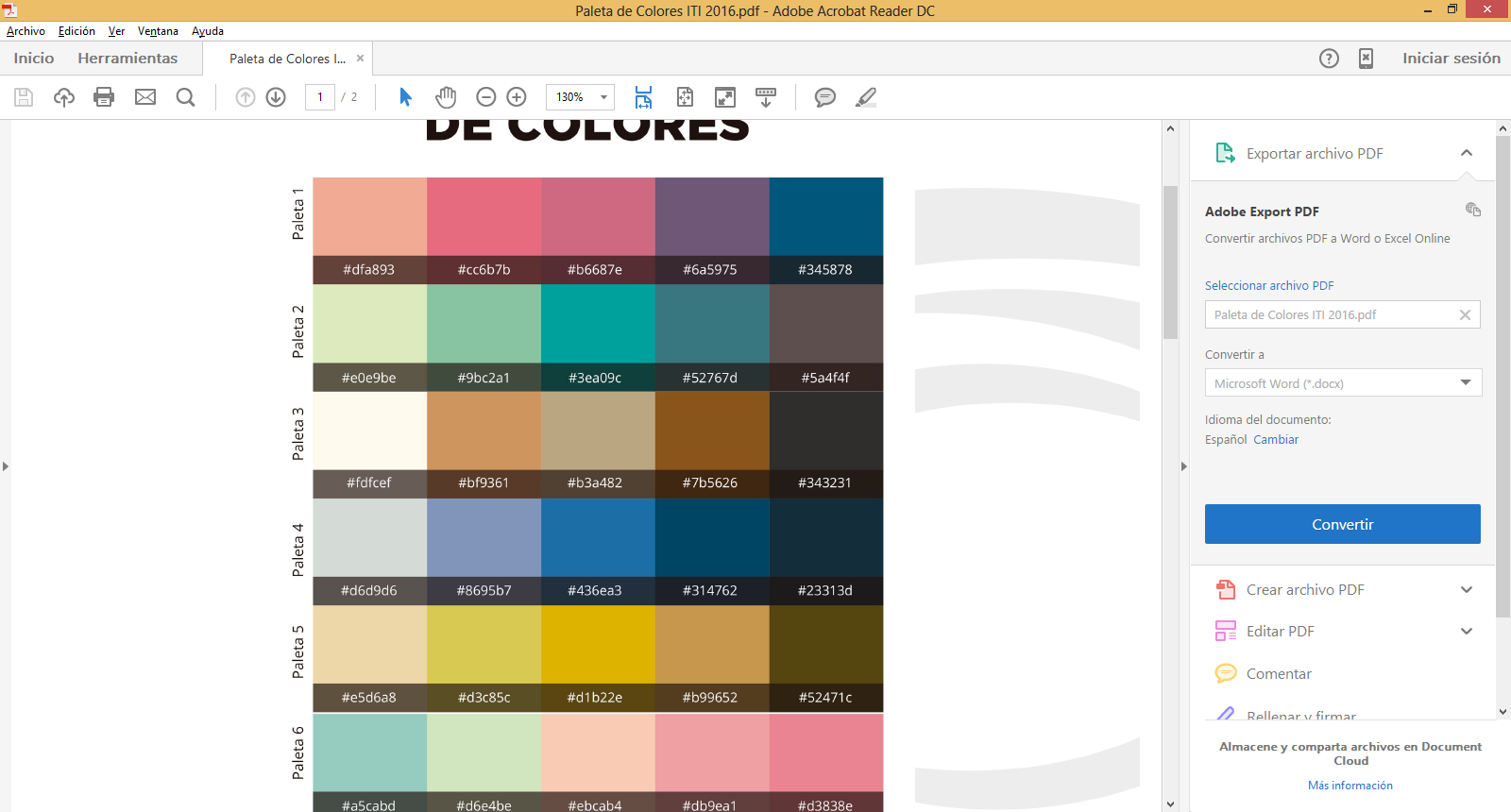
**Tema Procesos de desarrollo de software**



**Contexto del tema 1 (planteamiento inicial)**

|  |
| --- |
| Los retos que enfrentan las empresas dedicadas al desarrollo de software son cada vez más difíciles: clientes con muy altas expectativas con relación al bajo costo y alto rendimiento, tiempos de entrega cada vez más cortos, incorporación de nuevos requerimientos para hacer frente a los cambios del mercado y una feroz competencia a nivel internacional. Todo ello hace que estas compañías busquen formas de hacer estimaciones más reales y encontrar maneras innovadoras de aumentar su productividad.    Este proceso de mejora continua las obliga a buscar balancear tres elementos que son críticos en cualquier proyecto: las personas (sus habilidades, entrenamiento, y motivación), las herramientas y los procesos o métodos que utilizan para llegar a generar resultados. Desafortunadamente, gran parte de los proyectos de software de apoyo a las operaciones de las empresas no alcanzan a generar los resultados esperados, debido principalmente a que se enfocan en uno de estos tres elementos. Deciden contratar personal muy capaz sin tener las herramientas adecuadas, o el control del proceso, o bien, se enfocan en utilizar el método de moda creyendo que el equipo de trabajo tiene las habilidades necesarias para cualquier proyecto.  Las tres dimensiones críticas. Diagrama tomado del libro: Chrissis, M., Konrad, M. y; Shrum, S. (2011). CMMI for Development: Guidelines for Process Integration and Product Improvement. 3ra Ed. EEUU: Pearson Education.  **¿Qué es lo que hace que los proyectos de software sean exitosos?, ¿qué procesos o modelos de desarrollo pueden ser utilizados?, ¿existe algún modelo que pueda ayudar a las empresas a saber qué tan maduros se encuentran sus procesos de desarrollo?** |

**Explicación del tema 1**

**Información del metadato tema 1**

Indique los siguientes datos para el metadato de contenido:

|  |  |
| --- | --- |
| **Nombre del curso** | Procesos de desarrollo de software |
| **Clave del curso** | TI13314 |
| **Nombre del tema** | **Tema 1.** Procesos de desarrollo de software |
| **Descripción** | En este tema revisarás las principales metodologías de desarrollo, la importancia de gestionar un proyecto de software y el modelo CMMI como base para autoevaluar la madurez de los procesos de desarrollo. |
| **Conceptos clave** | CMMI, cascada, RAD, RUP, V-Model, sashimi, espiral, caos |
| **Objetivo** | Identificar las principales metodologías de desarrollo y el modelo CMMI como parte del proceso de mejora. |
| **Tiempo estimado** | 2 horas |
| **Autor** | Ing. Francisco Javier García Enríquez, MTI |
| **Fecha** | 21/10/2015 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Computer crime concept : Stock Photo  El desarrollo de software, (ya sea para sistemas simples o complejos, pequeños, medianos o grandes), sigue un proceso específico que permite mantener un control sobre las actividades que se realicen. Tal y como se establece una estricta secuencia de actividades para construir un edificio, de la misma forma lo hace el proceso de desarrollo de software. Mantener el control sobre las diferentes actividades de implementación de un software se vuelve una imperiosa necesidad para las compañías desarrolladoras, sobre todo cuando se trata de proyectos en los que intervienen cientos de personas que forman parte del equipo de trabajo. El éxito radica en la planeación, coordinación y organización de las variables de cualquier proyecto: costo, tiempo, recursos, alcance y calidad.  Según El-Haik y Shaout (2010), tener un proceso para llevar a cabo el desarrollo de software tiene las siguientes ventajas:   * Mejora la comunicación entre las personas involucradas en el proyecto. * Permite uniformizar los procedimientos en la industria. * Asegura una mejor calidad en el producto * Mejora la productividad ya reduce el tiempo de entrenamiento. * Hace que cálculos del tiempo del ciclo de vida de los nuevos proyectos sean más precisos y, * Existen menos dependencias de las personas y de las compañías.   **1.1 Modelos de desarrollo**  En la industria del software existen algunos modelos del proceso de software que seguramente las conoces. Para que puedas hacer un breve repaso de cada una de ellas, utiliza la siguiente tabla:   |  |  |  |  | | --- | --- | --- | --- | | **Modelo** | **Descripción** | **Ventajas** | **Desventajas** | | **Cascada** | También conocido como el ciclo de vida del desarrollo de un sistema, es un modelo clásico que divide el desarrollo en etapas lineales y secuenciales. | Es un modelo que permite mantener el control administrativo y funcional.  Es efectivo para proyectos pequeños donde los requerimientos son estáticos. | No permite realizar ajustes o revisiones a etapas anteriores.  Una vez en la etapa de pruebas, es muy difícil regresar y hacer cambios sin afectar a la fecha de entrega.  No es posible incorporar cambios una vez avanzado el desarrollo. | | **Sashimi** | Su nombre proviene del parecido que tiene al platillo japonés sashimi  Es similar al modelo cascada, con la diferencia que las etapas se sobreponen dando la oportunidad de revisar la información de cada etapa antes de continuar con la siguiente. | Es posible encontrar a tiempo problemas en las etapas antecesoras. | Es un modelo que no es eficiente en el desarrollo de aplicaciones complejas donde se requieran constantes cambios. | | **V-Model** | Se presume sea una extensión del modelo cascada, sólo que en lugar de ser una secuencia lineal de actividades, este modelo hace la forma V, después de la fase de codificación. | El enfoque hacia la verificación y validación permite que aumentar la calidad del producto. | Requiere de altos costos y muchos recursos para implementarlo.  No se sabe hasta dónde terminan las revisiones o inspecciones. | | **Espiral** | Combina el modelo cascada con el modelo de prototipo, ya que pasa por etapas del ciclo de vida del desarrollo de un sistema apoyándose de la creación de prototipos que deben ser validados por el cliente en un proceso iterativo. | Es simple y permite involucrar al cliente en etapas tempranas del desarrollo.  Es posible ir incrementando su complejidad conforme avanza el desarrollo. | Es costoso y requiere de mucho tiempo. | | **Prototipo** | Se basa en el diseño del sistema utilizando bocetos que representan la funcionalidad y la interacción con el usuario. | Es una forma rápida de comunicar ideas con el usuario.  El usuario puede involucrarse desde el inicio del desarrollo aportando ideas. | Cuando se crean prototipos funcionales, el usuario cree que ya se está avanzando en el desarrollo del sistema.  Pueden llegarse a tomar decisiones a la ligera que dificulten su implementación | | **Caos** | Es un modelo que combina el de cascada y espiral. Se basa en el principio de definir, implementar e integrar las líneas de código, funciones, módulos, el sistema y el proyecto completo desde un inicio. | Se genera un sistema a través de bloques de construcción | Definir todo el proyecto a priori es impráctico. | | **RAD** | Rapid Application Development es un modelo en el que se construye el software por medio de grupos de enfoque en los que se definen los requerimientos, se utilizan prototipos y componentes de reuso | Agiliza la construcción por el ahorro en tiempo al reutilizar componentes.  Tiene un enfoque de calidad al involucrar al usuario.  Es flexible para incorporar cambios. | Existe la posibilidad de crear sistemas que cubren necesidades específicas de forma aislada y se vuelve un problema al tratar de integrarlos. | | **Desarrollo ágil** | Establece un proceso iterativo de colaboración abierta, que permite desarrollar pequeños incrementos del sistema totalmente funcionales en periodos de tiempo muy cortos. | Es posible verificar el avance del sistema.  Puede incorporar cambios en cada iteración.  Permite ir conociendo la aceptación del producto e incorporar funcionalidades innovadoras de una forma más rápida. | Se desconoce la fecha de fin del proyecto completo.  Es difícil hacer una estimación del tamaño software y la cantidad de recursos necesarios para terminar el proyecto. | | **Proceso unificado** | Es un proceso iterativo e incremental que establece como fases: incepción, elaboración, construcción y transición, divididas en series de tiempo iterativas. | Es posible estimar la carga de trabajo a lo largo del proyecto por áreas o responsabilidades. | Requiere de una alta especialización del trabajo.  El equipo necesita conocer el modelo para aplicarlo. |   Cada uno de estos modelos de desarrollo de software, puede ayudar a mantener el control de las actividades del desarrollo de software, sin embargo, por sí solos no aseguran el éxito de los proyectos de software.  **1.2 Importancia de la administración del software**  El software es una industria que ha venido creciendo año con año, pese a las difíciles condiciones macroeconómicas. De acuerdo con IDC- International Data Consulting (2015), el valor del mercado del software para Latinoamérica es de 13,305 mdd en el 2014, registrando un crecimiento anual del 12.8%. Ante este panorama y la cantidad de proyectos de desarrollo de software, existe una verdadera preocupación por los índices de productividad, pues al fin y al cabo, todas las compañías desean hacer más con menos y acaparar el mercado antes que su competencia. Las decisiones que tome la organización sobre la productividad estarán enfocadas en optimizar los recursos, agilizar procesos y mantener la calidad. Jensen (2014), asegura que muchos de los problemas que se enfrentaban las organizaciones dedicadas al desarrollo de software en los años 70’s, persisten el día de hoy: Falta de fiabilidad, entregas tardías, imposibilidad de mantener, desempeño inadecuado y excesos en los presupuestos.  ImageLa siguiente gráfica, muestra los datos comparativos de un estudio llevado a cabo por Standish Group International en 2013, en la que divide el resultado obtenido de los proyectos de desarrollo de software en tres categorías: exitosos, desafiantes, fracaso. Observa que la tasa de éxito en el 2004 no difiere mucho de la registrada en el 2012, menos de la mitad de los proyectos llevados a cabo son exitosos. Aproximadamente la mitad de los proyectos incurrieron en mayores costos y tiempo (Challenged). En comparación con el mismo estudio 8 años después los números siguen siendo desalentadores. Los diferentes métodos para llevar proyectos de software no han podido aumentar la tasa de éxito significativamente, por lo que se puede deducir que existen factores que no se están considerando o no se están cuidando.  Una parte del problema radica en las herramientas que se utilizan para estimar el costo y tiempo de los proyectos de software. Muchas de ellas nacieron en la década de los setenta y son tan efectivas como lo eran en aquel entonces. Por ejemplo, el número de líneas de código que fueron necesarias para generar un producto terminado, ha sido una medida para determinar el tamaño del software, independiente del lenguaje de programación utilizado, lo cual no ha dado como resultado una estimación real, dado los avances en los lenguajes de última generación, además que no toma en cuenta el retrabajo. Afortunadamente el modelo ágil de desarrollo de software ha venido a cambiar la forma en cómo se estima un proyecto de software, utilizando un proceso iterativo, incremental y participativo. Nuevas herramientas o lenguajes como la programación orientada a objetos, han facilitado bastante el desarrollo modular al reutilizar código que fue fabricado en otros proyectos, Barry W Boehmacortando el tiempo y mejorando la productividad.  Barry Boehm. Imagen tomada de <http://csse.usc.edu/new/barry-w-boehm> sólo para fines educativos.  La otra parte del problema, y se podría decir, la que más afecta a la tasa de éxito en los proyectos de software, lo constituye la gestión de las personas. Es decir la habilidad para llevar a cabo el proyecto. En otras palabras, el lenguaje de programación utilizado, la capacidad técnica de los analistas, diseñadores, programadores, testers, e implementadores, y la metodología de desarrollo, afectan al proyecto de manera marginal, sin embargo, las habilidades de la administración de proyectos es determinante en el resultado final del desarrollo.  “Una administración deficiente puede incrementar los costos más rápidamente que cualquier otro factor” (Barry Boehm).  **1.3 Integración de modelos de madurez de capacidades (CMMI)**  Lo que puedes encontrar detrás de cada proyecto exitoso de software es una serie de actividades centradas en la productividad. Un modelo que incorpora las mejores prácticas para gestionar, controlar, medir y monitorear los procesos de software es el modelo de madurez de capacidades. Este modelo no es exclusivo del desarrollo de software, ya que es posible aplicarlo en otras industrias, sin embargo, cuando se trata de proyectos de software, ha demostrado ser una herramienta muy eficaz en alcanzar el éxito. Algunas industrias que han utilizado este modelo son aeroespacial, banca, defensa nacional, automotriz, manufactura y telecomunicaciones. **CMMI** (*Capacity Maturity Model Integration*), permite reconocer las características que deben tener los procesos y provee de algunas líneas de acción que pueden seguir las empresas para crear y perfeccionar sus propios procesos de desarrollo de productos, lo conlleva a minimizar realizar estimaciones realistas de sus proyectos, minimizar sus costos, operativos y mejorar la calidad. El modelo CMMI para desarrollo de software (conocido también como CMMI-DEV) contiene las mejores prácticas que cubren los proyectos de desarrollo de nuevos productos, ingeniería de sistemas, ingeniería de hardware y procesos de soporte para el mantenimiento del software. Este modelo fue desarrollado por el Instituto de Ingeniería de Software (SEI por sus siglas en inglés) en la Universidad de Carnegie Mellon que se encuentra en la ciudad de Pittsburgh. Está basado en los conceptos introducidos por Watts Humphrey en 1987, considerado por muchos como el padre de la Calidad del Software. Actualmente CMMI es soportado por el Instituto CMMI.  Las organizaciones pueden evaluar el nivel de madurez de sus procesos mediante la siguiente tabla, con el propósito de establecer dónde se encuentran y cuál sería su plan para avanzar al siguiente nivel de manera que tengan una forma de mejorar sus propios procesos.   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | **Nivel** | **Característica** | **Problema** | **Comentario** | | 5 | **Optimizado** | Enfocado a la mejora de los procesos | Automatización | ¿Qué podemos automatizar | | 4 | **Cuantitativamente gestionado** | Los procesos se encuentran medidos y controlados | Tecnología | ¿Cuál es la mejor tecnología para cerrar el ciclo de retroalimentación? | | 3 | **Definido** | Procesos definidos por la organización y la proactividad | Medición | Necesita datos para evaluar la tecnología | | 2 | **Gestionado** | Procesos definidos por los proyectos y con frecuencia son reactivos | Entrenamiento | Necesidad de adquirir expertos | | 1 | **Inicial** | Procesos impredecibles, pobremente controlados y reactivos | Planeación | Falta de conocimiento de la gestión | | -1 | **Negligente** | Indiferente | Apatía, indiferencia, desorganización | No necesita el CMMI | | -2 | **Obstructivo** | Contraproducente | Excesivamente rígido y formal |  | | -3 | **Desdeñoso** | Arrogancia | Ajeno al desempeño real |  | | -4 | **Socavado** | Sabotaje | Descreditar el éxito, promueve la apariencia |  |   **Niveles del Proceso de Madurez del Software de Watts Humphrey.** Tabla tomada de: Jensen, R. (2014). Improving Software Development Productivity: Effective Leadership and Quantitative Methods in Software Management. EEUU: Prentice Hall. Sólo para fines educativos.  http://cmmiinstitute.com/sites/all/themes/cmmi/images/header_CMMI_institute_logo_full.pngUtilizar CMMI como parte de la gestión de procesos no es ciertamente la panacea a todos los problemas de productividad, incluso los resultados no son de la noche a la mañana, requieren meses, e incluso en algunos casos, años. Lo importante de este modelo es que ofrece algunas guías para conocer dónde se encuentra la organización y de allí poder mejorar los procesos. Sus recomendaciones no son una camisa de fuerza. Cada organización puede tomar lo que considere mejor para avanzar en su proceso de madurez.  Logo CMMI Institute. Tomado de <http://cmmiinstitute.com/> Sólo para fines educativos.  El Instituto de Ingeniería de Software afirma que el modelo ofrece las siguientes ventajas:   * Es una guía a los esfuerzos de mejora de procesos y ayuda a las organizaciones a establecer y alcanzar sus objetivos. * Proporciona un lenguaje común para la comunicación entre la organización y la evaluación comparativa. * Proporciona un marco de referencia para la integración de los procesos de la organización. * Ayuda a la organización a reconocer acciones que puedan perfeccionar su capacidad de mejora y reconocer las áreas del proceso debe enfocarse. |

