Introducción al curso

ICI4241- Ingeniería de Software
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

Introducción

- ¿Mundo sin software?
  - Automóviles, banca, negocios, educación, transporte, entre otros.
- Los sistemas de software son abstractos e intangibles.
  - No existen limites naturales al potencial de la Ingeniería de software.
  - Los sistemas de software pueden volverse rápidamente muy complejos, difíciles de entender y costosos de cambiar.

2

Software

- ¿Qué es software?
  - Programas de cómputo y documentación asociada. Los productos de software se desarrollan para un cliente en particular o para un mercado en general.
- ¿Cuáles son los atributos del buen software?
  - El buen software debe entregar al usuario la funcionalidad y el desempeño requeridos, y debe ser sustentable, confiable y utilizable.

3

Ingeniería de Software

- ¿Qué es la Ingeniería de Software?
  - La ingeniería de software es una disciplina de la ingeniería que se interesa por todos los aspectos de la producción de software.
- ¿Cuáles son las actividades fundamentales de la ingeniería de software?
  - Especificación, desarrollo, validación y evolución del software.

4

Ingeniería de software, sistemas y computación

- ¿Cuál es la diferencia entre ingeniería de software y ciencias de la computación?
  - Las ciencias de la computación se enfocan en teoría y fundamentos; mientras la ingeniería de software se enfoca en el sentido práctico del desarrollo y en la distribución de software.
- ¿Cuál es la diferencia entre ingeniería de software e ingeniería de sistemas?
  - La ingeniería de sistemas se interesa por todos los aspectos del desarrollo de sistemas basados en computadoras, incluidos hardware, software e ingeniería de procesos. La ingeniería de software es parte de este proceso más general.

5

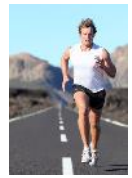
Historia de la Ingeniería de Software

- Se propuso originalmente en 1968, para discutir la crisis del software.
- Los enfoques individuales al desarrollo de programas no escalaban hacia los grandes y complejos sistemas de software.
- A lo largo de las décadas de 1970 y 1980 se desarrolló una variedad de nuevas técnicas y métodos de ingeniería de software, tales como la programación estructurada, el encubrimiento de información y el desarrollo orientado a objetos.

6

Retos de la Ingeniería de software

- Diversidad creciente:
  - Sistemas operativos, plataformas, tecnologías.
- Demandas por tiempos de distribución limitados .
- Desarrollo de software confiable.



7

Productos de software

- Productos genéricos:
  - Sistemas independientes que se producen por una organización de desarrollo y se venden en el mercado abierto a cualquier cliente. Ejemplo: Microsoft Office.
  - La organización que desarrolla el software controla la especificación del mismo.
- Productos personalizados (o a la medida):
  - Son sistemas que están destinados para un cliente en particular. Un contratista de software desarrolla el programa especialmente para dicho cliente. Traje de sastre.
  - La organización que compra el software generalmente desarrolla y controla la especificación, por lo que los desarrolladores de software deben trabajar siguiendo dicha especificación.

8

Lecturas y actividades del capítulo

- Desarrollar un ensayo sobre la crisis del software:
  - Formato IEEE LA disponible en el aula
  - Mantener reglas de redacción
  - Equipos de 2 a 3 personas.
  - Es parte del portafolio.

9

Ingeniería de software

IC14241- Ingeniería de Software
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

Hablemos de ingeniería

- Los ingenieros hacen que las cosas funcionen.
  - Aplican teorías, métodos y herramientas donde es adecuado.
- Los ingenieros también reconocen que deben trabajar ante restricciones organizacionales y financieras, de modo que buscan soluciones dentro de tales limitaciones.

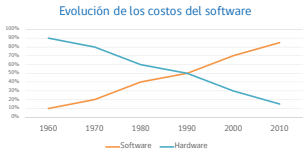
11

Hablemos de software

- La ingeniería de software no sólo se interesa por los procesos técnicos del desarrollo de software, sino también incluye:
  - Administración del proyecto de software y
  - el desarrollo de herramientas,
  - métodos y teorías para apoyar la producción de software.

12

Costos del software



13

Realidades del mundo del software

- Casos a considerar:
  - El 55% de los sistemas cuestan más de lo esperado, el 68% superan la fecha de entrega y el 88% tuvieron que ser sustancialmente rediseñados - IBM.
  - La media era 100 dólares por línea de código, se esperaba pagar 500 dólares por línea, y se terminó pagando entre 700 y 900 dólares por línea, 6.000 millones de dólares de trabajo fueron descartados - Advanced Automation System.
  - Cada 6 nuevos sistemas puestos en funcionamiento, 2 son cancelados, la probabilidad de cancelación está alrededor del 50% para sistemas grandes, la media de proyectos que sobrepasa el calendario es del 50%, 3 de cada 4 sistemas son considerados como fallos de operación. Bureau of Labor Statistics.

14

Realidades del mundo del software

- El 74% de todos los proyectos de tecnologías de la información fallan porque se pasan de presupuesto, porque no cumplen el plazo de entrega, y el 28% de los proyectos fallan completamente - The Standish Group.
- Cada año se gastan 75 billones de dólares en proyectos de tecnologías de la información fallidos en USA - The Standish Group.
- El 52,7% de los proyectos relacionados con las tecnologías de la información cuestan el 189% de su costo inicial estimado - The Standish Group.
- En grandes compañías sólo el 9% de los proyectos estuvieron en la fecha prevista y dentro del presupuesto - The Standish Group.
- El 31,1% de los proyectos se cancelan antes de completarse - Solutions Integrator.

15

Historias de software

- El servidor de un hospital comete un error fatal: "De acuerdo con esto, yo estoy muerto":
  - Durante una actualización programada de los archivos fuentes de una aplicación del Hospital Saint Mary's en octubre, Jeniffer Cammenga, la portavoz del hospital declaró: "Un numero (booleano) fue omitido en el código fuente, indicando que los pacientes habian fallecido, en lugar de indicar que habian sido dado de alta. - 8 de enero de 2003

16

Historias de software

- Otros emblemáticos casos:
  - Varias muertes de pacientes de cáncer acaecidas entre 1985–1987 se debieron a una sobredosis de radiación debida a un problema en las tareas concurrentes en el software de la máquina de radioterapia Therac-25 (http://sunnyday.mit.edu/therac-25.html).
  - En mayo de 2001 la Agencia Internacional para la Energía Atómica declaró una emergencia radiológica en Panamá. 28 pacientes sufrieron una sobre exposición, 8 murieron, y ¾ partes de los supervivientes pueden sufrir serias complicaciones que en algunos casos pueden llegar a ser mortales. Se concluyó que uno de los factores que provocaron el accidente se debió a un error en el software que controlaba ciertas entradas de datos (http://www.fda.gov/cdrh/ocd/panamaradexp.html)

17

Historias de software

- Otros emblemáticos casos:
  - Servicio de Ambulancias de Londres (1992). Falló en las pruebas de instalación del sistema y su compatibilidad con los ya existentes provocaron pérdida de llamadas y salidas múltiples debidas a llamadas duplicadas (http://www.cs.ucl.ac.uk/staff/A.Finkelstein/las.html.
  - El fallo en el lanzamiento del satélite Ariane 5 en 1996 fue causada por una rutina de excepción defectuosa en el código Ada, que se invocaba como resultado de una conversión errónea de un número en coma flotante de 64 bits a un entero de 16 bits (http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html).

18

Historias de software

- Otros emblemáticos casos:
  - Dos oficiales de policía en una región escocesa utilizaban una pistola de radar para identificar a motociclistas que infringían los límites de velocidad. Repentinamente la pistola de radar quedó bloqueada apuntando al cielo e indicando una velocidad de 300 millas por hora. Segundos más tarde, un caza Harrier, volando a baja altura, pasó por allí. El buscador de blancos del avión había detectado el radar y lo había tomado por un enemigo. Por fortuna, el Harrier volaba desarmado, ya que el comportamiento normal hubiera sido el disparo de un misil de contraataque automático (http://catless.ncl.ac.uk/Risks/17.67.html#subj1.1).

19

Ingeniería de software

- La ingeniería de software es una disciplina de ingeniería que se interesa por todos los aspectos de la producción de software, desde las primeras etapas de la especificación del sistema hasta el mantenimiento del sistema después de que se pone en operación.

Figura 1 - Atributos esenciales del buen software

22

Procesos de software

- El enfoque sistemático que se usa en la ingeniería de software se conoce en ocasiones como proceso de software.
- Un proceso de software es una secuencia de actividades que conducen a la elaboración de un producto de software.
- Actividades fundamentales de procesos de software:
  - Especificación del software, desarrollo del software, validación del software, evolución del software.
- Diferentes tipos de sistemas necesitan distintos procesos de desarrollo.

25

Historias de software

- Otros emblemáticos casos:
  - El 15 de enero de 1990 la red de comunicaciones de larga distancia de AT&T estuvo fuera de servicio durante nueve horas a consecuencia de un fallo de software. Millones de llamadas quedaron bloqueadas. Algunos negocios que dependían en gran medida de los servicios telefónicos, como agencias de viajes, quedaron prácticamente colapsados, con la consiguiente pérdida económica.
  - En febrero de 2003 un fallo en la red de Vodafone deja sin servicio a sus 8,6 millones de usuarios. La avería se produjo en el transcurso de las tareas de mantenimiento del software de la red llevadas a cabo por la operadora, impidiendo las comunicaciones con sus líneas en toda España desde las siete de la mañana .

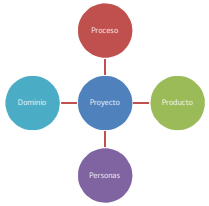
20

Ingeniería de software

- Relación directa entre calidad, fechas y recursos.
- Se debe adoptar un enfoque sistemático y organizado
- La ingeniería busca seleccionar el método más adecuado:
  - Desarrollo más creativo y menos formal suele ser más efectivo.

