Disclaimer: Este apunte no es autocontenido y fue pensado como un repaso de los conceptos, no para aprenderlos de aquí directamente.

1. Michael Jackson

1.1. Dominio de aplicación

Problema de desarrollo: Problema de construir una máquina (ej: word=máquina de escribir)

Dominio de aplicación: Parte de mundo en el que la máquina se hace sentir ⇒ lugar en el que es evaluada/aprobada

Identificarlo es necesario para enfocarse en los requerimientos. Puede incluir cosas concrectas y abstractas (reglas de negocios). Hay que prestarle mucha atención, entenderlo **es** entender lo que hay que hacer. Da la base para elegir apropiadamente el **marco** y los **métodos** para la solución.

Ej: Puede haber mecanismos preexistentes, que pueden tener que ser incorporados, modificados o eliminados, y es necesario entenderlos para hacerlo.

El dominio de aplicación es el material para trabajar, un **requerimiento** es qué hacer con él.

Se debe diferenciar el dominio de aplicación de la máquina a construir (es dificil en muchos casos).

1.2. Dominios

Dominios:
- **Estáticos**
- **Dinámicos:** El tiempo es una dimensión en la cual las cosas cambian. Noción de *estado*.
  - **Inertes:** El estado solo cambia a las órdenes del usuario (procesador de texto)
  - **Reactivos:** El estado cambia según sus propias reglas, pero sólo reaccionando ante una órden (base de datos)
  - **Activos:** El estado cambia todo el tiempo por sí mismo. (sistema operativo)
    - **Autónomo:** No controlable (personas de la calle). Considerar cambios la máquina no va a medir.
    - **Biddable:** Controlable por sugerencia (usuarios posibles de instruir)
    - **Programable:** Controlable totalmente. En general se convierte en parte de la máquina

  Estas categorizaciones depende mucho de los límites que se establezcan para el dominio.

Ej: Estados de una construcción vial, en c/u hay que desviar el tráfico en forma diferente, pero las rutas en general son estáticas para otros procesos (deliveries).

Para ver un dominio dinámico como estático saco una foto.

Unidimensional vs multidimensional: Comunicación punto a punto, texto vs grafos, imagen, teleconferencia

Tangibles: De mundo real. Inherentemente dificiles de describir porque son informales. Introducen reglas de tiempo real.
Intangibles: Abstractos. Son casi siempre parte de la máquina a construir

Interacciones de dominio: Dividimos el dominio de aplicación en varios dominios para modularizar, pero sólo si los **fenómenos** de cada dominio estan muy o enteramente separados.

Fenómenos compartidos: Lugar de interacción de los dominios (intersección). Los fenómenos compartidos limitan la independencia de cada dominio, pero esta limitación debe ser peque na en comparación con el tama no del dominio para que la partición tenga sentido.

Máquina: Interactúa con algunos dominios de la partición. Debe conectar el grafo de interacción de los dominios (un dominio que no tiene un camino de interacción hacia la máquina indica un error en el modelo).

Se debe entender cada dominio y las interacciones (**fenómenos compartidos**) entre éstos y de éstos con la máquina.

Contexto del problema: Dominio de aplicación ∪ Máquina = $\{D_1, ..., D_n\}$∪ Máquina donde los D_i son los dominios en los que se parte el dominion de aplicación

Prinicipio de relevancia de dominio: Todo lo que es relevante a un **requerimiento** debe aparecer en alguna parte del dominio de aplicación. Esto implica que haya partes importantes del dominio de aplicación no conectadas directamente a la máquina. Este principio se usa para establecer la partición como los sustantivos importantes que aparezcan en los **requerimientos**.

1.3. Requerimientos

La ingeniería de software tradicionalmente solo distinguía entre dentro y fuera de la máquina. Esto era un problema porque el cliente (eventualmente) sabe poco de software y el equipo de desarrollo poco del negocio del cliente. Para poder definir los **requerimientos** se precisa **incluir el dominio de aplicación dentro del espectro del problema**.

