Metodo Cascada

Requerimientos

Diseño

Implementación

Verificación

Operaciones y mantenimiento

Modelo de Referencia WRSPM (The World Machine Model)

En este video vamos a ver el modelo de referencia WRSPM, algo que llamamos el modelo de máquina mundial. Todo se trata de capturar lo correcto.

Los requisitos están siempre en el dominio del problema. Todo se trata de lo que el usuario quiere hacer para resolver algún problema que tiene. Nuestro trabajo es entonces tomar esos requisitos, capturarlos y luego determinar qué especificaciones de software necesitamos para restringir la solución en el dominio de la solución. Lo que vamos a hacer con nuestro ordenador, nuestro sistema con el fin de resolver el problema. Y hay varias capas de abstracción que existen entre, lo que llamamos requisitos, los objetivos del usuario, y las especificaciones del software, cuáles son los objetivos del sistema para supuestamente cumplir esos objetivos o requisitos del usuario. El modelo WRSPM es una forma de ver este sistema con el fin de determinar cuáles podrían ser las especificaciones de requisitos. Así que este es el modelo muy formal en la forma de ver cómo los separamos. Por lo tanto, verá aquí que tenemos un entorno y un sistema.

Así que el entorno es todos los elementos visibles por el usuario de todo en el mundo, que incluye alguna parte del sistema donde se superpone y esa superposición se llama interfaz.

S\_h Por lo tanto, nuestra interfaz del sistema suele ser una interfaz de usuario, una interfaz de usuario, o es posible que haya oído hablar de una GUI, una interfaz gráfica de usuario. Eso es lo que queremos decir con esa interfaz. Es la ubicación de la reunión entre lo que el usuario puede hacer y lo que el sistema puede hacer o mostrar al usuario con el fin de capturar la entrada y proporcionar salida. Entonces, tenemos el modelo WRSPM. Por lo tanto, tenemos cinco elementos diferentes en este modelo.

W es el mundo. Estas son todas las suposiciones del mundo, estas son cosas que sabemos que son verdaderas. Ahora, no se vuelve tan complejo como decir que la gravedad funciona. Por supuesto, esa es una suposición mundial, pero somos más específicos acerca de las suposiciones mundiales que tienen un impacto en nuestro sistema y en nuestro dominio problemático. Hay cosas que todo el mundo da por sentado y son una de las partes más difíciles de capturar.

R entonces son los requisitos. Esta es la comprensión del lenguaje del usuario de lo que el usuario quiere de la solución. Los usuarios, por ejemplo, quieren retirar dinero. Eso es lo que quieren hacer. Ellos van y encuentran un cajero automático para hacer eso. El cajero automático es la solución.

S es la especificación. La especificación se encuentra en esa área de interfaz. Es la interfaz entre cómo el sistema cumplirá esos requisitos. Así que todavía está escrito en lenguaje de sistema que es desde una perspectiva de usuario o lenguaje natural. Así que dice en lenguaje natural, simplemente Inglés, lo que hará el sistema. Así que va a decir cosas como, con el fin de retirar dinero del cajero automático usted tiene que insertar su tarjeta. Debe insertar su número PIN. Tienes que seleccionar de cheques o ahorros, tienes que ingresar, usando un bloc numérico, la cantidad a retirar y tiene que estar en incrementos de $20 para la mayoría de cajeros automáticos. Esas son todas las cosas que no le importan al usuario. El usuario quiere obtener dinero. La especificación es cómo lo hacen, cómo el sistema cumple ese requisito.

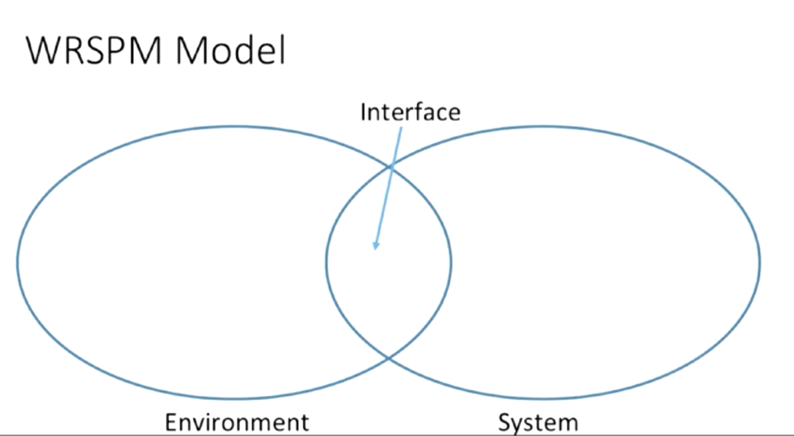
P entonces es el programa y notarás que está fuera del entorno en el sistema. Así que todo el camino por el otro lado de la línea. Ese programa es lo que los desarrolladores de software escribirán, el programa que cumplirá las especificaciones para proporcionar el objetivo del usuario para los requisitos. El programa es todo el código, marcos subyacentes, cualquier cosa similar que usaríamos desde una perspectiva de software, que deja M todo el camino al costado.

La M es la máquina. Es la especificación de hardware, por lo que incluye el rodillo para distribuir dinero, la caja de seguridad para asegurarse de que uno de los requisitos del negocio es que no se acaba de subir y tomar todo el dinero. Es el hardware detrás del sistema. Así que tienes esta comprensión mundial, los requisitos que se encuentran dentro del entorno que se ocupan de los requisitos del usuario, la especificación que define cómo se cumplirán los dos, el programa que cumple con esa especificación con el fin de cumplir con los requisitos, y la máquina que se todo funciona. Dentro de esto, entonces, nos preocupamos por cuatro variables. e\_h, e\_v, s\_v y s\_h.