23

Procesos de software



26

Historias de software

- Otros emblemáticos casos:
  - Un fallo humano, no negligente, motivado por la complejidad del sistema ergonómico de la pantalla del computador del centro de control de operaciones del Metro causó, el día 31 de octubre de 2004, el choque de dos trenes del Metro de Barcelona resultando heridas 50 personas.
  - La Mars Polar Lander se estrelló en su aterrizaje en Marte en diciembre de 1999 por un fallo de software. Los motores de descenso se apagaron prematuramente porque un fallo en los sensores indicaban que había tomado tierra cuando estaba a unos 40 metros [Clark, 2000]
  - El robot Spirit en Marte tuvo que ser reseteado y actualizado desde la tierra por fallos en su memoria flash (enero de 2004) http://www.msnbc.msn.com/id/3855168/

21

Importancia de la Ingeniería de software

- Las personas y la sociedad se apoyan en los sistemas de software:
  - Se requiere producir económica y rápidamente sistemas confiables.
- Es más factible a largo plazo usar métodos y técnicas de ingeniería de software para los sistemas de software, que diseñar los programas como si fuera un proyecto de programación personal:
  - Recurrentemente mayoría de los costos consisten en cambiar el software después de ponerlo en operación.

24

Procesos de software

- Cualquier proceso tiene las siguientes características [Pfleeger,2002]:
  - El proceso establece todas las actividades principales.
  - El proceso utiliza recursos, está sujeto a una serie de restricciones y genera productos intermedios y finales.
  - El proceso puede estar compuesto de subprocesos que se encadenan de alguna manera. Puede definirse como una jerarquía de procesos organizada de modo que cada subproceso tenga su propio modelo de proceso.
  - Cada actividad del proceso tiene criterios de entrada y de salida, de modo que se conoce cuándo comienza y cuándo termina una actividad.

27

Procesos de software

- Cualquier proceso tiene las siguientes características [Pfleeger,2002]:
  - Las actividades se organizan en secuencia de modo que resulta claro cuando una actividad se realiza en orden relativo a otras actividades.
  - Todo proceso tiene un conjunto de principios orientadores que explican las metas de cada actividad.
  - Las restricciones o controles pueden aplicarse a una actividad, recurso o producto.

28

Procesos de software

- Importancia de los procesos en el desarrollo de software:
  - Facilita la gestión del proyecto y establece una división del trabajo.
  - Facilita la comunicación de los miembros del equipo.
  - Permite la reasignación y la reutilización de personal especializado.
  - Mejora la productividad y el desarrollo se vuelve reproducible
  - Establece el contexto en el que se aplican los métodos técnicos.
  - Gestiona el cambio adecuadamente.
  - Asegura la calidad.

29

Diversidad de la Ingeniería de software

- El factor más significativo en la determinación de qué métodos y técnicas de la ingeniería de software son más importantes, es el tipo de aplicación que está siendo desarrollada.
- Existen muchos diferentes tipos de aplicación, incluidos los siguientes:
  - Aplicaciones independientes, interactivas basadas en transacción, de control embebido, de entretenimiento.
- Los limites entre estos tipos de sistemas son difusos. Si se desarrolla un juego para Smartphone, se debe considerar las mismas restricciones que las de los desarrolladores del software del teléfono.

30

Ética en la Ingeniería de software

- La ingeniería de software se realiza dentro de un marco social y legal que limita la libertad de la gente que trabaja en dicha área.
- Un Ingeniero no debe usar sus habilidades y experiencia para comportarse de forma deshonesta o de un modo que desacredite la profesión de ingeniería de software.
- Existen áreas donde los estándares de comportamiento aceptable no están acotados por la legislación, sino por la noción más difusa de responsabilidad profesional.

31

Ética en la Ingeniería de software

- Elementos de la ética en Ingeniería de software:
  - Confidencialidad: confidencialidad de sus empleadores o clientes sin importar si se firmó o no un acuerdo formal sobre la misma.
  - Competencia: No debe desvirtuar su nivel de competencia. Es decir, no hay que aceptar de manera intencional trabajo que esté fuera de su competencia.
  - Derechos de propiedad intelectual: Tiene que conocer las leyes locales que rigen el uso de la propiedad intelectual. Debe ser cuidadoso para garantizar que se protege la propiedad intelectual de empleadores y clientes.
  - Mal uso de equipos: No emplear habilidades técnicas para usar incorrectamente los equipos de otros individuos.

32

Lecturas y actividades del capítulo

- Educación ética en ingeniería del software: responsabilidad en la producción de sistemas complejos.
  - Gonzalo Génova, M. Rosario González, Anabel Fraga.
- Código Ético y Deontológico para Ingenieros en Informática .
  - Disponible en aula virtual.

33

Procesos de software

ICI4241- Ingeniería de Software
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

Introducción

- Un proceso de software es una serie de actividades relacionadas que conduce a la elaboración de un producto de software.
- Las actividades pueden incluir el desarrollo de software desde cero en un lenguaje de programación estándar como Java o C.
- Existen muchos diferentes procesos de software, pero todos deben incluir:
  - Especificación del software, Diseño e implementación del software, Validación del software, Evolución del software.

35

Introducción

- Los procesos de software son complejos y, como todos los procesos intelectuales y creativos, se apoyan en personas con capacidad de juzgar y tomar decisiones.
- No hay un proceso ideal; además, la mayoría de las organizaciones han diseñado sus propios procesos de desarrollo de software.

36

Introducción

- Los procesos de software pueden mejorarse con la estandarización de los procesos.
- Esto conduce a mejorar la comunicación, a reducir el tiempo de capacitación, y a que el soporte de los procesos automatizados sea más económico.
- La estandarización también representa un primer paso importante tanto en la introducción de nuevos métodos y técnicas de ingeniería de software, como en sus buenas prácticas.

37

Modelos de proceso de software

- Es una representación simplificada de este proceso.
- Cada modelo del proceso representa a otro desde una particular perspectiva y, por lo tanto, ofrece sólo información parcial acerca de dicho proceso.
- Se explora el marco (framework) del proceso, pero no los detalles de las actividades especificas.
- Los modelos genèricos no son descripciones definitivas de los procesos de software.

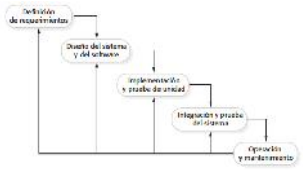
38

Modelo cascada

- El primer modelo publicado sobre el proceso de desarrollo de software.
- Debido al paso de una fase en cascada a otra conoce como "modelo en cascada" o ciclo de vida del software.
- Es un ejemplo de un proceso dirigido por un plan:
  - Se debe planear y programar todas las actividades del proceso, antes de comenzar a trabajar con ellas.

39

Modelo cascada



40

Modelo cascada

- Las principales etapas del modelo en cascada son:
  - **Análisis y definición de requerimientos**: Levantamiento de requerimientos se establecen mediante consulta a los usuarios del sistema → especificación del sistema.
  - **Diseño del sistema y del software**: Formalización de requerimientos. El diseño del software implica identificar y describir el software y su arquitectura.
  - **Implementación y prueba de unidad:** Consiste en verificar que cada unidad (por separado) cumpla con su especificación.
  - **Integración y prueba de sistema:** Los programas individuales se integran y prueban como un sistema completo para asegurarse de que se cumplan los requerimientos.
  - **Operación y mantenimiento:** El sistema se instala y se pone en producción.

41

Modelo cascada

- Recomendaciones
  - El resultado de cada fase consiste en uno o más documentos que se autorizaron quedando firmados.
  - La siguiente fase no debe comenzar sino hasta que termine la fase previa.
  - El proceso de software no es un simple modelo lineal, sino que implica retroalimentación de una fase a otra.
  - Sólo debe usarse cuando los requerimientos se entiendan bien y sea improbable el cambio radical durante el desarrollo del sistema.
  - Refleja el tipo de proceso utilizado en otros proyectos de ingeniería.

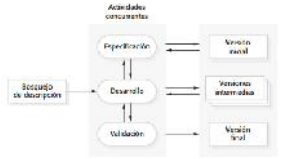
42

Desarrollo incremental

- Se basa en la idea de diseñar una implementación inicial, exponer ésta al comentario del usuario, y luego desarrollarla en sus diversas versiones hasta producir un sistema *adecuado*.
- Las actividades de especificación, desarrollo y validación están entrelazadas en vez de separadas, con rápida retroalimentación a través de las actividades.
- Al desarrollar el software de manera incremental, resulta más barato y fácil realizar cambios en el software conforme éste se diseña.

43

Desarrollo incremental



44

Desarrollo incremental

- Beneficios comparado con el modelo en cascada:
  - Se reduce el costo de adaptar los requerimientos cambiantes del cliente.
  - Es más sencillo obtener retroalimentación del cliente sobre el trabajo de desarrollo que se realizó.
  - Es posible que sea más rápida la entrega e implementación de software útil al cliente, aun si no se ha incluido toda la funcionalidad.

45

Desarrollo incremental

- Problemas:
  - **El proceso no es visible**: Se necesita de entregas regulares para medir el avance. Si los sistemas se desarrollan rápidamente, resulta poco efectivo en términos de costos producir documentos que reflejen cada versión del sistema.
  - **La estructura del sistema tiende a degradarse** conforme se tienen nuevos incrementos. A menos que se gaste tiempo y dinero en la refactorización para mejorar el software, el cambio regular tiende a corromper su estructura. La incorporación de más cambios de software se vuelve cada vez más difícil y costosa.
- Para dar respuesta a los problemas anteriores se necesita un buen levantamiento de requerimientos y un estricto control en los cambios.

46

Actividad grupal

- Necesitamos desarrollar un software que en general:
  - Permita conectarse con Facebook para el acceso.
  - Que muestre los indicadores comerciales y financieros (UF, Dólar, Euro, etc.)
  - Debe tener una versión de escritorio y una versión móvil
  - Debe permitir suscribirse a ciertos indicadores, lo cual debe llegar por correo electrónico.
- Indique el procesos de software a utilizar, montos y tiempos estimados preliminarmente. Debe justificar cada determinación.

