

Ayudas automatizadas para el desarrollo de “software” (Herramientas CASE). Tipos. Estructura. Prestaciones.

TEMA 58 (51 SAI)

ABACUS NT

Índice

1. Introducción

1.1. Prestaciones

1.2. Tipos

2. Componentes CASE

2.1. Integración

3. El Diccionario de Datos

4. Case y el Ciclo de Vida del Software

5. Metodologías de desarrollo

5.1. Principales Metodologías de Desarrollo

5.2. Enfoques de desarrollo de software

6. Conclusión.

6.1. Relación con el sistema educativo

7. Bibliografía

1. Introducción

A finales de los años setenta hay un fuerte tirón en la demanda de aplicaciones institucionales de gran complejidad con requerimientos muy exigentes a veces difíciles de identificar. Estos dos factores, aumento de demanda y aumento de complejidad, contribuyen a una importante crisis del software, que no es capaz de dar solución con las herramientas y metodologías existentes a las peticiones de los usuarios, a esto se le puede sumar el estado en que se encontraba el hardware.

A partir de estos momentos hay una evolución en varias líneas o frentes:

- Surgen nuevas técnicas metodológicas como superación del ciclo de vida clásico (técnicas estructuradas).
- Empiezan a surgir nuevas herramientas de desarrollo basadas en el ordenador.
- El hardware evoluciona, aparecen nuevos entornos, máquinas cada vez más pequeñas y a la vez más potentes sobre las que son posibles nuevos sistemas operativos y nuevos entornos de desarrollo basados en las nuevas metodologías y herramientas.

En este escenario en los años ochenta nace el término **CASE**, Computer Aided Software Engineering, o Ingeniería de Software Asistida por Ordenador, una evolución dentro del desarrollo de sistemas de información que pretende ser la solución a la crisis del software tal y como la hemos descrito anteriormente, es decir, en el entorno institucional con aplicaciones cada vez más complejas.

CASE no es identifiable con una herramienta sofisticada ni con una determinada metodología, si hubiera que definirlo podríamos decir que es una nueva "filosofía" que intenta conducir el desarrollo de software hacia una disciplina real de ingeniería, filosofía porque está por encima de una simple agregación de herramientas y métodos, ingeniería porque está a favor de métodos disciplinados y bien definidos que poco tienen que ver con la intuición y/o la artesanía. Es evidente que un cambio de filosofía lleva más tiempo que el aprendizaje de una determinada técnica y por supuesto un alto volumen de inversiones.

Desde un punto de vista más práctico, CASE puede percibirse, en principio, como:

- La automatización del desarrollo del software.
- La informatización de la informática.
- La redefinición del entorno software.

Profundizando más podemos estudiar CASE basándose en sus objetivos, entre los que podemos citar:

- Conseguir la aplicación práctica de las metodologías estructuradas.
- Mejorar la calidad del software producido.
- Acelerar el proceso de desarrollo.

- Facilitar la reutilización del software.
- Crear estándares de documentación.
- Disminuir y simplificar el mantenimiento de programas.
- Incrementar la "portabilidad" de las aplicaciones.
- Realizar prototipos en entornos complejos.

Para conseguir sus objetivos, CASE se basa tanto en el nuevo hardware, mucho más potente, como en las nuevas metodologías y herramientas, redefiniendo, como se ha dicho, un nuevo entorno de desarrollo, basado fundamentalmente en la utilización de metodologías actuales, herramientas CASE y el diccionario de datos integrado.

1.1. Prestaciones

Automatizando nuestras herramientas podemos aumentar sensiblemente los beneficios derivados del uso de las mismas, esta es definitiva la idea que da origen a este tipo de herramientas, generadores de aplicaciones, códigos y pantallas, sistemas de documentación automatizada, sistemas de diccionarios de datos...

