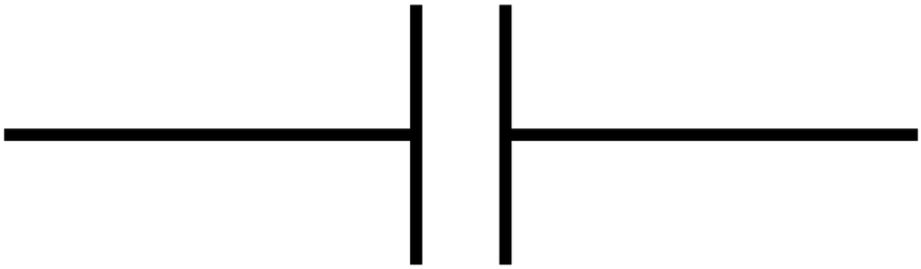


Las entradas del Diseño

Ejercicio Lista de Correo

por Fernando Dodino
aportes de Nicolás Passerini

Versión 1.2 - Marzo 2015

Indice

1 Las entradas del diseño

Si buscamos una definición formal de Diseño, Ralph y Wand intentaron construir un consenso a partir de muchas definiciones posibles:

> ***Design***
> *a specification of an object, manifested by an agent, intended to accomplish goals, in a particular environment, using a set of primitive components, satisfying a set of requirements, subject to constraints;*

Ahora bien, ¿cuáles son las entradas o *inputs* del diseño?

- Los **requerimientos** (*requirements*) son la entrada más obvia y tal vez la más importante. La fuente de los requerimientos es el proceso de análisis, que es también es una especificación: de lo que el sistema debe ser, de sus características externas. El diseño especifica cómo el sistema debe construirse.
    - Ralph y Wand diferencian un subconjunto de los requerimientos denominados **objetivos** (goals). Los objetivos resumen el impacto esperado del sistema en el ambiente.
    - Otra clasificación posible es en requerimientos funcionales y no funcionales. Una buena especificación de requerimientos debe contemplar tanto las cuestiones funcionales como las no funcionales, y el diseño deberá proponer ideas.
    - Una última diferenciación de los requerimientos es identificar (dentro de los requerimientos funcionales) a los que representan las formas en que un usuario puede interactuar con el sistema. A estos requerimientos los denominamos **casos de uso**.

- **Componentes primitivas** (*primitive components*) son los bloques de construcción provistos por la tecnología subyacente. En los niveles más bajos de abstracción del sistema, las abstracciones utilizadas tienden a coincidir con las herramientas provistas por la tecnología, tanto en cuanto a sus bloques primitivos de construcción, como en la forma de combinarlos entre sí.

- **Restricciones** (*constraints*), que pueden tener diferentes orígenes:
    - **Tecnológicas**, tenemos que tener en cuenta qué cosas permite o no permite la tecnología en la que vamos a construir el sistema.
    - **Leyes y otros reglamentos**, incluyendo a los reglamentos internos de la

empresa o licencias de las herramientas que usamos.
    - **Negocio**, el negocio puede imponer restricciones de tiempo de llegada al mercado, costos y beneficios, tiempo de vida del sistema, entre otras.
    - **De arquitectura**, la arquitectura puede imponer restricciones de integridad, correctitud, completitud, constructibilidad o robustez, entre otras.

- **Entorno** (*Environment*). Nos interesa tanto el entorno tecnológico (con qué tecnología contamos) como social (características de las personas que intervienen). Podemos diferenciar dos entornos:
    - **Entorno de desarrollo:** Es el ambiente en el que se diseña, construye y/o prueba el sistema.
        - Desde el punto de vista social debemos tener en cuenta sus conocimientos previos en cuanto a herramientas tecnológicas, conceptuales y metodológicas. Por ejemplo, antes de introducir una tecnología nueva puede ser una buena idea analizar la capacidad del equipo para aprenderla.
        - Desde el punto de vista tecnológico debemos tener en cuenta por ejemplo los equipos con los que contamos para trabajar, si están todos en un mismo lugar o separados, si la comunicación (redes) entre esos lugares es buena o mala, si tenemos licencias de los programas que queremos usar, etc.
    - **Entorno de uso:** El entorno en el que se usa el sistema.
        - Desde el punto de vista social debemos considerar al usuario que utilizará nuestro sistema, por ejemplo al diseñar la interfaz de usuario, o para instalarse el sistema en caso de ser necesario, sus costumbres previas, otros sistemas que usa.
        - Desde el punto de vista tecnológico podemos considerar el hardware del que puede disponer el usuario, su conexión a Internet, etc.

