La ingeniería de software basada en componentes (CBSE) (también conocida como desarrollo basado en componentes(CBD)) es una rama de la ingeniería de software que enfatiza la separación de asuntos (separation of concerns (SoC)) por lo que se refiere a la funcionalidad de amplio rango disponible a través de un sistema de software dado.

Es un acercamiento basado en la reutilización para definir, implementar, y componer componentes débilmente acoplados en sistemas. Esta práctica persigue un amplio grado de beneficios tanto en el corto como el largo plazo, para el software en sí mismo y para las organizaciones que patrocinan tal software.

Los ingenieros de software consideran los componentes como parte de la plataforma inicial para la orientación a servicios. Los componentes juegan este rol, por ejemplo, en servicios de web y, más recientemente, en las arquitecturas orientadas a servicios(SOA), por el que un componente es convertido por el servicio web en un servicio y consiguientemente hereda otras características más allá de las de un componente ordinario.

Los componentes pueden producir o consumir eventos y pueden ser usados para las arquitecturas dirigida por eventos (EDA).

Un componente de software individual es un paquete de software, un servicio web, o un módulo que encapsula un conjunto de funciones relacionadas (o de datos).

Todos los procesos del sistema son colocados en componentes separados de tal manera que todos los datos y funciones dentro de cada componente están semánticamenterelacionados (justo como con el contenimiento de clases). Debido a este principio, con frecuencia se dice que los componentes son modulares y cohesivos.

Con respecto a la coordinación a lo largo del sistema, los componentes se comunican uno con el otro por medio de interfaces. Cuando un componente ofrece servicios al resto del sistema, éste adopta una interface proporcionada que especifica los servicios que otros

componentes pueden utilizar, y cómo pueden hacerlo. Esta interface puede ser vista como una firma del componente – el cliente no necesita saber sobre los funcionamientos internos del componente (su implementación) para hacer uso de ella. Este principio resulta en componentes referidos como encapsulados. Las ilustraciones UML de este artículo representan a las interfaces proporcionadas, con un símbolo lollipop unido al borde externo del componente.

Sin embargo, cuando un componente necesita usar otro componente para poder funcionar, adopta una interface usada que especifica los servicios que necesita. En las ilustraciones de UML en este artículo, las interfaces usadas son representadas por un símbolo de zócalo abierto unido al borde externo del componente.

Un simple ejemplo de varios componentes de software – representados dentro de un hipotético sistema de reservaciones de días de fiesta representado en UML 2.0.

Otro atributo importante de los componentes es que son sustituibles, así que un componente puede sustituir a otro (en tiempo de diseño o tiempo de ejecución), si el componente sucesor cumple los requisitos del componente inicial (expresado por medio de las interfaces). Por lo tanto, los componentes pueden ser sustituidos por una versión actualizada o una alternativa sin romper el sistema en el cual operan.

Como una regla de oro general para los ingenieros que sustituyen componentes, el componente B puede sustituir inmediatamente al componente A, si el componente B proporciona por lo menos que el componente A provee y no usa más cosas que las que el componente A usa.

Los componentes de software con frecuencia toman la forma de objetos (no clases) o de colecciones de objetos (de la programación orientada a objetos), en una cierta forma binaria o textual, adhiriéndose a un cierto lenguaje de descripción de interface (IDL) de modo que el

componente pueda existir autónomo de otros componentes en una computadora.

Cuando un componente debe ser accesado o compartido a través de contextos de ejecución o enlaces de red, a menudo son empleados técnicas tales como serialización o marshalling para enviar el componente a su destino.

La reusabilidad es una importante característica de un componente de software de alta calidad. Los programadores deben diseñar e implementar componentes de software de una manera tal que diversos programas puedan reutilizarlos. Además, cuando los componentes de software interactúan directamente con los usuarios, debe ser considerada laprueba de usabilidad basada en componentes.