Los beneficios son múltiples, todos los que teníamos y otros entre los que se podrían citar:

- Disminución de tiempo que trae consigo cualquier automatización, cabe pensar, por ejemplo, en el desarrollo por prototipos.
- Automatización de tareas rutinarias como el dibujo de diagramas en el caso del Análisis Estructurado.
- Garantizar procesos consistentes asegurándonos que ciertos procedimientos cumplan las reglas establecidas, como por ejemplo la explosión de los diagramas de flujo de datos.
- Registro de los detalles del sistema, se pueden capturar y almacenar los datos y detalles de un sistema durante su desarrollo con objeto de recuperarlos y procesarlos con distintas finalidades en sucesivos momentos del proyecto o, incluso, en proyectos diferentes.

1.2. Tipos

En general, podemos clasificarlas en tres categorías:

- Herramientas de alto nivel.
- Herramientas de bajo nivel.
- Herramientas integrales e integradas.

Herramientas de alto nivel. Las herramientas de alto nivel, también denominadas de tipo "front-end", automatizan las fases correspondientes del proyecto, ayudando al analista en la preparación de especificaciones y descripciones del sistema, proporcionando, en general, un soporte gráfico, del que podrían ser un ejemplo representativo los diagramas de flujo del sistema.

Herramientas de bajo nivel. Este tipo de herramientas, denominadas también de tipo "back-end", ayudan al analista a describir los algoritmos y lógica de programas, descripción física de datos e interacciones con los dispositivos de entrada y salida, convirtiendo los diseños lógicos del software

en código, en un lenguaje de programación que, en definitiva, conforma la aplicación informática, por este motivo también se conoce a estas herramientas como de “programación asistida por ordenador”.

Herramientas integradas. La actividad del análisis es un todo que comienza en los requerimientos del usuario «traducidos» a especificaciones de «alto nivel», dando lugar a un proceso de descomposición que finaliza en unas especificaciones de «bajo nivel» programables, por tanto, lo deseable sería una herramienta que combinara los dos tipos vistos hasta ahora.

En general, las herramientas de tipo integral han sido poco corrientes, quedando como responsabilidad para el analista el paso del “front-end” al “back-end”. Es evidente que la integración, además de ahorrar tiempo, daría seguridad a la transición alto-bajo nivel.

Las herramientas integrales surgen generalmente asociadas con metodologías de desarrollo concretas y/o vinculadas a un determinado fabricante, la clave de la transición suele estar en el *diccionario de datos*, pieza clave en este entorno, así como en el ambiente CASE.

2. Componentes CASE

A continuación, se exponen, con excepción del diccionario de datos que, a pesar de que puede englobarse como un componente más, debido a su importancia y trascendencia, se tratará en un punto aparte, los componentes más comunes de una herramienta CASE:

- **Utilidades de «diagramación».** Las facilidades para producir diagramas de flujo de datos son fundamentales como apoyo a la metodología de análisis estructurado, las herramientas de diagramación ofrecen la capacidad de dibujar diagramas, guardando todos los detalles de forma interna, como soporte al análisis y documentación de requerimientos de la aplicación. Las herramientas CASE suelen incorporar uno o varios métodos de análisis estructurado.

La capacidad de reutilizar, cambiando y volviendo a dibujar los diagramas facilita enormemente una actividad que los analistas consideran aburrida y pesada.

- **Generadores de interfaces.** Basándose en la facilidad de crear menús y pantallas para la demostración del sistema, así como formatos de informes, los generadores de interfaces son fundamentales en la preparación de prototipos de este entorno, así como en los demás métodos de desarrollo.
- **Generadores de código.** De acuerdo con las especificaciones de bajo nivel, los generadores de código obtienen código ejecutable, si bien esta conversión aún no está muy perfeccionada.

En conexión con el diccionario integrado se puede conseguir además código reutilizable, de manera que, ante un cambio en las especificaciones, se pueda volver a generar el código a partir de los detalles registrados en el diccionario.

- **Utilidades de administración.** Algunas herramientas CASE ayudan al jefe de proyecto a planificar y llevar un control sobre todo el proceso de ejecución temporalizando y asignando recursos a ellas, también ayudan en la realización de informes.

2.1. Integración

Aunque lo deseable es que las herramientas CASE sean integrales, algunas se presentan como elementos discretos considerables por separado y otras como un grupo integrado de herramientas. Comercialmente esto se conoce como “toolkit” y “workbench”, respectivamente.

