- Hola. - Hola. En este video, vamos a obtener una introducción al diseño de software. En el modelo V de desarrollo de software, el diseño entra en el proceso en la cuarta etapa, después de la arquitectura y antes de la implementación. Se encuentra entre las decisiones a nivel de empresa en el diseño del subsistema y el esfuerzo de desarrollo. Quiero saber aquí que esta no es la única manera de representar esto. De hecho, van a ver la arquitectura y las fases de diseño que se muestran aquí agrupadas en varios lugares. La etapa combinada podría llamarse diseño arquitectónico o incluso podría ser llamada arquitectura o diseño solo a pesar de que contiene ambos conjuntos de tareas. Los dividimos para que podamos ser más específicos y proporcionar información sobre cada una de las responsabilidades por separado. Nuestro programa de maestría en ingeniería de software tiene un curso completo dedicado únicamente a la arquitectura de software y otro curso dedicado exclusivamente al diseño de software. Así que, independientemente de que la empresa para la que trabajas los separe, es probablemente solo una cuestión de semántica. Lo que nos importa es que hablaremos de ellos por separado. Entonces, ¿qué es el diseño de software entonces? El diseño de software es lo que llamamos el entregable, el diseño, el sustantivo, y lo que llamamos el proceso para hacer ese diseño, diseño, el verbo, es el proceso creativo de transformar el problema en una solución. En nuestro caso, transformando una especificación de requisito en una descripción detallada del software que está listo para el código. El sustantivo entonces es la descripción documentada de esa solución y las restricciones y explicaciones utilizadas para llegar a ella. Desarrollamos el diseño del software. Cubriremos la arquitectura por separado, pero como una breve visión general, la arquitectura se centra principalmente en las preocupaciones transversales, de nuestro sistema, especialmente en el contexto de toda la empresa. Decisiones a gran escala como , ¿debemos construir o comprar el software de otra empresa? ¿Cómo va a ser manejada la seguridad, por el servidor o por la aplicación? Muchas decisiones centradas en la empresa y la gestión entran en esto también como distribuir recursos y personal, decidir si el personal actual y el hardware pueden manejar el proyecto en sí mismo, y lo que va a costar para llevarnos allí. Asegurar la financiación interna para tales esfuerzos es a menudo considerado como una preocupación arquitectónica. Lo primero que hacemos es obtener un buen entendimiento del problema cuando se trata de diseño. La mayor parte de esto debe provenir de sus requisitos y documentos de especificaciones. Si tienes la mala suerte de no tener estos o tal vez al menos no tenerlos como documentos escritos, es aún más importante que tengas una comprensión completa de el dominio del problema antes de comenzar a desarrollar tu propia solución de diseño. TMTOWTDI, hay más de una forma de hacerlo. Es un acrónimo bastante común en tecnología porque a menudo es cierto. No seas visionado por túneles en ninguna solución a gran escala como siempre la única manera de resolver el problema. Casi siempre hay otra manera de alcanzar el mismo objetivo singular, así que considera múltiples alternativas antes de decidir definitivamente cuál perseguir. Cuando decimos abstracciones de soluciones, nos referimos esencialmente a cualquier documentación de la solución que no sea tecnológica. Mayormente, eso significa cualquier cosa que no sea código o hardware. Gráficos incluyendo maquetas o tramas, descripciones formales incluyendo lenguaje de modelado unificado o diagramas UML como diagramas de clases y diagramas de secuencia, y otras notaciones descriptivas se deben utilizar para capturar su descripción de la solución que pretende construir o tener hecho para ti. Lo que vas a hacer es repetir para todas las abstracciones, subsistemas, componentes, etc. bajo todo el diseño y hasta que todo el diseño se exprese en términos primitivos. Ahora, los términos primitivos son un poco subjetivos. Lo que quiero que imagines es esto, sigue abstrayendo y diseñando hasta que estés convencido de que puedes entregar este diseño a un equipo de desarrolladores que no conoces y estás seguro de que volverán con una solución correcta. Y haces todo esto sin ser específico de