Estructura de un equipo de desarrollo

Desarrollo Colaborativo de Aplicaciones

01: Introducción

02: Estructuras típicas del desarrollo software

03: Dictador Benevolente De Por Vida

04: Conclusiones

Introducción

Equipos de desarrollo software

La estructura organizativa es clave en el desarrollo de software. Es la diferencia entre el éxito o el caos.

Equipos de desarrollo software

No existe una fórmula universal

Cada aplicación tiene sus propios requisitos y esto influencia en la estructura y la dinámica del equipo de desarrollo

La eficiencia como factor clave

El desarrollo de un proyecto debe de ser **fluido y eficiente** para cumplir con los plazos requeridos.

Se usa la **ortogonalidad** como concepto clave para garantizar ésta.

Ortogonalidad

La ortogonalidad significa que los componentes o tareas del sistema son independientes entre sí.

- Cambiar un componente no debería afectar a los demás.
- Reduce el acoplamiento y facilita el mantenimiento.

Ortogonalidad

Un buen consejo para conseguir un sistema ortogonal es separar lo que es la **infraestructura** de lo que es la **aplicación**.

Se establecen equipos o subgrupos para cada componente.

Ejemplo de ortogonalidad

En el desarrollo de un *ecommerce*…

Infraestructura

- Servidor web
- Base de datos
- Balanceador de carga

Aplicación

- Pantalla de búsqueda
- Carrito de la compra
- Pasarela de pago

Asignamos equipos de trabajadores a cada componente

Ejemplo de ortogonalidad

¿Qué pasa si ahora queremos cambiar el motor de la BDD?

- Solo implicamos al equipo responsable de este módulo
- El resto de programadores trabaja con independencia

> **Nota**
>
> Una medida fiable de ortogonalidad es la cantidad de personas se debe implicar por cada cambio que se pida en el proyecto. **A mayor número de implicados, menor ortogonalidad del sistema**

Últimas notas sobre ortogonalidad

> ⚠ **Advertencia**
>
> La ortogonalidad signigica **independencia, no aislamiento total**; la comunicación entre los desarrolladores de los equipos sigue siendo clave. De ahí nacen, en parte, las metodologías de desarrollo ágil que ya conocéis.

Estructuras típicas del desarrollo software

Estructura jerárquica

Estructura propuesta por Harlan Mills en 1972. También se conoce como *estructura quirúrgica*

Define roles con responsabilidades específicas

Estructura jerárquica

Núcleo principal

- Programador o Ingeniero jefe: Planifica, coordina y revisa las actividades técnicas del grupo.
- Personal técnico: Análisis y desarrollo.
- Ingeniero de apoyo: Ayuda al ingeniero jefe y tiene la capacidad

Estructura jerárquica

Especialistas: Unidad de apoyo que asiste al ingeniero en jefe. Por ejemplo, expertos en bases de datos, comunicaciones, etc.

Estructura jerárquica

Especialistas: Unidad de apoyo que asiste al ingeniero en jefe. Por ejemplo, expertos en bases de datos, comunicaciones, etc.

Bibliotecario de software: Desarrollador que cataloga e indexa los módulos u objetos reutilizables.

Estructura descentralizada democrática

- No tiene un jefe permanente. Las tareas se coordinan por un responsable y siempre tienen un corto plazo.
- Los líderes de tarea se sustituyen por cada iteración.
- Las decisiones y los enfoques se toman por **consenso**.
- La comunicación de los miembros del equipo es horizontal.

Ejemplos

- **Proyecto Apache HTTP Server**: La Apache Software Foundation coordina por votación y consenso en mailing lists y foros.
- **Debian Project**: Comunidad global de voluntarios, las decisiones se toman en base a debates y votaciones entre los desarrolladores.

Estructura descentralizada controlada

- Jefe principal y jefes secundarios.
  - El jefe principal coordina tareas específicas.
  - Los jefes secundarios se encargan de subtareas.
- Comunicación horizontal entre subgrupos.
- Comunicación vertical a través de una jerarquía de control.