La máquina puede asegurar los requerimientos solo a través de los **fenómenos compartidos** con los dominios $\in$ dominio de aplicación.

Un evento compartido sucede en 2 dominios ó en un dominio y la máquina, aunque de cada lado se ve distinto (de un lado la suba de temperatura y del otro la se nal eléctrica del sensor).

Gap entre requerimientos y lo que la máquina puede asegurar: Es producido por el hecho de que los requerimientos no están formulados exclusivamente en términos de fenómenos compartidos y las cosas que pasan en la parte de "afuera" del dominio de aplicación no es controlable directamente por la máquina (e incluso a veces incontrolable). Ej: El requerimiento dice que el usuario escribe una carta, pero éste se niega a usar el teclado.

Especificación: Fenómenos compartidos (la intersección) entre el dominio de aplicación y la máquina. Se derivan de los requerimientos, que son un subconjunto del dominio de aplicación.

La idea es que entre la computadora y el programa (la máquina) impliquen la especificación que a su vez implica la satisfacción de requerimientos (idealmente, ya que los requerimientos en muchos casos pertenecen al mundo informal y pueden tener errores).

1.4. Especificaciones

Especificación: Como fenómenos compartidos, no necesariamente tienen sentido en el entorno (dominio de aplicación) porque **no son requerimientos** sino que fueron derivados de ellos.

Las especificaciones no pueden ser solamente abstractas porque al pertenecer a la intersección pueden contener cosas del dominio de aplicación, o sea, del mundo real. Si la especificación es solo abstracta, esto implicará que el problema que se está atacando sea abstracto.

2. No silver bullet

Problemas de la ingeniería de software:

Esencia: Problemas inherentes

Accidente: Problemas coyunturales

Programa: Es una representación conceptual, lo dificil es especificarla, dise narla y testearla, no programarla.

Problemas esenciales:

Complejidad: Todas las partes son diferentes (si son iguales, se hacen comportamiento compartido) lo cual redunda en un enorme número de estados (esto hace dificil el testing y los controles como el de seguridad).

Conformidad: Hay que lidiar con muchas cosas diferentes (reglas, procesos existentes) hechas por diferentes personas o instituciones a lo largo de mucho tiempo, lo que provoca que no necesariamente hay una armonía entre ellas. Otras disciplinas (física) buscan una armonía que puede ser dificil de encontrar, pero al menos existente.

Cambiabilidad: Al existir la posibilidad de hacer cambios baratos (en el sentido material) éstos son exigidos, a diferencia de en otras industrias que el producto entero es reemplazado por uno nuevo. El programa tiene que ser configurable, y adaptable como mínimo al hardware sobre el cual está corriendo, que cambia muy rápidamente.

Invisibilidad: La inherente realidad abstracta del software hace que sea más dificil hacer un

diagrama que lo represente o, al hacerlo, el gap entre diagrama y realidad es mayor que entre un plano y el edificio.

Métodos anteriores: (atacaron accidentes, no escencia, por lo que su ganancia es limitada)
Lenguajes de alto nivel: Solo mejoran el desarrollo de la máquina
Multitarea: Solo mejora turnaround
Unificación de herramientas: Solo mejora el desarrollo

Lo dificil acerca de la ingeniería de software es decidir que se quiere decir, no decirlo.

La inteligencia artificial no sirve de todo, porque para hacer un sistema experto es necesario tener un experto humano previamente.

Programación automática es equivalente a más alto nivel, siempre se debe escribir el input (la especificación).

Lo importante es debuguear la especificacion, asi que los verificadores automáticos no sirven.

3. Máquinas de estado finitas (FSMs)

FSM: Nodos y transiciones con label

Deadlock: Estado sin trancisiones saliendo de él. No necesariamente implica un mal modelado (puede indicar una finalización del proceso).