2 Atributos de Calidad

Los atributos de calidad (también cualidades del software) son características no funcionales que se consideran deseables en un sistema de software:

- Simplicidad
- Correctitud, consistencia, completitud.
- Robustez

- Flexibilidad
- Escalabilidad
- Performance
- Seguridad
- Usabilidad
- Facilidad de construcción, etc..

No todos los sistemas de software deben tener en cuenta todos estos atributos o cualidades, algunos serán más importantes que otras dependiendo del sistema, y ciertamente no se pueden maximizar todos a la vez. Diferenciamos cualidades de requerimientos porque algunas de ellas pueden incorporarse como entrada al diseño por un camino distinto al del análisis (por ejemplo, como restricciones de arquitectura o influencias del entorno).

3 Objetivo: diseñar una lista de correo

3.1 Diseño y análisis de sistemas

Hemos visto que una de las principales entradas del diseño son los requerimientos. Entonces el papel de la actividad de análisis es fundamental para poder llegar a un diseño pertinente: tenemos que entender el negocio. ¿Qué es una lista de correo? ¿Cómo funciona?

En el análisis de requerimientos relevamos las necesidades, priorizamos los requerimientos, entramos en contacto con quien necesita la solución, aprendemos cómo funciona el negocio, abstraemos un modelo. Es posible que encontremos fallas en el proceso, o formas de hacer mejor las cosas, es parte del valor que agregamos. Pero por sobre todo, la prosa, diagramas, gráficos y todas las herramientas para comunicar el modelo son información importante para poder tomar las decisiones propias del diseño: dónde pongo los componentes, qué hacen y cómo se relacionan para construir un sistema que de soporte a una lista de correo.

Aquí vemos el resultado de ese relevamiento:

Relevamiento al 10/05/20xx

La lista de correo es una función especial del correo electrónico que permite la distribución de mensajes entre múltiples usuarios de Internet de forma simultánea. En

una lista de correo, al enviar un mensaje a la dirección de la lista, éste llegará a la dirección de todas las personas inscriptas en ella.

El sistema debe contemplar:

Suscripción a la lista
Suscribir un nuevo miembro a una lista de correo. La suscripción puede ser definida como abierta (cualquiera que se suscribe es admitido) o cerrada (los pedidos de suscripción deben ser aprobados por un administrador).

Envío de mensajes
Enviar un correo, recibiendo la dirección de e-mail origen del correo, título y texto. El envío de mensajes a la lista puede definirse como libre (cualquiera puede enviar mensajes a la lista) o restringido a los miembros de la lista. Cada usuario puede tener definida más de una dirección de e-mail, desde las que puede enviar mensajes a la(s) lista(s). De todas las direcciones de e-mail que tenga, una es a la que se le envían los mails.

