Modelos en la ingeniería de software

Tabla de contenido

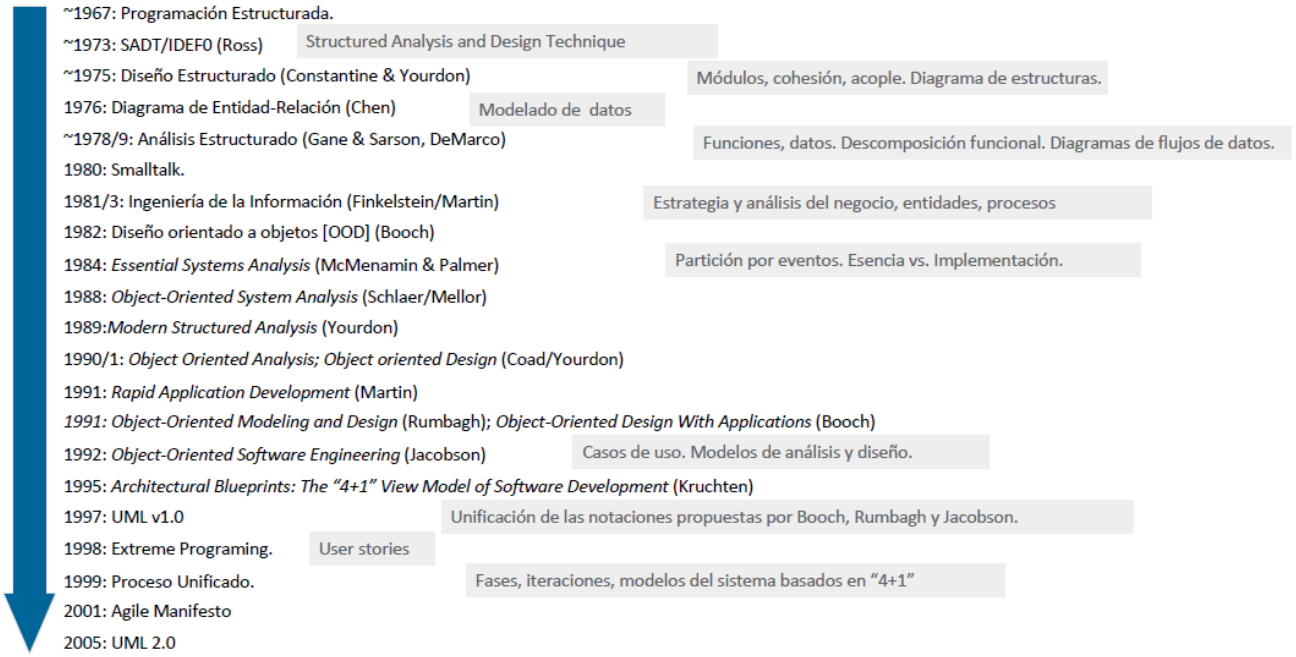
¿Qué es un modelo?

Antes que nada es importante que recordemos qué es un modelo. Un **modelo** es una simplificación de la realidad, es una representación de algún aspecto de la realidad que nos interesa. En el contexto del modelado de sistemas, desarrollamos modelos con el propósito de entender mejor el software que queremos construir y muchas veces también para entender el software que ya está construido y que necesitamos entender para poder mantenerlo, para poder darle evolución. Es muy difícil entender la totalidad de los aspectos de un sistema, por eso vamos a encontrarnos con que muchas veces vamos a tener que desarrollar varios y diferentes modos.

Brooks, en "No Silver Bullet", nos decía que al no existir el software en el espacio, es muy difícil poder representarlo mediante modelos, mediante diagramas. La ingeniería tradicional, la arquitectura, tienen la suerte, si se quiere, de poder utilizar planos, poder utilizar diagramas que no son ni más ni menos que abstracciones geométricas de *realidades* geometría. Cuando queremos llevar esa misma experiencia, esa misma práctica, al ámbito de la ingeniería de software, nos encontramos con que no hay un diagrama que sirva para mostrar todos los aspectos que nos interesan, terminamos utilizando varios. Y al no existir en el espacio es medio difícil hacer un mapeo entre lo que representan esos modelos y los que es el software. Por otra parte, recordemos que el código es, ni más ni menos, que otro modelo. Es una representación del comportamiento que tiene el software. Entonces, usualmente, cuando pretendemos modelar un sistema, nos vamos a encontrar con un montón de diagramas que representan en algunos casos la base de datos o el modelo conceptual de datos; nos vamos a encontrar con diagramas que representan la arquitectura o que representan el flujo de datos; o que representan el control. Entre todos esos tipos de diagramas, podemos terminar de entender qué es lo que el sistema hace o tiene que hacer.