La integración puede conseguirse de tres formas en general:

- Mediante una interfaz uniforme para todas las actividades.
- Mediante la transferencia de datos entre los distintos componentes.
- Creando enlaces, bien manuales o automáticos, bajo la responsabilidad del analista.

En cualquier caso, es importante señalar que la integración siempre pasará por el diccionario de datos.

3. El Diccionario de Datos

El diccionario de datos es un **catálogo, depósito de información o repositorio**, que contiene información amplia de todos los componentes del sistema, datos, procesos, flujos de datos, entidades, etc.

Está basado en una arquitectura de bases de datos fácilmente accesible aportando funciones básicas para el uso de la herramienta en general y facilidades sobre el uso del propio diccionario en particular.

Entre las *funciones* que aporta se pueden citar:

- Integración de componentes.
- Compartición de información.
- Estandarización de la documentación.
- Generación de la documentación.
- Generación de código.
- Reutilización y control de proyectos.
- Gestión y control de proyectos.

Entre las *facilidades* que puede proveer están:

- Control de seguridad.
- Concesión de privilegios.
- Bloqueos.
- Auditoría.
- Control de cambios.

4. Case y el Ciclo de Vida del Software

ISO/IEC 12207 (2017) - Information Technology / Software Life Cycle Processes es el estándar para los procesos de ciclo de vida del software de la organización ISO.

La creación de software consta generalmente de una serie de pasos claramente diferenciados:

1. **Análisis de requisitos:** se recopila documentación, se estudian los **requisitos** del usuario final (qué se va a implementar, cuáles son las necesidades del usuario). Se suele empezar con una descripción en lenguaje natural de lo que quiere el usuario. Al final de esta etapa tendremos una descripción clara y precisa de qué producto vamos a construir, qué funcionalidades aportará y qué comportamiento tendrá.
2. **Diseño:** una vez que sabemos qué debemos hacer, debemos determinar cómo lo haremos. Estudiaremos el problema y lo descompondremos en problemas más pequeños; definiremos la estructura de la solución, identificando los módulos que hay que implementar y sus relaciones; definiremos los algoritmos a emplear y el lenguaje de programación a utilizar, etc.
3. **Codificación:** se escribe el programa fuente en el lenguaje de programación establecido y se genera el código ejecutable.
4. **Pruebas:** se comprueba que el programa no tenga errores, y que se cumplen los criterios de calidad.
5. **Implementación:** En esta fase es en la que se instala el software en los equipos finales.
6. **Mantenimiento:** el programador se asegura de que el programa siga funcionando, y lo va adaptando también a nuevos requisitos que puedan ir surgiendo.

En el ciclo de vida clásico o convencional, cada proyecto atraviesa por algún tipo de análisis, diseño e **implantación** (implementación, prueba y mantenimiento). El ciclo de vida de proyecto utilizado, por ejemplo, en su organización, puede diferir en una o en todas las formas siguientes:

- La fase de análisis puede dividirse en una fase previa de estudio y otra de análisis (sobre todo en organizaciones en las cuales no aceptan cualquier proyecto).
- Puede no haber fase de análisis de hardware si se cree que cualquier sistema nuevo pudiera instalarse con los ordenadores existentes sin causar mayor problema operacional.
- Las fases de diseño, puede dividirse en dos: diseño preliminar y de diseño de detalles.
- Diversas fases de prueba pueden juntarse en una sola; de hecho, podrían incluirse total o parcialmente de forma paralela a la codificación.

De aquí que **el ciclo de vida del proyecto en una organización puede tener cinco fases o siete o doce**, pero seguir siendo todavía un ciclo de vida estructurado.

5. Metodologías de desarrollo

Una metodología de desarrollo es una recopilación de técnicas y procedimientos estructurados en fases que van a dar como resultado de su aplicación un producto software de calidad.

Al comienzo de la era de la informática las técnicas de desarrollo eran totalmente artesanas y sin metodología ninguna, A finales de los años setenta hay un fuerte tirón en la demanda de aplicaciones institucionales de gran complejidad con requerimientos muy exigentes a veces difíciles de identificar. Estos dos factores, aumento de demanda y aumento de complejidad, contribuyen a una importante crisis del software, que no es capaz de dar solución con las herramientas y

metodologías existentes a las peticiones de los usuarios, a esto se le puede sumar el estado en que se encontraba el hardware.