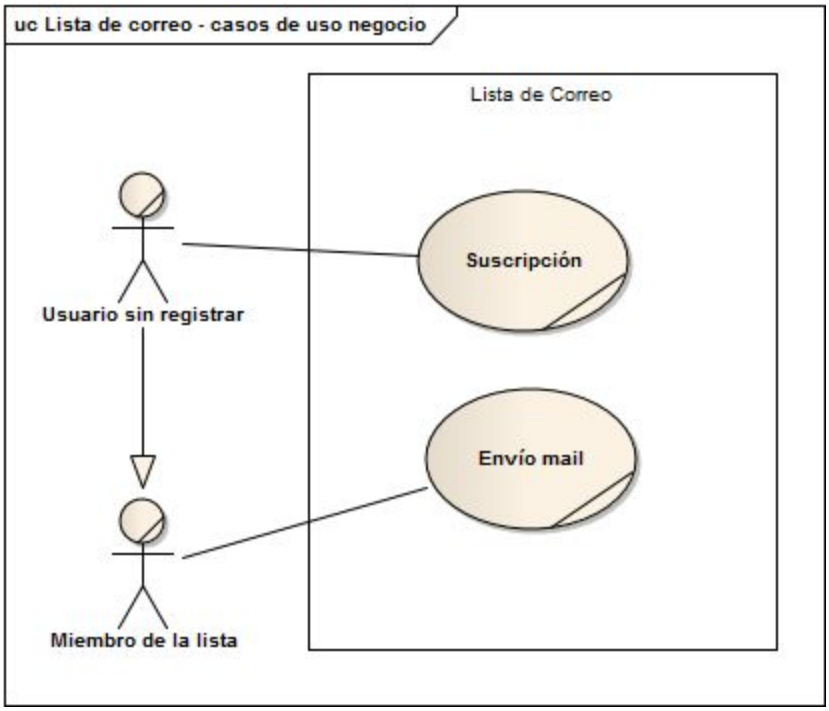
4 Requerimientos

Vamos a estudiar el ejercicio desde dos perspectivas diferentes, para entender cómo entran en juego los requerimientos con el posterior diseño de un problema.

4.1 Trabajo con casos de uso

4.1.1 Casos de uso de negocio

Podemos plantear los siguientes casos de uso de negocio:



¿Quiénes son los actores? Los interesados en el “negocio”, que toman dos formas posibles:

- usuarios que quieren suscribirse (y que podrían enviar mensajes en una lista abierta)
- usuarios registrados

El usuario es **externo a la organización para la cual desarrollamos el sistema**.

Respecto a los casos de uso:

- el diagrama los muestra como una “caja negra”, necesitamos complementarlo con otra herramienta para saber qué es lo que pasa internamente para el negocio[1]
- lo que pasa internamente lo terminan resolviendo
    - uno o varios sectores de una organización,
    - o componentes tecnológicos que incluyen hardware / software
    - con procedimientos manuales, automatizados o mixtos
- no podemos establecer su duración exacta: cuando tenemos una lista de suscripción cerrada, pueden pasar varios días hasta que el administrador acepte la suscripción pedida por el usuario

[1] De hecho es un error frecuente asociar los casos de uso a funciones del sistema e intentar explicar en este tipo de diagrama un flujo procedural. El diagrama de casos de uso son una herramienta de análisis, no es conveniente para definir algoritmos.

4.1.2 Casos de uso de sistema

Por supuesto, podríamos pensar una solución que no necesite resolverse mediante software: tendríamos una oficina comercial donde un usuario ingresa y le pide a un empleado el envío de una carta a todas las personas suscriptas (el soporte de la lista de correo podría ser un cuaderno donde están registrados los interesados). Otro mecanismo posible podría consistir en enviar una carta a un apartado postal. Entonces algún empleado debería tomar la carta, copiarla por algún medio (manual o bien utilizando tecnología de copiado rápido) y finalmente utilizar el servicio de mensajería estatal o privado para que una copia de esa carta llegue hasta las personas suscriptas. Así han funcionado las suscripciones de revistas por décadas.