Modelos y métodos

Una breve (e incompleta) perspectiva histórica



La preocupación por modelar sistemas nace prácticamente con el nacimiento de la ingeniería de software. En los años 70, mucho del desarrollo vino de la mano de la programación estructurada. Al surgir la programación estructurada, aparece la necesidad de tener un diseño estructurado, y al aparecer un diseño estructurado, aparece el análisis estructurado también. En los 90 del siglo pasado aparece la orientación a objetos muy fuertemente y, de vuelta, se repite el mismo patrón: aparecen distintas metodologías para realizar y representar diseño orientado a objetos y análisis orientado a objetos.

[V / **F**] Dentro de la ingeniería de software los modelos son relativamente simples de desarrollar.[1]

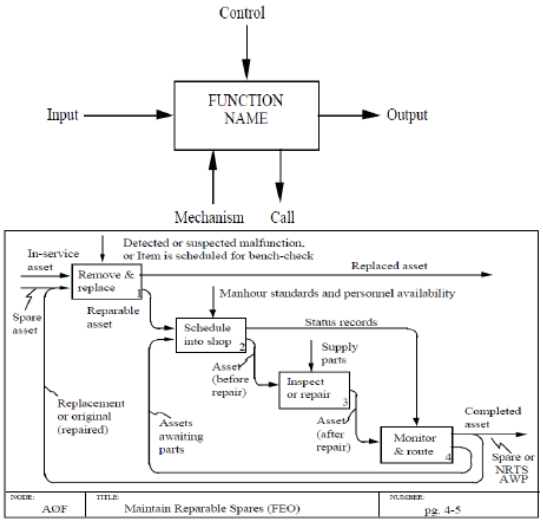
Modelos

La influencia del lenguaje de programación

El protagonista exclusivo a fines de los 90 era UML. UML tuvo un peso muy importante. Aparecieron también otras notaciones complementarias como BPMN, más recientemente aparece C4. Lo que es importante destacar es que estos lenguaje de modelado, estas notaciones, están muy influenciados por los paradigmas de programación. En principio podemos hablar de que existen como 2 grandes grupos de notaciones, algunas más orientadas a la descripción de la implementación y otras que tienen más que ver con el modelo conceptual del sistema. Los modelos y las notaciones más relacionadas con el diseño obviamente están mucho más cercanas al lenguaje de programación empleado. Los que tienen más que ver con el análisis de requisitos, con la ingeniería de requisitos, son modelos que describen lo que el sistema tiene que hacer, independientemente de cómo lo hagamos. Entonces se utilizan abstracciones del tipo actividades, a veces almacenamientos, a veces objetos; pero claro, tienen esos nombres, actividades, objetos, almacenamientos, pero no necesariamente son programas, aplicaciones, tablas u objetos del lenguaje orientado a objetos que se emplee. Entonces, lamentablemente, lo que termina pasando es que el lenguaje de programación influencia obviamente, en el lenguaje utilizado en el diseño, y esa anotación de diseño, termina influenciando lo que se utiliza para describir al sistema en términos tecnológicamente neutros, en términos abstractos. Vamos a verlo a continuación, con un ejemplo.

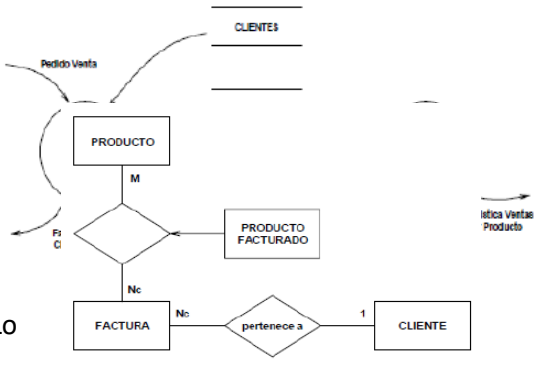
Perspectiva histórica

IDEF0



Uno de los primeros desarrollos que hubo en los 70, fue una técnica llamada SADT (Structured Analysis and Design Technique), que fue propietario durante mucho tiempo (la solicitó la fuerza aérea). Esta metodología, derivó en una notación llamada IDEF0 (modelado funcional) continuada por varios sucesores (IDEF1X: modelado de la información; IDEF3: modelado de procesos; IDEF4: modelado orientado a objetos; IDEF5: ontología). Lo que plantea es descomponer una función, una actividad en subfunciones. Esto supuestamente es una descripción abstracta de lo que tiene que hacer un sistema. Imaginémonos que en un primer nivel tenemos un único diagrama que representa todo el sistema, con los inputs con los outputs; si hacemos doble clic en esa primer cajita nos deberíamos encontrar con una cajita de menor nivel que representa las grandes actividades en las cuales se descompone esa función (subfunciones), y obviamente, el flujo de datos que hay entre cada una de esas situaciones. Eso se parece mucho a descomponer un programa de subrutinas. Y esta es una muestra clara de como la tecnología de implementación termina influenciando en la notación empleada para representar el análisis de requisitos, el análisis o la descripción del sistema en términos tecnológicamente neutros.