A partir de estos momentos hay una evolución en varias líneas o frentes:

- Surgen nuevas técnicas metodológicas como superación del ciclo de vida clásico (técnicas estructuradas).
- Empiezan a surgir nuevas herramientas de desarrollo basadas en el ordenador.
- El hardware evoluciona, aparecen nuevos entornos, máquinas cada vez más pequeñas y a la vez más potentes sobre las que son posibles nuevos sistemas operativos y nuevos entornos de desarrollo basados en las nuevas metodologías y herramientas.

En este escenario en los años ochenta nace el término **CASE**, Computer Aided Software Engineering, o Ingeniería de Software Asistida por Ordenador, una evolución dentro del desarrollo de sistemas de información que pretende ser la solución a la crisis del software tal y como la hemos descrito anteriormente, es decir, en el entorno institucional con aplicaciones cada vez más complejas.

5.1. Principales Metodologías de Desarrollo

MétricaV3

Se definía como “metodología de planificación, desarrollo y mantenimiento de sistemas de información” y fue creada por el Ministerio de Administraciones Públicas (MAP) español.

La versión 3 de Métrica fue, y es, la más famosa de las versiones. Y aún la usan muchas empresas, ya que en muchas subcontrataciones de la administración era, y es, un requisito obligado.

Aquella era una época en la que muchos países querían tener su propia metodología: **Merise** en Francia, **SSADM** en Reino Unido y **Métrica** en España.

Agile

Con esta metodología, el desarrollo de software se divide en proyectos más pequeños, por lo que es ideal para los ciclos de lanzamiento corto y finitos, como los requeridos para las actualizaciones de un producto existente. Es conveniente cuando se tiene experiencia en la preparación de documentación, el equipo de desarrollo trabaja estrechamente y, preferiblemente, también en la misma ubicación física.

Scrum

Los proyectos que utilizan esta metodología otorgan un peso considerable a la inteligencia, experiencia y habilidades que los miembros del equipo de desarrollo aportan a la hora de resolver problemas. Las tareas de proyecto se llevan a cabo en ciclos cortos conocidos como **sprints**, muy manejables y adecuadamente priorizados, que permiten mostrar el progreso de forma muy sencilla. Los proyectos más largos se beneficiarían de la estructuración de este método, en comparación con otros modelos de desarrollo de software y uno de los motivos es que los desarrolladores se sienten comprometidos con las metas y responsables del éxito de la iniciativa.

Programación Extrema

Su enfoque de evolución constante resulta muy beneficioso, sobre todo, si se tiene en cuenta que, uno de los procedimientos a aplicar es descomponer las tareas de desarrollo para incluir sólo las actividades críticas. Precisamente por eso resulta un método muy indicado para los casos en que el entorno de desarrollo carece de estabilidad o si los requisitos aplicables al proyecto son inciertos.