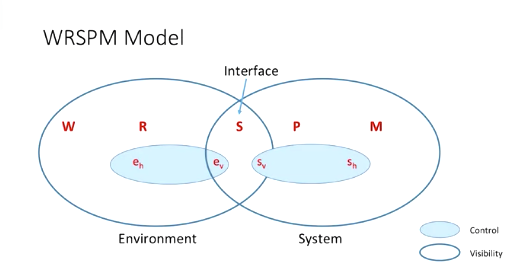
e\_h son los elementos del entorno que están ocultos del sistema. Así que muy bien pueden ser cosas que nos importan fuera del sistema que todavía tenemos que preocuparnos. Son las partes del entorno que el usuario quiere. Por ejemplo, la tarjeta. La tarjeta que insertan es algo que normalmente está fuera del sistema. Realmente no puedes leer la tarjeta. Lo que queremos asegurarnos es que el usuario que usa el cajero automático tiene una tarjeta. Demostramos que al hacer que inserten una tarjeta, leyendo la banda magnética y exigiendo al usuario que introduzca su PIN.

El e\_v, las partes que son visibles para el sistema en el entorno, son los datos que se generan al leer una tira mag en la tarjeta y el número PIN introducido. El número PIN existe y está oculto en el sistema hasta que se introduce en el sistema. Los datos que el usuario introduce entonces son la parte visible del entorno en ese caso. Así que el PIN, una vez introducido, sería información de entorno que ahora es visible para el sistema usando la interfaz.

El s\_v es los elementos del sistema que son visibles en el entorno. Esto incluye cosas como los botones, la información en la pantalla, las indicaciones pidiéndoles que inserten su tarjeta, el mensaje pidiéndoles que introduzcan su número PIN, las estrellas que aparecen cuando ingresas en los valores para que termines recibiendo cuatro asteriscos para que nadie mirando por encima de su hombro puede leer su PIN. Esos son todos los elementos del sistema que son visibles para el usuario y verá que, de nuevo, se encuentra dentro de esa intersección de interfaz. Es algo que está en la interfaz. Entonces todos los datos del sistema que está oculto de el entorno son todos los demás datos y elementos de nuestro sistema. Así que el rodillo detrás de las escenas dentro de la maquinaria que el usuario no puede ver en realidad, puede oírlo y que puede o no hacerlo visible para el usuario dependiendo de su definición, pero normalmente se entiende como está oculto para el usuario. Está oculto detrás de la máquina, está oculto en el código, está oculto en los datos del sistema. Por ejemplo, asegúrese de obtener un número de aprobación del banco real antes de distribuir dinero. Eso está todo oculto para el usuario, aunque sucede y es importante que suceda porque los usuarios esperan que nadie más pueda sacar su dinero. El modelo WRSPM es un modelo de referencia para cómo entendemos los problemas en el mundo real y nos ayuda a identificar la diferencia entre un requisito, la información del dominio del usuario y la especificación. El dominio del sistema, cómo vamos a resolver ese problema. Y tienes que tener mucho cuidado de separar los dos y entender los dos porque hay una gran diferencia entre escribir o capturar los requisitos y luego hacer especificaciones que realmente cumplan esos requisitos. Sólo porque un sistema puede hacerlo, no significa que necesariamente cumpla los requisitos y tener una buena comprensión de todo el modelo WRSPM le ayuda a asegurarse de que sus especificaciones cumplen con los requisitos.

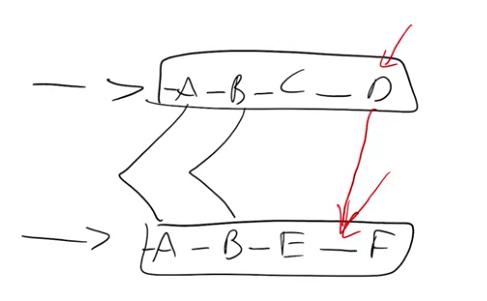


- Hola. - Hola. En este video, vamos a echar un vistazo mejor a un ejemplo del mundo real el modelo de referencia WRSPM en acción. Así que como actualización, este es el modelo WRSPM. Lo que tenemos es las suposiciones del Mundo, los requisitos tanto residiendo en el área del medio ambiente de visibilidad, el significado de la Especificación, sentado dentro de la interfaz entre el medio ambiente y nuestro sistema, el programa y la máquina en sí, el hardware, que es el software y el hardware que se encuentra dentro del nivel de visibilidad del sistema. Y luego usando estas cuatro variables de e\_h, e\_v, s\_v y s\_h, miramos el control. Así que e\_h son aquellos elementos del entorno que están ocultos del sistema. e\_v es los elementos del entorno que son visibles para el sistema. s\_v es los elementos del sistema que son visibles en el entorno. Y s\_h son los elementos del sistema ocultos del entorno. Ahora, todo esto es relativamente abstracto. Así que para obtener una mejor comprensión de la diferencia entre el requisito y la especificación, tomemos un