**Cierre del tema 1 (aterrizaje del alumno)**

|  |
| --- |
| Llevar a cabo un proyecto de desarrollo de software es una tarea que requiere habilidades no sólo técnicas sino también de administración. Finalmente, en estos proyectos se involucra a personas con diferentes intereses y habilidades, que requieren de la guía del administrador para http://cache3.asset-cache.net/xr/483797406.jpg?v=1&c=IWSAsset&k=3&d=B53F616F4B95E553CB7300DB25D3B6664466C576DD33AC9AC148D79621FB5D9D0511EF71C87E5061E30A760B0D811297resolver conflictos, negociar con el cliente y buscar las mejores alternativas que permitan generar el mejor producto de software. Los fracasos en los proyectos de desarrollo de sistemas se deben principalmente a la falta de estas habilidades que gestionen a tiempo las actividades del equipo de trabajo involucrado en las diferentes etapas del desarrollo del software. Sin duda, seguir el proceso de desarrollo de software propuesto por diferentes metodologías de desarrollo, ayuda a llevar el control de la secuencia de actividades necesarias para generar un producto de calidad, sin embargo, si existe una estimación deficiente del tamaño del software y los recursos necesarios (tiempo, presupuesto y personal), invariablemente afectará el resultado del proyecto.  El modelo de madurez de procesos (CMM) puede ser una herramienta muy útil para conocer el estado actual de los procesos de desarrollo de software que sigue una empresa, y de allí realizar un plan para escalar en los niveles de madurez que ayudarán a la productividad de cada desarrollo, lo que lleva a la eficiencia y a altos estándares de calidad. |

**Recursos de apoyo del tema 1**

|  |
| --- |
| **Videos educativos** (Cada tema debe presentar un video grabado por un experto, puede ser una entrevista, explicación de procesos o contenido) |
| **Videos obligatorios**  Analiza el siguiente video que explica los fundamentos del proceso de desarrollo:  Abitzar. (2014, septiembre 29). *¿Qué es un proceso?* [Archivo de video]. Recuperado de  <https://www.youtube.com/watch?v=PDl8z896s7g>  En ese video podrás encontrar otra explicación de los niveles de madurez de los procesos del modelo CMMI:  Abitzar. (2014, septiembre 17). *Niveles de CMMi – escalonado.* [Archivo de Video]. Recuperado de <https://www.youtube.com/watch?v=P5tF-MRTweA>  **Videos recomendados**  En este video encontrarás una breve explicación sobre diferentes modelos de proceso de desarrollo de software.  ByTDJ. (2012, mayo 27). *Modelos de Desarrollo de Software.* [Archivo de video]. Recuperado de <https://www.youtube.com/watch?v=0eiavKLGS9E>  En el siguiente video podrás revisar las características generales de CMMI.  CiberSur. (2011, junio 24). *Tendencias CMMI.* [Archivo de video]. Recuperado de: <https://www.youtube.com/watch?v=oN6uAFzFkt0>  Mediante los siguientes videos podrás encontrar una breve explicación sobre los modelos de desarrollo de software:   * Udacity. (2015, febrero 23). *The Software Crisis - Georgia Tech - Software Development Process.* [Archivo de video]. Recuperado de: <https://www.youtube.com/watch?v=0b5vp4Z2PKE> * Udacity. (2015, febrero 23). *Software Process Model Introduction - Georgia Tech - Software Development Process.* [Archivo de video]. Recuperado de <https://www.youtube.com/watch?v=laSrDtYtkXU> * Udacity. (2015, febrero 23). *Waterfall Model.* [Archivo de video]. Recuperado de <https://www.youtube.com/watch?v=5A5XCuWMG4o> * Udacity. (2015, febrero 23). *Spiral Model.* [Archivo de video]. Recuperado de <https://www.youtube.com/watch?v=mp22SDTnsQQ> * Udacity. (2015, febrero 23). *Evolutionary prototyping.* [Archivo de video]. Recuperado de <https://www.youtube.com/watch?v=bAEnaGG8Otc> |
| **Lecturas: artículos, recursos educativos abiertos** (Incluya al menos tres lecturas que permitan al participante tener mayor conocimiento del tema). |
| **Lecturas obligatorias**  Revisa los principales modelos del proceso de desarrollo de software mediante el siguiente artículo:  Letelier, P. (2003). *Proceso de Desarrollo de Software.* Recuperado de <http://ldc.usb.ve/~abianc/materias/ci4712/ProcesoSW-Letelier.pdf>  Revisa los conceptos fundamentales en los que se basa al modelo CMMI, con el siguiente artículo:  Microsoft. (s.f.). *Background to CMMI.* Recuperado de <https://msdn.microsoft.com/en-us/library/ee461556.aspx>  **Lecturas recomendadas**  Revisa los principales procesos de desarrollo de software a través de la siguiente información:  Universidad Carlos III. Madrid. (2010). *El proceso de desarrollo de software.* Recuperado de <http://ocw.uc3m.es/ingenieria-informatica/diseno-de-software-avanzado/material-de-clase-1/01-El_Proceso_de_Desarrollo_de_Software.pdf>  Si deseas conocer los principales obstáculos de la implementación del CMMI, te recomendamos el siguiente artículo:  Palacios, H. y Porcell, N. (2012). Obstáculos al implantar el modelo CMMI. Revista Escuela de Administración de Negocios, (72) 110-127. Recuperado de <http://www.redalyc.org/articulo.oa?id=20626818013>  Si deseas conocer sobre una caso real en la implementación del CMMI, te sugerimos el siguiente artículo: Ramos, F. (2010). *Implantación de CMMi nivel de madurez 2 en una PYME*. Recuperado de <http://www.redalyc.org/pdf/922/92218768003.pdf> |

**Checkpoint 1**

|  |
| --- |
| **Asegúrate de poder:**   * Identificar las ventajas y desventajas de los modelos del proceso de software para poder utilizarlos en las situaciones adecuadas. * Reconocer el papel que tiene la gestión en los proyectos de software para asegurar el éxito. * Conocer el modelo CMMI-DEV como base para hacer una autoevaluación de nivel de madurez del proceso de desarrollo de software. |

**Glosario del tema 1**

|  |
| --- |
| Ninguno |

**Referencias bibliográficas de la explicación del tema 1**

|  |
| --- |
| El-Haik, B. y Shaout, A. (2010). *Software Design for Six Sigma: A Roadmap for Excellence.*  EE. UU: John Wiley & Sons.  IDC. (2015). *El mercado de software creció en 2014 pese a las condiciones macroeconómicas: IDC*. Recuperado de <http://ar.idclatin.com/releases/news.aspx?id=1914>  Jensen, R. (2014). *Improving Software Development Productivity: Effective Leadership and Quantitative Methods in Software Management*. EE. UU: Prentice Hall. |

**Notas de enseñanza para el maestro/tutor 1**

|  |
| --- |
| Existen otras metodologías de programación como Top-down / Bottom-Up, eXtreme Process (XP), V-Model XT, Joint Application Development, Model Driven Engineering, Iterative Development Process, LEAN method (Agile), Wheel and Spoke Model, Constructionist Design Methodology. Se deja al criterio del profesor abordarlas en clase como parte de este tema.  La última actualización del modelo CMMI fue en 2010, y sigue siendo vigente para que las empresas de desarrollo puedan autoevaluar sus procesos de desarrollo. Es un modelo que se utiliza en empresas de más de 2000 empleados, con procesos de desarrollo complejos.  Los niveles de madurez de los procesos negativos (-1 Negligente, -2 Obstructivo, -3 Desdeñoso y -4 Socavado) fueron propuestos por Watts Humphrey, sin embargo, en el modelo actual del CMMI sólo aparecen los niveles positivos. |

**Actividades y/o Tareas del tema 1**

Utilizar formato que viene en la parte de Anexo I

**Tema PSP (Personal Software Process)**

**Contexto del tema 2 (planteamiento inicial)**

|  |
| --- |
| business morning : Stock PhotoGerardo es un ingeniero de software que trabaja en una empresa de renombre que desarrolla sistemas para empresas. Ha participado en algunos proyectos importantes como ingeniero programador, lo que le permite ir construyendo experiencia y reputación como un elemento clave en los proyectos importantes de la empresa. En reuniones con los administradores de los proyectos, es muy común que le pidan estimar el tiempo que requiere para construir un componente de software, sin embargo, sus respuestas no han sido del todo certeras. En ocasiones solicita tiempo de forma excesiva, lo que ha provocado que las estimaciones del costo del proyecto sean muy elevadas y el cliente decide mejor contratar a otra compañía. En otros casos, Gerardo ha ofrecido tiempos de entrega muy apretados, que le impiden entregar con calidad, lo que ha provocado que el cliente utilice un software lleno de defectos, lo que ha provocado mayores costos en las correcciones y mantenimientos necesarios al código.  Ya sea por sobreestimación o subestimación, los proyectos de software se ven impactados en sus costos. Ante tal situación, Gerardo se pregunta si existirá alguna forma en la que pueda medir mejor el proceso que sigue para construir un software, que genere una base más certera para realizar estimaciones reales.  **¿Qué debe hacer Gerardo?, ¿existirá algún mecanismo que le permita saber con mayor exactitud su productividad?, ¿es posible ofrecer una estimación más precisa para mantener los costos en un rango aceptable para el cliente y a la vez que no demerite en la calidad del software?** |

**Explicación del tema 2**

**Información del metadato tema 2**

Indique los siguientes datos para el metadato de contenido:

|  |  |
| --- | --- |
| **Nombre del curso** | Procesos de desarrollo de software |
| **Clave del curso** | TI13314 |
| **Nombre del tema** | **Tema 2.** PSP (Personal Software Process) |
| **Descripción** | En este tema revisarás las fases del proceso de software personal (PSP) que ayudan a tener una mejor base para reconocer la productividad individual. |
| **Conceptos clave** | PSP, Personal Software Process, Fases del PSP, proceso, planeación, desarrollo, postmortem. |
| **Objetivo** | Utilizar el modelo del PSP como parte de una metodología para calcular la productividad de los desarrollos de software. |
| **Tiempo estimado** | 2 horas |
| **Autor** | Ing. Francisco Javier García Enríquez, MTI |
| **Fecha** | 21/10/2015 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **2.1. Conceptos básicos del PSP**  PSP (Personal Software Process) es un modelo del proceso de desarrollo de software desarrollado por Watts Humphrey en 1997. Watts ideó este modelo con el propósito que pudiera ser utilizado por un ingeniero de software en proyectos de desarrollos pequeños o en componentes que pudieran ser construidos por una sola persona. Este modelo abarca las tareas de planificación, desarrollo y medición del software, de manera que el ingeniero pueda aplicarlo para mejorar sus propias habilidades de ingeniería de software, controlar su trabajo, cumplir con sus compromisos, medir su productividad y la calidad de sus productos, a través de un marco de referencia estructurado que incluye directrices y procedimientos de desarrollo de software.  http://www.sdtimes.com/images/watts.jpg  PSP es fundamentalmente un proceso de mejora, que, si se utiliza de forma estructurada y disciplinada, permite obtener resultados predecibles. Los ingenieros pueden apoyarse en el PSP para recopilar requerimientos precisos, diseñar y desarrollar programas limpios de errores, documentar su trabajo y mejorar la mantenibilidad del software existente. En opinión de Humphrey (2005), el PSP no una respuesta mágica a todos los problemas que enfrenta la ingeniería del software, pero sí ayuda a reconocer dónde se encuentran y qué hacer para mejorar.  Watts Humphrey (1947-2010) Recuperado de <http://sdtimes.com/watts-humphreys-road-to-excellence-is-worth-following/> Sólo para fines educativos.  A continuación revisarás algunos conceptos en los que se basa el modelo PSP.  **Proceso de software**  Un proceso es una secuencia de pasos para obtener un resultado esperado. El objetivo que persigue al documentar un proceso es obtener siempre el mismo resultado.  Según Humphrey (2005), cuando un proceso de software se encuentra adecuadamente diseñado, se convierte en una guía para realizar el trabajo, mejora la planeación y coordinación, comunicación y control de actividades involucradas con el resultado esperado. En contraste, cuando un equipo de trabajo no sigue un proceso definido, es parecido a tener un equipo de fútbol con jugadores que saben béisbol, básquetbol, tenis, incluso podrían ser los mejores jugadores en su disciplina, pero perderían los partidos porque ningún de ellos conoce cómo se debe patear un balón.  **Flujo del proceso de software**  El modelo PSP propone que todo ingeniero de software considere este flujo de actividades cada vez que intervenga en un proyecto de desarrollo, independientemente si la tarea asignada es simple o compleja. Esto le permitirá mantener una estructura que a la larga le ahorrará tiempo en la planeación y ejecución de las tareas, y sus entregables se harán en tiempo y forma. Este flujo de actividades no debe ser una camisa de fuerza que entorpezca el trabajo de ingeniero, sino simplemente una guía, con la seguridad que ha sido probada con éxito por ingenieros de software. PSP da la libertad de cambiar según se considere o se adapte mejor en cada caso.  “Cuando tu proceso no funcione, cámbialo hasta que lo haga” Humphrey, 2005.      **Flujo del proceso del PSP.** Tomado del libro Humphrey, W. (2005). *PSP(SM): A Self-Improvement Process for Software Engineers*. EE. UU: Pearson Education. Sólo para fines educativos.  Observa que el proceso PSP lo podrías dividir en tres partes generales: Planeación, Desarrollo y Postmortem. Más adelante en este mismo tema revisarás los elementos que lo componen.  **Marco de referencia para la medición**  Humphrey (2005) afirma que todo proceso definido debe establecer métricas con el principal objetivo de recabar información necesaria para conocer cuánto tiempo se lleva realizar una tarea, obtener el tamaño del producto resultante, contabilizar la cantidad de errores introducidos y detectados a tiempo, en otras palabras, medir un proceso te permite analizarlo, entender sus fortalezas y debilidades y, lo que es más importante, mejorarlo.  **Fundamento para la mejora**  Definir una secuencia de pasos para obtener un resultado te permite analizar cada paso de forma individual. Si no obtuviste el resultado esperado, quiere decir que un paso o la secuencia de pasos no fue la adecuada. Es parecido a lo que sucede en la repostería. Una persona podría seguir al pie de la letra una receta para hacer un pastel, pero si al final el pastel no tiene buen sabor, pudo deberse a que no utilizó los ingredientes adecuados o no siguió el procedimiento como lo marca la receta. Lo que obliga a la persona a estudiar paso a paso toda la receta y determinar qué fue lo que omitió o realizó de forma incorrecta, de tal suerte que la próxima vez que haga el pastel mejore el resultado. En ingeniería de software sucede lo mismo, sin embargo, el mayor problema es que no se analiza detenidamente el proceso de software para saber dónde necesita hacer ajustes y mejorarlo.    **2.2 Elementos del PSP**  **Fase de Planeación**  El proceso PSP establece una fase de planeación, que permite general y documentar el plan que guiará el proceso de software. En un principio esta fase de planeación puede llegar a considerarse como una pérdida de tiempo, sobre todo cuando la tarea asignada es muy simple, sin embargo, el hábito de dedicarle algunos minutos para analizar el problema y decidir cómo abordarlo, resultará en grandes beneficios posteriores, ahorrando tiempo en recopilar la información necesaria, abordar de mejor manera la tarea, y asegurar los entregables. Si el ingeniero de software planea en las tareas pequeñas mejorará sus planes en las tareas complejas.  **Proceso de la fase de planeación:**   |  |  |  | | --- | --- | --- | | **Propósito** | | Guiar al proceso de planeación del PSP | | **Criterios de entrada** | | Descripción del problema.  Resumen del plan del proyecto.  Registro del tiempo. | | **Paso** | **Actividades** | Descripción. | | 1 | Requerimientos del programa | Genere y obtenga los requerimientos del programa.  Asegúrese que los requerimientos sean claros y libres de ambigüedades.  Resuelva cualquier pregunta que le surge. | | 2 | Estimación de recursos | Realice su mejor estimación del tiempo que requiere para desarrollar este programa. | | **Criterios de Salida** | | Requerimientos documentados.  Reporte del plan de proyecto completado con la estimación del tiempo de desarrollo.  Registro del tiempo completado. |   Tabla tomada de Humphrey, W. (2005). *PSP(SM): A Self-Improvement Process for Software Engineers*. EEUU: Pearson Education. Sólo para fines educativos.  **Fase de desarrollo**  Las actividades de la fase de desarrollo incluyen el diseño del programa, la construcción o codificación, compilación y pruebas unitarias. Varias de estas actividades suceden al mismo tiempo por lo que no debes considerarlas como secuenciales. Por ejemplo, una primera compilación del programa sirve también como prueba a nivel de sintaxis de los comandos y parámetros utilizados, incluso algunos compiladores utilizados en las aplicaciones IDE suelen alertar sobre errores de lógica, como en el caso de *visual studio*. El objetivo que busca PSP al detallar las actividades del desarrollo es generar el hábito del análisis del problema antes de iniciar la programación. Esto permitirá que el ingeniero encuentre la mejor forma de solucionarlo, minimizando la cantidad de defectos que pueda introducir en el código, haciendo así que las pruebas sean más ágiles y efectivas.  **Proceso de la fase de desarrollo:**   |  |  |  | | --- | --- | --- | | **Propósito** | | Guiar en el proceso de desarrollo de programas pequeños. | | **Criterios de entrada** | | Declaración de requerimientos.  Formulario del Resumen del Plan de Proyecto con una estimación del tiempo de desarrollo.  Registros de los defectos y el tiempo requerido.  Estándar del tipo de defectos. | | **Paso** | **Actividades** |  | | 1 | Diseño | Revisar los requisitos y producir un diseño para cumplirlos.  Registrar en el registro de Tiempos cualquier defecto encontrado en la definición de requerimientos.  Registrar el tiempo en el registro de Tiempos. | | 2 | Codificación | Implementar el diseño siguiendo el Estándar de codificación.  Utilizar el registro de Defectos cualquier defecto de requisito o de diseño encontrado.  Documentar el tiempo en el registro de tiempos. | | 3 | Compilación | Compilar el programa hasta que esté libre de errores de  Compilación.  Reparar todos los defectos encontrados.  Registrar defectos en registro de defectos.  Registrar el tiempo en el Log de Registro de Tiempos. | | 4 | Probar | Probar hasta que todas las pruebas se ejecuten sin errores.  Reparar todos los defectos encontrados.  Documentar los defectos en el registro de defectos.  Registrar el tiempo en el Log de Registro de Tiempos. | | **Criterios de Salida** | | Un programa probado completamente de acuerdo al Estándar de Codificación  − Registros de los tiempos y defectos de Tiempos y Defectos completados |   Tabla tomada de Humphrey, W. (2005). *PSP(SM): A Self-Improvement Process for Software Engineers*. EE. UU: Pearson Education. Sólo para fines educativos.  **Fase Postmortem**    La parte final del PSP establece realizar un reporte postmortem, que se utiliza para documentar detalles importantes de la tarea realizada, qué problemas surgieron y como se solucionaron. Es un momento que el ingeniero de software debe dedicar para documentar las lecciones aprendidas. Este reporte es necesario que se realice de manera inmediata, justo cuando se ha entregado el producto. Si se deja pasar tiempo, es muy probable que se olviden detalles importantes que puedan ser valiosos para otras tareas o proyectos, y podría exigir más tiempo tratando de encontrar la información de sustento. Esta actividad sólo requiere de algunos minutos de tiempo cuando se tienen frescos los detalles importantes.  Al igual que la fase de planeación, realizar el reporte postmortem exige disciplina por parte del ingeniero de software y podría considerarla como una actividad tediosa, sin embargo, es un hábito que rendirá frutos en el futuro.  **Proceso de la fase de Postmortem**   |  |  |  | | --- | --- | --- | | **Propósito** | | Guiar al proceso de postmortem del PSP. | | **Criterios de entrada** | | Descripción del problema y declaración de los requerimientos  Formulario del Resumen del Plan de Proyecto con información del tamaño del programa.  Registros de los defectos y el tiempo requerido.  Programa completado y probado. | | **Paso** | **Actividades** | Descripción. | | 1 | Registro de Defectos | Revise el Resumen del Plan de proyecto y verifique que todos los defectos encontrados en cada fase del proyecto fueron registrados.  Registre los defectos que fueron omitidos. | | 2 | Consistencia de información de defectos | Revisar que la información de cada defecto registrado sea precisa y completa.  Verificar que la cantidad de defectos introducidos y removidos por cada fase son razonables y correctos.  Corregir la información incorrecta o de defectos omitidos. | | 3 | Tiempo | Revisar posibles errores u omisiones en el registro del tiempo.  Corregir cualquier información incompleta o incorrecta sobre el tiempo registrado. | | **Criterios de Salida** | | Un programa probado de forma completa.  Formulario del Resumen del plan de proyecto completado.  Registros del tiempo y defectos. |   Tabla tomada de Humphrey, W. (2005). *PSP(SM): A Self-Improvement Process for Software Engineers*. EE. UU: Pearson Education. Sólo para fines educativos.  **2.3 Medidas del PSP**  El PSP sugiere 2 medidas del proceso de software: el tiempo transcurrido por fase y los defectos encontrados por fase. El objetivo será que ayuden a planear y administrar mejor los proyectos de software, al determinar la productividad y la calidad del trabajo realizado. Ambas acciones requerirán de tiempo para realizar mediciones precisas, sin embargo, es posible que el ingeniero de software se apoye de algunas herramientas que registran el tiempo de una forma muy simple. El ingeniero de software deberá llevar la contabilidad de minutos transcurridos en cada una de las fases y la cantidad de defectos que aparecen en el código cada vez que realice un cambio para solucionarlo, considerando que un cambio puede ser tan simple como agregar un punto y coma al final de una línea o hasta agregar varias instrucciones nuevas. Es importante que no se consideren como defectos el número de errores o advertencias arrojados en la compilación, ya que en muchos casos están relacionados a un solo error.    **Registro del tiempo**  A continuación, se muestra una forma simple para registrar el tiempo por fases:  **Haz clic en campo del formato para conocer más detalle**    Tabla tomada de Humphrey, W. (2005). *PSP(SM): A Self-Improvement Process for Software Engineers*. EE. UU: Pearson Education. Sólo para fines educativos.  **Campos del formato:**   |  |  | | --- | --- | | **Apartado** | **Descripción** | | **Encabezado** | Introduzca los datos generales que le ayuden a identificar el proyecto o tarea asignada: Autor, fecha de elaboración, nombre del supervisor o Project manager, número de programa, el lenguaje de programación. | | **Proyecto** | Nombre del programa o número identificador. | | **Fase** | Fase en la que se está trabajando: Planeación, Desarrollo, Postmortem.  Es recomendable detallar la parte del desarrollo en la que se está trabajando. Si consideras que el tiempo de compilación está siendo un problema, puedes considerarla como una fase. | | **Fecha y hora de inicio** | Fecha y hora en la que empezó a realizar el proceso de la actividad. | | **Tiempo de interrupción** | Se registra cualquier tiempo en el que se interrumpió la actividad en la que se estaba trabajando (llamadas telefónicas, preguntas, asesorías). Si existen varias interrupciones, contabilice el tiempo total. Conviene agregar comentarios de la razón de interrupción.  En caso de que este tiempo sea mínimo se puede ignorar. | | **Fecha y hora fin de la actividad** | Ingresa la fecha y hora en la que se terminó la actividad. | | **Tiempo Diferencial** | Es el tiempo que se obtiene al restar el tiempo empleado menos el tiempo de interrupción. | | **Comentarios** | Agrega los comentarios que sea útiles para recordar cualquier circunstancia inusual en relación a la actividad. |   El registro del tiempo requiere disciplina, por lo que es conveniente hacer uso de cronómetros que puedas activarlos fácilmente. Esto permitirá una medición precisa del tiempo, y es posible que te des cuenta de la cantidad de veces que eres interrumpido y los motivos, ello te permitirá afrontar mejor el problema que te impide ser más productivo.  **Registro de defectos**  El registro de defectos tiene información que servirá para identificar la cantidad de defectos detectados, el tiempo utilizado en la solución y el tipo de defectos. A continuación puedes observar un ejemplo del registro de defectos.  **Haz clic en campo del formato para conocer más detalle**  http://my.safaribooksonline.com/getfile?item=cmNhbTAvNzk4ZDNwaS9nZWE3MDAvOXJzZzIzdDE1aWgydF9hbHBjMDdhYnQvMGcuc2Zp  **Campos del formato:**   |  |  | | --- | --- | | **Apartado** | **Descripción** | | **Encabezado** | Se incluye los datos generales del programa, fecha de elaboración, lenguaje de programación y nombre del programador. | | **Proyecto** | Cuando en el registro de defectos intervienen varios archivos de código, se incluye el número identificador del programa. | | **Fecha** | Fecha en la que se detectó el defecto. | | **Número** | Número secuencia del defecto. | | **Tipo** | Es posible utilizar una tipificación de defectos. Es importante tratar de homogeneizar la tipificación cuando se trata de un grupo de programadores. | | **Fase de inyección** | Se ingresa la fase en donde el defecto fue introducido. Utiliza tu mejor juicio y determina de dónde proviene el error. | | **Fase de remoción** | Se ingresa la fase donde fue solucionado. Normalmente es la misma fase en la que fue encontrado. | | **Tiempo de solución** | Tiempo que te tomó encontrarlo y solucionar el defecto. | | **Solución referenciada** | Si el defecto encontrado tiene que ver con la solución incompleta de otro defecto, se coloca el número de defecto al que hace referencia. | | **Descripción** | Se incluye una breve descripción del defecto, sólo para recordar el error después. |   Estos formatos pueden ser modificados según se crea conveniente, para facilitar el trabajo del registro de la información que servirá para reconocer la productividad actual y realizar mejores estimaciones en futuros proyectos. |

**Cierre del tema 2 (aterrizaje del alumno)**

|  |
| --- |
| http://cache1.asset-cache.net/xr/179216716.jpg?v=1&c=IWSAsset&k=3&d=B53F616F4B95E5532451B219896FA49FF00FEA16D1007B333EE4B4BB63FC630E0511EF71C87E5061E30A760B0D811297El proceso PSP exige una disciplina de documentación a los programadores de software que les permite analizar su productividad en un proyecto particular y es la base para implementar un proceso de mejora continua personal. Los formatos de registro sugeridos por PSP pueden ser adaptados por el mismo programador, aunque se sugiere que al principio se utilicen sin cambios, mientras se acostumbra a documentar su propio trabajo. Es importante aclarar que PSP no debe ser tomado como un método de evaluación del personal, ya que, a diferencia de otras industrias, la construcción del software nunca es una tarea repetitiva, cada proyecto exige retos diferentes por lo que no sería posible comparar el esfuerzo o la productividad entre proyectos diferentes.  Lo mismo sucede con el registro de defectos propuesto por PSP. La cantidad de defectos detectados a tiempo no deben ser una métrica comparativa de la calidad del producto en diferentes proyectos, es decir, si el programador ha registrado una buena cantidad de defectos en un proyecto y otra una cantidad menor en otro, no significa que el primero tenga mayor calidad que el segundo. Probablemente en el segundo proyecto no se tuvo el tiempo suficiente para realizar pruebas, detectándose menos defectos, o se reutilizó código limpio que había sido implementado en otros proyectos. En cualquier caso, mantener un registro del trabajo del programador será una actividad muy útil sobre todo como base para realizar estimaciones del tamaño del software y el esfuerzo que implica. Estas estimaciones serían una herramienta para planear futuros proyectos. |

**Recursos de apoyo del tema 2**

|  |
| --- |
| **Videos educativos** (Cada tema debe presentar un video grabado por un experto, puede ser una entrevista, explicación de procesos o contenido) |
| **Videos obligatorios:**   * Revisa una conferencia del proceso del PSP a través del siguiente video:   Jeitson. (2014, marzo 1). *PSP Proceso*. [Archivo de video]. Recuperado de  <https://www.youtube.com/watch?v=d9PtsqrzG0o>  **Videos recomendados**   * Si deseas revisar los fundamentos del PSP, te sugerimos el siguiente video:   Pareja, J. (2014, mayo 13). *Tips de Personal Software Process - Tomando Disciplina PSP0*. [Archivo de video]. Recuperado de <https://www.youtube.com/watch?v=-UHUigqPo1U>   * Si deseas obtener una explicación general del modelo PSP, te sugerimos revisar el siguiente video:   Bartley, M. (2014, diciembre 3). *The SEI Personal Software Process: Improve Yourself.* [Archivo de video]. Recuperado de  <https://www.youtube.com/watch?v=nWScAkGn-zw> |
| **Lecturas: artículos, recursos educativos abiertos** (Incluya al menos tres lecturas que permitan al participante tener mayor conocimiento del tema). |
| **Lecturas obligatorias**   * Revisa los conceptos generales del modelo PSP en el siguiente trabajo de tesis: Pelaez, J. (2003). *PSP 0 y PSP0.1.* Recuperado de   <http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/pelaez_r_jj/capitulo3.pdf>  **Lecturas recomendadas:**   * Si te interesa tener una certificación en PSP, te recomendamos revisar la siguiente información:   SEI (2015). *SEI Certification.* Recuperado de  <http://www.sei.cmu.edu/certification/>   * Si estás interesado en complementar la información sobre PSP, te recomendamos revisar el PSP Body of Knowledge,   Pomeroy-Huff, M, Cannon, R., Chick, T., Mullaney, J. y Nichols, W. (2010). *The Personal Software Process (PSP) Body of Knowledge, Version 2.0.* Recuperado de  <http://repository.cmu.edu/cgi/viewcontent.cgi?article=1034&context=sei>   * Si deseas conocer uno de los primeros trabajos de Watts Humphrey, padre del PSP, te recomendamos el siguiente artículo:   Humphrey, W. (1994). *Personal Software Process.* Recuperado de  <http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.206.1410> |

**Checkpoint 2**

|  |
| --- |
| **Asegúrate de poder:**   * Reconocer las ventajas de utilizar el PSP como un modelo base para realizar estimaciones reales del esfuerzo necesario en un proyecto de software. * Reconocer las fases que propone el PSP para mejorar la productividad del desarrollo de software. * Utilizar los formatos sugeridos por el PSP para registrar el tiempo y los defectos detectados, con la finalidad recabar información para mejorar el proceso de desarrollo. |

**Glosario del tema 2**

|  |
| --- |
| **Compilar**: se refiere a la acción de tomar un código fuente de un programa y pasarlo a un código que pueda ser interpretado por la computadora.  **IDE (Integrated Deveoment Envionmental**) es un programa que sirve de ayuda a los programadores al proporcionales algunas herramientas que le facilitarán escribir y estructurar mejor el código. Algunos ejemplos son Visual Studio, Netbeans, Eclipse. |

**Referencias bibliográficas de la explicación del tema 2**

|  |
| --- |
| Humphrey, W. (2005). PSP(SM): A Self-Improvement Process for Software Engineers. EEUU: Pearson Education. |

**Notas de enseñanza para el maestro/tutor 2**

|  |
| --- |
| Las fases descritas en este tema constituyen la línea base sobre la que se fundamente el proceso PSP o lo que también se le llama PSP0.  Existen otros niveles del PSP que servirán más adelante para realizar estimaciones del tamaño del software y en la formación de equipos de software. Por lo pronto es importante sentar las bases del proceso.  Los formatos incluidos en este tema fueron tomados directamente del libro de apoyo del curso. Son únicamente una guía, aunque es muy conveniente que los alumnos los utilicen tal y como los propone Watts Humphrey antes de hacerles alguna modificación. |

**Actividades y/o Tareas del tema 2**

Utilizar formato que viene en la parte de Anexo I

**Tema TSP (Team Software Process)**

**Contexto del tema 3 (planteamiento inicial)**

|  |
| --- |
| Patricia es ingeniero en software, ha estado trabajando para una empresa de desarrollo de software. Desde su contratación hace ya más de un año, ha utilizado el modelo PSP y ha visto cómo su productividad ha ido en aumento. Ahora ella es parte del mayor reto de su carrera al incorporarse al proyecto más importante que lleva su empresa. She is a team leader. : Stock PhotoEl jefe de Patricia le ha informado algunos detalles del proyecto, por ejemplo, que se ha involucrado a un grupo de más de cien personas con diferentes habilidades y niveles de experiencia en proyectos de desarrollo de software, por lo que será todo un reto coordinar y controlar. Las fechas de entrega son una parte fundamental de la aceptación del usuario, así que no pueden darse el lujo de permitir ningún retraso. El mismo dueño de la empresa desarrolladora de software ha puesto toda su confianza en este equipo de trabajo para entregar un producto de calidad que le abra las puertas a otras oportunidades de firma de contratos de desarrollo en otros países.  Algunos de sus compañeros han trabajado en una metodología llamada TSP (Team Software Process) y le han comentado que será de gran ayuda para este proyecto.  **¿De qué trata este proceso?, ¿cuáles son las ventajas de aplicarlo en un proyecto que involucra a un grupo nutrido de ingenieros?, ¿cómo asegurar los entregables en las fechas comprometidas, sin afectar la calidad del software?, ¿cómo se relaciona el proceso TSP con PSP?** |