Ejemplos

- **Mozilla Firefox**: Hay un “release manager” y líderes de áreas, pero cada subequipo trabaja con relativa autonomía.
- **Kubernetes**: Proyecto CNCF con “SIGs” (Special Interest Groups) liderados por coordinadores, supervisados por un comité técnico principal.
- **Proyectos de videojuegos grandes** (conocidos como AAA): Se designan directores técnicos de áreas (arte, IA, sonido), coordinados con comunicación cruzada.

Estructura centralizada controlada

- El jefe de equipo se encarga de la solución de problemas a alto nivel y la coordinación interna del equipo.
- La comunicación entre el jefe y los miembros del equipo es vertical.

Ejemplos

- **Apple en la era Jobs**: Decisiones clave y diseño controlados directamente por él, con equipos ejecutando instrucciones precisas.
- **SpaceX**: Elon Musk mantiene control directo sobre prioridades y estándares técnicos, supervisando los avances de cada equipo.
- **Proyectos militares o aeroespaciales**

¿Cuál es la diferencia entre la estructura jerárquica y la centralizada controlada?

Criterios a tener en cuenta

Podemos fijarnos en diversos factores a la hora de estructurar un equipo de desarrollo:

- Grado de dificultad del problema.
- Modularidad permitida en el problema.
- Condiciones humanas del equipo.
- Grado de comunicación requerido.
- Fecha de entrega del producto.

Otros paradigmas

- Paradigma cerrado: Jerarquía de autoridad similar al equipo Centralizado Controlado. Es útil en la producción de nuevo software similar a otro existente.
- Paradigma aleatorio: El equipo se estructura de manera libre en función de la iniciativa individual de los miembros. Útil cuando se requiere innovación.
- Paradigma abierto: Estructura el equipo según una mezcla de los dos anteriores de manera que se consigan algunos de los controles asociados con el paradigma cerrado así como la innovación que aporta el paradigma aleatorio.
- Paradigma sincronizado: Las “partes” del problema nos sirven para organizar los miembros del equipo, los cuales suelen trabajar en constante comunicación.

Dictador Benevolente De Por Vida

¿Quién manda en el *open-source*?

Los proyectos *open-source* se mantienen a través de una comunidad activa y diversa.

No implica que **carezca de liderazgo**

Benevolent Dictator For Life

El Dictador Benevolente De Por Vida (**BDFL**) fue un título concebido por Guido Van Rossum en 2018.

Benevolent Dictator For Life

El BDFL es un título sarcástico que se le otorga a algunas personas en la comunidad de código abierto.

- Son los encargados de velar por el proyecto software que tutelan, de rodearse de las personas que consideren para abarcar todos los aspectos de ese proyecto.
- Es *importante* que sean originadores del proyecto.
- El dictador benevolente puede delegar funciones y responsabilidades pero también toma las decisiones finales en un momento determinado.

Benevolent Dictator For Life

BDFL famosos:

- **Linus Torvalds**: Kernel de Linux
- **Guido van Rossum**: Python
- **Jimmy Donal "Jimbo" Wales**: Wikipedia
- **Alexandre Juliard**: Wine

Benevolent Dictator For Life

Los BDFL proporcionan una visión global y centralizada sobre las direcciones del proyecto. No obstante, el desarrollo de software libre permanece siempre descentralizado. Sin embargo, son figuras importantes para la toma de decisiones globales del proyecto.

Riesgos de los BDFL

Una cuestión de ego

Riesgos de los BDFL

Una cuestión de ego

Riesgos de los BDFL

Los BDFL **no** siempre son de por vida

- Guido Van Rossum abandonó Python sin dejar sucesor para no sesgarlo a sus ideas
- Los BDFL de Django se apartaron de la posición para permitir que el proyecto creciese

Un *software* open-source debe constituirse con una comunidad fuerte e independiente.