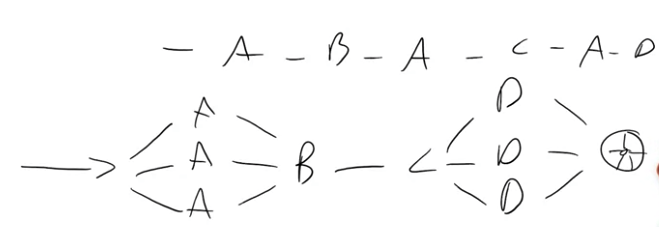


ejemplo real de esto. Así que lo que tenemos es un monitor de pacientes. Es algo que hace un seguimiento de cosas como la frecuencia cardíaca, el pulso , ese tipo de cosas. El deseo de tener un monitor de paciente es que es un sistema de alerta que notifica a una enfermera si el corazón del paciente se detiene. También hace otras cosas, pero centrémonos en este deseo de un usuario por el momento. Ese es el objetivo del usuario. El verdadero requisito entonces, es que cuando el corazón de un paciente se detiene, se notifica a una enfermera. Eso es lo que el usuario realmente quiere. Quieren un sistema que haga eso. Si el corazón se detiene, una enfermera es notificada, eso es realmente lo que quieren fuera de un sistema. Pero nuestro requisito del sistema, lo que vamos a proporcionar es que cuando el sonido de un sensor cae por debajo de un cierto umbral, se acciona una alarma. Entonces, cuando hablamos de un sensor, en este caso, lo que significa es que hay un micrófono conectado a el paciente que mide los latidos del corazón. Y de lo que estamos hablando es de medir la retroalimentación del sensor cuando está midiendo el volumen de ese sensor. Y cuando ese sensor cae por debajo de un cierto umbral, y lo que queremos decir con eso es un cierto volumen durante un cierto período de tiempo. Una vez que esto falla, queremos que la alarma se accione. La pregunta, sin embargo, es que, , ¿está satisfaciendo el requisito real, o lo que llamamos el requisito del usuario? Porque recuerda que el verdadero requisito era que si el corazón se detiene, se notifica a la enfermera. Eso es diferente a no escuchar más el sonido. Y como cualquiera que haya estado en un hospital y tratado con uno de estos monitores de pacientes sabe, hay un montón de falsos positivos. El sensor se cae, el sensor se mueve. El sensor simplemente no es lo suficientemente sensible para captar los latidos del corazón. Pero esos falsos positivos, especialmente en un área crítica de seguridad como un hospital, generalmente se consideran bien. Preferimos que la enfermera tenga que comprobar más a menudo que no. Por lo tanto, mientras este requisito del sistema se acerca, no está cumpliendo realmente el requisito real. Falta algo entre lo que decimos que vamos a hacer por nuestro sistema, el requisito del sistema o especificación del sistema, y el requisito del usuario, lo que el usuario realmente quiere. Así que aquí está el monitoreo de pacientes. La definición de requisitos. Lo que escribimos es que quieren un sistema de advertencia que notifique a la enfermera si el corazón de un paciente se detiene. Eso es lo que el usuario quería. Así que así es como capturamos lo que quieren. Luego hablamos de nuestro diseño del sistema ya que, podemos hacer un ordenador que utiliza un micrófono como sensor y un zumbador como actuador. Y usando este diseño del sistema, podemos hablar de esta especificación del requisito. Esto es lo que vamos a hacer. Si el sonido del sensor, nuestro micrófono, cae por debajo de algún umbral determinado que vamos a definir en función de las necesidades del usuario, este zumbador será accionado y funcionará si tomamos un ordenador que ha sido programado correctamente. Si programamos la computadora para zumbar el zumbador cuando el sonido del sensor cae por debajo del umbral, tenemos un sistema de advertencia que notifica a la enfermera si el corazón del paciente se detiene. Algo así, ese tipo de obras. Pero todavía nos faltan algunos elementos clave. Y estas son las W del modelo WRSPM, Son las suposiciones del Mundo. Esto funciona porque siempre habrá una enfermera lo suficientemente cerca como para escuchar el timbre. No hemos dicho nada sobre la enfermera escuchando el timbre, eso es una suposición mundial. Estamos haciendo una suposición de regla de negocios que hay una regla de negocios que una enfermera estará en la estación de enfermeras porque ahí es donde está el zumbador. Así que si el zumbador se apaga y hay una enfermera allí, que las reglas del negocio requieren, entonces deberían escuchar el zumbador. Ahora, también empezamos a hablar de, bueno, ¿qué pasa con las enfermeras sordas? No oyen el timbre. Así que ahora tenemos que hablar de accionar nuestro actuador, tal vez un zumbador no sea suficiente. Tal vez necesitamos un zumbador y una luz. Tal vez necesitamos un zumbador, una luz, y algo que realmente sacude físicamente el escritorio, algo así. Así que no sólo es necesario que una enfermera esté lo suficientemente cerca para escuchar un timbre, es que una enfermera está lo suficientemente cerca como para reconocer nuestra alarma una vez que la hayamos activado. Además, también tenemos una suposición Mundial de que si el sonido del corazón cae por debajo de ese umbral, significa que el corazón tiene o está a punto de detenerse. Ahora, esta es una definición bastante ligera porque sólo porque eso ha sucedido, el sonido cayendo por debajo del umbral, no significa necesariamente que el corazón tiene o está a punto de detenerse. Eso no es necesariamente cierto. Sin embargo, aceptamos que sea cierto porque la única otra razón para que caiga por debajo de el umbral significa que el sensor no está en el lugar correcto, o se está cayendo, o por alguna otra razón, definitivamente también queremos que la enfermera vaya y compruebe. Esas son suposiciones mundiales que todavía no hemos modelado correctamente cuando hablamos de nuestras definiciones aquí de nuestra definición de requisitos y especificación de requisitos. El modelo WRSPM ayuda a identificar los elementos involucrados en la solución de los problemas. Usted tiene que ser muy cuidadoso con las suposiciones del Mundo. Los requisitos se cumplen cuando estas cosas son ciertas. Las Especificaciones cumplen esos requisitos cuando y solo cuando estas cosas son ciertas. Lo que quiere ser capaz de hacer desde un punto de vista formal es que, si W y S, las suposiciones del Mundo y las Especificaciones implican que se cumplen los requisitos y la Máquina y el Programa implican que la Especificación se ha cumplido, entonces y sólo entonces su solución es correcta, y donde se encuentran la mayoría de los problemas está en esa definición. Es que W y S implican R, tenemos que hacer muchas suposiciones sobre el Mundo y hacer muchas otras cosas que están fuera de nuestro control la mayor parte del tiempo, para asegurarnos de que eso es verdad. Lo mismo vale para M y P, la Máquina y el Programa. Solo porque tengamos nuestra Especificación, no significa que la hayamos programado correctamente, no significa que la hayamos probado correctamente, no significa que la máquina no fallará a veces. Esas son todas las cosas que tenemos que asumir cuando implican S. Así que hace que este modelo WRSPM sea muy importante para asegurarse de que usted realmente está investigando todas las suposiciones, para asegurarse de que estas dos ecuaciones terminan siendo cumplidas.