49

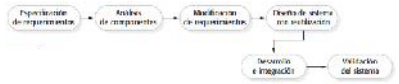
Especificación de software

- La especificación del software consiste en comprender y definir qué servicios se requieren del sistema,.
- Se deben identificar las restricciones sobre la operación y el desarrollo del sistema.
- La ingeniería de requerimientos es una etapa particularmente crítica del proceso de software:
  - Los errores en esta etapa conducen de manera inevitable a problemas posteriores tanto en el diseño como en la implementación del sistema.

52

Ingeniería de software orientada a la reutilización

- Los enfoques orientados a la reutilización se apoyan en una gran base de componentes de software reutilizable y en la integración de marcos para la composición de dichos componentes.

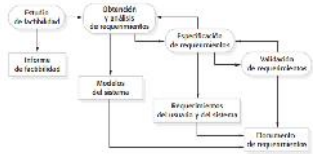


47

Actividades del proceso de software

ICI4241 - Ingeniería de Software
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

Especificación de software



53

Ingeniería de software orientada a la reutilización

- Principales etapas de la Ingeniería de software orientada a la reutilización:
  - **Análisis de componentes**: Dada la especificación de requerimientos, se realiza una búsqueda de componentes para implementar dicha especificación
  - **Modificación de requerimientos:** Durante esta etapa se analizan los requerimientos usando información de los componentes descubiertos. Luego se modifican para reflejar los componentes disponibles.
  - **Diseño de sistema con reutilización:** Durante esta fase se diseña el marco conceptual del sistema o se reutiliza un marco conceptual existente.
  - **Desarrollo e integración:** Se diseña el software que no puede procurarse de manera externa, y se integran los componentes.

48

Generalidades

- Las cuatro actividades básicas de proceso de especificación, desarrollo, validación y evolución se organizan de diversa manera en diferentes procesos de desarrollo.
- La forma en que se llevan a cabo estas actividades depende del tipo de software, del personal y de la inclusión de estructuras organizativas.

51

Diseño e implementación de software

- Un diseño de software se entiende como una descripción de la estructura del software que se va a implementar, los modelos y las estructuras de datos utilizados por el sistema, las interfaces entre componentes del sistema y, en ocasiones, los algoritmos usados.
- Los diseñadores no llegan inmediatamente a una creación terminada, sino que desarrollan el diseño de manera iterativa.

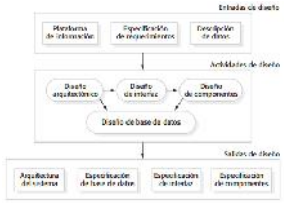
54

Diseño e implementación de software

- Corresponde al proceso de convertir una especificación del sistema en un sistema ejecutable.
- Incluye procesos de diseño y programación de software, también puede involucrar la corrección en la especificación del software (enfoque incremental).

55

Diseño e implementación de software



56

Diseño e implementación de software

- En el diseño se presentan 4 actividades:
  - **Diseño arquitectónico**: se identifica la estructura global del sistema, los principales componentes (subsistemas o módulos), sus relaciones y cómo se distribuyen.
  - **Diseño de interfaz**: se definen las interfaces entre los componentes de sistema. Los componentes se diseñan y se desarrollan de manera concurrente.
  - **Diseño de componentes**: en él se toma cada componente del sistema y se diseña cómo funcionará.
  - **Diseño de base de datos**: donde se diseñan las estructuras del sistema de datos y cómo se representarán en una base de datos.

57

Validación de software

- La validación de software se usa para mostrar que un sistema cumple tanto con sus especificaciones como con las expectativas del cliente.
- Las pruebas del programa, donde el sistema se ejecuta a través de datos de prueba simulados, son la principal técnica de validación.
- Con excepción de los programas pequeños, los sistemas no deben ponerse a prueba como una unidad monolítica.

58

Validación de software

- Las etapas en el proceso de pruebas son:
  - **Prueba de desarrollo**: Las personas que desarrollan el sistema ponen a prueba los componentes que constituyen el sistema. Cada componente se prueba de manera independiente, es decir, sin otros componentes del sistema.
  - **Pruebas del sistema**: Los componentes del sistema se integran para crear un sistema completo. Este proceso tiene la finalidad de descubrir errores que resulten de interacciones no anticipadas entre componentes y problemas de interfaz de componente, así como de mostrar que el sistema cubre sus requerimientos funcionales y no funcionales.

59

Validación de software

- Las etapas en el proceso de pruebas son:
  - **Pruebas de aceptación**: Ésta es la etapa final en el proceso de pruebas, antes de que el sistema se acepte para uso operacional. El sistema se pone a prueba con datos suministrados por el cliente del sistema, en vez de datos de prueba simulados. Las pruebas de aceptación revelan los errores y las omisiones en la definición de requerimientos del sistema, ya que los datos reales ejercitan el sistema en diferentes formas a partir de los datos de prueba.

60

Validación de software

61

Actividad de desarrollo de competencias

- Sabemos que el software posterior a su entrega necesita de un mantenimiento también conocido como "Evolución del software". En razón a ello:
  - Genere un modelo de procesos (BPMN u otra notación) para modelar la evolución de software.
  - Debe entregar las especificaciones y reglas de su modelo (por escrito).

62

Variabilidad en proyectos de software

IC14241- Ingeniería de Software
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

Cómo enfrentar el cambio

- El cambio es inevitable en todos los grandes proyectos de software.
- Los requerimientos del sistema varían conforme a presiones externas y prioridades administrativas.
- A medida que se ponen a disposición nuevas tecnologías, surgen nuevas posibilidades de diseño e implementación.
- Sucede cualquiera que sea el modelo del proceso de software utilizado.

64

Evolución de los sistemas

65

Cómo enfrentar el cambio

- El cambio se agrega a los costos del desarrollo de software ya que el trabajo ya terminado debe volver a realizarse (rehacer).
- Existen dos enfoques que se usan para reducir los costos del rehacer:
  - **Evitar el cambio**, donde el proceso de software incluye actividades que anticipan cambios posibles antes de requerirse la labor significativa de rehacer. (Prototipos)
  - **Tolerancia al cambio**, donde el proceso se diseña de modo que los cambios se ajusten con un costo relativamente bajo. (Incremental)

66

Prototipado

- Un prototipo es una versión inicial de un sistema de software que se usa para demostrar conceptos.
- El rápido desarrollo iterativo del prototipo es esencial, de modo que se controlen los costos.
- Busca que los interesados en el sistema experimenten por anticipado con el prototipo durante el proceso de software.

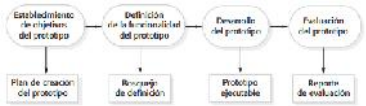
67

Prototipado

- Se usa para anticipar los cambios que se requieran:
  - En el proceso de ingeniería de requerimientos, un prototipo ayuda con la selección y validación de requerimientos del sistema.
  - En el proceso de diseño de sistemas, un prototipo sirve para buscar soluciones específicas de software y apoyar el diseño de interfaces del usuario.

68

Prototipado



69

Problemas del prototipado

- Un problema es que quizás el prototipo no se utilice necesariamente en la misma forma que el sistema final.
- El revisor del prototipo tal vez no sea un usuario típico del sistema.
- En ocasiones, los desarrolladores están presionados para entregar prototipos desechables, sobre todo cuando existen demoras en la entrega de la versión final del software. Esto tiene varios problemas!

70

Problemas del prototipado

- Problemas de los prototipos desechables:
  - Puede ser imposible corregir el prototipo para cubrir requerimientos no funcionales:
    - Rendimiento, seguridad, robustez y fiabilidad.
  - El cambio rápido durante el desarrollo significa que el prototipo no está documentado.
  - Los cambios realizados durante el desarrollo de prototipos degradarán la estructura del sistema, y este último será difícil y costoso de mantener.
  - Durante el desarrollo de prototipos se hacen más flexibles los estándares de calidad de la organización.

71

Entrega incremental

- Es un enfoque al desarrollo de software donde algunos de los incrementos se entregan al cliente y se implementan para usarse en un entorno operacional.
- En un proceso de entrega incremental, los clientes identifican cuáles servicios son más importantes y cuáles son menos significativos para ellos.
- Cada incremento proporciona un subconjunto de la funcionalidad del sistema.

72

Entrega incremental

- Los servicios de más alta prioridad se implementan y entregan primero.
- Una vez completado y entregado el incremento, los clientes lo ponen en funcionamiento. (funcionalidad parcial del sistema).
- Pueden experimentar con el sistema que les ayuda a clarificar sus requerimientos, para posteriores incrementos del sistema.

73

Entrega incremental

74

Ventajas de la entrega incremental

- La entrega incremental tiene algunas ventajas:
  - Los clientes pueden usar los primeros incrementos como prototipos y adquirir experiencia que informe sobre sus requerimientos
  - El primer incremento cubre sus requerimientos más críticos, de modo que es posible usar inmediatamente el software.
  - Los servicios de sistema más importantes reciben mayores pruebas.

75

Problemas de la entrega incremental

- Existen problemas con la entrega incremental:
  - Dado que los requerimientos no están definidos con detalle sino hasta que se implementa un incremento, resulta difícil identificar recursos comunes que necesiten todos los incrementos.
  - El desarrollo iterativo resulta complicado cuando se diseña un sistema de reemplazo.
  - En el enfoque incremental, no hay especificación completa del sistema, sino hasta que se define el incremento final. Esto requiere una nueva forma de contrato que los grandes clientes, como las agencias gubernamentales, encontrarían difícil de adoptar.

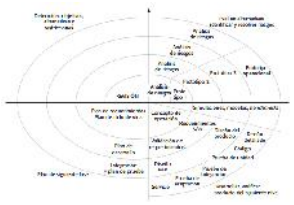
76

Modelo en espiral de Boehm

- Boehm propuso un marco del proceso de software dirigido por el riesgo.
- Aquí, el proceso de software se representa como una espiral, y no como una secuencia de actividades.
- Cada ciclo en la espiral representa una fase del proceso de software.
- El modelo en espiral combina el evitar el cambio con la tolerancia al cambio.