**Explicación del tema 3**

**Información del metadato tema 3**

|  |  |
| --- | --- |
| **Nombre del curso** | Procesos de desarrollo de software |
| **Clave del curso** | TI13314 |
| **Nombre del tema** | **Tema 3.** TSP (Team Software Process) |
| **Descripción** | En este tema encontrarás algunas técnicas para administrar un equipo de trabajo al que se le asigna un proyecto de desarrollo de software. |
| **Conceptos clave** | TSP, técnicas, equipo, involucramiento, problemas, estilos de equipos, líder de equipo. |
| **Objetivo** | Reconocer las características de un equipo de personas a las que se les asigna un proyecto de desarrollo de software. |
| **Tiempo estimado** | 2 horas |
| **Autor** | Ing. Francisco Javier García Enríquez, MTI |
| **Fecha** | 21/10/2015 |

|  |
| --- |
| El trabajo en equipo en un proyecto de desarrollo de software es fundamental para generar un producto de calidad. Sería prácticamente imposible que el todo el desarrollo pueda realizarse con un solo ingeniero y cumplir con las expectativas en las fechas de entrega del cliente.  Bastaría con realizar una comparación entre los sistemas desarrollados hace algunos años con los actuales y verificar la cantidad de requerimientos documentados. Las necesidades del cliente, la competencia y el mercado han cambiado, al igual que lo ha hecho la misma tecnología, por lo que para responder a estas expectativas se necesita la fuerza y el empuje de todo un equipo de personas con habilidades y experiencias diferentes.  **3.1 Conceptos básicos del TSP**  Según Humphrey (2006), TSP (Team Software Process) es un proceso diseñado para guiar a equipos de personas en la planeación, diseño y desarrollo de sistemas de software de calidad. Es una metodología que ha sido probada en varios proyectos y ha demostrado que mejora la productividad del equipo y se mantienen motivados.  Algunos conceptos básicos del TSP son los siguientes:  **Haz clic en cada concepto para conocer más detalle**    **Equipo de trabajo (*Team*)**  http://cache2.asset-cache.net/xr/470013530.jpg?v=1&c=IWSAsset&k=3&d=B53F616F4B95E553A61D9D3B78D70BCE39C0A06F6CC350793C98F2063D6BA66D0511EF71C87E5061E30A760B0D811297Es un grupo de personas que comparten una meta en común. En un equipo de trabajo eficaz, cada miembro está comprometido con el objetivo del grupo, hará todo lo que sea necesario para alcanzar la meta propuesta y sabe que para alcanzar el éxito necesita de sus compañeros. Una parte fundamental del trabajo en equipo es que todos conocen los lineamientos y las prácticas que son comunes. Esto es parecido a establecer un equipo de algún deporte. Todos sus miembros deben conocer las reglas del juego si desean ganar los partidos.  **Roles y responsabilidades**  El equipo debe asignar uno o más roles a cada uno de sus miembros. Según Humphrey (2006), los roles proveen del sentido de propiedad y pertenencia, además que se asegura que se asigna una responsabilidad para cada una de las tareas necesarias del proyecto.  Algunos roles de gestión que se deben asignar son los siguientes:   * **Planeación:** el miembro del equipo con este rol se hace responsable de planear las actividades que involucran el proyecto, asignar los recursos necesarios y dar seguimiento al cumplimiento de las tareas. * **Proceso:** se asegura que se encuentren definidos los procesos que seguirán los miembros del equipo de trabajo, como generación de la documentación, reportes de progreso y estatus semanal, análisis de propuestas de mejora al proceso y solución a problemas detectados en la secuencia de actividades. * **Calidad:** es responsable de generar un plan de calidad, establecer los parámetros de evaluación del trabajo, llevar el control del registro periódico de defectos, crear reportes de calidad y ponerlos a la disposición de todos los miembros del equipo. * **Soporte:** su principal responsabilidad es asegurar que los miembros del equipo tengan las herramientas necesarias y los métodos adecuados para realizar el trabajo del proyecto. Entre sus actividades se encuentra mantener un sistema para el registro de defectos y tiempos las tareas de los miembros del equipo, control de versiones y configuración de ambientes de desarrollo, además de estar atento de generar oportunidades que permitan reutilizar de código de otros proyectos. * **Interfaz con el cliente:** este rol permite asegurar que el equipo entiende las necesidades del cliente e impulsa formas que permitan exceder a sus expectativas. Confirma con el cliente los requerimientos, gestiona nuevos requerimientos o posibles cambios al documento de especificaciones de requisitos una vez empezado el proceso.   *“El trabajo de calidad no se hace por error, es producto del trabajo cuidadoso, razonado y de personas motivadas”.* (Humphrey, 2006)   * **Diseño:** se encarga de las actividades de la planeación, refinamiento y verificación del diseño del software. * **Implementación:** el gerente de implementación es responsable de generar un producto de calidad, conforme al diseño y a las especificaciones del cliente, así como la documentación necesaria que describa los detalles del código, versiones, manuales de instalación y soporte. * **Pruebas:** este rol se encarga de la verificación y validación del software, es decir de llevar a cabo las pruebas necesarias para detectar a tiempo posibles defectos al software antes de que se encuentre en las manos del cliente y asegurarse que el producto de software hace lo que se espera.   **Equipos de desarrollo**  En equipos dedicados al desarrollo de software puedes encontrar básicamente 2 tipos:  Business team looking at camera : Ilustración de stock  **Tamaño de los equipos**  En opinión de Humphrey (2006), el tamaño de los equipos tiene una relación directa con su efectividad y recomienda que una cantidad adecuada de miembros se encuentra entre las 12 y 15 personas. Los equipos de más de 20 personas deben ser divididos en grupos más pequeños para que el proceso TSP sea más efectivo.  Jigsaw puzzle being built by teamwork : Foto de stock**Cooperación e interdependencia**  La ayuda mutua es fundamental para el equipo. Cada miembro debe entender que depende en cierta medida del desempeño de otros. Es aquí donde el todo es más que la suma de sus partes.  **Líder del equipo**  Cualquier equipo de trabajo de software requiere de un líder que dirija, coordine y sea el máximo responsable del proyecto asignado al equipo, por lo que debe proveer del entusiasmo, positivismo y energía al equipo para alcanzar los objetivos planeados.  Businessman Addressing Meeting Around Boardroom Table : Foto de stock  **3.2 Equipos de Trabajo**  El proceso TSP sugiere que los equipos de trabajo deben ser autodirigidos para lograr el éxito.  Business people working together. : Stock Photo**Los equipos autodirigidos** son aquellos que generan sus propias estrategias, procesos, planes y establezcan sus propias negociaciones con la gerencia.  Los equipos son conformados por personas con diferentes áreas de experiencia, que ponen sus habilidades al servicio del resto del equipo y realizan tareas en las que puedan especializarse para encontrar mejores soluciones a problemas complejos. Este tipo de equipos se caracterizan especialmente por dar los resultados esperados sin necesidad de supervisión, se comprometen con el equipo de trabajo y son esenciales para proyectos de desarrollo complejos que requieren un trabajo creativo.  **Propiedades de los equipos autodirigidos:**  Según Humphrey (2006), los equipos autodirigidos tienen las siguientes propiedades:  **Haz clic en cada propiedad para conocer más detalle**   * **Un sentido de propiedad y pertenencia:** son grupos que conocen muy bien quién pertenece al equipo y quién no. Son parte distintiva y cohesiva del grupo. Saben trabajar en equipo a través de la cooperación, la comunicación y cada uno sabe que su contribución es indispensable para lograr la meta propuesta. * **Comprometidos con el objetivo del equipo:** alcanzar el objetivo es importante, pero es más importante lo que el equipo ha hecho para lograr el objetivo. Para mantener este compromiso, es importante establecer un mecanismo de retroalimentación que les permita conocer en qué lugar o momento se encuentran en su camino por lograr la meta, por lo que el equipo debe estar informado de su progreso de manera periódica. * **Hacen suyo el proceso y el plan:** estos equipos se hacen responsables por el trabajo, por lo que deciden cómo hacerlo, monitorean su proceso, generan sus propios reportes de avance sin ser un mero trámite de rendición de cuentas para la gerencia. Esto los mantiene motivados porque no se ven amedrentados por un supervisor que está sobre sus espaldas.   *“La disciplina es lo que separa a los expertos de los amateurs en cualquier campo”.* (Humphrey, 2006)   * **Habilidad y disciplina:** tienen la habilidad de realizar un plan, la convicción de defenderlo y la disciplina de seguirlo. Enfocan todas sus energías en su disposición para ensayar, practicar y mejorar sus procesos, lo que les permite echar mano de su creatividad. * **Comparten un compromiso por comportarse de forma honesta, verdadera y respetuosa:** cada miembro se considera una persona en la que el resto del equipo puede confiar. Construyen la confianza a base del respeto y en la honestidad. Se sienten tomados en cuenta para las decisiones importantes y respetan las decisiones de su líder de equipo. * **Están dedicados a alcanzar la excelencia:** su enfoque hacia la mejora continua los impulsa a hacer más que sólo su parte del trabajo asignado. Están abiertos a probar nuevas formas de hacer las tareas y festejan no sólo sus éxitos, sino también sus fracasos porque saben que les permitió reconocer que no era el camino adecuado.     Diferencia entre el equipo tradicional y el equipo autodirigido. Tomado de: Humphrey, W., Chick, T., Nichols, W. y Pomeroy-Huff, M. (2010). Team Software Process (TSP), Body of Knowledge (BOK). Recuperado de: <http://www.sei.cmu.edu/reports/10tr020.pdf> Sólo para fines educativos.  **3.3 Administración de equipos de trabajo**  Una vez formado el equipo, será necesario identificar su estilo de trabajo que ayude al líder del equipo obtener el mayor provecho de la sinergia que pueda encontrar en el equipo. Humphrey (2006), describe cuatro estilos: Abierto, Cerrado, Aleatorio y Síncrono.  **Haz clic en cada estilo para conocer más detalle**  **Abierto:** sus miembros son capaces de realizar tareas que se requieren. Participan en la planeación, diseño y asignan roles de común acuerdo. El consenso es la actividad que los describe.  **Aleatorio**: es un equipo en el que se propicia la creatividad para encontrar soluciones innovadoras a problemas complejos. Sus reuniones pueden ser calificadas como caóticas por lo que requieren de una guía para redirigir el grupo. Su principal ventaja es la generación de un ambiente creativo, por lo que suelen hallar soluciones que rompen paradigmas.  **Cerrado:** los equipos cerrados están enfocados a la tarea, saben los pasos a realizar para alcanzar el objetivo y tratan hacer el trabajo lo más eficientemente posible. No dedican tiempo para el debate o las discusiones, principalmente porque requieren terminar lo antes posible sus asignaciones. Este estilo de equipo de trabajo aparece generalmente cuando existe un apremio por cumplir con las fechas de entrega. La mayor desventaja es que no le dedican esfuerzo para la mejora continua.  **Sincrónico:** ss un grupo en el que sus miembros trabajan de manera relativamente independiente, ya que tienen poca necesidad de interactuar con sus compañeros de equipo. Es frecuente encontrar este tipo de equipos cuando son tareas específicas que son asignadas a una persona. El resultado del trabajo es de fácil acoplamiento.  **Estilos de Equipo**. Tomado del libro: Humphrey, W. (2006). TSP(SM) Coaching Development Teams. EEUU: Pearson Education. Sólo para fines educativos.  Como puedes observar en el diagrama, la característica principal del equipo con un estilo cerrado es que requiere de una mayor dirección que los abiertos, y los equipos aleatorios establecen una mecánica más interactiva que los sincrónicos.  **Problemas comunes:**  Los equipos de trabajo suelen terminar fracasando cuando se presentan los siguientes problemas, que Humphrey (2006) las categoriza en 4 grupos:  **Haz clic para conocer más detalle**   * **Recursos inadecuados**: cuando la cantidad de miembros no es proporcional al tamaño del software asignado, sus miembros no tienen las habilidades necesarias o las herramientas sin inapropiadas. * **Problemas de liderazgo:** aparece cuando el líder del equipo no tiene claro cómo guiar y motivar a sus miembros en el trabajo que se requiere hacer. * **Metas imposibles:** surgen debido a la planeación de fechas de entrega poco realistas o demasiado optimistas, lo que suele desgastar las energías del equipo y minar la motivación. * **Problemas de la moral del equipo:** la moral del equipo puede venirse abajo cuando está más preocupado por permanecer en el equipo que por trabajar en equipo. Problemas de evaluación, niveles salariales, seguridad en el trabajo y ambientes con mucha tensión, pueden generar climas organizacionales que no contribuyen al verdadero trabajo en equipo.   Business people cheering with hands together : Stock Photo  **Técnicas del involucramiento**  Para poder trabajar verdaderamente en equipo, el involucramiento de todos los miembros es una de las estrategias más efectivas, sobre todo cuando en el equipo está conformado por personas de recién ingreso y personas con expertas.  Humphrey (2006) propone utilizar las siguientes técnicas que puede utilizar un líder de equipo:  **Haz clic en cada técnica para conocer más detalle**   * **Pregunte, no ordene:** realizar preguntas en las reuniones de equipo es una buena forma de conocer las opiniones de todos. Esto obligará a los nuevos miembros a salir de la obscuridad y emitir su opinión profesional. * **Hágase el tonto:** cuando el líder el equipo es un reconocido experto en el proceso, surge la idea entre los miembros del equipo que por su experiencia se debe hacer lo que el líder diga, provocando así que se siga el proceso que éste indica y no el proceso que defina el equipo, lo que finalmente provoca es que pocos se interesen en seguirlo y mucho menos en mejorarlo. Lo que el líder debe hacer es asignar un gerente de proceso que establezca una mecánica que fomente la participación de los miembros del equipo y establezcan su propio proceso. * **Busque con frecuencia llegar a un acuerdo:** escuchar las opiniones de cada miembro siempre será una buena opción para involucrarlos en la toma de decisiones que afectan al equipo. Incluso es importante que nadie se reserve sus opiniones cuando crea que ayudan realizar un mejor trabajo. * **Señales tácitas de preocupación o desacuerdo:** analizar el lenguaje corporal de los miembros del equipo debe ser una de las actividades del líder del equipo durante las reuniones. Fruncir el ceño, actuar agitado, alejarse de la mesa de reunión o no ofrecerse como voluntarios, suelen ser señales tácitas de desacuerdo. El líder debe hacer preguntas inteligentes que hagan aflorar las preocupaciones de los miembros del equipo. * **No dejes que nadie monopolice la discusión:** es muy frecuente que los miembros expertos o veteranos en proyectos de desarrollo ofrezcan ideas muy valiosas, sin embargo, es importante que el líder del equipo funja como moderador en las reuniones de trabajo y evite que sean un monólogo del experto. Todos deben participar y sentirse tomados en cuenta. * **Maneje a los expertos:** tener a expertos en un equipo es una gran ventaja cuando éstos saben trabajar en equipo. El líder debe aprovechar su experiencia sin que ellos tengan actitudes altaneras que hagan sentir al resto del equipo menospreciado. Tratarlos como consultores o asesores, suele ser una buena estrategia que aporte al trabajo en equipo. * **Mantenga el enfoque en los hechos y los datos:** es frecuente que las personas generen opiniones personales basadas corazonadas, lo que provoca tomar decisiones sin un verdadero fundamento. Es necesario entonces basarse en hechos reales y datos concretos para tratar de llegar a acuerdos que involucren a sus miembros por propio convencimiento. |

**Cierre del tema 3 (aterrizaje del alumno)**

|  |
| --- |
| http://cache4.asset-cache.net/xr/78806323.jpg?v=1&c=IWSAsset&k=3&d=E937CC63F8A0DC3DB4B232F7AFAC57EF9F1418E06368D55F64F4494B60994BC4BCC685C059D63657Si alguna vez has trabajado en equipo elaborando alguna evidencia o trabajo final en alguna de tus materias, sabrás lo difícil que puede llegar a ser poner en sintonía a todos los integrantes, es decir, lograr que todos se interesen por alcanzar la mejor calificación, aporten ideas, escuchen a sus compañeros, acepten una crítica y estén dispuestos a dar su mejor esfuerzo. Si esto es difícil en un grupo de 5 personas o menos, cómo será trabajar en un grupo de 15 personas, y además coordinarse con otros grupos igual de numerosos. Las compañías dedicadas al desarrollo de software necesitan establecer una mecánica de trabajo en equipo adecuada si desean producir software de calidad. Tan sólo piensa en la cantidad de personas que se requieren para crear las nuevas versiones del sistema operativo de Windows o de Mac  TSP te ofrece algunos lineamientos para formar y administrar equipos de trabajo que se vuelvan autodirigidos, es decir, equipos que se encuentran convencidos que sólo trabajando unidos, colaborando y ofreciendo su mutuo apoyo, lograrán cualquier meta que se propongan.  TPS es una técnica que complementa las metodologías PSP y CMM. En su conjunto ayudan a las organizaciones a mantener una estructura enfocada al proceso de desarrollo de software, al trabajo en equipo y a la mejora continua. Cada organización debe tomar lo que le sirve y adecuarlo a sus condiciones particulares, así lograrán encontrar su propia fórmula para el éxito. |