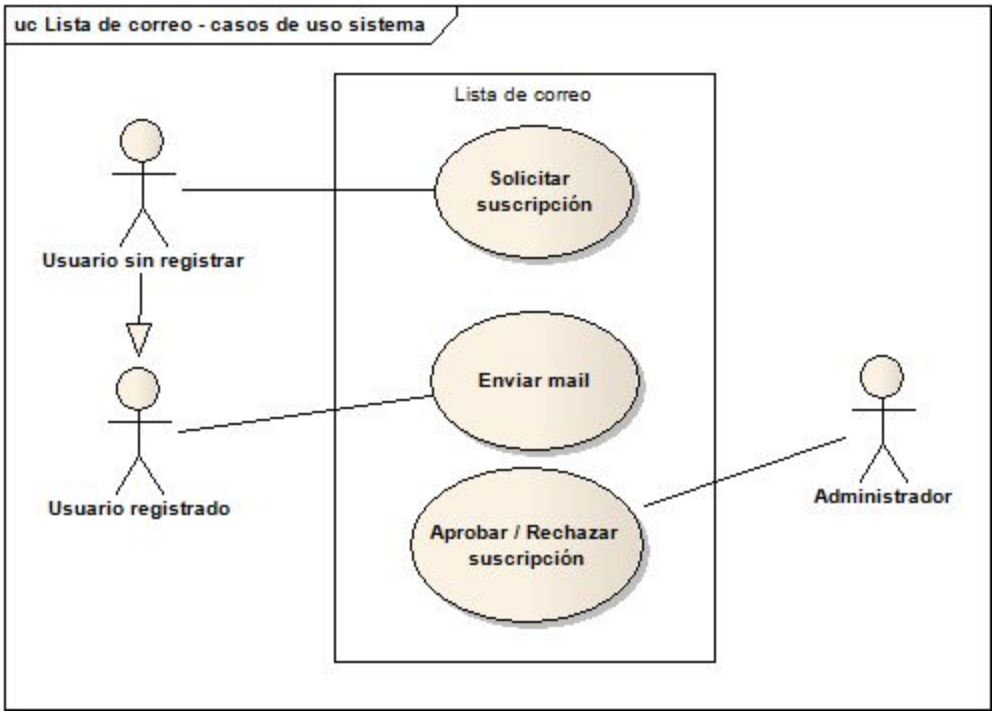
Pero vamos a asumir que queremos resolver este problema a través de un software. Entonces tenemos que graficar un nuevo diagrama de casos de uso, pensando en los límites de nuestro software, en casos de uso que son objetivos de nuestra aplicación y que serán iniciados por perfiles o **actores del sistema**. **Cada caso de uso de negocio se particiona en uno o muchos casos de uso de sistema,** como la suscripción a la lista de correo:

1. El usuario pide la suscripción a la lista (momento m_1)
2. Aquí se plantea una disyuntiva
    a. En el caso de la lista de suscripción abierta, necesitamos un sistema que procese el pedido del usuario y lo agregue al grupo
    b. En el caso de la lista de suscripción cerrada, el sistema **no puede resolver el pedido sincrónicamente**. Se puede enviar un mail al administrador (esto lo tiene que determinar nuestro cliente), *pero* **no podemos agregar al usuario directamente al grupo ni tampoco podemos esperar a que el administrador nos conteste** (no esperamos ningún “input” del teclado ni automatización posible para esta decisión que recae en una persona). Tendremos que diseñar una solución para trabajar este paso intermedio. Pero esto lo resolveremos más adelante, recordando que una buena metodología es la que me lleva a responder las preguntas en el momento adecuado.
3. El administrador revisa las suscripciones pendientes (en otro momento m_2) y aprueba o rechaza el pedido de suscripción.

Para el caso de uso de negocio “Suscripción” tenemos como soporte dos casos de uso de sistema:

- Solicitar suscripción
- Aprobar / Rechazar suscripción

Cada caso de uso es resuelto por un perfil diferente: al solicitar la suscripción estamos hablando de un usuario nuevo. Para procesar la suscripción necesitamos alguien con perfil de administrador.



Resumiendo: los casos de uso de sistema dan soporte a los casos de uso de negocio, son los que automatizan las operaciones que el negocio necesita. La duración de un caso de uso de sistema es puntual[2], y cuando hablamos de actores del sistema estamos hablando del perfil de los usuarios que van a utilizar nuestro software y no de los interesados en el negocio.

4.2 Trabajo con user stories

Las metodologías ágiles proponen una herramienta diferente para trabajar los requerimientos de usuario: las *user stories*. Rápidamente

- Una user story define una característica del sistema que agrega valor para el cliente. Está escrita *por el cliente en prosa*, tiene pocos detalles sobre la funcionalidad pero sí debe explicar claramente cuál es el objetivo que persigue y para qué se necesita.