En este video, vamos a introducir la arquitectura de software y tener una mejor idea de cuál es realmente la definición de arquitectura de software desde nuestra perspectiva . Hay muchos paralelismos entre la arquitectura de software y lo que la mayoría de la gente piensa de la arquitectura cuando se trata de construir edificios. Arquitectos, no importa qué campo sea, son esa interfaz entre el cliente, lo que quieren, y el contratista, el implementador, la persona que construye la cosa. Y siempre es demasiado en toda la arquitectura que el mal diseño arquitectónico no puede ser rescatado por una buena construcción. Un mal diseño no se puede arreglar cuando estás construyendo ese diseño, es increíblemente raro. Y también hay paralelismos en cómo funcionan los arquitectos. Hay tipos de proyectos especializados que requieren cierta experiencia de arquitecto. Si se puede imaginar, es muy diferente construir rascacielos de lo que es construir, digamos, una presa, o un reactor nuclear o cualquier otra cosa que pueda necesitar ser diseñada de una manera específica. Así pues, hay tipos especializados de proyectos dentro de cada campo de la arquitectura. Y en todos esos campos, ya han surgido escuelas de pensamiento y estilos de arquitectura y continúan emergiendo a medida que trabajamos a través del ciclo de vida de todos los proyectos arquitectónicos. Hay un arquitecto relativamente famoso Frank Gehry. Este es el Museo de Arte Weisman de la Universidad de Minnesota. Frank Gehry es uno de los más, si no el más importante arquitecto contemporáneo. Él tiene su propio estilo muy único, y esto es un ejemplo de eso. Cuando vas y ves un edificio que está en este estilo particular, es muy fácil atribuirlo a un arquitecto específico o al menos a una escuela de pensamiento arquitectónica específica, un estilo. Esto es lo mismo ya sea un arquitecto de edificios o un arquitecto de software. Aunque en su mayor parte, la arquitectura de software normalmente no se atribuye a una sola persona, a un artista, sino más bien a una escuela de pensamiento, una forma de construir sistemas a gran escala. Entonces, ¿qué es realmente la arquitectura de software? Así que aquí hay tres definiciones separadas, dos de ellas de libros de texto de arquitectura muy populares, bien respetados que se utilizan en el campo. Y además de uno de los principales organismos informáticos internacionales, IEEE. Tómese un momento, detenga el video y lea estas definiciones. Así, cada una de estas definiciones tiende a caer en la misma idea, esta idea de componentes , esta idea de separar componentes, que comprende externamente propiedades visibles de esos componentes y relaciones entre todos ellos. Aquí hay otra definición para usted, esta viene de Mike Whalen, él es el director de nuestro Centro de Ingeniería de Software de la Universidad de Minnesota. Usted nota que esto es un poco más práctico de una definición. Lo que realmente nos importa es dividir sistemas grandes en sistemas más pequeños. Y estos sistemas más pequeños aún individualmente y de forma independiente tienen valor para el negocio. Y que pueden, supuestamente, si están escritos correctamente, ser integrados entre sí y con otros sistemas existentes muy fácilmente. Ahora hay una serie de razones por las que esta partición es un componente clave. Una de las cosas de las que hablamos en términos de arquitectura de software es una decisión, por ejemplo, comprar versus construir. Usted necesita decidir si el proyecto global que va a construir debe ser construido por un equipo interno o debe salir y encontrar un producto existente, pieza de software fuera de Internet, fuera del estante y simplemente personalizarlo a sus necesidades específicas. Ahora, comprar versus construir es un tipo de problema bastante común con el que tendrás que lidiar. Y tienes que tomar esa decisión y ahí es donde entra la arquitectura, tomando todo el sistema grande y dividiéndolo en pequeños que pueden o no ser construidos individualmente por tu equipo. O contratada para construir por otra persona y entonces simplemente los integramos en nuestro sistema. Es por eso que ese valor empresarial individual y la capacidad de integrarse fácilmente entre sí se vuelve tan importante. Hay una serie de razones por las que deberíamos preocuparnos. La mayoría de las cosas buenas están bien diseñadas. Si se puede imaginar la World Wide Web, se ha diseñado muy bien. Si no estuviera tan bien diseñado, sería mucho, mucho más frágil y colapsará mucho más de lo que hace ahora. Pero ha sido diseñado en un enfoque de arquitectura cliente-servidor muy descentralizado y que lo hace muy resistente a muchas cosas que de otro modo podrían bloquear un sistema tan grande, porque es un sistema grande, pero ha sido distribuido. La buena arquitectura también es muy difícil. Mucho éxito de los sistemas de gran escala trata de asegurarse de que la arquitectura sea buena. Y es realmente difícil hacerlo bien, así que debes asegurarte de que pasas el tiempo haciéndolo. También nos importa porque los errores en el nivel arquitectónico, el nivel a gran escala, son casi, de nuevo, imposible de arreglar simplemente mediante la codificación de la parte de construcción. Cuando cometes errores a nivel arquitectónico, tienden a tener implicaciones de gran alcance. Y cuando encuentras errores en la arquitectura, si estás en la fase de construcción, es casi imposible arreglarlo y seguir adelante. Muchas veces tienes que hacerlo, al igual que una presa, si hay un gran problema de diseño tienes que deconstruir, demoler todo y empezar de nuevo. Eso también puede ser cierto para todo su sistema de software, y eso es algo que la mayoría de las empresas simplemente no pueden resolver. Las arquitecturas de software son sobre muchas cosas diferentes, ¿de acuerdo? Una gran parte de ella es la financiación. Una de las razones por las que descomponemos sistemas en estos componentes que son independientes es para que podamos hablar de paralelización. ¿Quién y qué equipo va a trabajar, gestionar el proyecto, desarrollar y probar individual todavía potencialmente grandes conjuntos de software que eventualmente se integrarán en este sistema a gran escala. Cuando hablamos de patrones arquitectónicos y escuelas arquitectónicas de pensamiento, estamos hablando principalmente de software de nivel empresarial. La arquitectura a pequeña escala generalmente no es tan importante. Pero tan pronto como usted comienza a entrar en piezas de software de tamaño moderado en una empresa, usted tiene que lidiar con este tipo de problemas. Incluyendo, de dónde vamos a obtener el dinero, el presupuesto para pagar por los desarrolladores, los gerentes de proyecto, los diseñadores, los probadores y beta testing, pruebas de usuario, pruebas de aceptación, para realmente asegurarse de que este proyecto es un éxito? Todo eso tiene que venir desde el principio porque necesitas asegurar la financiación para hacerlo. Así que la arquitectura de software se trata de mirar esos componentes, determinar cómo separarlos para que realmente sea práctico que resuelva la solución de cualquier manera.