**Recursos de apoyo del tema 3**

|  |
| --- |
| **Videos educativos** (Cada tema debe presentar un video grabado por un experto, puede ser una entrevista, explicación de procesos o contenido) |
| **Videos obligatorios:**   * Revisa la exposición de los fundamentos del TSP en el siguiente video:   González, J. (2014). *TPS.* [Archivo de video]. Recuperado de  <https://www.youtube.com/watch?v=Oee6j0S1qRI>  **Videos recomendados:**   * Si deseas una explicación de cómo se aplican TSP en empresas de software, te recomendamos la entrevista a Linda Taborda, Directora Ejecutiva de Intersoftware, Colombia. Negocios. (2011, diciembre 11). *La adopción de las prácticas internacionales PSP y TSP en Colombia.* [Archivo de Video]. Recuperado de <https://www.youtube.com/watch?v=-VHmBL2QX7s> * Revisa esta interesante charla de Watt Humphrey sobre el trabajo ideal en software.   McGraw, S. (2009, agosto 12). *The Ideal Software Job - Watts S. Humphrey | SEI | FREE PREVIEW.* [Archivo de Video]. Recuperado de <https://www.youtube.com/watch?v=eMnZ-9mvH34> |
| **Lecturas: artículos, recursos educativos abiertos** (Incluya al menos tres lecturas que permitan al participante tener mayor conocimiento del tema). |
| **Lecturas obligatorias**   * Revisa los fundamentos del proceso TSP, te recomendamos revisar el siguiente documento:   Humphrey, W., Chick, T., Nichols, W. y Pomeroy-Huff, M. (2010). *Team Software Process (TSP), Body of Knowledge (BOK).* Recuperado de <http://www.sei.cmu.edu/reports/10tr020.pdf>  **Lecturas recomendadas**   * Si deseas revisar cómo se relacionan las metodologías TSP y CMMI, te recomendamos revisar el siguiente trabajo de investigación:   Davis, N. y McHale, J. (2003). *Relating the Team Software ProcessSM (TSPSM) to the Capability Maturity Model for Software (SW-CMM.* Recuperado de <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.115.9727&rep=rep1&type=pdf>   * Si te interesa revisar las fases del proceso TSP, te recomendamos consultar el capítulo 2 de la siguiente tesis:   González, L. (2009). *A Gap Analysis Methodology for the Team Software Process*. Recuperado de <https://repositorio-aberto.up.pt/bitstream/10216/60227/1/000136060.pdf> |

**Checkpoint 3**

|  |
| --- |
| **Asegúrate de poder:**   * Identificar las características de un equipo de trabajo autoridigido para obtener los mejores resultados en los proyectos de desarrollo de software en los que participes. * Reconocer las técnicas de involucramiento que puedes utilizar cuando seas el líder del equipo de desarrollo de software. * Reconocer los diferentes estilos de equipos de trabajo para poder administrarlos de una forma adecuada según lo requiera el proyecto. |

**Glosario del tema 3**

|  |
| --- |
| Ninguno |

**Referencias bibliográficas de la explicación del tema 3**

|  |
| --- |
| Humphrey, W. (2006). *TSP(SM) Coaching Development Teams.* EE. UU: Pearson Education.  Humphrey, W. (1999). *Introduction to the Team Software Process(SM*). EE. UU: Addison Wesley Longman.  Humphrey, W., Chick, T., Nichols, W. y Pomeroy-Huff, M. (2010). *Team Software Process (TSP), Body of Knowledge (BOK).* Recuperado de <http://www.sei.cmu.edu/reports/10tr020.pdf> |

**Notas de enseñanza para el maestro/tutor 3**

|  |
| --- |
| La importancia de este tema radica en hacer conciencia en el alumno de la importancia del trabajo en equipo. Se sugiere hacer un breve ejercicio sobre los problemas que los alumnos enfrentan cuando realizan algún trabajo en equipo y qué sugerencias proponen para obtener mejores resultados.  TSP es una metodología que, si es adecuadamente aplicada, no sólo servirá para el trabajo en equipo en proyectos de software sino para cualquier otro proyecto que involucren un gran número de personas. |

**Actividades y/o Tareas del tema 3**

Utilizar formato que viene en la parte de Anexo I

**Tema Medidas del software**

**Contexto del tema 4 (planteamiento inicial)**

|  |
| --- |
| Karla es ingeniero de software que ha estado trabajando con PSP desde que entró a trabajar a una compañía que se ha convertido en un referente en la consultoría de proyectos de software. Al inicio del año su jefe le asignó un proyecto al que se le asignaron 600 horas de trabajo de un programador a tiempo completo. A Karla esta estimación del esfuerzo requerido para obtener la funcionalidad completa le pareció muy baja, pues considera que el cliente espera cubrir una necesidad crítica para su negocio. Ella estima que deberían incluirse al menos 200 horas extras a la estimación inicial.  Consultation between businessman and customer at office. : Stock PhotoAnte esta discrepancia, Karla aprovecha una reunión con el gerente del proyecto para consultarle cómo es que llegaron a esa estimación. El gerente le confirma que fue una estimación de la cantidad de líneas de código requeridas considerando que reutilizarán código, y se realizó un cálculo de los Puntos de Función tomando en cuenta uno de los proyectos anteriores en los que se requirió una funcionalidad similar y manteniendo la misma arquitectura de software. Karla se pregunta ¿cómo se estima la cantidad de líneas de código cuando se incorporan funciones de otros programas?, ¿a qué se refirió el gerente del proyecto cuando mencionó el término de Puntos de Función y qué relación tiene con la estimación del tamaño del software? |

**Explicación del tema 4**

**Información del metadato tema 4**

Indique los siguientes datos para el metadato de contenido:

|  |  |
| --- | --- |
| **Nombre del curso** | Procesos de desarrollo de software |
| **Clave del curso** | TI13314 |
| **Nombre del tema** | **Tema 4** Medidas del Software |
| **Descripción** | En este tema revisarás la forma en la que las líneas de código pueden ser una forma de medir el tamaño del software y conocerás el término de puntos de función como método para estimación. |
| **Conceptos clave** | Estimación, tamaño del software, LOC, Puntos de Función. |
| **Objetivo** | Utilizar las líneas de código y los puntos de función como técnicas para calcular el tamaño del software. |
| **Tiempo estimado** | 2 horas |
| **Autor** | Ing. Francisco Javier García Enríquez, MTI |
| **Fecha** | 21/10/2015 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| * 1. **Tamaño del software**   Los proyectos de software deben basarse en un plan que incluya una secuencia de actividades, a las que se les asigna recursos. En este ejercicio de planeación será necesario un cálculo de cantidades como el tiempo requerido y el personal que realizará las tareas, a esto se le llame estimación. La estimación es una actividad tan importante en los proyectos que podría transformarse en un aliado o en el peor enemigo de los proyectos. Al estimar el tamaño del software a desarrollar se calcula el tiempo que el proyecto durará y la cantidad de recursos necesarios. En opinión de Humphrey (2006), para que el tamaño del software sea una medida útil debe ser precisa, específica, y cuantificable de forma automática.  **Haz clic para conocer más detalle**  Una **medida precisa** se refiere a obtener un valor exacto del tamaño del producto. Para ser precisos en esta medición se requiere de una adecuada definición del contenido del producto, de tal suerte que cada vez que se estime el tamaño se llegue al mismo valor, evitando así estimar a partir de un juicio u opinión de alguna persona.  Una **medida específica** es aquella que se obtiene de las propiedades del producto de software. El cálculo del tamaño de una aplicación construida con Java no sería igual a si se utilizara C#, Python o Turbo Pascal.  Una **medida cuantificable** es aquella que puede realizarse utilizando alguna herramienta que la pueda obtener de forma automática. Realizar el cálculo del tamaño del software de manera manual sería una actividad que consume mucho tiempo y es inexacta.  Existe una correlación entre la estimación del tamaño del software y el tiempo requerido para desarrollarlo. Entre el tamaño del software sea más grande se requiere más tiempo para analizar los requerimientos del cliente, el diseño es más complejo, la cantidad de casos de prueba será mayor, al igual que el tiempo que le dediquen a documentar el software. Por ejemplo, para establecer una medida de la base de datos que se utilizaría en un sistema es posible contabilizar la cantidad de tablas y campos, incluso un estimado de la cantidad de registros que se crearían de forma periódica. En un desarrollo se podría establecer la cantidad de elementos de la interfaz gráfica (cantidad de campos, etiquetas, botones, opciones del menú, etc.), el tamaño de los programas podría estimarse con la cantidad de líneas de código.  Cualquier elemento que sirva para estimar el esfuerzo necesario para generar un producto de software serían muy útiles.  **SLOC (source lines of code)**  La cantidad de líneas de código es una medida que está directamente relacionada con el esfuerzo del programador, y probablemente la más sencilla. Sin embargo, es importante primero definir el método para contabilizar de manera consistente. En el siguiente ejemplo, puedes ver dos formas en las que puede aparecer una misma estructura de código:   |  |  | | --- | --- | | **Ejemplo 1** | **Ejemplo 2** | | if A>B  then  begin  A := A–B  end  else  begin  A := A+B  end; | if A>B  then begin A := A–B end  else begin A := A+B end; |   Ejemplos tomados del libro: Humphrey, W. (2005). *PSP(SM): A Self-Improvement Process for Software Engineers.*  EE. UU: Pearson Education. Sólo para fines educativos.  En el ejemplo 1 podríamos contabilizar cada renglón como una línea de código, lo que implica que cantidad de LOC’s=9, mientras que en el ejemplo 2 sería LOC’s=3. Podrías decir que el primer código tiene una carga de esfuerzo mayor que el segundo, lo cual no sería del todo cierto.  Otra manera sería contabilizar cada línea cada vez que aparezca un punto y coma (;), eso equivale a que en el ejemplo 1 y el ejemplo 2 tienen un valor de LOC’s=1, siendo una contabilización más consistente.  La descripción completa del método sería:   1. Contabilice las sentencias lógicas de código. Es decir, aquellas que terminan con un punto y coma. 2. Los comentarios y los renglones en blanco no se cuentan. 3. Considere aplicar estas reglas para un lenguaje de programación específico.   Lo importante de tener esta definición radica en la posibilidad de automatizar el método, lo que facilita el cálculo.  El siguiente paso sería determinar cómo evaluar el esfuerzo del programador al asignarle un programa que debe modificar para agregarle funcionalidad.  Humphrey (2006), ofrece las siguientes definiciones:   |  |  | | --- | --- | | **Base** | Es la cantidad de LOC de un programa la primera vez que se genera. | | **Añadidas** | Son LOC’s que se agregan a un código base. | | **Modificadas** | Líneas de código base que fueron modificadas. | | **Eliminadas** | Líneas de código base que fueron borradas o código que no fue utilizado en la nueva versión. | | **Reusadas** | Cuando se reutiliza código de otros programas sin ninguna modificación, se consideran como líneas de código reusadas. | | **Añadidas y modificadas** | En ocasiones un nuevo requerimiento implicará agregar nuevas líneas de código y modificar líneas de código existente. En este caso es conveniente tener un contador particular. | | **Nuevo código reusable** | Es posible que ciertos componentes sean independientes o bien sean funciones comunes que podrán ser utilizadas en otros programas. Esto implica un esfuerzo extra por parte del programador, ya que deberá pensar generar un módulo flexible que pueda ser utilizado en varios escenarios. | | **Totales** | Total de líneas de código consideradas para el programa. |   Una forma gráfica para comparar 2 versiones de código la puedes ver en el siguiente diagrama. El total de líneas de código de la versión 2 es de 910, que se componen de las líneas de código base (sin modificar) + modificadas + reusadas + añadidas. Total = 175 + 75 + 600 + 60. En este caso no fueron incorporadas líneas de nuevo código reusable.    Diagrama tomado del libro: Humphrey, W. (2005). PSP(SM): A Self-Improvement Process for Software Engineers.  EE. UU: Pearson Education. Sólo para fines educativos.   * 1. **Uso de los datos del tamaño**   Una vez que se ha calculado el tamaño del software, estos datos pueden ser utilizados para planear, administrar la calidad y medir la productividad.  **Haz clic para conocer más detalle**    **Planeación**  Los datos que se han registrado en otros proyectos son una fuente de información que se puede tomar para poder estimar el tamaño del software que está por elaborarse en un nuevo proyecto de desarrollo y al mismo tiempo estimar el esfuerzo que requerirá.  **Calidad**  Los registros de los defectos son una buena base para estimar la densidad de defectos que puede tener un software, que determinan el esfuerzo que se realizará en las pruebas, mantenimiento y soporte del sistema. El costo del esfuerzo es un dato importante para la estimación del presupuesto para realizar proyectos de desarrollo de software similares. En opinión de Humphrey (2005), la densidad de defectos en los programas está determinada por la cantidad de defectos encontrados por cada 1000 LOC, mientras que en productos terminados la tasa debe plantearse en el número de defectos por cada 1, 000,000 de LOC. Esta diferencia se debe a que los productos terminados han pasado por un proceso de pruebas exhaustivo antes de ser puestos en producción, por lo que la tasa de errores debería ser mucho menor.  **Productividad**  La productividad se mide como la cantidad de horas necesarias para realizar una unidad de trabajo. Es posible calcular la productividad a través de la cantidad de LOC generadas entre las horas que se requirieron. Por ejemplo, si para producir un programa de 600 LOC se requirieron 60 horas de trabajo, entonces la productividad sería 10 LOC por hora. El problema al medir la productividad radica en que el resultado es un promedio, que no significa que un programador mantenga ese nivel de productividad en cualquier proyecto. Cada programa asignado puede tener variaciones importantes en su complejidad. Algunas tareas podrían incluir la modificación de un programa ya elaborado, reutilizar código, realizar ajustes a funcionalidades existentes. Todas estas variaciones impiden tomar una sola medida de productividad para aplicarlo a cualquier situación. Humphrey (2005), recomienda realizar un ejercicio donde se registren diferentes escenarios y así tomar el valor que más se asemeje al nuevo proyecto, así la estimación del tiempo necesario sería menos imprecisa. Toma por ejemplo el siguiente registro, considerando que el tiempo utilizado fue 60 hras:   |  |  |  | | --- | --- | --- | | Opción de Productividad | Tamaño LOC | Productividad LOC / Hras | | Añadidas (350 + 100 + 50) | 500 | 8.33 | | Añadidas + Modificadas (500+100) | 600 | 10.00 | | Añadidas + Modificadas + Eliminadas (600+200) | 800 | 13.33 | | Añadidas + Modificadas + Reusadas (600+600) | 1200 | 20.00 | | Añadidas + Modificadas + Eliminadas + Reusadas (1200 + 200) | 1400 | 23.33 | | Total al final del producto | 900 | 15.00 |   El proceso para obtener la cantidad de líneas de código de forma manual sería una actividad que consume tiempo y bastante impráctica en programas muy extensos. Es importante que se haga uso de herramientas que puedan obtener estas cantidades de forma automática, asegurando que el método de obtención sea consistente y precisa cada vez que se utilice.  **4.3 Puntos de función**  Mientras medir el producto del software a través de la cantidad de líneas de código se realiza a través del punto de vista del programador, existe otro método para medir el tamaño del software desde el punto de vista del usuario. A este método se le conoce como FP - Functional Points o Puntos de Función y en opinión de Chemuturi (2009) es un método muy popular y ha sido aceptado por muchas organizaciones como un estándar. Esta técnica se basa en medir la funcionalidad que se ofrece el software al usuario. Para cuantificar la funcionalidad se requiere descomponer el sistema en partes pequeñas que puedan ser entendidas y analizadas tanto por el usuario como por el programador. Se divide entonces en 5 aspectos: entradas, salidas, archivos, consultas e interfaces que utiliza.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  | **Factor de Peso (1)** | | |  |  | | **Variable** | **Conteo** |  | **Simple** | **Promedio** | **Complejo** |  | **Subtotal** | | No. De Entradas |  | X | 3 | 4 | 6 | = |  | | No. De Salidas |  | X | 4 | 5 | 7 | = |  | | No. De Consultas |  | X | 3 | 4 | 6 | = |  | | No. De Archivos |  | X | 7 | 10 | 15 | = |  | | No. De Interfaces |  | X | 5 | 7 | 10 | = |  | | **Conteo Total:** |  |  |  |  |  |  |  |  1. El factor de peso ha de elegirse de forma subjetiva según el tipo de funcionalidad requerida.   Se realiza el siguiente cálculo:  PF= [Conteo Total] x [0.65 + 0.01 x ∑(*Fi*)]  Donde ***Fi*** = Factores de ajuste de valor (*i=1 al 14*) Dependiendo de las respuestas a las preguntas que aparecen en la siguiente tabla:   |  |  |  |  | | --- | --- | --- | --- | | Factor de ajuste | Característica del Sistema | Pregunta | Valores del 0 al 5 (2) | | 1 | Respaldo | ¿El sistema requiere respaldo y recuperación confiables? |  | | 2 | Comunicación de datos | ¿Se requieren comunicaciones de datos especializadas para transferir información hacia o desde la aplicación? |  | | 3 | Procesamiento distribuido de datos | ¿Existen funciones de procesamiento distribuidas? |  | | 4 | Rendimiento | ¿El desempeño es crucial? |  | | 5 | Configuración de uso pesado | ¿El sistema correrá en un entorno operativo existente enormemente utilizado? |  | | 6 | Tasa de Transacciones | ¿El sistema requiere entrada de datos en línea? |  | | 7 | Entradas On-Line | ¿La entrada de datos en línea requiere que la transacción de entrada se construya sobre múltiples pantallas y operaciones? |  | | 8 | Actualización On-Line | ¿Los ALI se actualizan en línea? |  | | 9 | Complejidad de la interface | ¿Las entradas, salidas, archivos o consultas son complejos? |  | | 10 | Complejidad del procesamiento | ¿El procesamiento interno es complejo? |  | | 11 | Reusabilidad | ¿El código se diseña para ser reutilizable para cumplir con las necesidades otros tipos de usuarios? |  | | 12 | Implementación | ¿La conversión y la instalación se incluyen en el diseño? |  | | 13 | Multiplicidad de sitios | ¿El sistema se diseña para sitios múltiples en diferentes organizaciones? |  | | 14 | Facilidad del cambio | ¿La aplicación se diseña para facilitar el cambio y su uso por parte del usuario? |  |   (2) Los valores se determinan usando la siguiente escala de valores:     |  |  | | --- | --- | | Factor | Significado | | 0 | No presente | | 1 | Incidental | | 2 | Moderado | | 3 | Medio | | 4 | Significativo | | 5 | Esencial |   Los puntos de función son de especial ayuda para calcular el esfuerzo, costo y posibles defectos en el software de la siguiente manera:  Según el estándar propuesto por el Grupo Internacional de Usuarios de Puntos de Función IFPUG (2015), se estima que el esfuerzo requerido para desarrollar un Punto de Función es de 14 horas-hombre:  **Esfuerzo Requerido** = PF \* 14 (Horas-Hombre)  Tomando en cuenta lo anterior, es posible calcular el costo del esfuerzo realizado:  **Costo Requerido**= Esfuerzo Requerido \* Costo Hora-Hombre  Además, este mismo estándar establece que los defectos potenciales en el desarrollo de un sistema tienen la siguiente proporción:  **Defectos Potenciales** = 5.5 \* PF  Y una vez entregado el sistema, habiendo pasado todas las pruebas, se estima que los defectos entregados al cliente son los siguientes:  **Entrega de Defectos** = 0.22 \* PF  Para que estos cálculos sean una base adecuada que pueda utilizarse para planear el esfuerzo de un proyecto, es importante considerar los siguientes elementos:   * **Tipo de** **plataforma de hardware**: Mainframe, Cliente-Servidor, PC. * **Lenguaje de programación**: Java, C++, C#, PHP, Python * **Tipo de Proyecto**: Software del Sistema, Software intermedio, Software de Aplicación * **Tipo de Sistema** Operativos: Windows, Linux, Unix * **PF históricos**: Cálculo de otros proyectos similares * **Costos históricos**: Costos incurridos en proyectos similares. |