la tecnología. Dejaste que los desarrolladores, aquellos que codifican en C Sharp o Java o Python decidan qué tipo de colección usar en el nivel de código. Deje que ellos decidan exactamente cómo ordenar. Esas cosas son muy específicas del idioma y deben dejarse a los desarrolladores. Esas decisiones no necesitan ser tomadas a nivel de diseño. Todavía debe centrarse en lo que hace que la solución general funcione correctamente, dejando toda la optimización específica del idioma al equipo de desarrolladores. Ellos siguen su diseño. Ellos trabajan las optimizaciones a nivel de idioma. En arquitectura y diseño, seguimos estas seis etapas. Los tres primeros son arquitectónicos. Los últimos tres, diseño. Después de decidir sobre una arquitectura del sistema, separar la responsabilidad de comportamiento en componentes, y determinar cómo interactuarán esos componentes a través de sus interfaces, nos propuse diseñar los componentes individuales. Cada componente está diseñado de forma aislada, el beneficio de la encapsulación y la dependencia de las interfaces que diseñamos. Una vez que cada componente está completamente diseñado de forma aislada, cualquier estructura de datos que sea inherentemente compleja, importante, o compartida entre las clases, o incluso compartida entre componentes, se diseñan entonces para la eficiencia. Lo mismo ocurre con los algoritmos. Cuando el algoritmo es particularmente complejo, novedoso, o importante para el cumplimiento exitoso del comportamiento requerido de los componentes, es posible que vea diseñadores de software en lugar de desarrolladores, escribiendo pseudo código para asegurarse de que el algoritmo está construido correctamente. El diseño de software toma requisitos abstractos y luego construyes el detalle y hasta que estés satisfecho de que puedes entregarlo y se desarrollará correctamente. Entonces, vas a decidir cosas como clases, métodos, tipos de datos, ese tipo de cosas pero no las optimizaciones individuales específicas del idioma que entrarán en el código eventual. Por lo tanto, va a proporcionar detalles, que está listo para la implementación, pero no incluye detalles de implementación. El diseño de software consiste en diseñar una solución, creando los entregables y la documentación necesaria para permitir al equipo en desarrollo construir algo que satisfaga las necesidades del usuario o del cliente. La mejor gente para hacer eso es el equipo de diseño. Este es un paso crucial que pasa de nuestra comprensión del lenguaje natural a soluciones listas para código.

En este vídeo, vamos a echar un vistazo más de cerca a lo que nos referimos cuando decimos modularidad. Cuando hablamos de modularidad, estamos hablando principalmente de estas cuatro cosas.

Acoplamiento, cohesión son medidas de cómo funcionan bien los módulos juntos y qué tan bien cada módulo individual cumple una determinada tarea bien definida y tienden a ir juntos, así que hablaremos de ellos por separado. La información oculta describe nuestra capacidad de abstraer información y conocimiento de una manera que nos permite completar el trabajo complejo en paralelo sin tener que saber todos los detalles de implementación relativos a cómo la tarea se completará eventualmente.

La encapsulación de datos se refiere a la idea de que puede contener constructos y conceptos dentro de un módulo permitiéndonos entender y manipular más fácilmente el concepto cuando lo estamos viendo en un relativo aislamiento. La realidad es que el desarrollo de software es lo más complejo que hemos hecho como humanos. Construir un puente es difícil, pero se puede ver. Puedes rodar un camión a través de él, ver y ver si el puente se derrumba o se balancea o se agrieta mientras estás haciendo eso. Hay propiedades físicas del mundo que debe adherirse a que podemos aprovechar para asegurarnos de que hemos hecho nuestro trabajo. El software no es así. Tal vez nunca sepas que hay una gran fisura en la línea principal de tu programa esperando a que ingreses el valor equivocado y colapsas delante de ti. Intentar mantener todo el concepto de un programa a gran escala en tu mente todo a la vez es un manado de tontos. No tenemos otra opción que dividir el problema en partes más pequeñas que podríamos ser capaces de comprender. Para hacerlo correctamente, vamos a centrarnos en tres conceptos, tres enumerados aquí.