Análisis estructurado (DFD, DER, DD)



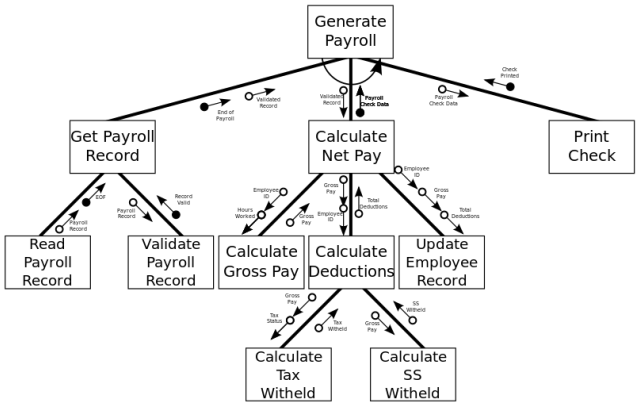
Otro ejemplo similar, es el del análisis y el diseño estructurado en versión Yourdon, Larry Constantine, Tom DeMarco. En esta notación, que estuvo lejos de estar estandarizada, un sistema se descompone en actividades y en subactividades (igual que en el caso anterior). Las actividades están representadas por burbujas. En este diagrama vemos un diagrama en el cual tenemos 2 burbujas y varios almacenamientos. Esas burbujas se pueden descomponer a su vez en subburbujas, en subactividades, es decir, que podemos ir desde lo general a lo particular, es una estructura jerárquica. Lo

[1] Los modelos son la parte más difícil de desarrollar.

diferente de este tipo de diagramas es que aparecen los almacenamientos. Los almacenamientos representan, de alguna manera, los objetos de ese diagrama de entidad, relación que vemos más abajo. En principio, son almacenamientos esenciales, es decir, que no son archivos ni son tablas, sino que representa lo que la terminología del análisis estructuro llamaba almacenamiento esencial, cosas que el sistema tiene que recordar. Esto obviamente da lugar a un posterior diseño de una base de datos o un sistema de archivo. Pero nuevamente acá el problema que tenemos es que estamos fuertemente influenciados por la tecnología de implementación, aunque estemos usando una abstracción (programas y archivos).

Diseño estructurado (DE)

En la misma línea, está el diagrama de estructuras, el diseño estructurado, en ese momento también planteado por Larry Constantine y Ed Yourdon. Se descompone un programa en módulos. Entre los módulos se intercambian parámetros. Y esto también está muy asociado a los lenguajes de programación que se utilizaban en ese entonces, que eran muy relacionados con el paradigma estructurado.



Unified Modeling Language (UML)

Ya más cerca de los 90 aparece UML. UML es un lenguaje gráfico que se usa para especificar, visualizar, también para construir, documentar, sistemas basados en software. UML fue el resultado de fusionar las notaciones más populares que había en la década del 90, las de Jacobson, Rumbaugh y Booch. Es un estándar del Object Managment Group (OMG) desde el año 97 (actualmente está en la versión 2.5.1). En principio es independiente de la metodología o del proceso de desarrollo y plantea una serie de diagramas.

UML

Diagrama de clases

El diagrama de clases describe la estructura interna de un sistema en términos de clases, asociaciones; entre esas clases colaboraciones, interfaces. Es el diagrama que quizás se utiliza más, y muestra una estructura estática. Este diagrama puede utilizarse para modelar aspectos de diseño más cercano a la implementación y, con algunas particularidades, con algunas limitaciones, para modelar también aspectos conceptuales que no tienen que ver con la implementación (puntualmente vamos a utilizar una variante de este diagrama para desarrollar algo que llamamos modelo de dominio).

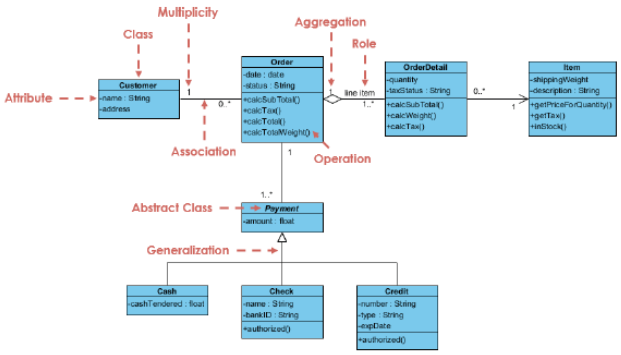


Diagrama de casos de uso

Un modelo más cercano al mundo del análisis de requisitos es el casos de uso. Cada una de esas elipses que vemos ahí está representando un caso de uso. Un caso de uso es un paquete de funcionalidad que nos ofrece el sistema. El diagrama de caso de uso además incluye actores que son las personas que interactúan con el sistema. Es un diagrama estático, es un diagrama que no ofrece mucha información, por eso, más adelante vamos a ver que para cada uno de estos casos de uso vamos a necesitar algún tipo de especificación en formato texto para poder dar más información.