77

Modelo en espiral de Boehm



78

Modelo en espiral de Boehm

- Cada ciclo en la espiral se divide en cuatro sectores:
  - **Establecimiento de objetivos**: Se definen objetivos específicos para dicha fase del proyecto.
  - **Valoración y reducción del riesgo**: En cada uno de los riesgos identificados del proyecto, se realiza un análisis minucioso. Se dan acciones para reducir el riesgo.
  - **Desarrollo y validación**: Después de una evaluación del riesgo, se elige un modelo de desarrollo para el sistema.
  - **Planeación**: El proyecto se revisa y se toma una decisión sobre si hay que continuar con otro ciclo de la espiral.

79

Actividad de formación de competencias

- Desarrolle un completo estado del arte usando el formato de informe publicado en el aula.
  - Tema: El Proceso Unificado Racional (R.U.P)
  - Entrega jueves 31 vía aula virtual.

80

Desarrollo ágil de software

IC14241- Ingeniería de Software
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

Introducción

- Las empresas operan en un entorno global que cambia rápidamente y deben responder frente a nuevas oportunidades y mercados, así como al surgimiento de productos y servicios competitivos.
- Los procesos de desarrollo del software rápido se diseñan para producir rápidamente un software útil.
- El software no se desarrolla como una sola unidad, sino como una serie de incrementos, y cada uno de ellos incluye una nueva funcionalidad del sistema.

82

Introducción

- Existen muchos enfoques para el desarrollo de software ágil, comparten algunas características fundamentales:
  - Los procesos de especificación, diseño e implementación están entrelazados.
  - El sistema se desarrolla en diferentes versiones. Los usuarios finales y otros colaboradores del sistema intervienen en la especificación y evaluación de cada versión.
  - Las interfaces de usuario del sistema se desarrollan usando con frecuencia un sistema de elaboración interactivo, que permita que el diseño de la interfaz se cree rápidamente en cuanto se dibujan y colocan iconos en la interfaz.

83

Métodos ágiles

- Primeros inicios en los años 90.
- Centrado en negocios pequeños y medianos.
- Busca que el equipo de desarrollo se enfoque en el software y no en el diseño y la documentación.
- Se apoyan en el enfoque incremental para la especificación, el desarrollo y la entrega del software.
- Apropiados para requerimientos inestables.

84

Principios

Principio	Descripción
Participación del cliente	Los clientes deben involucrarse estrechamente durante el proceso de desarrollo. Su función consiste en ofrecer y priorizar nuevos requerimientos del sistema y evaluar las iteraciones del mismo.
Entrega incremental	El software se desarrolla en incrementos y el cliente especifica los requerimientos que se van a incluir en cada incremento.
Personas, no procesos	Tienen que reconocerse y aprovechar las habilidades del equipo de desarrollo. Debe permitirse a los miembros del equipo desarrollar sus propias formas de trabajar sin procesos establecidos.
Adoptar el cambio	Esperar a que cambien los requerimientos del sistema y, de esta modo, diseñar el sistema para adaptar dichos cambios.
Mantener simplicidad	Enfocarse en la simplicidad tanto en el software a desarrollar como en el proceso de desarrollo. Siempre que sea posible, trabajar de manera activa para eliminar la complejidad del sistema.

85

Desarrollo dirigido por un plan y desarrollo ágil

- Los **enfoques ágiles** en el desarrollo de software consideran el diseño y la implementación como las actividades centrales en el proceso del software.
- Un **enfoque basado en un plan** para la ingeniería de software identifica etapas separadas en el proceso de software con salidas asociadas a cada etapa.

86

Desarrollo dirigido por un plan y desarrollo ágil

87

Desarrollo dirigido por un plan y desarrollo ágil

- Para decidir entre un enfoque basado en un plan y uno ágil, se debe responder:
  - ¿Es importante tener una especificación y un diseño muy detallados antes de dirigirse a la implementación? Si → enfoque basado en un plan.
  - ¿Es práctica una estrategia de entrega incremental, donde se dé el software a los clientes y se obtenga así una rápida retroalimentación de ellos? Si → método ágil.
  - ¿Qué tan grande es el sistema que se desarrollará? Pequeño → método ágil.
  - ¿Qué tipo de sistema se desarrollará? Con mucho análisis → enfoque basado en un plan.
  - ¿El sistema está sujeto a regulación externa? Si → enfoque basado en un plan .

88

Desarrollo dirigido por un plan y desarrollo ágil

- ¿Cuál es el tiempo de vida que se espera del sistema? Sistema a largo plazo → enfoque basado en un plan.
- ¿Qué tecnologías existen para apoyar el desarrollo del sistema? Buenas herramientas (menos análisis) → método ágil.
- ¿Cómo está organizado el equipo de desarrollo? Subcontratación → enfoque basado en un plan (más documentación).
- ¿Existen problemas culturales que afecten el desarrollo del sistema? Conocimiento informal → método ágil.
- ¿Qué tan buenos son los diseñadores y programadores en el equipo de desarrollo?. Buen desempeño → método ágil.

89

Administración de un proyecto ágil

- La responsabilidad principal de los administradores del proyecto de software es dirigir el proyecto:
  - El software se entregue a tiempo
  - Y con el presupuesto planeado para ello.
- El desarrollo ágil también tiene que administrarse de tal modo que se busque el mejor uso del tiempo y de los recursos disponibles.
- El enfoque de Scrum es un método centrado en la administración iterativa del desarrollo.

90

Administración de un proyecto ágil

- La responsabilidad principal de los administradores del proyecto de software es dirigir el proyecto:
  - El software se entregue a tiempo
  - Y con el presupuesto planeado para ello.
- El desarrollo ágil también tiene que administrarse de tal modo que se busque el mejor uso del tiempo y de los recursos disponibles.
- El enfoque de Scrum es un método centrado en la administración iterativa del desarrollo.

91

Actividad grupal:

- Buscar información y generar un documento sobre Scrum:
  - Formato a elección del grupo
  - Se valorará innovación y creatividad
  - Formato puede ser PDF, PPT, entre otros.
  - Entrega: Martes 12 de abril.

92

Tipos de requerimientos

ICI4241 - Ingeniería de Software
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

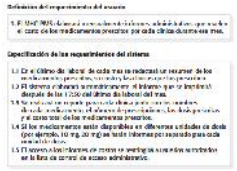
Introducción

- Los requerimientos para un sistema son descripciones de lo que el sistema debe hacer.
- Tales requerimientos reflejan las necesidades de los clientes por un sistema que atienda cierto propósito.
- Al proceso de descubrir, analizar, documentar y verificar estos servicios y restricciones se le llama ingeniería de requerimientos.
- Existen requerimientos de usuario y requerimientos de sistema.

94

Introducción

- Requerimientos de usuario versus requerimientos de sistema:



95

Lectores de requerimientos

- En razón a lo anterior se definen los lectores de requerimientos:

96

Requerimientos funcionales y no funcionales

- **Requerimientos funcionales**: Son enunciados acerca de servicios que el sistema debe proveer, de cómo debería reaccionar el sistema a entradas particulares y de cómo debería comportarse el sistema en situaciones específicas.
- **Requerimientos no funcionales**: Son limitaciones sobre servicios o funciones que ofrece el sistema. Incluyen restricciones tanto de temporización y del proceso de desarrollo, como impuestas por los estándares.

97

Requerimientos funcionales

- Refieren lo que el sistema debe hacer.
- Dependen del tipo de software que se esté desarrollando, de los usuarios esperados y del enfoque de la organización al escribir los requerimientos:
  - Al expresarse como requerimientos del **usuario**, los requerimientos funcionales se describen por lo general de forma abstracta que entiendan los usuarios del sistema.
  - Desde el **sistema**, detallan las funciones del sistema, sus entradas y salidas, sus excepciones, etc.

98

Requerimientos no funcionales

- Los requerimientos no funcionales, como el rendimiento, la seguridad o la disponibilidad, especifican o restringen por lo general características del sistema como un todo.
- Los requerimientos no funcionales a menudo son más significativos que los requerimientos funcionales individuales.

99

Tipos de requerimientos no funcionales

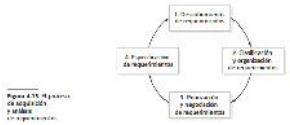
100

Actividad grupal:

- Buscar información y realizar un resumen breve sobre "Requerimientos de dominio" :
  - Formato a elección del grupo
  - Se valorará innovación y creatividad
  - Formato puede ser PDF, PPT, entre otros.
  - Entrega: Martes 12 de abril.

103

Proceso de adquisición y análisis



106

Requerimientos no funcionales

- Un problema común es que los usuarios proponen estos requerimientos como metas generales, como facilidad de uso, capacidad de que el sistema se recupere de fallas, o rapidez de respuesta al usuario.
- Las metas establecen buenas intenciones; no obstante, ocasionan problemas a los desarrolladores del sistema, pues dejan espacio para la interpretación y la disputa posterior una vez que se entregue el sistema.

101

Adquisición y análisis de requerimientos

ICI4241 - Ingeniería de Software
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

Problemas en adquisición y compresión

- La adquisición y la comprensión de los requerimientos es un proceso difícil :

107

Métricas para requerimientos no funcionales

Propiedad	Medida
Rapidez	Transacciones/seg procesadas Tiempo de respuesta usuario/sistema Tiempo de regeneración de pantalla
Tamaño	Mbytes Número de chips ROM
Facilidad de uso	Tiempo de capacitación Número de cuadros de ayuda
Confiabilidad	Tiempo medio para fallas Probabilidad de no disponibilidad Tasa de ocurrencia de fallas Disponibilidad
Robustez	Tiempo de reinicio después de falla Porcentaje de eventos que causan falla Probabilidad de corrupción de datos en la falla
Portabilidad	Porcentaje de enunciados dependientes de objetivo Número de sistemas objetivo

102

Introducción

- En esta actividad, los ingenieros de software trabajan con clientes y usuarios finales del sistema para descubrir:
  - El dominio de aplicación
  - Qué servicios debe proporcionar el sistema
  - El desempeño requerido de éste
  - Las restricciones de hardware.