**Cierre del tema 4 (aterrizaje del alumno)**

|  |
| --- |
| Determinar cuánto esfuerzo requiere un proyecto de software suele ser una actividad basada en la experiencia de un grupo de programadores expertos, que en muchas ocasiones ni siquiera son parte activa en el mismo proyecto. Esto provoca generar proyectos subestimados en recursos, que se vuelven una dura carga para el equipo de programadores, que tratarán de cumplir con fechas de entrega que son prácticamente imposibles, o bien, se sobreestima el Business meeting : Stock Photoesfuerzo requerido, lo que encarece el producto y merma las oportunidades del cierre de contratos. Sin duda, nadie tiene una bola mágica que prediga con exactitud lo que implica realmente la realización de un proyecto de software, sin embargo, cuando el tamaño del software está basado en una técnica que se aplica de manera consistente, resulta ser una opción mucho mejor que basarse en opiniones subjetivas.  En la actualidad no existe una única forma de estimar el tamaño del software que sea utilizada por la industria de las compañías dedicadas al desarrollo del software. Cada una aplica la técnica que mejor le ha funcionado. Lo cierto es que recuperar la mayor cantidad de datos para medir la productividad del equipo de desarrollo es la mejor forma que pueden aplicar para estimar con mayor precisión sus nuevos proyectos de software. |

**Recursos de apoyo del tema 4**

|  |
| --- |
| **Videos educativos** (Cada tema debe presentar un video grabado por un experto, puede ser una entrevista, explicación de procesos o contenido) |
| **Videos obligatorios:**   * Revisa el siguiente video donde explica los **conceptos generales sobre estimar el tamaño** del software.   Tabaka, J. (2010). *Agile Chalk Talk: Sizing and Estimating*. [Archivo de video]. Recuperado de <https://www.youtube.com/watch?v=1SBX8BKuUoo>   * Revisa el siguiente video sobre **Puntos de Función**:   Functionpoints. (2011, octubre 23). *Introduction to Function Points*. [Archivo de video]. Recuperado de <https://www.youtube.com/watch?v=X0WLX8iAFb0>  **Videos recomendados**   * Si te interesa saber cómo **utilizar Puntos de Función para medir a proveedores** de IT, te recomendamos revisar la entrevista a Barbara Beech en el siguiente video:   Cast Software. (2014, noviembre 12). *Using Function Points to Measure Your IT Vendors*. [Archivo de video]. Recuperado de:<https://www.youtube.com/watch?v=GQc7Nnz4t64>   * Si estás interesado(a) en **el análisis de puntos de función** como método para medir el software, te recomendamos revisar el siguiente video:   ITMPI. (2013, junio 20). *A Primer on Function Points.* [Archivo de video]. Recuperado de: <https://www.youtube.com/watch?v=zLkc0QHHWmg> |
| **Lecturas: artículos, recursos educativos abiertos** (Incluya al menos tres lecturas que permitan al participante tener mayor conocimiento del tema). |
| **Lecturas obligatorias**   * Revisa las **desventajas de utilizar las líneas de código LOC** como una métrica para calcular el tamaño del software en el siguiente artículo:   Bhatt, K, Tarey, V. y Patel, P. (2012). *Analysis Of Source Lines Of Code* (SLOC) Metric. Recuperado de <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.413.4352&rep=rep1&type=pdf>   * Revisa la **comparación entre la métrica LOC vs Puntos de Función** en el siguiente artículo:   Gollapudi, K. (2005). *Function Points or Lines of Code? – An Insight.* Recuperado de  <http://2911segrdghndfkgjhldfigjldfnvlkdfhgoledkjfngvbldfghjlo2012.googlecode.com/svn/trunk/SMA/Team%20Assignment%236/Ref/Function%20Points%20Or%20Lines%20Of%20Code.doc>  **Lecturas recomendadas**   * Si deseas revisar un método para estimar el tamaño de proyectos Web, te recomendamos revisar el siguiente artículo:   De Andrés, J., Fernández-Lanvin, D. y Lorca, P. (2015). Cost estimation in software engineering projects with web components development. Dyna,82(192) 266-275. Recuperado de <http://www.redalyc.org/articulo.oa?id=49640676030>   * Si deseas revisar los fundamentos sobre los Puntos de Función para estimar el tamaño del software, te recomendamos revisar el siguiente artículo:   Elliott, J. (2007). *Puntos de Función*. Recuperado de  <http://zirks.ublog.cl/archivos/9323/05dsiipuntosdefuncion.pdf>   * Si deseas revisar un caso en el que aplicaron Puntos de Función para estimar el tamaño del software, te recomendamos revisar el siguiente artículo:   Salazar-B., G. (2009). Estimación de proyectos de software: un caso práctico. *Ingeniería y Ciencia*, 5(9) 123-143. Recuperado de <http://tuxchi.redalyc.org/articulo.oa?id=83512213006> |

**Checkpoint 4**

|  |
| --- |
| Asegúrate de poder:   * Reconocer la importancia de medir el tamaño del software como una base para planear el proyecto, la calidad y la productividad. * Aplicar las técnicas de Líneas de Código y Puntos de Función para estimar el tamaño del software. |

**Glosario del tema 4**

|  |
| --- |
| **Correlación:** es el grado en el que dos variables se relacionan entre sí. |

**Referencias bibliográficas de la explicación del tema 4**

|  |
| --- |
| Chemuturi, M. (2009). *Software estimation best practices, tools & techniques*. EE. UU: J. Ross Publishing,  Humphrey, W. (2005). *PSP(SM): A Self-Improvement Process for Software Engineers*. EE. UU: Pearson Education.  IFPUG (2015). *Function Point Alignment*, Recuperado de:  <http://www.ifpug.org/Metric%20Views/MetricViewsJan2015.pdf> |

**Notas de enseñanza para el maestro/tutor 4**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Existen otros métodos para calcular el tamaño del software, entre ellos se encuentran Puntos de Casos de Uso, Análisis de Puntos de función Mark II, Puntos de características (Feature Points), Monte Carlo, Story Point, Puntos de Casos de Prueba.  Watts Humphrey considera que no existe una medición que pueda adaptarse a cualquier escenario, La mejor estrategia será reunir la mayor información, estimar y después comparar lo estimado con lo real para hacer los ajustes necesarios.  Existen algunos autores que desdeñan el uso de la técnica LOC para estimar el tamaño del software, sin embargo, es importante que el alumno la reconozca como parte de su formación. Es importante que se forme un criterio sobre las desventajas de su uso.   |  |  |  | | --- | --- | --- | |  | **Líneas de código** | **Puntos de función** | | **Lenguaje y tecnología** | LOC es dependiente del lenguaje de programación | PF es independiente del lenguaje de programación | | **Comunicación con los clientes** | Muchos clientes no están familiarizados con LOC | La funcionalidad es relativamente más fácil de comunicarla al cliente | | **Herramientas** | Existen muchas herramientas que pueden automatizar el conteo de LOC | No existen ninguna herramienta que pueda automatizar los PF. | | **Uso** | LOC ofrece buenos resultados cuando el proyecto se encuentra en la etapa de Diseño | Se puede utilizar PF desde la etapa de requerimientos | |

**Actividades y/o Tareas del tema 4**

Utilizar formato que viene en la parte de Anexo I

**Tema Estimación del software**

**Contexto del tema 5 (planteamiento inicial)**

|  |
| --- |
| Una empresa de software desea saber cuánto esfuerzo en horas-hombre requerirá el nuevo proyecto para un cliente importante. Dado que es un proyecto estratégico para el cliente, la empresa desea planear muy bien sus recursos, cumpliendo con las expectativas del desarrollo.  Business meeting : Stock PhotoRoberto, el gerente del proyecto, ha recopilado toda la información en relación con los requerimientos generales del desarrollo, de tal manera que puedan ser la base para que los gerentes de las diferentes áreas de sistemas puedan estimar el tiempo y la cantidad de recursos necesarios. Los gerentes han estado acostumbrados a dar estimaciones basadas en su intuición y experiencia, sin embargo, han provocado que subestimen los recursos necesarios o sobreestimen demasiado en el tiempo requerido en proyectos pasados. Para este proyecto, Roberto les ha solicitado que ofrezcan una estimación basada en datos históricos y en algún método de estimación de software, de tal manera que puedan compararlo con la estimación que ellos hacen.  **¿Cómo se estima el esfuerzo y el tamaño del software?, ¿cuáles son los métodos de estimación del software que pueden utilizar los gerentes?, ¿existirá algún método estadístico que puedan utilizar para precisar su estimación?** |

**Explicación del tema 5**

**Información del metadato tema 5**

Indique los siguientes datos para el metadato de contenido:

|  |  |
| --- | --- |
| **Nombre del curso** | Procesos de desarrollo de software |
| **Clave del curso** | TI13314 |
| **Nombre del tema** | **Tema 5** Estimación del software |
| **Descripción** | En este tema revisarás la importancia que tiene la estimación del tamaño del software como parte de la planeación de proyectos de software y analizarás algunos métodos de estimación. |
| **Conceptos clave** | Estimación, tamaño del software, tiempo, proyecto de software, horas hombre, costos, planeación. |
| **Objetivo** | Aplicar métodos de estimación del tamaño de software a proyectos de desarrollo en los que participes. |
| **Tiempo estimado** | 2 horas |
| **Autor** | Ing. Francisco Javier García Enríquez, MTI |
| **Fecha** | 21/10/2015 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5.1 Fundamentos de la estimación del software**  Estimar el esfuerzo que requiere un proyecto es fundamental para calcular los recursos (económicos, humanos, equipamiento y tiempo) que se necesitarán para producir un software de calidad. La estimación permite anticiparse al trabajo que debe realizarse. Una adecuada estimación debe basarse en una adecuada metodología que permita realizar cálculos de manera consistente, de tal suerte que se pueda obtener el mismo resultado cada vez que se calcule, independientemente que quién realice el cálculo.  Según Chemuturi (2009), en la estimación en los proyectos de software intervienen los siguientes elementos:   * Cantidad y tamaño de los componentes necesarios. * Esfuerzo para desarrollar el software * Costo monetario que debe cubrirse. * Duración o plan de ejecución.   Composite image of shocked elegant businesswoman looking through binoculars : Stock PhotoEstas variables permiten revisar las fuentes de donde se obtendrán estos recursos y establecer un presupuesto al cual apegarse para mantener el control de la correcta asignación y uso de recursos.  Una buena administración de proyectos de software debe establecer una estimación antes de dar inicio oficial al proyecto, debe revisarla durante el desarrollo y finalmente comparar el estimado vs el presupuesto real una vez desarrollado y entregado. Esto aplica ya sea si es un desarrollo interno o bien si se ha contratado a un tercero para desarrollarlo.  **Principios de la estimación:**  El PSP-Bok (2009) nos ofrece los siguientes principios de la estimación:   1. **La estimación es incierta**: nadie sabe qué tan grande será el producto, ni mucho menos cuando se encuentra en etapas tempranas del proceso de desarrollo. La estimación puede sesgarse debido a las necesidades del negocio o a presiones de diversa índole. 2. **La estimación es un proceso de aprendizaje:** existe una mejora en la estimación con forme se adquiere experiencia y con datos. 3. **La estimación es una habilidad:** es un hecho que algunas personas tiene mejores habilidades para estimar que otros. La mayoría de las personas puede mejorar su habilidad de estimar el software con la práctica. 4. **Trabaja por ser consistente:** la meta en el proceso de estimación es establecer un proceso que ofrezca resultados de manera consistente, manteniendo un balance entre la sobreestimación y subestimación. 5. **Utiliza un método definido para generar estimaciones:** este método permite establecer con claridad los pasos a seguir para estimar, facilita que otras personas puedan utilizarlo y es una base para poder mejorar. 6. **La estimación está sujeta al error:** es posible que las estimaciones no reflejen con exactitud lo que vaya a suceder en realidad, aunque sus variaciones se acercarán al promedio. 7. **Realice estimaciones al detalle:** al estimar por partes se evita omitir algún elemento importante. Los posibles errores que pueden aparecer al estimar por partes de forma independiente serán mejores que una estimación general. 8. Composite image of businessman standing on ladder : Stock Photo**Utilice datos al hacer estimaciones:** basarse en datos históricos de otros proyectos es una buena base para realizar ajustes a las nuevas estimaciones, al igual que una buena descripción de los requerimientos del cliente.   La idea general del proceso PSP es que un programador tenga la habilidad de estimar su propio trabajo y así pueda utilizar esa experiencia al estimar proyectos más complejos. La precisión de una estimación está basada en estimar pequeñas partes que al final puedan contribuir al cálculo de los requerimientos necesarios del proyecto en general.  **5.2 Métodos de estimación del esfuerzo**  Existen varios métodos para estimar el tamaño del software y el esfuerzo requerido para completar un sistema, a continuación, se describen algunos.  **Haz clic en cada método para conocer más detalle**    **Modelado algorítmico de costos**  Para estimar el costo que implicará el desarrollo de un software se requiere de una estimación del tamaño del proyecto, del tipo de software a desarrollar y de la productividad del equipo de desarrollo.  Sommerville (2011) ofrece una explicación del modelo algorítmico estima el esfuerzo a través de la siguiente fórmula:  **Esfuerzo** = A X TamañoB X B  **Donde:**  **A:** es un factor constante que depende de las prácticas utilizadas por la organización y del tipo de software a desarrollar.  **Tamaño**: es una estimación del tamaño del código fuente (SLOC) o bien los PF Puntos de Función o alguna otra medida que pueda servir para determinar el tamaño de la aplicación.  **B**: es un exponente que se encuentra entre 1 y 1.5, dependiendo del tamaño y complejidad del desarrollo. Este valor exponencial permite relacionar la complejidad del software con los costos, ya que los costos no están relacionados de forma lineal conforme aumenta la complejidad, sino que se van integrando mayores costos.  **M:** es un número que representa la integración entre los atributos del proceso, producto y el desarrollo.  Tanto el valor de B y M son estimaciones totalmente subjetivas, dependen de los antecedentes y experiencia de quien lo calcula.    Este modelo requiere de una constante calibración conforme se vaya usando en posteriores proyectos. Sus variables deben ser ajustadas conforme obtengan los costos reales de los proyectos.  **Modelo COCOMO II**  El modelo COCOMO viene es el acrónimo de Constructive Cost Model. Proviene del análisis de los datos de muchos proyectos, hasta determinar algunas fórmulas que describieron las observaciones obtenidas. Según Sommerville, estas fórmulas vinculan el tamaño del sistema con el esfuerzo para desarrollarlo.  “COCOMO II es un modelo de estimación bien documentado y no registrado” (Sommervile, 2011)  La ventaja que tiene este modelo de estimación es que considera el uso de otros estilos de desarrollo de software modernos como el desarrollo ágil y desarrollo espiral.  COCOMO II incluye los siguientes modelos de estimación: modelo de composición de aplicación, modelo de diseño temprano, modelo es reutilización y modelo posarquitectónico.  Por ejemplo, el modelo de composición de aplicación COCOMO II se obtiene de la siguiente manera:  **Esfuerzo** = 2.94 X TamañoB X M  **Donde:**  **B** varía entre 1.0 y 1.24, dependiendo de la novedad del proyecto, flexibilidad, mitigación de riesgo, cohesión del equipo y madurez del proceso de desarrollo.  **Tamaño:** Se expresa en KSLOC (miles de líneas de código) que se estima se requieren para entregar los Puntos de Función del desarrollo.  **M** es un valor que se obtiene a través de la siguiente fórmula:  **M**= PERS X RCPX X RUSE X RDIF X PREX X FCIL X SCED  **Donde:**  PERS: Habilidad del personal  RCPX: Complejidad del producto  RUSE: Reutilización requerida  PDIF: Dificulta de la plataforma  PREX: Experiencia del personal  SCED: Calendario  FCIL: Facilidad de soporte.  Cada uno de estos atributos se les asigna un valor entre 1 y 6 puntos, donde 1 se considera muy bajo y 6 muy alto.  **Estimación del esfuerzo en programación orientada a objetos**  Esta es una técnica muy sencilla para estimar desarrollo en los que se utilice el paradigma orientado a objetos. El parámetro base de estimación es el número de clases. Para calcular el esfuerzo se requieren seguir 4 pasos:   1. Estimar el número de clases necesarias. Esto es posible a través del diagrama de clases obtenido del análisis de requerimientos. 2. Categorizar el tipo de interfaces y asignarle un peso:  |  |  | | --- | --- | | No se requiere interfaz de usuario (UI): | 2.0 | | La interfaz es simple (sólo texto): | 2.25 | | Requiere una interfaz gráfica (GUI) | 2.5 | | Requiere una interfaz de usuario compleja | 3.0 |  1. Multiplicar el número de clases con el paso asociado a la interfaz de usuario requerida. Se suma este producto al número de clases originalmente estimado con el fin de obtener un estimado del número de clases requerido al final del proyecto. 2. Multiplicar el valor obtenido en el paso 3 por un valor ente 15 y 20 que es un estimado de productividad del número de días-hombre requeridos para desarrollar cada clase.   http://cache4.asset-cache.net/xr/485745248.jpg?v=1&c=IWSAsset&k=3&d=B53F616F4B95E55339A380D20B34CC42C3E39EBE2FD5FF1EDA38F361ADA7CD4B0511EF71C87E5061E30A760B0D811297  Por ejemplo, imagina que un estimado inicial de clases necesarias para un desarrollo es de 50, con una interfaz gráfica de usuario compleja. Entonces 50 x 3.0 = 150 clases. Se agregan 150 clases a las 50 inicialmente estimadas, obteniendo un total de 200 clases. Asumiendo que la productividad del equipo de desarrollo es de 18 días, nos daría un total de 3600 días-hombre. Hay que recordar que no deja de ser un cálculo que ofrece un estimado del esfuerzo necesario, por lo que es importante realizar ajustes convenientes según el criterio de cada organización.  **5.3 Método de estimación PROBE**  El método **PROBE** (*Proxy Based Estimating*) es utilizado para estimar el tamaño y el esfuerzo de desarrollo. Se considera como parte del proceso PSP y se basa principalmente en información histórica que puede ser recopilada para la estimación. Este método utiliza el concepto de “representante” o proxy, que es una unidad de medida que ayuda a estimar las partes que componen el software. Los proxies pueden ser las pantallas, elementos de bases de datos (campos, tablas, consultas, registros), las clases en programación orientada a objetos, o incluso Puntos de Función (PF).    Según Humphrey (2005), esta unidad de medida debe tener las siguientes características:   * Debe permitir contabilizar el esfuerzo requerido para desarrollar el software. * Es preferible utilizar un proxy que pueda ser contabilizado de forma automática. * Debe ser una unidad que aparezca desde el inicio del proyecto. * Tener la posibilidad de adaptarse a las necesidades de cada proyecto y desarrollador. * Debe ser sensible a las variaciones de implementación que afectan al costo o al esfuerzo.   **Ejemplo del método PROBE**  Humphrey (2005) ofrece el siguiente ejemplo para mostrar el método PROBE:  Suponiendo que en tamaño de la unidad representativa (Proxy) es de 126 líneas de código, entonces: E = 126 LOC (dato estimado tomado de información histórica).  **Tabla de ejemplo de datos históricos**   |  |  |  |  | | --- | --- | --- | --- | | **Número del programa** | **Tamaño estimado del Proxy** | **Tamaño del código añadido o modificado** | **Horas totals reales** | | 1 |  | 83 | 11.2 | | 2 |  | 116 | 9.3 | | 3 |  | 186 | 21.6 | | 4 | 97 | 81 | 6.9 | | 5 | 81 | 114 | 10.2 | | Totals | 178 | 580 | 59.2 |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Número del programa** | **Tamaño estimado del Proxy (Size)**  **– *x*** | **Horas reales**  **–*y*** | **Size\*Size**  ***x*\**x*** | **Size\*Hours**  ***x*\**y*** | **Hours\*Hours**  ***y*\**y*** | | 1 | 83 | 11.20 | 6,889 | 929.6 | 125.44 | | 2 | 116 | 9.30 | 13,456 | 1,078.8 | 86.49 | | 3 | 186 | 21.60 | 34,596 | 4,017.6 | 466.56 | | 4 | 81 | 6.90 | 6,561 | 558.9 | 47.61 | | 5 | 114 | 10.20 | 12,498 | 1,162.8 | 104.04 | | **Total** | **580** | **59.20** | **74,498** | **7,747.7** | **830.14** | | **Promedio** | **116** | **11.84** |  |  |  |   Se obtiene el coeficiente de correlación entre el tamaño y las horas:    Dato que el coeficiente de correlación se acerca a 1 podemos decir que existe una correlación entre el tamaño y el tiempo. El método PROBE sugiere que el valor de r sea mayor o igual a 0.7.  Se procede a calcular los valores de los parámetros de regresión β0 y β1:    El tiempo estimado de desarrollo para este caso sería =β0 + β1 \* E  Tiempo Estimado = -2.31046 + 126 \* 0.121987  Tiempo Estimado = 13.060 horas  Para calcular el intervalo de predicción se utiliza una distribución t.  http://my.safaribooksonline.com/getfile?item=NS9kYWNyNzAvOXBtOGlzdGdyLzNnMGEwMzc5ZTEyZ25waGlyYnBlY2UxYmFzZC9maS1wLmE-  **Donde:**  ***p*** es la probabilidad de encontrar un valor en la distribución *t* de student. El valor típico de *p* es 0.7 o 0.9 (70% o 90%). El número de grados de libertad es n-2.  ***Xk*** = El tamaño o tiempo estimado.  **σ** es la varianza (el cuadrado de la desviación estándar).  El valor de t para 0.70 con 3 grados de libertad es 0.58439  El intervalo de predicción superior es 13.060+5.4252 = 18.49 y el inferior es 13.606-5.4252 = 7.63  Es decir que el 70% de los valores estimados caerán entre 6.13 y 19.99 horas |