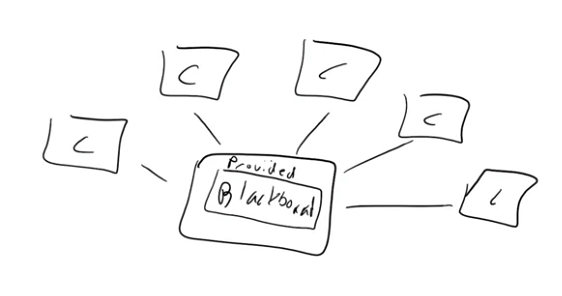
Hola, en este video, vamos a echar un vistazo más de cerca a algunos de los modelos que van junto con la arquitectura de software. Así que hay una variedad de modelos que se han convertido esencialmente en modelos para una serie de diferentes problemas comunes. Así que estos modelos son efectivamente soluciones de mejores prácticas para los problemas que ocurren comúnmente a nivel empresarial, y verá lo que quiero decir aquí en un segundo. Así que el primero que quería ver es algo llamado Pipe-and-Filter. Así que permítanme darles un ejemplo de un problema que podrían enfrentar en el mundo real. Así que tenemos estas dos fuentes de datos entrantes en dos entidades lógicas de negocios relativamente a gran escala. Dentro de esto, hay una variedad de diferentes tareas con las que podría tener que lidiar para procesar los datos que entran. Y luego, por supuesto, todo lo que fuera de eso, todos los datos transformados continuarían en una lógica de negocio adicional. Lo que podrían ver aquí es lo que vemos cuando miramos esto desde una perspectiva arquitectónica . Lo que veo es que A y B son tareas compartidas en sistemas mucho más grandes. Esto es principalmente algo que queremos poder aprovechar a nivel empresarial en lugar de tener dos equipos separados construir esencialmente la misma tarea dos veces. porque lo que termina sucediendo es que, si esa tarea tiene que cambiar de alguna manera, lo que inevitablemente lo hará, entonces A y B podrían potencialmente comenzar a divergir en cómo funcionan. Y eso no es lo que quieres ver, especialmente si por ejemplo , eventualmente en el camino, uno de ellos tiene un problema y el otro no con los mismos flujos de datos entrando. Eso presentaría un problema realmente grande que es mucho más difícil de depurar. Así que lo que queremos hacer entonces es echar un vistazo a una solución a esto que proporciona para la reutilización de estos servicios compartidos, estos subsistemas compartidos. Lo que queremos ser capaces de hacer es algo como esto en su lugar. Por lo tanto, todavía tenemos las dos fuentes de datos llegando, pero en su lugar ahora, queremos construir estos subsistemas ahora de tal de una manera que literalmente, estos puedan ser compartidos. Estos son los mismos módulos compartidos a través de los sistemas globales, así que que eventualmente, nuestros subsistemas hacen uso de este a su vez. Ahora, la forma en que hacemos eso es con Tubería y Filtro. Por lo tanto, la tubería y el filtro tiende a significar que eventualmente comenzamos a construir y mirar estas interfaces entre los elementos individuales. Quiero que imagines si estás trabajando en un sistema Unix. Lo que terminas viendo es algo así como la información que se emite de un programa que se puede insertar directamente en el siguiente programa, y en la línea. Es bastante simple cuando piensas en texto regular, pero podemos expandir esta idea de solución en casi cualquier cosa. Siempre que tenga el mismo formato para entrada y salida , puedo reorganizar y reutilizar cualquiera de estos componentes en cualquier orden. Por ejemplo, digamos que más adelante en la línea nos damos cuenta, queremos aplicar algo como, elemento D, pero realmente queremos aplicar eso aquí entre las partes E y F en el segundo subsistema. Bueno, si y sólo si estamos usando un sistema de tuberías y filtros, deberíamos saber con certeza que podemos simplemente insertar D directamente en el flujo del proceso sin tener que cambiar nada. Y eso solo funciona si está particularmente enfocado en asegurarse de que los formatos de entrada y salida sean exactamente los mismos.