105

Problemas en adquisición y compresión

- La adquisición y la comprensión de los requerimientos es un proceso difícil :
  - Los participantes con frecuencia **no saben lo que quieren**, excepto en términos muy generales; pueden encontrar difícil articular **qué quieren que haga el sistema**; pueden hacer **peticiones inalcanzables** porque no saben qué es factible y qué no lo es .

108

Problemas en adquisición y compresión

- La adquisición y la comprensión de los requerimientos es un proceso difícil:
  - Los participantes en un sistema expresan **naturalmente los requerimientos** con sus términos y **conocimientos implícitos de su trabajo**. Los ingenieros de requerimientos, sin experiencia en el dominio del cliente, podrían no entender dichos requerimientos.

109

Problemas en adquisición y compresión

- La adquisición y la comprensión de los requerimientos es un proceso difícil:
  - Diferentes participantes tienen **distintos requerimientos** y pueden expresarlos en **variadas formas**. Los ingenieros de requerimientos deben descubrir todas las fuentes potenciales de requerimientos e identificar similitudes y conflictos.

110

Problemas en adquisición y compresión

- La adquisición y la comprensión de los requerimientos es un proceso difícil:
  - **Factores políticos** llegan a influir en los requerimientos de un sistema. Los administradores pueden solicitar requerimientos específicos del sistema, porque éstos les permitirán aumentar su influencia en la organización.

111

Descubrimiento de requerimientos

- El descubrimiento de requerimientos es el proceso de recopilar información sobre el sistema requerido y los sistemas existentes.
- Las fuentes de información durante la fase de descubrimiento de requerimientos incluyen:
  - documentación,
  - participantes del sistema y
  - especificaciones de sistemas similares.

114

Problemas en adquisición y compresión

- La adquisición y la comprensión de los requerimientos es un proceso difícil:
  - El ambiente económico y empresarial donde ocurre el análisis es **dinámico**. Inevitablemente **cambia durante el proceso de análisis**. Puede cambiar la importancia de requerimientos particulares; o bien, tal vez surjan nuevos requerimientos de nuevos participantes a quienes no se consultó originalmente.

112

Problemas en adquisición y compresión

- Es imposible complacer por completo a cada participante.
- Si algunos suponen que sus visiones no se consideraron de forma adecuada, quizás intenten deliberadamente **boicotear el proceso** de ingeniería de requerimientos o construcción de software.

113

Problemas de las entrevistas

- Todos los especialistas en la aplicación usan terminología y jerga que son específicos de un dominio. Es imposible que ellos discutan los requerimientos de dominio sin usar este tipo de lenguaje.
  - Por lo general, usan la terminología en una forma precisa y sutil, que para los ingenieros de requerimientos es fácil de mal interpretar.
- Cierto conocimiento del dominio es tan familiar a los participantes que encuentran difícil explicarlo, o bien, creen que es tan fundamental que no vale la pena mencionarlo.

117

Entrevistas

- La interacción con los participantes es a través de entrevistas y observaciones, y pueden usarse escenarios y prototipos para ayudar a los participantes a entender cómo será el sistema.
- Las entrevistas con participantes del sistema son una parte de la mayoría de los procesos de ingeniería de requerimientos.
- El equipo de ingeniería de requerimientos formula preguntas a los participantes sobre el sistema que actualmente usan y el sistema que se va a desarrollar.

115

Entrevistas

- Los requerimientos se derivan de las respuestas a dichas preguntas.
- Las entrevistas son de dos tipos:
  - **Entrevistas cerradas**: donde los participantes responden a un conjunto de preguntas preestablecidas.
  - **Entrevistas abiertas**: en las cuales no hay agenda predefinida. El equipo de ingeniería de requerimientos explora un rango de conflictos con los participantes del sistema y, como resultado, desarrolla una mejor comprensión de sus necesidades.

116

Recomendaciones para las entrevistas

- Tener mentalidad abierta: evitar ideas preconcebidas sobre los requerimientos y escuchar a los participantes:
  - Si el participante aparece con requerimientos sorprendentes, entonces tener disposición para cambiar su mentalidad acerca del sistema.
- Instar al entrevistado con una pregunta de trampolín para continuar la conversación, dar una propuesta de requerimientos o trabajar juntos en un sistema de prototipo.

118

Escenarios

- Las personas encuentran más sencillo vincularse con ejemplos reales que con descripciones abstractas.
- Pueden comprender y criticar un escenario sobre cómo interactuar con un sistema de software.
- Los ingenieros de requerimientos usan la información obtenida de esta discusión para formular los verdaderos requerimientos del sistema.
- Los escenarios son particularmente útiles para detallar un bosquejo de descripción de requerimientos.

119

Escenarios

- Cada escenario abarca comúnmente una interacción o un número pequeño de interacciones posibles.
- Un escenario puede incluir:
  - Una descripción de qué esperan el sistema y los usuarios cuando inicia el escenario.
  - Una descripción en el escenario del flujo normal de los eventos.
  - Una descripción de qué puede salir mal y cómo se manejaría.
  - Información de otras actividades que estén en marcha al mismo tiempo.
  - Una descripción del estado del sistema cuando termina el escenario.

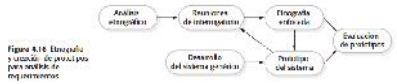
120

Etnografía

- Una razón por la que muchos sistemas de software se entregan, y nunca se utilizan, es que sus requerimientos no consideran de manera adecuada cómo afectaría el contexto social y organizacional la operación práctica del sistema.
- La etnografía es una técnica de observación que se usa para entender los procesos operacionales y ayudar a derivar requerimientos de apoyo para dichos procesos.

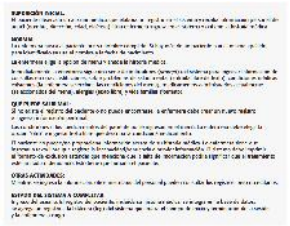
122

Etnografía y requerimientos



123

Ejemplo de escenario



121

Etnografía

- La etnografía es muy efectiva para descubrir dos tipos de requerimientos:
  - Requerimientos que se derivan de la forma en que realmente trabaja la gente, en vez de la forma en la cual las definiciones del proceso indican que debería trabajar.
  - Requerimientos que se derivan de la cooperación y el conocimiento de las actividades de otras personas.

124

Actividad grupal:

- Buscar información y realizar un resumen breve sobre "Utilización de casos de uso (UML) para la captura de requerimientos" :
  - Formato a elección del grupo
  - Se valorará innovación y creatividad
  - Formato puede ser PDF, PPT, entre otros.
  - Entrega: Martes 2 de mayo.

125

Pruebas de software

ICI4241- Ingeniería de Software
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

Introducción

- Las pruebas intentan demostrar que un programa hace lo que se intenta que haga, así como descubrir defectos en el programa antes de usarlo.
- Al probar el software, se ejecuta un programa con datos artificiales. Hay que verificar los resultados de la prueba que se opera para buscar errores, anomalías o información de atributos no funcionales del programa.

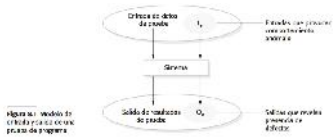
127

Introducción

- El proceso de prueba tiene dos metas distintas:
  - Demostrar al desarrollador y al cliente que el software cumple con los requerimientos.
  - Encontrar situaciones donde el comportamiento del software sea incorrecto, indeseable o no esté de acuerdo con su especificación.
- Las pruebas pueden mostrar sólo la presencia de errores, mas no su ausencia.

128

Introducción



129

Validación y verificación

- Boehm, pionero de la ingeniería de software, expresó de manera breve la diferencia (Boehm, 1979):
  - "Validación: ¿construimos el producto correcto?".
  - "Verificación: ¿construimos bien el producto?".

130

Validación y verificación

- El objetivo final de los procesos de verificación y validación es establecer confianza de que el sistema de software es "adecuado":
  - Propósito del software: Cuanto más crítico sea el software, más importante debe ser su confiabilidad.
  - Expectativas del usuario: Debido a su experiencia con software no confiable y plagado de errores, muchos usuarios tienen pocas expectativas de la calidad del software, por lo que no se sorprenden cuando éste falla.
  - Entorno de mercado: Cuando un sistema se comercializa, los vendedores del sistema deben considerar los productos competitivos, el precio que los clientes están dispuestos a pagar por un sistema y la fecha requerida para entregar dicho sistema.

131

Inspecciones

- Las inspecciones se enfocan principalmente en el código fuente.
- Se utiliza el conocimiento del sistema, su dominio de aplicación y el lenguaje de programación o modelado para descubrir errores.
- Hay tres ventajas en la inspección del software sobre las pruebas:
  - Durante las pruebas, los errores pueden enmascarar (ocultar) otras fallas.
  - Las versiones incompletas de un sistema no se pueden inspeccionar sin costos adicionales.
  - Puede considerar también atributos más amplios de calidad de un programa, como el cumplimiento con estándares, la portabilidad y la mantenibilidad.

132

Inspecciones

- Las inspecciones de programa son una idea antigua y la mayoría de estudios y experimentos indican que las inspecciones son más efectivas para el descubrimiento de defectos, que para las pruebas del programa.

133

Modelo de procesos de pruebas

- Por lo general, un sistema debe pasar por tres etapas de pruebas:
  - **Pruebas de desarrollo**, donde el sistema se pone a prueba durante el proceso para descubrir errores (bugs) y defectos.
  - **Versiones de prueba**, donde un equipo de prueba por separado experimenta una versión completa del sistema, antes de presentarlo a los usuarios.
  - **Pruebas de usuario**, donde los usuarios reales o potenciales de un sistema prueban el sistema en su propio entorno.

134

Modelo de procesos de pruebas

135

Pruebas de desarrollo

- Las pruebas de desarrollo incluyen todas las actividades de prueba que realiza el equipo que elabora el sistema.
- Durante el desarrollo, las pruebas se realizan en tres niveles:
  - Pruebas de unidad, donde se ponen a prueba unidades de programa o clases de objetos individuales.
  - Pruebas del componente, donde muchas unidades individuales se integran para crear componentes compuestos.
  - Pruebas del sistema, donde algunos o todos los componentes en un sistema se integran y el sistema se prueba como un todo.