Composición paralela $A||B$: Los nodos son pares ordenados de estados. Se avanza en la coordenada necesaria cuando uno avanza para las etiquetas no compartidas o cuando **ambos** avanzan en las etiquetas compartidas. Esto sirve para "comunicar" distintas FSMs.

Extensiones:
Agregar a las etiquetas **condiciones** o **acciones** sobre variables.
Agregar a los nodos condiciones (solo se puede reposar en ellos si se cumple la condición).

Temporizadas: Se agrega el tipo de variable `Timer`

Se puede decidir hacerlas no determinísticas.

Las FSMs especifican aspectos dinámicos (el mapa conceptual solo estáticos). Se pueden bindear al modelo y luego usar OCL.

4. Descripciones y análisis de software

Ingeniería de software: Aplicación del conocimiento científico al proceso de desarrollo.

Proceso de desarrollo: Forma de transformar requerimientos en productos de software.

Calidad: Depende del punto de vista (programador, usuario, manager)

No silver bullet, pero buenas prácticas probadas mejoran el funcionamiento.

Requerimientos: Funcionales y no funcionales, éstos últimos pueden ser sobre el producto (seguridad, performance) o sobre el proceso (tiempos de entrega, modalidad de trabajo).

El usuario no siempre necesita lo que quiere, se debe intentar mantener el sistema lo mas simple posible.
Definir prioridad para los requerimientos (esencial, importante, deseable).

Análisis de escenarios: Ejemplos de sesiones de interacciones. Criterios de aprobación para testing.
Lenguajes: Casos de uso, diagramas de secuencia, diagramas de actividad, castellano
no son requerimientos, los requerimientos se derivan de ellos.

Casos de uso: Actores, descripción en castellano pero con estructura
Actor: Todo usuario o sistema que interactua con la ma'quina. Una persona en particular puede ser varios actores (tipos de usuario).
Diagrama de casos de uso: Relación entre casos (usa, extiende). Modularización, D&C, cada diagrama es interpretable independientemente.

Los casos de uso son disparados por un actor o por tiempo. Se pueden ir desgranando incrementalmente. Son los servicios a los usuarios y sirven para guiar el desarrollo (dise no, implementación y testing). Forma adecuada de generar modelo que son los requerimientos. Los requerimientos funcionales son un caso de uso, los no funcionales suelen asociarse a alguno(s).

Otros documentos: Glosario, prototipo de interfaz.

Diagrama de colaboración: Describe interacciones entre instancia de actor y de caso de uso

Perfiles de trabajo: Analista (director), especificador de casos de uso, dise nador de GUI, arquitecto

Pasos del proceso de modelado:

1. Identificar actores y casos de uso (títulos)
2. Asignar prioridades a casos de uso
3. Detallar casos de uso (precondición, postcondición, caminos alternativos y prohibidos). Incluye interfase con actores y recursos.
4. Prototipar interfase de usuario
5. Hacer modelo de casos de uso (ayuda a generalizar casos de uso)

Rastro de requerimientos: Al cambiar los requerimientos se analizan los casos de uso afectados en el modelo y se ven que partes son afectadas del dise no y la implementación

Matriz de trazabilidad: Relación de cada caso de uso con su impacto. En las columnas aparecen los distintos aspectos del sistema (módulos, programas, recursos, procesos) y las filas los casos.

Cuantificación: Volúmen afecta solución tecnológica posible.

Confiabilidad: Se debe definir una métrica, siempre asumiendo uso aceptable

Restricciones: Externas (mundo real), internas (reglas del cliente) y de interfaz (con otros sistemas).

5. Arquitecturas de software

6. Resúmen de documentos

- Modelo conceptual
- Caso de uso
- Diagrama de casos de uso
- Diagrama de colaboración
- Glosario
- Prototipo de interfaz
- Máquina de estado finita (FSM)
- Diagrama de secuencia
- Diagrama de actividad
- Diagrama de flujo de datos (DFD)
- Diagrama de entidad relación (DER)
- Diagrama de flujo de trabajo (workflow)