Esto nos permite hacer muchas cosas diferentes. Así que este es un ejemplo, pero déjame mostrarte otro. Este es igual de común. Lo que verá en su lugar es que los datos entrantes se canalizarán a través de versiones paralelas de A, que luego todos se canalizaron a B, que canalizaron a C, que luego volvió a paralelizar a, la lógica empresarial. Nuestra capacidad de paralelizar esta información permite tres procesos diferentes, todos usando la misma base de código de A. Todos al mismo tiempo, simultáneamente, nos permite pasar muy rápidamente a través de ese conjunto si eso es lo más, por ejemplo, el rendimiento pesado. Lo mismo con la parte D. Pero entonces si B es relativamente rápido, cada proceso individual para A puede apuntar al mismo proceso B.



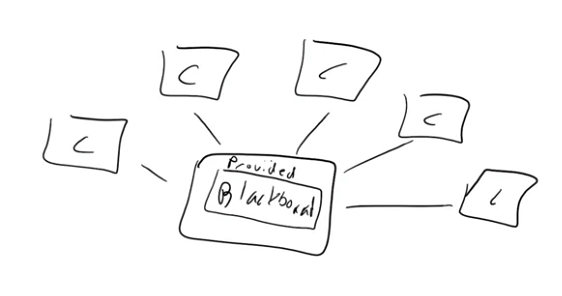
La capacidad de hacer esto, de ser capaz de reutilizar o incluso, por ejemplo, reutilizar de una manera más lineal, el mismo proceso y sobre se basa en esa idea de tubería y filtro. La idea de que los mismos formatos de entrada y salida se están utilizando en todos los procesos o todos los módulos dentro de los componentes. Esta arquitectura de tuberías y filtros es particularmente útil en aquellos casos en los que podemos querer expandir, paralelizar, o reutilizar componentes en sistemas grandes como este. Así que ese es un enfoque en el estilo arquitectónico que se puede aplicar a algo como esto. Y ves eso, por ejemplo, en compiladores. Los compiladores, por ejemplo, tendrán cosas como análisis lógico, análisis de pares, análisis semántico y generación de código. Todos esos deberían tener esencialmente los mismos tipos de entrada y salida para que puedan ser reutilizados.

El siguiente objetivo principal para un estilo arquitectónico para los datos compartidos es Blackboard. Así que una arquitectura de pizarra es algo que podría verse así donde tenemos alguna pizarra que se puede imaginar la tiza de estilo antiguo y pizarra blanca, pizarras en la pizarra.

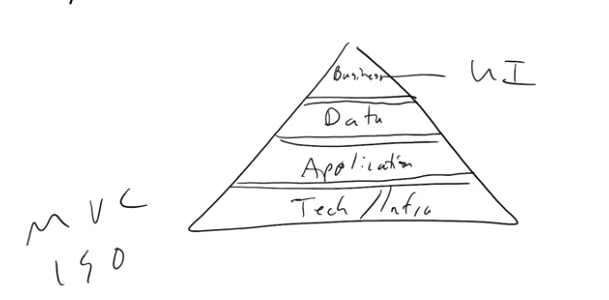


Es un lugar central donde puedes ir y buscar información. Esto está utilizando efectivamente un centro de datos donde todos estos componentes utilizan alguna forma de datos compartidos o procesamiento compartido en todos los componentes. Entonces, cuando tiene datos compartidos importantes para los que, de nuevo, probablemente no desee usar variables globales, puede usar algo así como una pizarra. Esa pieza de funcionalidad mantendrá la integridad de esos datos, o la integridad de las operaciones en muchas llamadas de muchos componentes diferentes. Así que los componentes realmente no se hablan entre sí. Hablan con la pizarra. Así que cuando necesitan comunicarse entre sí, realmente no lo hacen. Se comunican con Blackboard, sirviendo como algún tipo de cola de mensajes o alguna otra forma de repositorio de datos que cada componente puede hacer uso. Así que lo que tendremos aquí es nuestra interfaz proporcionada que necesita estar muy bien definida para que cualquier componente pueda usarla. Ahora, la tendencia de estas interfaces de pizarra tiende a ser muy, muy grande. Tienden a incluir mucha funcionalidad que otros componentes no usan. Hay alguna pérdida de facilidad de uso, comprensibilidad, mantenibilidad cuando se trata de ese tipo de cosas. Pero si el aspecto más importante son los datos compartidos, este estilo arquitectónico podría ser útil y pensar en este formato podría ser bueno.

También tenemos cosas como arquitecturas en capas. Las arquitecturas en capas son muy, muy comunes. Es probable que hayas visto alguna forma de esto a lo largo de tu tiempo. Pero es posible que vea algo como la capa de lógica empresarial separada de una capa de datos , la capa de aplicación y luego su tecnología o lo que normalmente llamamos la capa de infraestructura en la parte inferior.