136

Desarrollo dirigido por pruebas

- El desarrollo dirigido por pruebas es un enfoque de diseño de programas donde se entrelazan el desarrollo de pruebas y el de código.
- El código se desarrolla incrementalmente, junto con una prueba para ese incremento.
- No se avanza hacia el siguiente incremento sino hasta que el código diseñado pasa la prueba.

137

Desarrollo dirigido por pruebas

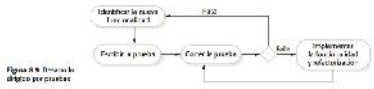


Figura 8.4 Desarrollo dirigido por pruebas

138

Desarrollo dirigido por pruebas

- Algunos beneficios:
  - **Cobertura de código**: cualquier segmento de código que escriba debe tener al menos una prueba asociada. Por lo tanto, puede estar seguro de que cualquier código en el sistema se ejecuta realmente.
  - **Pruebas de regresión**: un conjunto de pruebas se desarrolla incrementalmente conforme se desarrolla un programa. Siempre es posible correr pruebas de regresión para demostrar que los cambios al programa no introdujeron nuevos bugs.

139

Desarrollo dirigido por pruebas

- Algunos beneficios:
  - **Depuración simplificada**: cuando falla una prueba, debe ser evidente dónde yace el problema. Es preciso comprobar y modificar el código recién escrito.
  - **Documentación del sistema**: las pruebas en sí actúan como una forma de documentación que describen lo que debe hacer el código. Leer las pruebas suele facilitar la comprensión del código.

140

Pruebas de versión

- Las pruebas de versión son el proceso de poner a prueba una versión particular de un sistema que se pretende usar fuera del equipo de desarrollo.
- Distinciones entre las pruebas de versión y las pruebas del sistema durante el proceso de desarrollo:
  - Un equipo independiente que no intervino en el desarrollo del sistema debe ser el responsable de las pruebas de versión.

141

Pruebas de versión

- Distinciones entre las pruebas de versión y las pruebas del sistema durante el proceso de desarrollo:
  - Las pruebas del sistema por parte del equipo de desarrollo deben enfocarse en el descubrimiento de bugs en el sistema (pruebas de defecto).
  - El objetivo de las pruebas de versión es comprobar que el sistema cumpla con los requerimientos y sea suficientemente bueno para uso externo (pruebas de validación).

142

Pruebas de versión

- La principal meta del proceso de pruebas de versión es convencer "a calidad" de que éste es suficientemente apto para su uso.
- Las pruebas de versión deben mostrar que el sistema entrega su funcionalidad, rendimiento y confiabilidad especificados, y que no falla durante el uso normal.
- Las pruebas de versión, por lo regular, son un proceso de prueba de caja negra, donde las pruebas se derivan a partir de la especificación del sistema.

143

Pruebas de versión

- Tipos de prueba de versión:
  - Pruebas basadas en requerimientos
  - Pruebas de escenario
  - Pruebas de rendimiento

144

Pruebas de usuario

- Las pruebas de usuario o del cliente son una etapa en el proceso de pruebas donde los usuarios o clientes proporcionan entrada y asesoría sobre las pruebas del sistema.
- Esto puede implicar probar de manera formal un sistema, o podría ser un proceso informal donde los usuarios experimentan con un nuevo producto de software, para ver si les gusta y si hace lo que necesitan.
- Las pruebas de usuario son esenciales, aun cuando se hayan realizado pruebas abarcadoras de sistema y de versión.

145

Introducción a la gestión de proyectos

IC14241 - Ingeniería de Software
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

Historia

- Las legiones romanas diferían de sus contrincantes en:
  - La tecnología que utilizaban era mejor: bronce, hierro, entre otros
  - La planificación y seguimiento estaba más elaborada, es decir:
    - Tenían los objetivos más claros,
    - Tenían mejor preparada la estrategia a seguir,
    - Estaban mejor organizados,
    - Informaban rápidamente a sus compañeros, para que estos pudieran corregir cualquier desviación (Maratón).

151

Pruebas de usuario

- Hay tres diferentes tipos de pruebas de usuario:
  - **Pruebas alfa**, donde los usuarios del software trabajan con el equipo de diseño para probar el software en el sitio del desarrollador.
  - **Pruebas beta**, donde una versión del software se pone a disposición de los usuarios, para permitirles experimentar y descubrir problemas que encuentran con los desarrolladores del sistema.
  - **Pruebas de aceptación**, donde los clientes prueban un sistema para decidir si está o no listo para ser aceptado por los desarrolladores del sistema y desplegado en el entorno del cliente.

146

Gestión de proyectos informáticos

149

¿Qué es un proyecto?

- Final de la carrera (Memoria de Título)
- Planos y especificaciones
- Forma de organizar el trabajo

152

Actividad grupal:

- Buscar información y realizar un resumen breve sobre "Entornos automatizado de pruebas" :
  - Formato a elección del grupo
  - Se valorará innovación y creatividad
  - Formato puede ser PDF, PPT, entre otros.
  - Entrega: Martes 24 de mayo.

147

Historia

- Los seres humanos transformamos la realidad que nos rodea con nuestras manos e ingenio:
  - Desde la antigüedad, vimos que para ser más productivos necesitábamos organizarnos ante los objetivos que pretendíamos alcanzar.
  - En la actualidad las empresas tienen aprendida la lección.

150

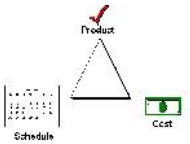
Características de un proyecto

- Existe un objetivo claro.
- Se puede identificar un conjunto de tareas.
- Necesaria la intervención de especialistas.
- Existen limitaciones en los recursos.
- Tiene principio y fin en el tiempo.
- Se requiere un nivel de calidad.
- Se requiere una planificación.

153

Gestión de proyectos: Triángulo de compromisos

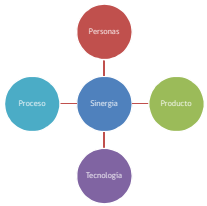
- Se ponen los cimientos del proyecto:
  - Calidad
  - Costos económico
  - Duración del proyecto
- Rápido, barato, bueno
  - Elegir 2



154

Dimensiones de un proyecto

- Se deben potenciar
  - Para maximizar la velocidad de desarrollo



155

Dimensiones de un proyecto: Personas

- Diferencias en productividad 10:1
- Mejoras posibles:
  - Selección del personal
  - Organización del Equipo
  - Motivación
- Otros factores de suceso:
  - Asignación de personas a tareas
  - Desarrollo de la carrera
  - Equilibrio entre lo individual y el equipo

156

Dimensiones de un proyecto: Proceso

- 2 tipos: de Gestión y Técnicos
- Evitar rigidez y burocracia
- No abusar del proceso → Negligencia
- Otros factores de suceso:
  - Control de calidad y bases del desarrollo
  - Gestión de riesgos
  - Atención a los recursos
  - Planificación del ciclo de vida
  - Orientación al cliente

157

Dimensiones de un proyecto: Producto

- La dimensión más tangible
- Definición del tamaño del producto
- Características y requerimientos

158

Dimensiones de un proyecto: Tecnología

- Generalmente, la dimensión menos importante
- Selección del lenguaje, herramientas, etc.
- Valor y costo de reutilizar (código, componentes, etc.)

159

Errores: personas

- Motivación débil
- Personal mediocre
  - Capacidad vs. Experiencia
- Empleados problemáticos incontrolados
- Hazañas
- Agregar personal a un proyecto atrasado
- Oficinas repletas y ruidosas
- Fricciones clientes-desarrolladores

160

Errores: personas

- Expectativas poco realistas
- Ilusiones
- Falta de un promotor efectivo
- Falta de participación de los implicados
- Falta de participación del usuario

161

Errores: proceso

- Planificación excesivamente optimista
- Gestión de riesgos insuficiente
- Fallas de los contratistas
- Planificación insuficiente
- Abandono de la planificación bajo presión
- Pérdida de tiempo en el inicio difuso
- Escatimar en las actividades iniciales

162

Errores: proceso

- Diseño inadecuado
- Escatimar en el control de calidad
- Insuficiente control de la dirección
- Omitir tareas necesarias en la estimación
- Planificar: ponerse al día más adelante
- Programación a destajo

163

Errores: producto

- Exceso de requerimientos
- Cambio de las prestaciones
- Desarrolladores meticulosos
- Tiras y aflojas en la negociación
- Desarrollo orientado a la investigación
- Síndrome de la panacea
- Sobreestimar ahorros por uso de nuevas herramientas o métodos
- Cambiar herramienta a mitad del proyecto

164

El Software como obra humana

- Algunos autores comparan el software a la escritura de libros.
  - Fruto del intelecto,
  - Descripción de realidades y ficciones.
- Cuando el software es grande es como una novela de varios tomos.

165

¿Porqué es difícil desarrollar software ?

- Es complicado explicar los motivos que hacen tan difícil desarrollar software.
- Lo cierto es que muchos proyectos de desarrollo de software fracasan.
- Centraremos el tema mediante:
  - Una estadística realizada sobre 8 proyectos de Software Estadounidenses.
  - Características del Software.
  - Aplicaciones del Software.

167

El Software como obra humana

- Para nosotros será el conjunto de información:
  - capaz de producir en las máquinas el comportamiento deseado, de forma eficaz y eficiente,
  - que los usuarios puedan utilizar el sistema de forma eficiente.
  - Al que los desarrolladores puedan dar mantenimiento de forma eficaz y eficiente.

166

¿Porqué es difícil desarrollar software ?

168

Características del software

- Es inmaterial e invisible
- El comprador lo puede evaluar cuando ya ha sido construido.
- El Software se desarrolla, no se fabrica.
- Es complejo.
  - Sistemas formados por miles de funciones
  - Con interfaces complejas entre ellas
  - Distribuidos, heterogéneos, etc.
- Es excesivamente maleable (cambios en tiempo de desarrollo).