**Cierre del tema 5 (aterrizaje del alumno)**

|  |
| --- |
| En cualquier proyecto en el que participes, será necesario utilizar la estimación como herramienta básica para conocer de antemano la cantidad de recursos necesarios. La estimación se fundamenta en la información que se tiene a la mano, si la información es suficiente y correcta, la estimación será de mejor calidad. Si, por lo contrario, no se tiene suficiente información o tiene errores, la estimación que se haga seguramente no se acercará a los datos reales. We are successful team! : Stock Photo  Estimar el tamaño el software es una actividad que requiere de un proceso iterativo en el que se realicen ajustes para que la estimación sea más precisa. Esto requiere de la disciplina de los programadores para poder documentar con precisión el trabajo que realizan, y para ello será muy útil las recomendaciones que se tengan del proceso PSP. Cada nuevo proyecto será una nueva oportunidad para mejorar el proceso de estimación anterior.  Debes tener presente que rara vez las estimaciones se cumplen en los proyectos de software, debido a que es difícil predecir los problemas a los que se enfrentará el equipo de desarrollo. Existirán muchos imponderables que provocan que el proyecto se salga de control, sin embargo, si el proyecto cuenta con un buen equipo de trabajo, se podrán solucionar los problemas y cumplir con las fechas estimadas de entrega sin mermar la calidad del producto. |

**Recursos de apoyo del tema 5**

|  |
| --- |
| **Videos educativos** (Cada tema debe presentar un video grabado por un experto, puede ser una entrevista, explicación de procesos o contenido) |
| **Videos obligatorios**   * Revisa la explicación el método de estimación COCOMO II en el siguiente video:   Poveda, J. (2014, octubre 24). *Primera parte COCOMO II*. [Archivo de video]. Recuperado de: <https://www.youtube.com/watch?v=rvrBUEgZ9_Y>  Poveda, J. (2014, octubre 24). *Segunda parte COCOMO II*. [Archivo de video].  <https://www.youtube.com/watch?v=Z4wU10qouLg>  **Videos recomendados:**   * Si deseas conocer la opinión de un experto sobre la **estimación en proyectos**, te recomendamos revisar el siguiente video.   SSW TV. (2014, febrero 23). *How to Estimate in Software Development with Gerard Beckerleg | #NoEstimate.* [Archivo de video]. Recuperado de <https://www.youtube.com/watch?v=bicBUVbeR58>   * Si te interesa conocer cómo realizar estimaciones en un modelo de desarrollo ágil, te recomendamos revisar el siguiente video:   Agile Samurai. (2013, mayo 26). *Agile Estimation*. [Archivo de video]. Recuperado de:   * Puedes revisar en el siguiente video **el concepto de estimación de proyectos en general.**   Project Management Videos. (2015, abril 26). *How to Estimate Your Project*. [Archivo de video]. Recuperado de <https://www.youtube.com/watch?v=rN0FrDpQNUk> |
| **Lecturas: artículos, recursos educativos abiertos** (Incluya al menos tres lecturas que permitan al participante tener mayor conocimiento del tema). |
| **Lecturas obligatorias**   * Revisa el ejemplo de la estimación del tamaño del software utilizando el método PROBE en el siguiente material:   SEL. (2006). *Estimación con PROBE*. Recuperado de  <http://ocw.uc3m.es/ingenieria-informatica/principios-de-ingenieria-informatica/probe-i>  <http://ocw.uc3m.es/ingenieria-informatica/principios-de-ingenieria-informatica/probe-ii>  **Lecturas recomendadas**   * Si te interesa revisar a profundidad del modelo COCOMO II, te recomendamos el siguiente sitio de internet:   Center for Systems and Software Engineering. (s.f). *COCOMO II*. Recuperado de  <http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html>   * Si deseas utilizar la tabla t de student para realizar el cálculo del intervalo de predicción, te recomendamos revisar el siguiente material:   Universidad de Leida. (s.f). *Cómo se usa la tabla t de student*. Recuperado de  <http://web.udl.es/Biomath/Bioestadistica/Dossiers/Temas%20especiales/Estimacion/Como%20se%20utiliza%20la%20tabla%20t%20de%20Student%20(formulas).pdf>   * Puedes revisar un caso de estudio en el que utilizaron COCOMO II para estimar el esfuerzo y el costo de un sofware.   Milicic, D. (2004). *Applying COCOMO II. A case study*. Recuperado de  <http://www.diva-portal.org/smash/get/diva2:831513/FULLTEXT01.pdf> |

**Checkpoint 5**

|  |
| --- |
| Asegúrate de poder:   * Reconocer los elementos y los principios que intervienen en una adecuada estimación como parte del proceso de planeación de un proyecto de desarrollo. * Identificar algunos métodos de estimación de software que puedes utilizar cuando te asignen un proyecto de software. * Practicar el método PROBE como parte de la metodología PSP para realizar estimaciones del tamaño del software y el tiempo requerido. |

**Glosario del tema 5**

|  |
| --- |
| Ninguno |

**Referencias bibliográficas de la explicación del tema 5**

|  |
| --- |
| Chemuturi, M. (2009). *Software estimation best practices, tools & techniques*. EE. UU: J. Ross Publishing.  Pomeroy-Huff, M, Cannon, R., Chick, T., Mullaney, J. y Nichols, W. (2010). *The Personal Software Process (PSP) Body of Knowledge, Version 2.0*. Recuperado de  <http://repository.cmu.edu/cgi/viewcontent.cgi?article=1034&context=sei> |

**Notas de enseñanza para el maestro/tutor 5**

|  |
| --- |
| En el tema 5.3 se requerirá que el alumno se encuentre familiarizado con conceptos estadísticos de correlación, varianza y regresión lineal. Se sugiere solicitarles un breve repaso a estos conceptos antes de revisar el método PROBE.  El coeficiente de correlación permite identificar si dos variables están relacionadas entre sí.  El valor de r se encuentra entre -1.0 y +1.0. Entre más cercano esté de +1.0 se considera que existe una correlación fuerte entre ambas variables. Un valor de mayor a 0.7 se dice que existe una relación suficiente para poder ser utilizada en estimaciones.  La fórmula estadística del índice de correlación es la siguiente:  http://my.safaribooksonline.com/getfile?item=NS9kYWNyNzAvOXBtOGlzdGdyLzNnMGEwMzc5ZTEyMHBoaWZjNGlwcy4vZw--  Se recomienda el uso de paquetes computacionales que calculen el coeficiente como Excel o SPSS. |

**Actividades y/o Tareas del tema 5**

Utilizar formato que viene en la parte de Anexo I

**Anexo I. Tema 1**

**PRESENCIAL Y EN LINEA**

|  |  |
| --- | --- |
| Clave de actividad | TI13314T01A01 |
| Nombre de la actividad | Metodología SCRUM |
| Descripción de la actividad: | En esta actividad realizarás una investigación sobre la metodología de desarrollo de software SCRUM. |
| Objetivo de la actividad | Familiarizarte con la metodología SCRUM como una de las más popular en el desarrollo de software. |
| Técnica didáctica: casos, solución de problemas, ejercicio, etc. | Investigación. |
| Palabras clave: | Scrum, desarrollo de software, CMMI. Claves de éxito, sprint, artefactos. |
| Duración | 2 horas |
| Requerimientos para la actividad | * Lectura del tema 1 * Computadora |
| Desarrollo de la actividad | Investiga la metodología de desarrollo de software SCRUM y contesta lo siguiente:   1. Describe la metodología o el proceso que sigue. 2. Diagrama una línea del tiempo donde puedas explicar sus orígenes, evolución e historia. 3. ¿Cómo se conforma un equipo de Scrum? 4. ¿Qué características tiene un sprint de Scrum? 5. ¿Qué es un artefacto de scrum y cómo los utiliza? 6. ¿Cuáles serían las claves del éxito en un proyecto de desarrollo de software en el que utilice la metodología Scrum? 7. Busca un video en YouTube donde muestre una sesión de trabajo Scrum. Incluye la liga apegándote al formato APA y agrega un breve comentario de lo que observaste en tu trabajo. 8. ¿Cómo podrías combinar CMMI y SCRUM? |
| Criterios de evaluación de la actividad (de 3 a 5 criterios):  Defina cómo se va a evaluar la actividad (el conocimiento y la aplicación de la información) a través de mínimo 3 y máximo 5 enunciados. A cada uno de los enunciados ponerle un valor en porcentaje sobre el 100 %, en la suma de todos deberá de dar un 100. | |  |  | | --- | --- | | **Criterio** | **Puntaje** | | 1. Incluye elementos que le dan contexto al uso de la metodología Scrum para el desarrollo de software. | 20 | | 2. Describe las características de la metodología Scrum. | 30 | | 3. Describe al menos 5 factores que llevan al éxito a proyectos que utilicen Scrum. | 20 | | 4. Describe cómo combinar las técnicas CMMI + Scrum en un mismo proyecto de desarrollo de software. | 30 | | **Total** | **100** | |
| Entregable(s): | Realiza un documento que tu investigación de la metodología SCRUM que incluya el contexto, metodología, factores de éxito y comparación con CMMI. |
| Año de creación: | 2015 |

**Notas de enseñanza para el maestro/tutor**

|  |
| --- |
| SCRUM es una metodología de desarrollo de software muy popular. El objetivo de esta actividad es que el alumno pueda familiarizarse con esta metodología y pueda utilizarla en su práctica profesional.  Se omitió la descripción de esta metodología en el tema a propósito, con la idea que el alumno investigue por su propia cuenta los detalles de SCRUM.  Se sugiere elegir algunos de los videos que los alumnos encuentren para presentarlos en clase.  En el siguiente recurso podrá encontrar una guía muy completa de esta metodología escrita por sus fundadores:  Schwaner, K y Sutherland, J. (2013). *La guía de Scrum*. Recuperado de  <http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-ES.pdf#zoom=100> |