Descomponibilidad. Esencialmente es el concepto antiguo, posiblemente romano, de dividir y conquistar. Cuando el problema es demasiado grande y complejo para obtener un manejo adecuado en él, dividirlo en partes más pequeñas hasta que puedas resolver la parte más pequeña es el camino a seguir. Entonces usted acaba de resolver todas las partes más pequeñas. Pero entonces tenemos que poner todas esas partes más pequeñas juntas y ahí es donde entra en juego la composabilidad. Esto a menudo no es tan simple como uno quisiera. Si no me crees, pregúntale a la NASA. El orbitador climático de Marte desintegró durante su misión debido a un error en unidades con un módulo usando libras segundos y el otro usando Newton-segundos al calcular los valores de impulso totales de su propulsor. Son las pequeñas cosas las que hacen que esto sea tan difícil. Estamos hablando de tomar algo complejo para comenzar y luego hacerlo aún más complejo mediante agregando quién sabe cuántos puntos de intersección entre diferentes desarrolladores o incluso equipos de desarrolladores. La probabilidad de una mala comunicación lo hace aún más difícil. Así que tratamos de enfocarnos en la facilidad de comprensión por desglosando los componentes que esperamos proporcionar una facilidad de comprensión, que entonces esperemos que conduzca a una facilidad de comunicación. Cuando se trata de ocultar información, lo que queremos es la capacidad de usar algo por entender lo que hace, pero no necesariamente cómo. Esto tiende a funcionar bien hasta que la comprensión de lo que hace es tensa. Tome rand (). ¿Qué hace la función rand ()? Bueno, podrías responder que te proporciona un número aleatorio, pero eso no es del todo correcto. Para la mayoría de los propósitos, genera un valor suficiente aleatorio para la mayoría de los usos, pero en realidad, rand () es un generador de números pseudo-aleatorio. Se basa en un valor predefinido y no es exactamente aleatorio. Así que cuando necesitas verdadera aleatoriedad, no encaja del todo en el proyecto de ley. Pero, funciona en muchas situaciones típicas donde la verdadera aleatoriedad no es del todo necesaria. Aquí hay otro ejemplo. ¿Alguna vez has usado la función de ordenación interna en tu idioma? ¿Sabes cómo ordena los valores? ¿Es quicksort, merge sort, or insertion sort, o bubble sort? Bueno, espero que no sea tipo burbuja, pero realmente no lo sabes. Y en su mayor parte, no te importa. En la mayoría de las situaciones, todo lo que desea es que cuando proporciona la matriz se ordene. Siempre que se ordene y no haya un problema de rendimiento, la ocultación de la información es perfectamente aceptable. Usted sabe cómo usarlo y lo que hace detrás de las escenas no le concierne. Debido a la encapsulación de datos, proteger los datos del acceso no autorizado y mantener la integridad es un punto clave. El desarrollador de un módulo tiene la mejor idea de cómo y cuando los atributos deben ser modificados, y luego tratamos de permitirles mantener tanto control como sea posible. Nadie más puede meterse con esos datos. Si se corrompe, debe haber sido hecho por el módulo, esa es la intención. Y por robusto aquí, quiero decir que es probable que las nuevas adiciones no vayan a romper el diseño actual. Así que cuando hablamos de modularidad, estamos hablando realmente de descomponer y reensamblar todos estos componentes. Así que cuando se trata del acoplamiento, la cohesión, la ocultación de información y la encapsulación de datos, son solo cualidades, perspectivas diferentes sobre lo que realmente significa la modularidad.