Toma un significativo esfuerzo y conciencia para escribir un componente de software que sea efectivamente reutilizable. El componente necesita estar:

- completamente documentado
- probado a fondo
    - robusto – con una comprensiva comprobación para la validez de la entrada
    - capaz de devolver mensajes de error apropiados o códigos de retorno
    - diseñado con conciencia de que será puesto en usos imprevistos

En los años 1960, los programadores construyeron bibliotecas de subrutinas científicas que eran reusables en un amplio arreglo de aplicaciones de ingeniería y científicas. Aunque estas bibliotecas de subrutinas reusaban algoritmos bien definidos de una manera efectiva, tenían un limitado dominio de aplicación. Los sitios comerciales rutinariamente crearon programas de aplicación a partir de módulos reusables escritos en ensamblador, COBOL, PL/1 y otros lenguajes de segunda y tercera generación, usando bibliotecas de aplicación tanto de sistema como de usuario.

Por 2010, los componentes reusables modernos encapsulan las estructuras de datos y los algoritmos que son aplicados a las estructuras de datos. Esto [clarificación necesaria] se basa en teorías anteriores a los objetos, arquitecturas, frameworks y patrones de diseño de

software, y la extensa teoría de la programación orientada a objetos y el diseño orientado a objetos de todos éstos. Afirma que los componentes de software, como la idea de componentes de hardware, usados, por ejemplo, en telecomunicaciones, pueden en última instancia ser hechos intercambiables y confiables. Por otro lado, se argumenta que es un error enfocarse en componentes independientes en vez del framework (sin el cual el componente no existiría).

Historia

La idea de que el software deba estar componentizado – construido de componentes prefabricados – por primera vez llegó a ser prominente con el discurso de Douglas McIlroy en la conferencia de la OTAN sobre la ingeniería de software en Garmisch, Alemania, 1968, titulado *Mass Produced Software Components* (*componentes de software producidos en masa*). La conferencia se propuso para hacer frente la llamada crisis del software. La inclusión subsecuente de McIlroy de tuberías y filtros en el sistema operativo Unix fue la primera implementación de una infraestructura para esta idea.

Brad Cox de Stepstone en gran parte definió el concepto moderno de un componente de software. Los llamó *Software ICs* y precisó crear una infraestructura y un mercado para estos componentes inventando el lenguaje de programación Objective-C. Él sumariza esta visión en su libro de 1986 *Object-Oriented Programming – An Evolutionary Approach*(Programación orientada a objetos – Un acercamiento evolutivo).

A principio de los 1990, IBM encabezó esta trayectoria con su System Object Model (SOM). Como reacción, Microsoft pavimentó la vía para el despliegue real de componentes de software con OLE y COM. Por 2010 existen muchos modelos exitosos de componentes de software.

Diferencias con la programación orientada a objetos

Los que proponen la programación orientada a objetos (OOP) mantienen 2 posiciones diferentes para modelar el software:

1) que el software debe ser escrito según un modelo mental de los objetos reales o imaginarios que representan. La OOP y las disciplinas relacionadas de análisis orientado a objetos y el diseño orientado a objetos están enfocados en el modelado de interacciones del mundo real e intentan identficar los «sustantivos» y los «verbos» en las reuniones de levantamiento de requerimientos, para esos conceptos sean programados directamente en clases y métodos, que pueden ser usados en más formas humanamente legibles, idealmente por los usuarios finales así como por los programadores que codifican para esos usuarios finales. Esta manera de pensar acerca de la OOP es más bien una ilusión, históricamente no ha dado buenos resultados.

2) que primero debe realizarse un levantamiento de requerimientos con los «stakeholders» (clientes, usuarios finales, etc.) y que luego debe modelarse el sistema en casos de uso de negocio que luego son divididos en casos de uso de usuario, que son directamente implementables en 3 capas: objetos visuales de presentación, objetos de servicios y objetos de persistencia, pudiendo haber más capas dependiendo de las necesidades de la aplicación.

Por contraste, la ingeniería de software basado en componentes no hace tal asunción, y en lugar ello expresa que los desarrolladores deben construir el software pegando entre sí componentes prefabricados – como en los campos de la electrónica o la mecánica. Algunos pares incluso hablan de sistemas modularizados como componentes de software como un nuevo paradigma de programación.

Algunos argumentan que los científicos de la computación tempranos hicieron esta distinción, con la teoría de la programación literaria de Donald Knuth asumiendo optimisticamente que había una convergencia entre los modelos intuitivo y formales, y la teoría de Edsger Dijkstra en el artículo *The Cruelty of Really Teaching Computer Science*(la crueldad de realmente enseñar ciencias de la computación), que indicaba que la programación era simplemente, y solamente, una rama de las matemáticas.