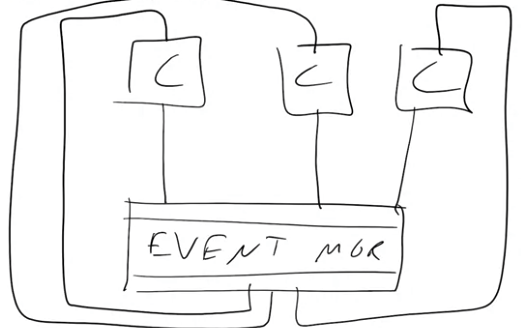
De lo que normalmente estamos hablando aquí son cosas como, queremos asegurarnos de separar estas capas por una razón particular. Y otra que vemos a veces es en lugar de negocios, también puede ver la capa de interfaz de usuario listada aquí. Así que cuando piensas en escuchar algo como, MVC, O el modelo ISO para topología de Internet, es posible que escuches muchas arquitecturas de capas diferentes que se hablan. Lo que estamos tratando de hacer aquí es separar algo elementos centrales de la arquitectura, elementos centrales de todo el sistema en capas tales que podemos modificar o variar, por ejemplo, la capa de datos de una manera que es diferente de cómo lo usamos antes sin afectar necesariamente la aplicación y capa de negocio que tenemos realmente aquí. Interfaces tanto proporcionadas como requeridas entre las capas que se protegen perfectamente entre sí, cada capa externa de los cambios dentro, por lo que podemos hacer cambios en los datos más adelante sin tener que cambiar nada más. Que podemos hacer cambios en la capa de aplicación sin tener que cambiar nada más. Esto es particularmente cierto, por ejemplo, cuando normalmente cambiamos la capa de interfaz de usuario. Si estoy cambiando la capa de interfaz de usuario, no debería tener que modificar la capa de datos de ninguna manera cuando lo hago. Y eso es muy común, que desea actualizar la interfaz de usuario. Esto también es cierto si desea, por ejemplo, cambiar de MariaDB a algo de NoSQL, o Cassandra, o Mangoose o algo así en la capa de infraestructura. Debería poder hacer eso sin necesariamente impactar o cambiar cualquiera de los códigos en el nivel de la aplicación. Así que pensar en una arquitectura de capa es también una forma de dividir su sistema.



Otro de los que probablemente hayas oído hablar es Cliente-Servidor. El Cliente-Servidor es muy, muy sencillo. Usted tiene su portátil y luego salir a la nube, y entonces en algún lugar por aquí está el servidor de Google, y hablas con eso. Internet es el ejemplo principal de una arquitectura cliente-servidor. De lo que estamos hablando aquí es de que la información primaria, o compartida, o almacenada, o persistente se almacena en un servidor, y potencialmente muchas de las operaciones y la programación va allí también, y el cliente simplemente usa ese servicio para realizar cualquier tarea que sea. Así que muchas veces la interfaz de usuario termina siendo empujada al cliente. Y con el tiempo, hemos visto muchos cambios en este modelo.



Por ejemplo, el uso de AJAX y JavaScript muy fuertemente en el lado del cliente ha desplazado esta idea de un modelo puro, delgado, fat server a más de un modelo mixto o incluso fat client, tipo de servidor delgado dependiendo del tipo de programación que termina sucediendo en el cliente de su portátil cuando carga una página web, por ejemplo. Pensar en este formato, cambia la forma en que vas a construir tu sistema porque ahora necesitas pensar en el tipo de cosas que suceden aquí en el servidor versus el tipo de cosas que van a suceder en el cliente. Y, por último, podríamos querer mirar algo así como un gestor de eventos. Así que esto va a parecer similar a nuestra pizarra, pero va a tener algunas diferencias clave bastante. Así que tenemos nuestro gestor de eventos y , a continuación, cada componente se enumera aquí. Ahora, lo que tendemos a ver es que el evento proporciona una interfaz para los componentes para poder usar. Pero también el administrador de eventos usa cada uno de estos componentes en función de los eventos, por lo que siempre se basa en algún componente dispara un evento. Ese evento es manejado por el administrador de eventos que luego decide qué componente controlará ese evento. Así que en cada caso individual, un componente será llamado por algo, potencialmente el gestor de eventos. Decidirá que este evento ha sucedido, empujará ese evento a el administrador de eventos, que luego encontrará el componente correcto para manejar ese evento , generalmente agregando manejadores adjuntos al administrador de eventos. Que entonces, a su vez, llamará al componente, que luego se ejecutará, decidirá qué evento ha ocurrido, y luego continuará usando el administrador de eventos en ese bucle una y otra vez, y otra vez. Esto se centra principalmente en cosas que están basadas en eventos. Cuando estás mirando a esperar a que ocurra algún evento, y luego tener algo de acción en respuesta a eso. Estos modelos de arquitectura de software son soluciones de mejores prácticas. Así que estos fueron solo un par de ejemplos. Hay muchos otros modelos arquitectónicos que debe usar si su problema puede encajar en este tipo de solución. Debería, porque son las mejores prácticas, y potencialmente codifican marcos que ya existen que lo ayudarán a integrarse en uno de estos modelos arquitectónicos. También es bueno ver estos modelos como una forma de saber la mejor manera de particionar el trabajo para que puedas encontrar a las mejores personas, la cantidad correcta de hardware y software , y trabajar con la presupuestación y estimación de los plazos para asegurarte de que tu proyecto de software estará en el mejor posición que puede ser para un éxito.



