiento de software, se reutiliza el proceso seguido al hacer un cambio: se comienza modificando los requerimientos y se continúa con los mismos pasos utilizados en el desarrollo del producto original.

La reusabilidad es un factor clave que caracteriza la madurez de un área industrial: se pueden ver altos niveles de reusabilidad en áreas maduras como la industria automotriz donde se construye un auto ensamblando varios componentes que están altamente estandarizados y son utilizados en muchos modelos que produce la misma industria, además el proceso de manufactura es en general reutilizado. El bajo grado de reusabilidad que presenta el software es otra clara indicación de que se debe evolucionar para alcanzar el estatus de disciplina ingenieril bien establecida.

Portabilidad (Portability):

El software es portable si puede ser ejecutado en distintos ambientes, refiriéndose este último tanto a plataformas de hardware como a ambientes de software como puede ser determinado sistema operativo. Si bien se ha transformado en un tema importante debido a la proliferación de procesadores y sistemas operativos distintos, puede ser importante incluso en una misma familia de procesadores debido a las variaciones de capacidad de memoria e instrucciones adicionales, por lo que una forma de lograr portabilidad es asumir una configuración mínima y

utilizar un subconjunto de las facilidades provistas que se garantiza estarán disponibles en todos los modelos de la arquitectura, como instrucciones de máquina y facilidades del sistema operativo. También es necesario utilizar técnicas que permitan al software determinar las capacidades del hardware y adaptarse a éstas.

En general la portabilidad se refiere a la habilidad de un sistema de ser ejecutado en plataformas de hardware distintas, y a medida que la razón de dinero gastado en software versus hardware crece, la portabilidad gana importancia. Algunos sistemas de software son de por sí específicos para una máquina: un sistema operativo es escrito para controlar una computadora especifica, y un compilador produce código para una máquina específica, pero incluso en estos casos es posible alcanzar algún nivel de portabilidad, por ejemplo UNIX fue portado a varios sistemas de hardware distintos y aunque requirió meses de trabajo el esfuerzo fue mucho menor que escribirlo nuevamente desde cero. Para muchas aplicaciones es importante ser portable entre sistemas operativos, o de otra forma, los sistemas operativos proveen portabilidad entre plataformas de hardware.

Comprensibilidad (Understandability):

Algunos sistemas de software son más fáciles de comprender que otros, algunas tareas son inherentemente más complejas que otras. Por ejemplo, un sistema que realiza predicción del clima, sin importar lo bien que esté escrito, será más difícil de comprender que uno que imprime una lista de correo. Dadas dos tareas con dificultad similar, se pueden seguir ciertas guías para producir diseños y escribir programas más comprensibles.

La comprensibilidad es una cualidad interna del producto y ayuda a lograr muchas de las otras cualidades como evolucionabilidad y verificabilidad. Desde un punto de vista externo, un usuario considera que un sistema es comprensible si su comportamiento es predecible, en este caso la comprensibilidad es un componente de la amigabilidad al usuario.

Interoperabilidad (Interoperability):

La interoperabilidad se refiere a la habilidad de un sistema de coexistir y cooperar con otros sistemas, por ejemplo, la habilidad de un procesador de texto de incluir gráficas producidas por un paquete de gráficos. Aunque rara en los productos de software, la interoperabilidad abunda en otros productos de la ingeniería, por ejemplo, estéreos de distinta marca pueden conectarse juntos y también a televisiones y videograbadores, de hecho equipos producidos hace décadas se adaptan a nuevas tecnologías como discos compactos, mientras que casi todos los sistemas operativos tuvieron que ser modificados – en algunos casos significativamente – antes de que pudieran trabajar con los nuevos discos ópticos. Una vez más, el ambiente UNIX con sus interfaces estándares ofrece un ejemplo de interoperabilidad limitada en un único ambiente: promueve que las aplicaciones tengan una interfaz simple y estándar, lo que permite que la salida de una aplicación sea utilizada como entrada de otra. Por otro lado, esta interfaz es primitiva y orientada a caracteres, no es fácil para una aplicación utilizar datos estructurados como una imagen producida por otra aplicación. UNIX tampoco puede el mismo operar conjuntamente con otro sistema operativo

Un concepto relacionado con la interoperabilidad es el de *sistema abierto*, que es una colección extensible de aplicaciones escritas en forma independiente que cooperan para funcionar como un sistema integrado y permiten la adición de nuevas funcionalidades por parte de organizaciones independientes, luego de ser liberado. Esto se logra, por ejemplo, liberando el sistema conjuntamente con una especificación de sus interfaces “abiertas”, que podrán ser utilizadas por distintos desarrolladores para comunicación entre distintas aplicaciones o sistemas. Los sistemas abiertos permiten que distintas aplicaciones escritas por distintas organizaciones interoperen.

Productividad (Productivity):

La productividad es una cualidad del proceso de producción de software, mide la eficiencia del proceso y como se vio antes, es la cualidad de performance aplicada al proceso. Un proceso eficiente resulta en una entrega más rápida del producto.

Los ingenieros producen software individualmente a cierta tasa, la cual puede variar considerablemente entre individuos con habilidad distinta. Cuando los individuos conforman un equipo, la productividad de éste es alguna función de las productividades individuales, y en general esta productividad combinada es menor que la suma de las partes.