169

Iniciación al proceso de gestión de proyectos

IC14241- Ingeniería de Software
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

Éxito de los Proyectos

- El Chaos Report by Standish Group mostró una importante mejora en las tasas de éxito de Proyectos TI respecto a los últimos 15 años:
  - Tiempo: excesos bajaron a 63% comparado al 222%
  - Costo: excesos decrecieron a un 45% comparado a un 189%
  - Características requeridas: crecieron a 67% comparado al 61%
  - En USA, 78.000 proyectos fueron exitosos vs. 28.000 anterior.
  - 28% de los proyectos de TI fueron exitosos comparados al 16%
- ¿A qué se debe esta mejoría?

171

PMI: Management Body of Knowledge (PMBOK)

- Define una estructura la gestión de proyectos en:
  - Procesos
  - Áreas de conocimiento
- Procesos:
  - **Procesos de gestión de proyectos**: describiendo y organizando el trabajo del proyecto
  - **Procesos orientados al producto**: especificando y construyendo el producto del proyecto

172

PMI: Grupos de procesos

- Se definen los procesos:
  - **Inicio**: Reconoce que un proyecto/fase deben comenzar, comprometiéndose a eso.
  - **Planeación**: Desarrollar y mantener un esquema trabajable para completar la necesidad del negocio del cual el proyecto fue desarrollado.
  - **Ejecución**: Coordinar a las personas y otros recursos para desarrollar el plan.
  - **Control**: Aseguran que los objetivos del proyecto sean cumplidos a través del monitoreo y medición de avance y tomar acción correctiva cuando sea necesario.
  - **Cierre**: Formalizan la aceptación del proyecto/fase y los llevan a una terminación ordenada.

173

PMI: Grupos de procesos

- Cada proceso es descrito:
  - **Entradas (Inputs):** Documentos o ítems documentables sobre los que se actuará.
  - **Herramientas & Técnicas:** Los mecanismos aplicados a las entradas para generar las salidas.
  - **Salidas (Outputs):** Documentos o ítems documentables que corresponden al resultado de un proceso.

174

Declaración de trabajo

- Debe existir una descripción del trabajo requerido para el proyecto.
- Esta declaración fija las "condiciones de borde o límites" (alcance).
- Puede ser usado en el contrato final: *sea cuidadoso, específico y claro.*
- Típicamente realizado después de la aprobación (después del "Vamos")
- Puede tener múltiples versiones:
  - Lista de entregables para un llamado a propuestas
  - Más detallado dentro del *llamado final*
  - Versión obligatoria del contrato.

175

Declaración de trabajo

- Template de declaración de trabajo:
  - **Alcance del Trabajo:** Describir el trabajo a realizar en detalle. Especificar el hardware y el software involucrados y la naturaleza exacta del trabajo.
  - **Localización del Trabajo:** Describir dónde debe ser realizado el trabajo. Especificar la localización del hardware y software y dónde la gente debe realizar el trabajo.
  - **Período de Operación:** Especificar cuándo se espera que el trabajo comience y termine, horas de trabajo, cantidad de horas que se pueden cargar por semana, información relacionada a la programación. *Sección opcional de "remuneración".*
  - **Programación de los Entregables:** Enumerar los productos entregables específicos, describirlos detalladamente y especificar cuándo se deben entregar.

176

Declaración de trabajo

- Template de declaración de trabajo:
  - **Estándares Aplicables**: Especificar cualquier estándar específico a la compañía o industria, que sean relevantes en la realización del trabajo.
  - **Criterios de Aceptación**: Describe cómo la organización del comprador determinará si el trabajo es aceptable.
  - **Requerimientos Especiales**: Especificar cualquier requisito especial tal como certificaciones de hardware o software, grado o experiencia mínima del personal, requisitos de viaje, documentación, pruebas, soporte, etc.

177

Planificación y recursos

ICI4241- Ingeniería de Software
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

Charter del proyecto

- Descripción del proyecto a *alto nivel.*
  - Necesidad del negocio, producto, supuestos
- Generalmente precede a la Declaración de Trabajo.
- Generalmente es un documento conciso.
- *Ver ejemplo pdf.*

178

Charter del proyecto

- Estructura general adaptable:
  - Descripción
    - Necesidad del negocio
    - Objetivos
    - Método o enfoque
  - Alcance general del trabajo
  - Programación & presupuesto general
    - Muy general.
  - Roles & responsabilidades
  - Supuestos

179

El plan

- El plan de proyecto define el trabajo y cómo será hecho.
- Para qué sirve un Plan?
  - Como base de acuerdo de costos y calendarios para el trabajo.
  - Como estructura para organizar la tarea.
  - Como marco para obtener los recursos requeridos.
  - Como registro de lo inicialmente comprometido.

181

Planificar, estimar, programar

- Planificar: Identifica actividades. No hay fechas de inicio/fin.
- Estimar: Determinar el tamaño & duración de las actividades.
- Programar: Agrega fechas de inicio/fin, relaciones y recursos específicos.

182

Bases de la planificación

- Identificar "qué" necesita ser hecho:
  - Estructura de Desglose del Trabajo (Work Breakdown Structure - WBS)
- Identificar "Cuánto" (el tamaño)
  - Técnicas de estimación del esfuerzo
- Identificar las dependencias entre tareas
  - Gráfico de dependencias, diagrama de red
- Estimar la duración total del trabajo a realizar
  - La programación actual

183

Ejemplos de WBS

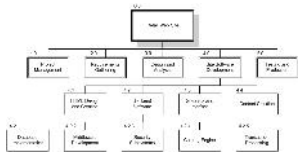
- Lista:

```
0.0 Retail Web Site
1.0 Project Management
2.0 Requirements Gathering
3.0 Analysis & Design
4.0 Site Software Development
    4.1 HTML Design and Creation
    4.2 Backend Software
        4.2.1 Database Implementation
        4.2.2 Middleware Development
        4.2.3 Security Subsystems
        4.2.4 Catalog Engine
        4.2.5 Transaction Processing
    4.3 Graphics and Interface
    4.4 Content Creation
5.0 Testing and Production
```

185

Ejemplos de WBS

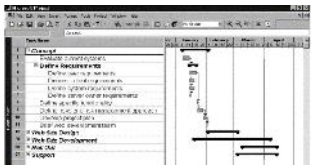
- Gráfico - árbol:



184

Ejemplos de WBS

- Gantt con WBS:



186

Asignación de recursos

- Consiste en asociar a cada una de las tareas, en el proyecto, las personas y materiales necesarios para que éstas se pueda realizar.
- Los recursos humanos constituyen el componente más importante de los proyectos informáticos.
  - Por encima de los recursos físicos (HW e Instalaciones)

187

Sub estimación

- Forzar la planificación por debajo de lo previsible:
  - Condenan al proyecto independientemente de la calidad del personal o de la disponibilidad de herramientas, lenguajes y procesos.
  - Si se comprime la duración o el presupuesto
    - El personal no será eficientemente,
    - No se forzará si ve imposible alcanzar la meta.
  - Peor aún, cuando los retrasos empiecen,
    - Sufrirá la moral y el proyecto probablemente cueste más que de haberse hecho de forma razonable.

188

Fecha de entrega

- Puntos de vista:
  - Del informático:
    - Aplicación es el objetivo de la creación.
    - Proyecto es el medio.
  - Del Usuario y cliente
    - Aplicación: "Es lo que me hace falta para poder alcanzar mis objetivos empresariales"
    - Proyecto: "Un mal trago que hay que pasar"
- Determinar plazos:
  - La negociación.
  - Selección de una alternativa
  - Método empírico de Putnam y Norden

189

Fecha de entrega

- Determinar plazos:
  - La negociación.
  - Selección de una alternativa
  - Desafío personal: Buscar sobre el método empírico de Putnam y Norden.

190

Recursos usuales

- Trabajo
- Lugar de trabajo
- Equipamiento
- Material básico para el desarrollo
- Material fungible

191

Recurso humano

- Es mejor disponer de un equipo pequeño de buenos profesionales
  - Con la gente correcta aún con herramientas, lenguajes y procesos insuficientes, pueden tener éxito.
  - Lo contrario parece imposible.
- Pero:
  - Si confiamos todo a unas pocas personas
  - ¿Qué ocurre si se van?
- Hay que equilibrar el personal.

192

Recurso humano

- Asignación de personal
  - A una tarea podemos asignar una cantidad determinada de personas.
  - La proporción entre cantidad de personas asignadas a una tarea y el esfuerzo, no tienen relaciones lineales.
  - Asignar más gente a un proyecto a mitad de éste no reduce necesariamente su duración.

194

Recurso humano

- Asignación consistente de las tareas
  - Distinta visión del director y los empleados es sobre el trabajo.
  - Asignar las tareas a quienes quieren.
  - Trabajar las asignaciones con los empleados.
  - Hacer una lista de objetivos por trabajador.
  - Ir haciendo reuniones hasta que este clara la asignación.

195

Recurso humano

- Aspecto cognitivo (KAS), la capacidad técnica:
  - Los conocimientos para realizar la tarea
  - La capacidad de realizarla, y
  - La experiencia sobre la materia.
- Aspecto conativo (MAC), la voluntad:
  - La motivación de la persona,
  - El compromiso que asumirá, y
  - La seguridad que tiene en si para realizarla

193

Consideraciones finales

- Costo mínimo de desarrollo
  - En tiempo:
    - Especialistas ya formados en cada área de trabajo
    - Tantos como se pueda y sea necesario
  - En dinero
    - Utilizar el personal necesario para que se lleven a cabo las tareas y
    - que ya conozcan las áreas que se les asignan.

196

Consideraciones finales

- Costo mínimo a largo plazo
  - Pensar en el mantenimiento y otros proyectos
  - Hacer que el personal menos experimentado trabaje en el desarrollo:
    - Dando formación en caso necesario.
  - Hacer que el personal se sienta promocionado.
    - Detectar los objetivos de cada empleado y
    - hacer que cada nuevo proyecto sea un paso en la consecución de éstos.

197

Consideraciones finales

- Conviene recordar, al asignar personas a tareas, que:
  - Que la productividad de los programadores es muy variable, es habitual la relación 1:5.
  - En un estudio se dieron diferencias de 1 a 26 en los niveles de productividad.
  - En las tareas criticas convendrá poner al personal con mayor experiencia y reputación, ya que se espera sean más productivos.