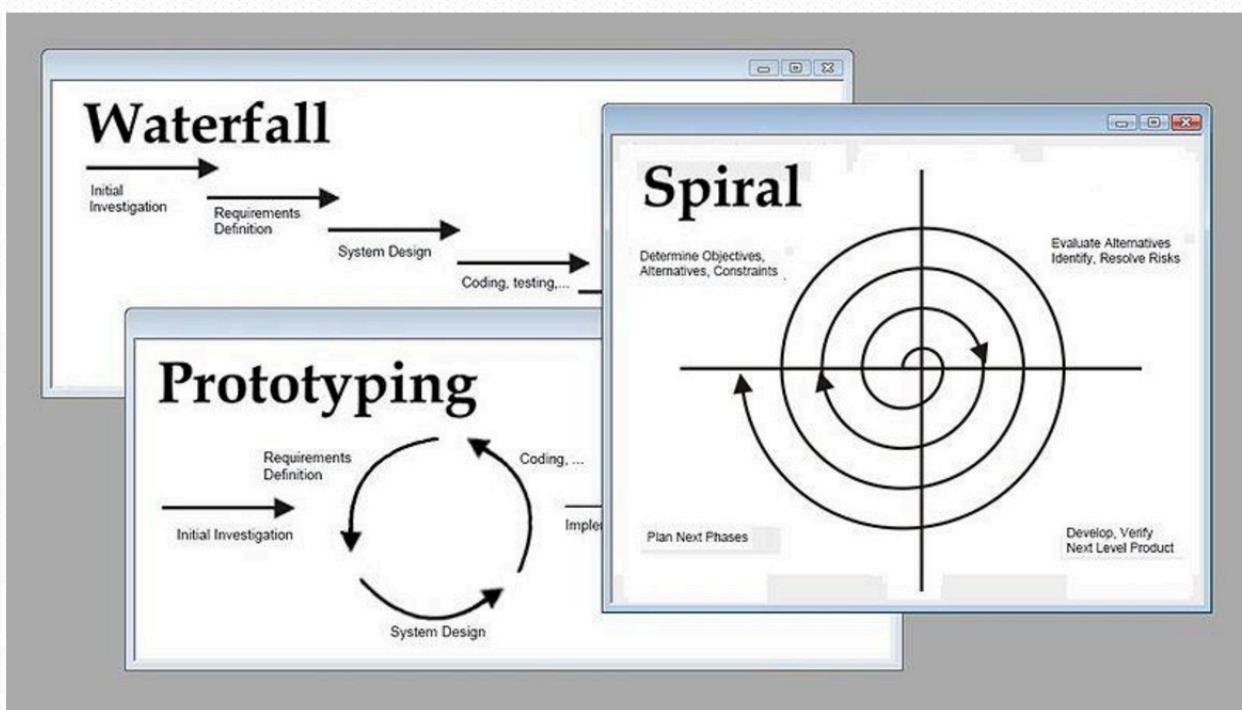
Programación en pareja

Consiste en juntar a dos programadores para que trabajen juntos en una sola estación de trabajo, facilitando que cada desarrollador se centre en diferentes aspectos del proyecto para conseguir que se reduzcan los costes de las pruebas y también los defectos en los programas. Merece la pena apostar por esta alternativa cuando el equipo de desarrollo es pequeño y seguro, y siempre que los desarrolladores estén de acuerdo.

Kanban

en base al sistema just in time, esta metodología trata de evitar los cuellos de botella, mediante una identificación muy precisa de las tareas en curso que funciona muy bien en proyectos que no sean demasiado complejos. Permite ser complementado por otras metodologías y es muy recomendable su utilización cuando se tiene confianza plena en el equipo de desarrollo.

5.2. Enfoques de desarrollo de software



Cada **metodología de desarrollo de software** tiene más o menos su propio enfoque para el desarrollo de software. Estos son los enfoques más generales, que se desarrollan en varias metodologías específicas. Estos enfoques son los siguientes:

Modelo en cascada

Es un proceso secuencial, fácil de desarrollo en el que los pasos de desarrollo son vistos hacia abajo (como en una cascada de agua) a través de las fases de análisis de las necesidades, el diseño, implantación, pruebas (validación), la integración, y mantenimiento.

Los principios básicos del modelo de cascada son los siguientes:

- El proyecto está dividido en fases secuenciales.
- Se hace hincapié en la planificación, los horarios, fechas, presupuestos y ejecución de todo un sistema de una sola vez.
- Un estricto control se mantiene durante la vida del proyecto a través de la utilización de una amplia documentación escrita, así como a través de comentarios y aprobación.

Iterativo

El prototipo permite desarrollar modelos de aplicaciones de software que permiten ver la funcionalidad básica de la misma, sin necesariamente incluir toda la lógica o características del modelo terminado. El prototipo permite al cliente evaluar en forma temprana el producto, e interactuar con los diseñadores y desarrolladores para saber si se está cumpliendo con las expectativas y las funcionalidades acordadas. Los Prototipos no poseen la funcionalidad total del sistema pero si condensa la idea principal del mismo, Paso a Paso crece su funcionalidad, y maneja un alto grado de participación del usuario.

