Diseño de Interfaces

Edgar Quiroz

9 de marzo de 2021

Índice

1. UML

En cualquier intento colaborativo para desarrollar software debe haber una forma de **especificar** que se va a hacer. Hay dos casos extremos potencialmente problemáticos. Primero en **equipos grandes** y multidisciplinario. Aquí la comunicación directa no es práctica, y las diferentes áreas de todos no permiten usar lenguaje técnico. El otro caso sería cuando hay un **proyecto grande**, tal que no es factible que todos los miembros del equipo entiendan todo el sistema.

Para resolver esto se creó **UML** (Universal Modelling Language). Es un estándar de decenas de diferentes tipos de diagramas, cada uno de ellos especializado en **especificar un aspecto** del sistema de software de manera **simple y no técnica**

Nota: al iniciar un trabajo en una empresa de software, pedir toda la documentación disponible para entender como funciona el proyecto donde vas a trabajar.

1.1. Diagramas de clases

Permiten modelar parte del sistema de software usando **orientación a objetos**. En breve, diferentes elementos y actores se agrupan en **clases**. Cada clase tiene ciertas caracterísicas propias, y puede interactuar con las otras clases a través de **métodos**.

1.2. Diagramas de flujo

Permiten modelar un **algoritmo**. Se pueden tomar decisiones, asignar variables y definir ciclos. Estos algoritmos normalmente reflejan un **proceso** que se quiere automatizar en el sistema.

Todos los sistemas tienen un flujo de actividades. A grandes rasgos siempre es útil modelar el flujo, sin importar el tamaño del proyecto.

1.3. Diagrama de secuencia

Describe **como y en que orden** interactuan las diferentes entidades que participan en un flujo. Pueden incluir tanto flujos **exitosos**, flujos **fallidos** o flujos **excepcionales**.

1.4. Diagrama de despliegue

Describe la **implementación física** del sistema, i.e. los servidores, y como estén conectados entre sí. Si se usa un servicio de alojamiento, generalmente tiene herramientas para crear estos diagramas, como es el caso de **AWS**.

1.5. Diagramas de casos de uso

Describe la **funcionalidad específica** (paso a paso) de un sistema para los diferentes **tipos de usuario**. Tener bien identificados todos las funcionalidades permite crear interfaces más especializadas.

Determinar las funcionalidades requiere un **análisis de requerimientos**.

Nota: el análisis de requerimientos es un buen punto de partida para diseñar una interfaz, aunque es algo bastante posterior.

2. Interacciones Humano-Máquina

Estudia el diseño, evaluación e implementación de sistemas computacionales interactivos. En otras palabras, mecanismos para comunicarse con una máquina.

Interacción: acción recíproca entre dos o más agentes

Gran parte del éxito de un sistema está dado por que tan **seguro**, **útil**, **eficaz** y **usable** es. Esto toma lugar en **interfaces**.

- **Seguro**: robusto a ataques
- **Útil**:
- **Eficaz**: hace el software productivo y ahorra costos de capacitación.
- **Úsabilidad**: se sencillo e intuitivo.

Tarea: diseñar reglas para jugar serpientes y escaleras usando únicamente texto.

Para llegar a mecanismos con estas características, en general se estudia como interactúan las personas interractúan con su ambiente y con otras personas, e intentar replicarla.

2.1. Algo de historia

Un ejemplo sería la **ENIAC**. Ésta se configuraba con más de 6000 interruptores. Para usarla había que tener un conocimiento muy profundo de la aquitectura de la máquina, y modificaciones a su funcionamiento tomaba semanas.

Una mejora a esto fue la introducción de **teletipos** (con tarjetas perforadas). Esto facilitaba la forma de modificar el software de la máquina.

En 1968, Xerox creó la primera **interfaz gráfica**. Poseía un mouse, ventanas, íconos y menúes.