Con respecto a la productividad se pueden adoptar distintas soluciones de compromiso al momento de elegir un proceso, por ejemplo, si se requiere especialización de los individuos se logrará productividad en la producción de cierto producto pero no de otros, si se quieren desarrollar componentes reusables se reducirá la productividad del grupo a cargo del desarrollo debido a que éstos son más difíciles de desarrollar, pero se logrará una mayor productividad a nivel de la organización involucrada, en el desarrollo de varios productos.

Medir la productividad es una tarea difícil, se necesita una métrica que permita comparar distintos procesos en términos de su productividad. Algunas métricas como LOC (lines of code) producidas tienen varias desventajas que serán vistas posteriormente.

Como en otras disciplinas de la ingeniería la eficiencia del proceso se ve fuertemente afectada por la automatización: las herramientas de ingeniería de software y ambientes modernos, traen asociado un aumento en la productividad.

Oportunidad (Timeliness)

La oportunidad es una cualidad del proceso que se refiere a la habilidad de entregar un producto a tiempo. Históricamente los procesos de producción de software no han tenido esta cualidad lo que llevó a la llamada “crisis del software” que a su vez trajo aparejada la necesidad y el nacimiento de la ingeniería de software. Incluso actualmente muchos procesos fracasan en lograr sus resultados a tiempo.

La oportunidad en sí misma no es una cualidad útil, aunque llegar tarde puede llevar a perder oportunidades en el mercado, entregar un producto a tiempo que carece de otras cualidades como confiabilidad o performance, no tiene sentido.

Entregar un producto a tiempo requiere una agenda planeada cuidadosamente, con un trabajo de estimación acertado y puntos de revisión especificados claramente y verificables. Todas las demás disciplinas de la ingeniería utilizan técnicas estándares de gestión de proyectos, incluso hay varias herramientas informáticas para hacerlo. Estas técnicas estándares de gestión de proyectos son difíciles de aplicar en la ingeniería de software debido a la dificultad en medir la cantidad de trabajo requerido para producir una pieza de software dada, la dificultad en medir la productividad de los ingenieros y el uso de puntos de revisión imprecisos y no verificables. Otra razón de la dificultad en lograr la entrega en tiempo en un proceso de software es el cambio continuo en los requerimientos del usuario.

Una técnica para alcanzar la entrega en tiempo de un producto es la liberación incremental del mismo, esto es subconjuntos del producto completo cuyo uso ayuda a redefinir los requerimientos incrementalmente. La aplicación de esta técnica depende de la habilidad para partir las funciones requeridas del sistema, en subconjuntos que se puedan entregar en forma incremental, utilizando un proceso de desarrollo incremental, ya que un proceso no incremental no permite este tipo de producción aún si se pueden identificar los subconjuntos para la construcción incremental.

Visibilidad (Visibility):

Un proceso de desarrollo de software es visible si todos sus pasos y su estado actual son claramente documentados. Otros términos utilizados son transparencia y apertura. La idea es que los pasos y el estado del proyecto están disponibles y fácilmente accesibles para ser examinados externamente.

En muchos proyectos de software la mayoría de los ingenieros e incluso los gerentes no conocen el estado exacto del proyecto, algunos están diseñando, otros codificando, otros testeando y todos al mismo tiempo. Esto no está mal en si mismo, pero si un ingeniero comienza a rediseñar una gran parte del código justo antes de que el software sea liberado para una prueba de integración, los riesgos de generar problemas y retrasos son altos.

La visibilidad permite a los ingenieros pesar el impacto de sus acciones y por lo tanto, los guía al tomar decisiones, permite que los integrantes del equipo trabajen todos en la misma dirección. En el ejemplo anterior se puede observar que en lugar de esto, mientras el grupo de testeo ha trabajado en testear una versión y espera que la siguiente tenga corrección de errores y mínimas diferencias con ésta, un ingeniero decide rediseñar gran parte del código para corregir un defecto menor. Esto producirá, además de lo mencionado previamente, tensiones entre los grupos involucrados ya que por un lado se intenta estabilizar el software y por el otro se está desestabilizando, por supuesto sin intención.

Esta cualidad del software es tanto interna como externa, ya que en el transcurso de un proyecto muchas veces se requieren informes del estado del mismo, como presentaciones formales o informales, generalmente por parte de los gerentes de la organización para futuras planificaciones o por parte del mismo cliente. Si el proceso de desarrollo tiene poca visibilidad, los reportes de estado no serán acertados o insumirá mucho esfuerzo realizarlos cada vez.

Una dificultad en la gestión de proyectos grandes es la gestión del conocimiento, muchas veces información crítica sobre los requerimientos y diseño tiene forma de “folklore”: es conocida solo por personas que han estado en el proyecto desde el inicio o por un largo tiempo. En esos casos es bastante difícil recuperarse de la pérdida de un ingeniero o agregar nuevos, de hecho, esto último en general reduce la productividad de todo el proyecto debido al tiempo que demanda la transmisión de conocimientos al nuevo integrante.

Para lograr visibilidad es importante no solo documentar los pasos sino también mantener en forma adecuada el estado de los productos intermedios como la especificación de requerimientos y de diseño, o sea tener también visibilidad del producto.

A partir de: Fundamentals of Software Engineering – Carlo Ghezzi, Mehdi Jazayeri, Dino Mandrioli. Prentice-Hall, Inc. 1991, edición en inglés. ISBN-0-13-820432-2, Capítulo 1 – Software: its nature and qualities.