[2] Alistair Cockburn recomienda establecer el “coffee break test”: una vez que el actor termine cada caso de uso de sistema debería poder irse a tomar un café sin necesidad de recordar nada de lo hecho anteriormente

- Un caso de uso es generalmente más grande que una user story. Podríamos asumir que es más fácil entonces particionar en varios requerimientos la solicitud de suscripción de un usuario nuevo, pero veremos en el ejemplo que hay que tener cierto cuidado con dejar el requerimiento demasiado “en el aire”

El usuario escribe entonces su primer user story[3]:

Como interesado en la lista de correo quiero poder suscribirme a ella para poder enviar mensajes al conjunto de personas que la conforman.

Escribimos los detalles al dorso

Confirmaciones: ● Las listas de suscripción abierta inscriben directamente al interesado ● Las listas de suscripción cerrada deja al interesado en una lista de espera ● El envío de suscripción se hace a partir de un mail que tiene el siguiente formato “suscribir-XXXXXX” donde XXXXXX es el identificador de la lista

Por supuesto, la visión del usuario es una visión de negocio, entonces la historia que está escribiendo está contada desde esa perspectiva. Dijimos antes que un caso de uso englobaba varias user stories, ¿no es contradictorio que nuestro primer ejemplo no cumple esta afirmación? Mmm... lo que sucede es que la user story no está contemplando la separación de momentos entre el pedido de suscripción y su posterior aprobación o rechazo.

Hay varios momentos en donde puede surgir un cambio en la redacción de la historia:

- el usuario mismo detecta que se necesita separar una historia en dos o tres
- en las estimaciones de tiempo el equipo de desarrollo detecta que una historia lleva un gran esfuerzo de construcción[4], entonces puede sugerirle al usuario que

[3] El formato sugerido de cada user story es

- Descripción de la funcionalidad (actor, objetivo y justificación de para qué se quiere)
- Conversaciones, que aclaran dudas o bien aportan *algunos* detalles (como hemos dicho antes, no se intenta formalizar todos los detalles del requerimiento, eso ocurre en el momento en que se encara una user story)
- Pruebas de aceptación, que veremos más adelante

No es intención del apunte estudiar en profundidad la técnica provista por las metodologías ágiles, en todo caso puede verse el capítulo 18 de *Planning Extreme Programming* de Kent Beck, un paper que explica las user stories y las recomendaciones para escribirlas

[4] esto podría significar 4 ó 5 semanas

la divida en dos o más historias. La responsabilidad sobre las historias sigue siendo del cliente/usuario[5]
- en el momento de diseñar nos damos cuenta de que necesitamos un administrador que suscriba al usuario, o bien al desarrollar los casos de prueba.

No importa cuándo ocurra, lo importante es que en ese momento todos estén de acuerdo en que se puede mejorar la redacción de la historia que explica la suscripción:

> Como interesado en la lista quiero poder **enviar un pedido de suscripción** a la lista para poder enviar mensajes al conjunto de personas que la conforman (...)

En los detalles de la primera historia ya aclaramos que “Las listas de suscripción cerrada deja al interesado en una lista de espera”, es importante que todos (cliente y equipo de desarrollo) entiendan lo que esto define.

A su vez, escribiremos una nueva historia en otra ficha:

> Como administrador de la lista de correo quiero poder aprobar o rechazar pedidos de suscripción de una lista cerrada para identificar a las personas que conforman la lista y evitar el ingreso de usuarios automáticos que generen spam

De esta manera logramos mantener la user story lo suficientemente pequeña como para poder hacer una estimación pero además para que construirla tenga sentido para el negocio.

4.3 Cómo impacta en el diseño

Hemos visto dos técnicas diferentes que nos permiten trabajar los requerimientos de los usuarios. Podemos elegir una u otra, o incluso combinarlas, el diseño no depende tanto del formato sino de la capacidad y el compromiso que asumen las personas para entender qué es lo que el negocio necesita.