198

Estimación

IC14241 - Ingeniería de Software
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

Estimación: Problemática

- Ejercicio: "Estimar cuanto les va a tomar llegar a casa hoy"
  - En qué se basa?
  - Experiencia no?
  - Como una especie de probabilidad "promedio"
  - Para muchos proyectos de software no hay tal "promedio"
- Las estimaciones exactas son muy poco probables
- Generalmente las estimaciones de software tienen un error de entre un 25-100%

Estimación: Problemática

- La estimación es difícil pero necesaria
- Creada, utilizada o refinada durante:
  - Planeación estratégica
  - Estudio de factibilidad o la *declaración de trabajo*
  - Propuestas
  - Evaluación de vendedores o contratistas
  - Planificación del proyecto (iterativamente)

Estimación: Problemática

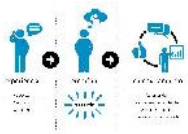
- Proceso básico
  - Estimar el tamaño del producto
  - Estimar el esfuerzo (meses-hombre)
  - Estimar la programación
- No siempre se realizan de manera explícita

Estimación: Métodos utilizados

- Métodos utilizados para la estimación de proyectos:
  - Basados en la experiencia.
  - Basado exclusivamente en los recursos.
  - Método basado exclusivamente en el mercado.
  - Basado en los componentes del producto o en el proceso de desarrollo.
  - Basado en métodos paramétricos/algorítmicos

Estimación: Métodos utilizados

- Métodos basados en la experiencia:
  - Basados en la experiencia en proyectos
  - Un experto estudia las especificaciones y hace su estimación.
  - Problemas:
    - Depende de personas (el experto)
    - Precisión depende de la experiencia
  - Mejor tomar varias opiniones



Estimación: Métodos utilizados

- Métodos basados en los recursos, Parkinson:
  - La estimación consiste en ver de cuanto personal se necesita y durante cuanto tiempo se dispone de él, haciendo esa estimación.
  - En la práctica:
    - "El trabajo se expande hasta consumir todos los recursos disponibles" (Ley de Parkinson).

Estimación: Métodos utilizados

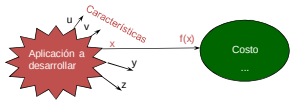
- Métodos basados en el mercado: Precio de venta
  - Lo importante es conseguir el contrato.
  - El precio se fija en función de lo que creemos que está dispuesto a pagar el cliente.
  - Ventaja:
    - Si se usa en conjunción con otros métodos puede ser aceptable para ajustar la oferta.
  - Desventaja
    - Peligro si es el único método utilizado.

Estimación: Métodos utilizados

- Métodos basados en componentes o proceso de desarrollo
  - Top-Down
    - Se ve todo el proyecto, se descompone en grandes bloques o fases.
    - Se estima para cada componente.
  - Bottom-up
    - Se decompone el proyecto en las unidades lo menores posibles.
    - Se estima cada unidad y se calcula el costo total.

Estimación: Métodos utilizados

- Métodos basados en algoritmos
  - Se basan en la utilización de fórmulas, generalmente aplicadas sobre modelos top-down o bottom-up, produciendo una estimación de costo del proyecto.



208

Estimación: Métodos utilizados

- Métodos basados en algoritmos
  - Líneas de Código (LOC)
  - Puntos de Función
  - N° de "círculos" del DFD
  - N° de relaciones de un ER
  - N° de procesos en un diagrama de estructura

209

Estimación de esfuerzo

- Ahora que se conoce el "tamaño", determinar el "esfuerzo" necesario para construirlo
- Varios modelos: empíricos, matemáticos, subjetivos,...
- Expresados en unidades de duración
  - Meses-hombre (o meses-equipo)

210

Estimación de esfuerzo

- Mc' Connell muestra tablas para convertir tamaño a esfuerzo
- Como ocurre con el tamaño, las técnica paramétricas funcionan mejor con datos históricos
- Nuevamente, no son comunes en los proyectos
- Generalmente los pasos de estimación de tamaño y esfuerzo se combinan
  - No es recomendable, pero se hace
- Se utiliza mucho programar por "compromiso"
  - Se le pide al desarrollador "comprometerse" con una estimación (propia)

211

Estimación: aspectos relevantes

- Estimaciones de calidad se necesitan tempranamente pero la información es limitada
- Se tienen datos de estimaciones precisas al final
  - Pero ya no se necesitan o si? y para el próximo proyecto?
- La mejores estimaciones se basan en experiencias pasadas
- Parte política:
  - Se puede anticipar un "recorte" de parte de la administración
- En muchos proyectos hay poco o nada de
  - Cambios tecnológicos
  - Inexistencia de datos históricos
  - Amplia variedad de experiencias y tipos de proyecto

212

Sobre y sub estimación

- Sobre Estimación
  - El proyecto no será financiado
    - Estimaciones conservadores asegurando 100% de éxito pueden significar probabilidad de financiamiento cero
  - Ley de Parkinson
    - El trabajo se expande hasta tomar todo el tiempo disponible
  - Problemas con el alcance y características

213

Planificación

IC14241- Ingeniería de Software
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

Sobre y sub estimación

- Sub Estimación
  - Aspectos de Calidad (ajustar fases clave como las pruebas)
  - Incapacidad de cumplir con las fechas
  - Moral y otros aspectos motivacionales del equipo

214

Objetivo

- Dado que ya hemos identificado:
  - los entregables, fases y tareas
  - los recursos a asignar a cada tarea, y
  - que tarea que se asignan a cada persona.
- Tendremos que crear un calendario de realización, con dos objetivos:
  - que quede claro lo que se espera y para cuando,
  - comprobar que es posible, un día 24 h.

216

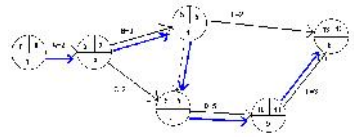
Creación de un calendario aceptable

- Creación del calendario y camino crítico (Grafos-IO):
  - Ordenación de las tareas
  - Creación del calendario
- Revisión y ajuste del calendario:
  - En función del uso de recursos
  - Según las necesidades del usuario
- Aceptación generalizada del plan.

217

Camino critico

- Ordenación de las tareas.
- Cálculo de fechas.



218

Ordenar tareas

- Identificar y documentar dependencias.
  - Restricciones
  - Supuestos
  - Dependencias obligatorias
  - Dependencias discrecionales
  - Dependencias externas

219

Identificar dependencias

- De forma genérica, situándonos en cada tarea, nos planteamos las siguientes cuestiones:
  - ¿Qué debe haberse hecho antes de esto?
  - ¿Qué puede hacerse a la vez?
  - ¿Que debe seguir a lo que hacemos ahora?
- Añadiremos a cada ficha de tarea la lista de tareas precedentes.

220

Restricciones

- Son los factores que limitan las opciones del equipo de desarrollo.
- Son impuestas por el cliente o la dirección de la empresa desarrolladora.
- Ejemplo:
  - Lenguaje de desarrollo,
  - Equipo en que deberá funcionar,
  - personal del que se dispondrá.

221

Supuestos

- Factores que se consideran verdaderos durante la planificación,
- Tienen un grado de riesgo y no cumplirse durante el desarrollo.
- Están directamente relacionados con los riesgos del proyecto, como veremos.
- Ejemplo: Se dispondrá de un computador en casa del cliente.

222

Dependencias obligatorias

- Son las inherentes a la naturaleza del trabajo (aspectos técnicos).
- Se suelen deber a la necesidad de disponer de un entregable que es punto de partida en la tarea.
- Ejemplo:
  - "Prueba del programa XYZ", debe ser precedida de "Codificación del programa XYZ"

223

Dependencias discrecionales

- Las que define el equipo del proyecto.
- Hay que ser cautelosos, pueden condicionar la programación del proyecto en el futuro.
- Se basan en:
  - Las "Mejores Prácticas",
  - Se prefiere una secuencia por que será más fácil de controlar.
  - Limitaciones en la asignación de personal.

224

Dependencias externas

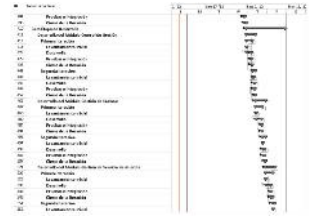
- Vienen impuestas desde el exterior.
- Se refieren a la interdependencia:
  - Con otros proyectos.
  - Con empresas externas o contratos y no podemos ejercer ninguna presión.
- Una actividad no puede comenzar hasta que no dispone de un producto ajeno.
  - Ejemplo: pruebas de programas sobre el hardware.

225

Ejemplo planificación

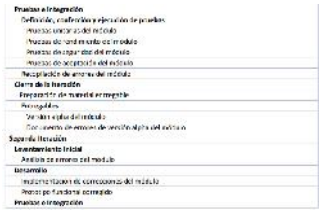
226

Ejemplo planificación



229

Ejemplo planificación



232

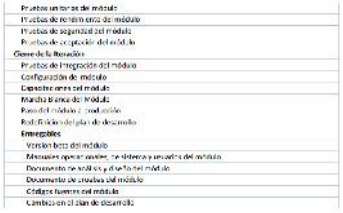
Ejemplo planificación

227

Ejemplo planificación

230

Ejemplo planificación

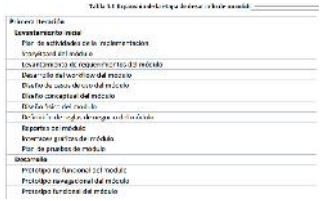


233

Ejemplo planificación

228

Ejemplo planificación



231

Cierre del curso

ICI4241- Ingeniería de Software
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

¿Qué aprendimos en este curso?

- Comprender y aplicar técnicas de ingeniería en todas las etapas del ciclo de vida de software.
- Comparar los modelos de proceso y metodologías apropiadas para el desarrollo de software
- Adaptar el proceso de desarrollo de software a nuevas situaciones y/o nuevos ambientes (sistemas web, sistemas críticos, etc.).
- Estimación y gestión básica de proyectos

235

Ingeniería de Software

Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso
Versión 01- 2016

236