- Hola. - Hola. En este video, vamos a hablar un poco sobre el proceso de arquitectura de software. Entonces, el proceso de diseño de la arquitectura de software se ve un poco así. Hay tres preocupaciones principales cuando empezamos a hablar del proceso de diseño. La estructuración del sistema se refiere a cómo el sistema se descompone en estos varios subsistemas principales y comunicaciones entre esos subsistemas son luego identificadas. Estamos intercediendo particularmente en cosas como las interfaces. Tan pronto como empezamos a descomponer o separar elementos o componentes, subsistemas a gran escala de nuestro sistema general, tenemos que empezar a preocuparnos por cómo van a continuar comunicándonos ahora que los hemos separado. Esto es particularmente importante cuando vas a paralelizar el trabajo de desarrollo, una vez que los dividimos, dando un componente a un equipo de desarrollo, otro componente a otro equipo de desarrollo y aún más a otros equipos de desarrollo. Tener esa interfaz como un contrato acordado es un componente muy clave para asegurarse de que todo funcione una vez que vuelva a reunirse y la estructuración del sistema es parte de eso. Modelado de control entonces, es cómo las arquitecturas crean un modelo de las relaciones de control entre las diferentes partes del sistema que se establece. Esto es particularmente cierto cuando tenemos estos componentes separados que tratan con el flujo de control, no sólo las interfaces sino el flujo de control en la comunicación entre sí, sino cómo el software funcionará una vez que esté funcionando. Y luego, la descomposición modular es cómo identificamos esas particiones del subsistema. Estamos particularmente mirando cosas como la simplicidad. Estamos buscando cosas como mantenibilidad, confiabilidad, seguridad, todo ese tipo de atributos de calidad, pero también, estamos preocupados por cosas como la gestión de recursos, quién necesita ir a dónde y qué equipos son más adecuados para manejar qué tipo de responsabilidades. Entonces, cuando hablamos de subsistema versus módulos, es importante aquí notar la distinción. Los subsistemas siguen siendo un sistema independiente que podría tener valor comercial. No es nuestro producto completo, pero sigue siendo parte de ese producto que, por sí mismo, tiene valor empresarial y puede funcionar de forma totalmente independiente, o, por lo menos, se puede desarrollar totalmente de forma independiente. Eso es realmente lo que estamos buscando cuando se trata de un subsistema y ese es el objetivo principal para la arquitectura. Eso es versus módulos. El diseño de módulos es lo que consideramos el diseño de software. Eso es un componente individual de un subsistema que no puede funcionar como un sistema independiente o una pieza independiente del producto en general. No tiene, por sí mismo, valor para el negocio. Tienes que reunir e integrar múltiples módulos para conformar un subsistema. Y, aunque la distinción todavía puede ser un poco gris, no se sienta solo en eso. Eso no es una distinción de corte y secado para cualquier desarrollador o ingeniero. La clave aquí va a ser, ¿puede estar sola? Otra cosa que nos importa a nivel arquitectónico son los atributos de calidad del software. Ahora, ven un conjunto enumerado aquí y nos referimos principalmente a estos como las «ilities», principalmente porque muchos de ellos terminan en ese mismo sufijo como ven aquí. Cada uno de estos tiende a ser algo que está asociado con la arquitectura. No significa que no pueda asociarse con otras etapas en el proceso como requisitos, diseño, o incluso implementación, pero es algo en lo que nos enfocamos particularmente en la etapa arquitectónica. Podrías debatirme sobre usabilidad aquí, y creo que eso es justo, pero veamos la seguridad. No hay mucho que uno pueda hacer en la etapa de diseño para afectar la seguridad. Estás haciendo diagramas de clase. Tu decisión sobre qué métodos usar. No hay tantas preocupaciones de seguridad en esa etapa. Sin embargo, desde una perspectiva arquitectónica, puedes hablar de cosas como la configuración del servidor, qué sistemas operativos usarás, cómo separar subsistemas vulnerables de otros, y cómo hablar de instalar el software adecuado y las herramientas de seguridad junto con el que está creando para que todo el producto de software funcione. Cuando se trata de la arquitectura, normalmente estamos hablando de planificación a nivel empresarial. Entonces, ¿dónde encajará este eventual sistema con todo lo demás? ¿Cómo encaja dentro de las restricciones de seguridad actuales? ¿Cómo encaja dentro de las restricciones de rendimiento actuales? ¿Qué pasa con el hardware? ¿Cómo vamos a conseguir eso? ¿Dónde obtenemos fondos para que los desarrolladores construyan esto? ¿Cómo podemos dividir el trabajo para que podamos hacerlo lo más rápido posible utilizando a las personas adecuadas? Esas son todas preocupaciones arquitectónicas. Y, mucho de esto se vuelve a cómo afectamos estos atributos de calidad. ¿Cómo rompemos el sistema y dividimos las responsabilidades de tal manera que podamos garantizar la fiabilidad, tal que podamos garantizar el rendimiento y la seguridad? El proceso de arquitectura de software se ocupa principalmente de esas tres cosas. Estimación; estimación del trabajo y total, decidir cuánto va a tomar, cuánto tiempo va a tomar, quién va a hacerlo. Centrándonos en la calidad; ¿cómo aseguramos cosas como la fiabilidad y el rendimiento , especialmente en este nivel arquitectónico, ya que esas son principalmente preocupaciones centradas en el hardware y el desarrollador? Y luego particionar; en realidad dividiendo el trabajo en subsistemas o módulos que luego pueden ser pasados a la etapa de diseño porque entonces, etapa de diseño entra en la construcción real de los módulos individuales.