El objetivo del diseño será cumplir con los requerimientos, pero hay que tener en cuenta que

- es muy frecuente encontrar objetivos que se contraponen, situaciones en las que un ingeniero tiene que sopesar entre dos objetivos y las soluciones que benefician a uno de los objetivos perjudican al otro. Por ejemplo, agregar nuevas

[5] en algunas bibliografías también son sinónimos interesado o *stakeholder*

funcionalidades quita simplicidad al usuario.
- nuestro trabajo es tomar decisiones, para eso hay que saber cada decisión qué cualidades impacta, cómo hago para minimizar el impacto

4.4 ¿Alcanza con los Casos de Uso?

Los Casos de Uso y las User Stories proporcionan una forma de capturar los requisitos funcionales centrándose en el valor añadido para el usuario, y se utilizan en muchas metodologías para dirigir el proceso de desarrollo. En términos de Booch, Jacobson y Rumbaugh[6], el diseño, la implementación y la prueba pueden planificarse y organizarse en función de los Casos de Uso. De esta forma, cada iteración incorpora (entre otros posibles objetivos) un conjunto de nuevos Casos de Uso para el sistema.

Sin embargo, los casos de uso no son la herramienta más adecuada para la descripción de requerimientos que afectan a múltiples casos de uso o incluso a la totalidad del sistema. Por ejemplo, si una regla de seguridad debe verificarse en cada caso de uso, es conveniente encontrar una forma de expresarla que no obligue a agregarla como primer paso en cada caso de uso ("el usuario debe estar logueado"). Hacerlo de esa manera suele ser inadecuado porque:

- obliga al Analista a escribir muchas veces lo mismo.
- en caso de cambiar el requerimiento o agregarse nuevos requerimientos similares es difícil de garantizar la correctitud general, al tener que corregir muchos o todos los casos de uso.
- escribir muchas veces lo mismo además multiplica la posibilidad de introducir errores humanos.
- no brinda una visión global y por lo tanto no ayuda al Diseñador o Desarrollador a detectar que es un requerimiento global, que excede al Caso de Uso que uno está mirando en este momento.

Por eso, muchas metodologías proponen complementar la Vista por Casos de Uso con otra vista transversal, que permita visualizar los requerimientos del sistema que no aplican a un único caso de uso. Booch, Jacobson y Rumbaugh[7] sugieren que el resultado del flujo de trabajo de Requerimientos se puede modelar combinando:

- Un **modelo de casos de uso** que capture los requisitos funcionales, y los no funcionales que son *específicos de casos de uso concretos*. El modelo de casos de uso incluye Diagramas de Casos de Uso y Especificaciones de Casos de

[6] *El proceso unificado de desarrollo de software*, de Ivar Jacobson, Grady Booch & James Rumbaugh, Rational Software Corporation, Addison Wesley, Capítulo 3
[7] *El proceso unificado de desarrollo de software*, *op.cit,* Capítulo 7

Uso.

- Una **especificación de requerimientos adicionales** para los requerimientos que son genéricos y no específicos de un caso de uso en particular, que incluye tanto requerimientos funcionales como no funcionales
- Un **modelo de negocio** o un **modelo del dominio** para establecer el contexto del sistema.
- Un conjunto de **esbozos de interfaces de usuario** y de prototipos para cada actor, que representan el diseño de las interfaces de usuario.

En general la vista por Casos de Uso resulta insuficiente y al menos se la debe complementar con una especificación de requerimientos adicionales (o bien con un modelo de dominio). A los requerimientos adicionales de naturaleza *funcional* se los suele denominar **Reglas de Negocio**.