En ambas formas, esta noción ha llevado a muchos debates académicos [palabras de comadreja] sobre los pros y los contra de los dos acercamientos y de las posibles estrategias para unir los dos. Algunos consideran las diversas estrategias no como competidoras, sino

como descripciones del mismo problema desde diversos puntos de vista.

Arquitectura

Un computador corriendo varios componentes de software con frecuencia es llamado un servidor de aplicaciones. Usando esta combinación de servidores de aplicaciones y componentes de software es usualmente llamado computación distribuida. La usual aplicación del mundo real de esto es por ejemplo el software de aplicaciones o de negocios.

Modelos[editar]

Un modelo de componentes es una definición de estándares para la implementación, documentación y el despliegue de componentes. Ejemplos de modelos de componentes son: el modelo Enterprise Java Beans (EJB), el modelo COM+ (modelo .NET), el modelo de componentes Corba. El modelo de componentes especifica como deben ser definidas las interfaces y los elementos que deben ser incluidos en una definición de interface.[7]

Tecnologías

- Tecnologías de Objeto de negocio
    - Newi
- Frameworks basados en componentes para dominios específicos
    - Earth System Modeling Framework (ESMF)
- Programación orientada a componentes
    - Paquetes definidos por la plataforma de servicios OSGi
    - Common Component Architecture (CCA) – Foro de arquitectura de componentes comunes, Software de componentes científico/HPC
    - TASCS – SciDAC Center for Technology for Advanced Scientific Component Software (Centro para la Tecnología para el Avance del Software Científico de Componentes)
    - Lenguaje de programación Eiffel

    - Enterprise JavaBeans de Sun Microsystems (ahora Oracle)
    - Programación basada en flujos
    - Modelo de componentes fractal de ObjectWeb
    - Framework de componentes MidCOM de Midgard y PHP
    - Oberon, Component Pascal, y BlackBox Component Builder
    - rCOS Método de diseño de manejo de modelo basado en componentes diseñado desde UNU-IIST
    - SOFA component system de ObjectWeb
    - El espacio de nombres `System.ComponentModel` en el Microsoft .NET
    - Unity3D desarrollado por Unity Technologies
    - UNO de la suite de oficina OpenOffice.org
    - Visual Component Library (VCL) y Component Library for Cross Platform (CLX) de Borland y la biblioteca libre similar LCL de Lazarus.
    - Visual Basic Extension, OCX/ActiveX/COM y DCOM de Microsoft
    - XPCOM de Mozilla Foundation
- Tecnologías de documentos compuestos
    - Active Documents en Oberon System y BlackBox Component Builder
    - Bonobo (component model), una parte de GNOME
    - Fresco
    - KPart, la tecnología de documentos compuestos de KDE
    - Object linking and embedding (OLE)
    - OpenDoc
- Componentes de software de computación distribuida
    - .NET Remoting de Microsoft
    - 9P Protocolo distribuido desarrollado por Plan 9, y usado por Inferno y otros sistemas.
    - CORBA y el CORBA Component Model del Object Management Group
    - D-Bus de la organización freedesktop.org
    - DCOP de KDE (obsoleto)
    - DCOM and later versions of COM (and COM+) from Microsoft
    - DSOM y el IBM System Object Model de IBM (ahora deshechado)
    - ICE de ZeroC

    - Java EE de Sun
    - Universal Network Objects (UNO) de OpenOffice.org
    - Servicios web
        - Representational State Transfer
    - Zope de Zope Corporation
- La programación genética enfatiza la separación de algoritmos de la representación de datos
- Lenguajes de descripción de interfaces (IDLs)
    - Open Service Interface Definitions (OSIDs)
    - Algunas partes de COM y de CORBA
    - Platform-Independent Component Modeling Language
    - SIDL – Scientific Interface Definition Language
        - Algunas partes del lenguaje de Babel. Sistema de interoperabilidad de lenguaje de programación científica (SIDL y Babel son tecnología nucleo de CCA y SciDACTASCS Center – ver arriba.)
    - SOAP IDL del World Wide Web Consortium (W3C)
    - WDDX
    - XML-RPC, el predecesor de SOAP
- Componentes de frameworks Inversion of Control (IoC) y Plain Old C++/Java Object (POCO/POJO)
- Tuberías y filtros
    - Unix sistema operativo