Incremental

Provee una estrategia para controlar la complejidad y los riesgos, desarrollando una parte del producto software reservando el resto de aspectos para el futuro.

Espiral

La atención se centra en la evaluación y reducción del riesgo del proyecto dividiendo el proyecto en segmentos más pequeños y proporcionar más facilidad de cambio durante el proceso de desarrollo, así como ofrecer la oportunidad de evaluar los riesgos y con un peso de la consideración de la continuación del proyecto durante todo el ciclo de vida.

Cada viaje alrededor de la espiral atraviesa cuatro cuadrantes básicos: (1) determinar objetivos, alternativas, y desencadenantes de la iteración; (2) Evaluar alternativas; Identificar y resolver los riesgos; (3) desarrollar y verificar los resultados de la iteración, y (4) plan de la próxima iteración.³

Cada ciclo comienza con la identificación de los interesados y sus condiciones de ganancia, y termina con la revisión y examinación.

Rapid Application Development (RAD)

El desarrollo rápido de aplicaciones (RAD) es una metodología de desarrollo de software, que implica el desarrollo **iterativo** y la construcción de **prototipos**.

Intenta reducir los riesgos inherentes del proyecto partiéndolo en segmentos más pequeños y proporcionar más facilidad de cambio durante el proceso de desarrollo.

Orientación dedicada a producir sistemas de alta calidad con rapidez, principalmente mediante el uso de iteración por prototipos, promueve la participación de los usuarios y el uso de herramientas de desarrollo computarizadas.

Hace especial hincapié en el **cumplimiento de la necesidad comercial**, mientras que la ingeniería tecnológica o la excelencia es de menor importancia.

Otros enfoques de desarrollo de software

Metodologías de desarrollo Orientado a objetos, Diseño orientado a objetos (OOD) de Grady Booch, también conocido como **Análisis y Diseño Orientado a Objetos (OOAD)**. El modelo incluye seis diagramas: de clase, objeto, estado de transición, la interacción, módulo, y el proceso.

Top-down programming, evolucionado en la década de 1970 por el investigador de IBM Harlan Mills (y Niklaus Wirth) en Desarrollo Estructurado.

Proceso Unificado, es una metodología de desarrollo de software, basado en UML. Organiza el desarrollo de software en cuatro fases, cada una de ellas con la ejecución de una o más iteraciones de desarrollo de software: creación, elaboración, construcción, y las directrices. Hay una serie de herramientas y productos diseñados para facilitar la aplicación. Una de las versiones más populares es la de **Rational Unified Process**.

6. Conclusión.

Los paradigmas más actuales de la Ingeniería de Software son el uso de Orientación a Objetos en las etapas de Análisis y Diseño, la construcción de aplicaciones Cliente/Servidor, el uso de interfaces gráficos de usuarios y la preocupación por la calidad del software. La demanda creciente de desarrollo de software en la década de los noventa precisa de herramientas CASE ágiles y potentes.

En el desarrollo de Sistemas de Información se ha pasado de organizaciones centralizadas, basadas en aplicaciones de tipo carácter, a estructuras distribuidas bajo arquitecturas cliente/servidor, con interfaces gráficos de usuario. El entorno de desarrollo debe aprovechar el menor coste de una red local y la accesibilidad de las interfaces gráficas de usuario, pero debe aportar además una flexibilidad a un entorno cambiante, manteniendo una curva de aprendizaje lo más suave posible.

6.1. Relación con el sistema educativo

- GS – DAW - DAM –Entornos de Desarrollo (PES)

7. Bibliografía

Pressman, R. S. Ingeniería del Software. Un enfoque práctico, 3^a edición. Ed. McGraw-Hill, 2000.

Sommerville, I.: Ingeniería de Software. 6^a Edición. Addison-Wesley Iberoamericana, 2002.

Cerrada, J. A.: Introducción a la Ingeniería del Software. 1^a Edición de 2000. Editorial Centro de Estudios Ramón Areces, S.A.

Piattini, M. y otros: Aplicaciones Informáticas de Gestión. RA-MA, 2003.

https://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software