**Anexo I. Tema 2**

**PRESENCIAL Y EN LINEA**

|  |  |
| --- | --- |
| Clave de actividad | TI13314T02A02 |
| Nombre de la actividad | Video de la metodología PSP. |
| Descripción de la actividad: | En esta actividad realizarás un video con el que podrás convencer a programadores a utilizar PSP en su trabajo. |
| Objetivo de la actividad | Conocer a detalle la metodología PSP para poder explicarla a otras personas. |
| Técnica didáctica: casos, solución de problemas, ejercicio, etc. | Video |
| Palabras clave: | PSP, video, programación, formatos del PSP, ventajas, escenario, desarrollo de software. |
| Duración | 2 horas |
| Requerimientos para la actividad | * Lectura del tema 2 * Dispositivo con cámara de video y micrófono * Apoyos visuales * Script o guion de apunte |
| Desarrollo de la actividad | Eres líder de un equipo de programadores al que se le ha asignado un proyecto de desarrollo de software importante, sin embargo, ninguno de ellos conoce la metodología PSP.  Realiza un video donde expliques la metodología PSP a personas como tu equipo de trabajo que no saben cómo aplicarla. Tu objetivo será convencerlos a utilizar PSP en su trabajo diario.    Características del video.   * La duración del video deberá ser de 10 a 15 minutos. * Incluye el logo del Tecmilenio como introducción al video y explica al inicio el objetivo del video. * Deberás incluir apoyos visuales. * Recuerda limitar la cantidad de texto. * No te limites a leer las filminas. * Puedes agregar imágenes que te ayuden a la explicación. * Puedes apoyarte de un script como teleprompter. * No es necesario que tu persona salga a cuadro. * Trata de optimizar el sonido realizando el video en un lugar con un mínimo de ruido ambiental. * Recuerda incluir los créditos y bibliografía en forma APA al final de la presentación.   Contenido del video:   1. Describe el escenario donde pueda utilizarse el PSP 2. Menciona los antecedentes del proceso. 3. Explica el proceso PSP (Incluye cada una de sus fases). 4. Menciona las ventajas de hacer uso de esta metodología de trabajo personal. 5. Explica por qué son importante los formatos de registro y muestra cómo llenarlos. 6. Incluye una breve conclusión de tu trabajo. 7. Haz público el video en Youtube y comparte la liga a tu profesor. |
| Criterios de evaluación de la actividad (de 3 a 5 criterios):  Defina cómo se va a evaluar la actividad (el conocimiento y la aplicación de la información) a través de mínimo 3 y máximo 5 enunciados. A cada uno de los enunciados ponerle un valor en porcentaje sobre el 100 %, en la suma de todos deberá de dar un 100. | |  |  | | --- | --- | | **Criterio** | **Puntaje** | | 1. Genera un video explicando el PSP con una duración mínima de 10 minutos. | 40 | | 2. Incluye una explicación de los antecedentes, contexto y fases del PSP. | 20 | | 3. Ofrece la descripción de las ventajas del uso del PSP. | 20 | | 4. Explica la importancia de los formatos de registro como parte del proceso PSP. | 20 | | 1. Ofrece sus conclusiones personales relacionado el contenido del video. | 20 | |
| Entregable(s): | Liga de YouTube con el video donde explicas los beneficios de utilizar PSP como una metodología |
| Año de creación: | 2015 |

**Notas de enseñanza para el maestro/tutor**

|  |
| --- |
| Si lo considera prudente puede asignar esta actividad como trabajo en equipo de 2 personas.  En internet existe poco material en video que describe el PSP como una metodología de desarrollo útil para todo programador, por lo que es una excelente oportunidad para que alumnos del Tecmilenio propongan material que puede ser visto en YouTube.  Fomente en el grupo la creatividad y exija un trabajo bien hecho. Nuestros alumnos pueden llegar a sorprendernos. |

**Anexo I. Tema 3**

**PRESENCIAL**

|  |  |
| --- | --- |
| Clave de actividad | TI13314T03A03 |
| Nombre de la actividad | Implementando TSP en una compañía de software. |
| Descripción de la actividad: | En esta actividad realizarás en equipo una breve obra (sketch) donde representen la implementación de TSP en una empresa dedicada a desarrollar software. |
| Objetivo de la actividad | Identificar problemas de trabajo en equipo y cómo puede ayudar la metodología de TSP al formar equipos autodirigidos. |
| Técnica didáctica: casos, solución de problemas, ejercicio, etc. | Representación teatral. |
| Palabras clave: | TSP, trabajo en equipo, problemas, soluciones, empresas de software. |
| Duración | 2 horas |
| Requerimientos para la actividad | Lectura del tema 3  Trabajo en equipo para representar un sketch en clases. |
| Desarrollo de la actividad | Reúnete en equipo y realicen lo siguiente:  Son un grupo de asesores entusiastas dedicados a ayudar a las empresas a implementar TSP. Una de sus mejores herramientas para convencer a los equipos de trabajo de IT es el uso de sketches, en donde representan a personas y buscan un cambio de actitud.  Tu equipo de trabajo deberá realizar un sketch en el que describan una situación problemática en un típico equipo de trabajo al que se le asignó un proyecto de desarrollo de software.  Conviertan esta situación inicial en un equipo autoridigido tal y como lo propone TSP.  El trabajo deberá contener lo siguiente:   * Deberán crear un guion de la obra para que lo puedan representar ante el grupo. * Asignen roles específicos para que cada miembro pueda participar en algún momento del sketch. Puedes colocarles algún letrero o gafete que indique el rol que desempeñan. * Utilicen el mismo inmobiliario de su salón como escenografía. * Es importante que ensayen el guion antes de presentarlo ante el grupo. * Ofrezcan una conclusión como cierre de la obra, expresando la utilidad de TSP en una empresa de software. Las conclusiones deben ser parte del guion. * La representación de la obra no debe exceder los 15 minutos. |
| Criterios de evaluación de la actividad (de 3 a 5 criterios):  Defina cómo se va a evaluar la actividad (el conocimiento y la aplicación de la información) a través de mínimo 3 y máximo 5 enunciados. A cada uno de los enunciados ponerle un valor en porcentaje sobre el 100 %, en la suma de todos deberá de dar un 100. | |  |  | | --- | --- | | **Criterio** | **Puntaje** | | 1. Entreguen el guion de la obra en un documento. | 20 | | 2. Representen la obra (sketch) en el salón de clases, participando todos los integrantes del equipo. | 50 | | 3. Expresen sus conclusiones sobre el uso de TSP en empresas de software. | 30 | | **Total** | **100** | |
| Entregable(s): | Entreguen un documento con el guion de la obra, representen la obra y terminen con sus conclusiones. |
| Año de creación: | 2015 |

**Notas de enseñanza para el maestro/tutor**

|  |
| --- |
| * Puede mencionarles que la representación será video grabada para que sus alumnos tomen en serio la actividad. * Es importante que si decide video-grabarlos tenga el consentimiento de ellos para tomarlo como material didáctico en otros cursos o compartirlos con el resto de la clase. Decidan en grupo si será material que puede ser público. * Puede ajustar el tiempo que tienen disponible por equipo para realizar la presentación. Considere unos 3 a 5 minutos entre las presentaciones de los equipos. * Es importante que le presenten un borrador antes de realizar la representación para evitar la improvisación o adecuar el lenguaje utilizado. |

**EN LINEA**

|  |  |
| --- | --- |
| Clave de actividad | TI13314T03A03L |
| Nombre de la actividad | Implementando TSP en una compañía de software. |
| Descripción de la actividad: | En esta actividad realizarás en equipo una breve obra (sketch) donde representen la implementación de TSP en una empresa dedicada a desarrollar software. |
| Objetivo de la actividad | Identificar problemas de trabajo en equipo y cómo puede ayudar la metodología de TSP al formar equipos autodirigidos. |
| Técnica didáctica: casos, solución de problemas, ejercicio, etc. | Representación teatral. |
| Palabras clave: | TSP, trabajo en equipo, problemas, soluciones, empresas de software. |
| Duración | 2 horas |
| Requerimientos para la actividad | Lectura del tema 3  Uso de la herramienta de animación Powtoon  Uso de algún servicio en la nube: Onedrive, Google Drive o Dropbox |
| Desarrollo de la actividad | Revisa el caso y contesta lo siguiente:  Eres parte de un grupo de asesores entusiastas dedicados a ayudar a las empresas a implementar TSP. Una de sus mejores herramientas para convencer a los equipos de trabajo de IT es el uso de un video animado, en donde se explique las ventajas de esta metodología.  Deberás realizar un video animado en el que describas una situación problemática en un típico equipo de trabajo al que se le asignó un proyecto de desarrollo de software. Puedes hacer uso de la herramienta POWTOON <http://www.powtoon.com/>.  Convierte esta situación inicial del equipo tradicional a un equipo autodirigido tal y como lo propone TSP.  El trabajo deberá contener lo siguiente:   * Deberás crear un guion de la animación. * Asigna roles específicos para cada uno de los personajes animados que participarán. Puedes colocarles alguna etiqueta para identificarlos más fácilmente. * Trata de hacer algunas inflexiones de voz para diferenciar los personajes. * Al final del video incluye una conclusión en la que expreses tu opinión sobre la utilidad de TSP en una empresa de software. Las conclusiones deben ser parte del guion. * El video animado no debe exceder los 10 minutos. |
| Criterios de evaluación de la actividad (de 3 a 5 criterios):  Defina cómo se va a evaluar la actividad (el conocimiento y la aplicación de la información) a través de mínimo 3 y máximo 5 enunciados. A cada uno de los enunciados ponerle un valor en porcentaje sobre el 100 %, en la suma de todos deberá de dar un 100. | |  |  | | --- | --- | | **Criterio** | **Puntaje** | | 1. Entrega el guion del video en un documento | 20% | | 2. Exporta el video y compártelo en Youtube o bien guárdalo en algún servicio en la nube como OneDrive, Google Drive o DropBox  Incluye la liga del video en tu trabajo y publícala en el foro de discusiones que tu profesor ha colocado en Blackboard para esta actividad. | 50% | | 3. Incluye tus conclusiones sobre el uso de TSP en empresas de software | 30% | | **Total** | **100** | |
| Entregable(s): | Entreguen un documento con el guion de la obra, el video animado y tus conclusiones. |
| Año de creación: | 2015 |

**Notas de enseñanza para el maestro/tutor**

Incluya de tres a cinco comentarios o sugerencias para que el maestro/tutor pueda llevar a cabo las actividades de manera efectiva.

|  |
| --- |
| * Puede mencionarles que el grupo votará por el mejor video animado. El ganador puede ganarse algunos puntos extras en la siguiente actividad. * Se requiere que genera un foro de discusiones específico para compartir las ligas de los videos animados. * Se sugiere el uso de una encuesta en blackboard que sirva para votar la animación que más les gustó. * Se sugiere el uso de la herramienta de animación POWTOON ya que es sencilla y contiene elementos de animación necesarios para esta actividad. Sin embargo, si el alumno se siente más cómodo con otra herramienta, es libre de hacerlo. |

**Anexo I. Tema 4**

**PRESENCIAL Y EN LINEA**

|  |  |
| --- | --- |
| Clave de actividad | TI13314T04A04 |
| Nombre de la actividad | Métricas del tamaño del software |
| Descripción de la actividad: | En esta actividad revisarás un comparativo entre la métrica de número de líneas de código y los puntos de función. |
| Objetivo de la actividad | Identificar las ventajas y desventajas que tiene cada medida del software. |
| Técnica didáctica: casos, solución de problemas, ejercicio, etc. | Resumen |
| Palabras clave: | SLOC, Puntos de función, software, métricas, tamaño del software. |
| Duración | 2 horas |
| Requerimientos para la actividad | Lectura del tema 4 del curso  Lectura del artículo Function Point sor Line of code. [Ver link](http://2911segrdghndfkgjhldfigjldfnvlkdfhgoledkjfngvbldfghjlo2012.googlecode.com/svn/trunk/SMA/Team%20Assignment%236/Ref/Function%20Points%20Or%20Lines%20Of%20Code.doc) |
| Desarrollo de la actividad | De manera individual realiza lo siguiente:   1. Revisa el siguiente artículo y realiza un resumen del comparativo que hacen los autores sobre el uso de líneas de código vs puntos de función como métrica para estimar el tamaño del software.   Gollapudi, K. (2005). *Function Points or Lines of Code? – An Insight.* Recuperado de: [link](http://2911segrdghndfkgjhldfigjldfnvlkdfhgoledkjfngvbldfghjlo2012.googlecode.com/svn/trunk/SMA/Team%20Assignment%236/Ref/Function%20Points%20Or%20Lines%20Of%20Code.doc)   1. Realiza una tabla comparativa entre las ventajas y desventajas de cada una de las métricas. 2. ¿Crees que sea posible convertir los puntos de función en líneas de código? Justifica tu respuesta. 3. Ofrece en tus conclusiones tu opinión sobre cuál te parece una mejor métrica. 4. Busca en internet una aplicación que pueda medir las líneas de código (SLOC) de forma automática.   Describe cómo realiza la contabilidad y qué configuraciones puede tener para adecuarla a diferentes criterios.  Ofrece un ejemplo del reporte que genera. |
| Criterios de evaluación de la actividad (de 3 a 5 criterios):  Defina cómo se va a evaluar la actividad (el conocimiento y la aplicación de la información) a través de mínimo 3 y máximo 5 enunciados. A cada uno de los enunciados ponerle un valor en porcentaje sobre el 100 %, en la suma de todos deberá de dar un 100. | |  |  | | --- | --- | | **Criterio** | **Puntaje** | | 1. Resumen del artículo propuesto. | 30 | | 2. Tabla comparativa entre las métricas de SLOC y Puntos de Función. | 20 | | 3. Explicación de una aplicación que contabilice de forma automática las líneas de código. | 30 | | 4. Conclusiones con opiniones personales del uso de ambas medidas. | 20 | | Total | 100 | |
| Entregable(s): | Realiza un resumen del artículo propuesto que incluya un comparativo entre las métricas del tamaño de software y la explicación de una aplicación que ayude a contabilizar las líneas de código. |
| Año de creación: | 2015 |

**Notas de enseñanza para el maestro/tutor**

|  |
| --- |
| Puede utilizar la siguiente página para mostrarles un ejemplo de una aplicación que contabiliza las líneas de código:   <http://cloc.sourceforge.net/>  En lugar del artículo propuesto en esta actividad puede sustituirlo por la explicación de un ejercicio completo en el que utilizaron puntos de función.  Santana, A. (2005). *Caso práctico sobre análisis de puntos de función*. Recuperado de:  Parte 1: <http://sg.com.mx/content/view/216>  Parte 2: <http://sg.com.mx/content/view/202>  Parte 3: <http://sg.com.mx/content/view/305>  Es posible convertir los puntos de función a líneas de código.  En la sección 4.4.4 del siguiente artículo sus autoras ofrecen una tabla de conversión según el lenguaje de programación:  Gómez, A., López, M., Migani, S. y Otazú, A. (s.f.). *Un modelo de estimación de proyectos de software*. Recuperado de  <http://www.academia.edu/4853590/UN_MODELO_DE_ESTIMACION_DE_PROYECTOS_DE_SOFTWARE> |

**Anexo I. Tema 5**

**PRESENCIAL Y EN LINEA**

|  |  |
| --- | --- |
| Clave de actividad | TI13314T05A05 |
| Nombre de la actividad | Estimaciones utilizando PROBE y COCOMO II |
| Descripción de la actividad: | En esta actividad realizarás un ejercicio de estimación considerando algunos datos históricos. |
| Objetivo de la actividad | Familiarizarte con el método de estimación PROBE y COCOMO II |
| Técnica didáctica: casos, solución de problemas, ejercicio, etc. | Ejercicio |
| Palabras clave: | Estimaciones, PROBE, COCOMO II, ejercicio, software |
| Duración | 2 horas |
| Requerimientos para la actividad | * Lectura del tema 5 del curso * Cálculos estadísticos: Desviación estándar, Promedios, Coeficiente de correlación. |
| Desarrollo de la actividad | Realiza los siguientes ejercicios de estimación  Utilizando el método PROBE realiza los cálculos necesarios para lo siguiente:   1. Estimar el tamaño del software utilizando la siguiente información. Tamaño del proxy(E)=366 LOC, β0=62 y β1=1.3 2. ¿Cuál sería el tiempo estimado de desarrollo para   (E)=366 LOC, β0=108 y β1=2.95 (la unidad de medida es minutos).   1. Explica los 4 métodos que utiliza PROBE cuando tenemos datos limitados (método A, B, C y D). 2. ¿Cuál sería el rango del intervalo de predicción en tiempo para los siguientes datos?, considerando un E=126 LOC.  |  |  |  | | --- | --- | --- | | **Número del programa** | **Tamaño (LOC)**  **(*x)*** | **Tiempo (Horas)**  **(*y)*** | | 1 | 123 | 11 | | 2 | 110 | 12 | | 3 | 201 | 12.8 | | 4 | 98 | 11.5 | | 5 | 89 | 9.1 |  1. ¿Cuál es el coeficiente de correlación entre el tamaño y el tiempo de los datos históricos proporcionados? 2. ¿Qué tan diferente es la estimación del rango con referencia a su desviación estándar? 3. Realiza un estimado del esfuerzo para producir un software utilizando el modelo de composición de aplicación COCOMO II con los siguientes datos: Tamaño estimado 1024 KSLOC   PERS: 2, RCPX: 5, RUSE: 1, PDIF: 2, PREX: 3, SCED: 6, FCIL: 1. B=1.19 |
| Criterios de evaluación de la actividad (de 3 a 5 criterios):  Defina cómo se va a evaluar la actividad (el conocimiento y la aplicación de la información) a través de mínimo 3 y máximo 5 enunciados. A cada uno de los enunciados ponerle un valor en porcentaje sobre el 100 %, en la suma de todos deberá de dar un 100. | |  |  | | --- | --- | | **Criterio** | **Puntaje** | | 1. Estimación del tamaño del software utilizando PROBE. | 20 | | 2. Estimación del tiempo estimado del software utilizando PROBE. | 20 | | 3. Explicación de los 4 métodos PROBE. | 20 | | 4. Calculo del intervalo de predicción y su comparación con su desviación estándar. | 20 | | 5 Estimación del esfuerzo de un proyecto de software utilizando COCOMO. | 20 | | Total | 100 | |
| Entregable(s): | Entrega el procedimiento de los cálculos realizados, incluyendo las ecuaciones utilizadas. |
| Año de creación: | 2015 |

**Notas de enseñanza para el maestro/tutor**

|  |
| --- |
| Los cuatro métodos de PROBE se encuentran explicados en el siguiente material, que es parte de las lecturas obligatorias del curso:  SEL. (2006). *Estimación con PROBE*. Recuperado de  <http://ocw.uc3m.es/ingenieria-informatica/principios-de-ingenieria-informatica/probe-ii> |