La idea de un **Modelo de Dominio** es particularmente controversial, dado que es un modelo que establece un puente entre los requerimientos y el diseño. Si una metodología propone una separación fuerte entre las actividades y los roles *funcionales* (extracción de requerimientos, análisis) y las actividades y roles *técnicos* (diseño, programación, prueba), debe evitarse que un Modelo Lógico funcione como una restricción a los Modelos Físicos[8]. En esa forma de trabajo, el Modelo Lógico debe interpretarse como una descripción funcional y será responsabilidad del Diseñador decidir cuáles de las ideas de ese modelo lógico pueden ser traducidas linealmente al modelo físico y cuáles deben ser implementadas de otra manera. Si consideramos al análisis y al diseño como actividades bien separadas y llevadas a cabo por grupos distintos de personas, entonces debemos evitar que las personas responsables de definir “qué debe hacer el sistema” tomen decisiones sobre “cómo debe hacerlo”, eso corresponde a los Diseñadores.

En otras metodologías se propone una relación más estrecha entre ambos modelos. **Domain-Driven Design**[9] es una metodología que propone la definición de un Modelo de Dominio consensuado entre los Usuarios y los Desarrolladores. De esta forma se intenta minimizar la brecha entre el problema y la solución, haciendo que el cliente y el equipo de desarrollo puedan hablar en los mismos términos y compartan un único modelo que describe el comportamiento del sistema. Esto permite a los usuarios dar feedback mucho más rápidamente y también se elimina el trabajo que produciría tener

[8] Algo que pasa con mucha frecuencia en la práctica, en particular al definir modelos de datos mediante Diagramas de Entidad-Relación (DER)

[9] *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Eric Evans, Addison-Wesley Professional, 2003.

que mantener múltiples modelos con información similar.

5 Diseño: ¿de lo general a lo particular?

Repasamos dos visiones que son tanto diferentes como complementarias:

5.1 Diseño Top Down

En un diseño top-down

- se trabaja con un *big picture*, o una idea global de lo que hace el sistema
- dividimos un sistema en partes o componentes
- lo que hace cada componente permanece como una caja negra, hasta que lo vuelvo a descomponer en subprocesos
- el nivel de complejidad decrece al aumentar el nivel de detalle porque me concentro en una parte del problema y no en resolver el todo
- tenemos que pensar estrategias de prueba para poder probar ciertos componentes

5.2 Diseño Bottom Up

En un diseño bottom up

- comenzamos por componentes de bajo nivel
- especificados con un gran nivel de detalle (casi en su totalidad),
- luego se van integrando los componentes y se forman subsistemas.
- cada subsistema se va integrando hasta llegar a formar el sistema final.
- los componentes van creciendo en complejidad y completitud a medida que se avanza con el desarrollo.

5.3 Mezclando ambas perspectivas

En cualquiera de las dos formas de trabajo separamos un problema en componentes,

- esto permite que varias personas trabajen en distintos problemas en forma paralela
- necesita que luego ese trabajo se sincronice (surge la interfaz entre componentes)
- asociado a esto último se define el grado de acoplamiento entre módulos

En general, podemos decir que nuestra estrategia es tomar las decisiones de diseño en el momento en que podamos hacerlo, evitando definiciones prematuras.

- Seguir a rajatabla el esquema top-down nos obliga a diseñar todos los

componentes antes de empezar a programar.
- Seguir a rajatabla el esquema bottom-up implica no armar un mapa de requerimientos que tenemos que resolver, no "levantar la vista".

Al pensar el diseño también vamos a combinar ambas técnicas porque ninguna es enteramente satisfactoria

- Tenemos que balancear el entendimiento global del sistema y el nivel de especificación que necesitamos para poder validar nuestro modelo
- También tenemos que balancear dónde estamos parados, qué prioridad tiene cada requerimiento vs. los tiempos en los que se comienza a testear
- Los objetivos que resuelven cada componente específico deben apuntar a conseguir el objetivo central del sistema

6 Resumen final

El proceso del diseño necesita una serie de elementos como soporte para tomar decisiones. Los requerimientos, las restricciones, las componentes primitivas, el contexto tecnológico en el que debe trabajar la solución, los casos de prueba, son algunos de esos inputs o entradas fundamentales para poder avanzar sobre un modelo que agregue valor al usuario. Comienza aquí a gestarse la perspectiva de sistemas para dar soluciones a la problemática de negocio vista hasta entonces.