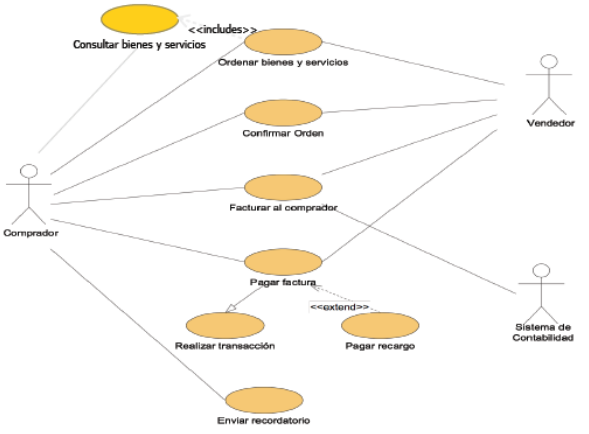


Diagrama de secuencia

El diagrama de secuencia nos muestra la interacción entre una serie de clases para completar una acción determinada. Normalmente para un caso de uso, probablemente necesitemos implementar esa funcionalidad que está representada en ese caso de uso, mediante una serie de objetos o de clases que van a colaborar entre sí, probablemente entonces utilicemos este tipo de diagrama para representar esa colaboración. Lo que muestra el diagrama es el paso del tiempo de izquierda a derecha y el intercambio de mensajes entre las clases. Esos mensajes, por supuesto, invocan a los distintos métodos que ofrecen cada una de esas clases.

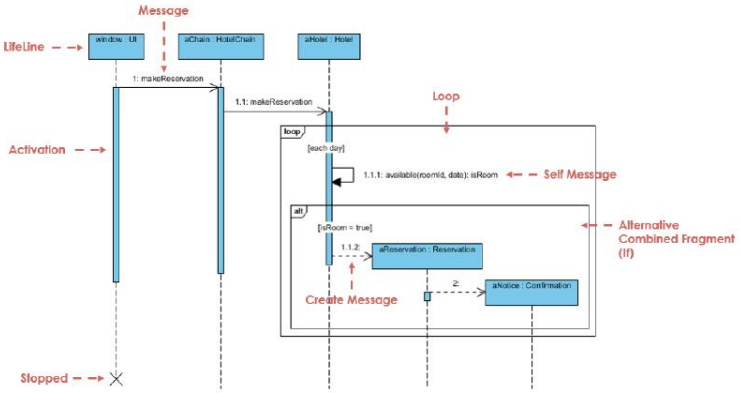
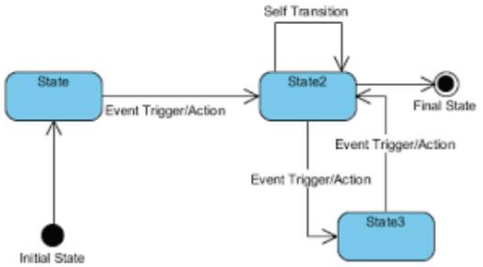


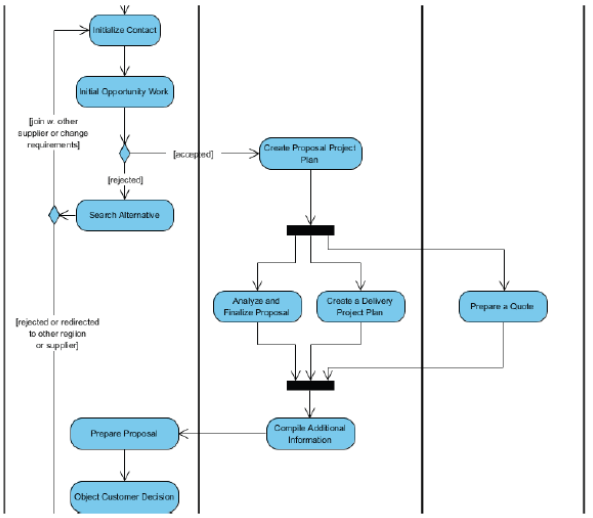
Diagrama de estado

Otro diagrama muy utilizado es el diagrama de estado. Básicamente es una máquina de estados que permite ilustrar el ciclo de vida de una clase o de una colaboración. La notación es muy sencilla, hay un nodo de inicio, un nodo de fin, hay estados y hay transiciones entre los estados que son representados mediante rectángulos y mediante flechas respectivamente.



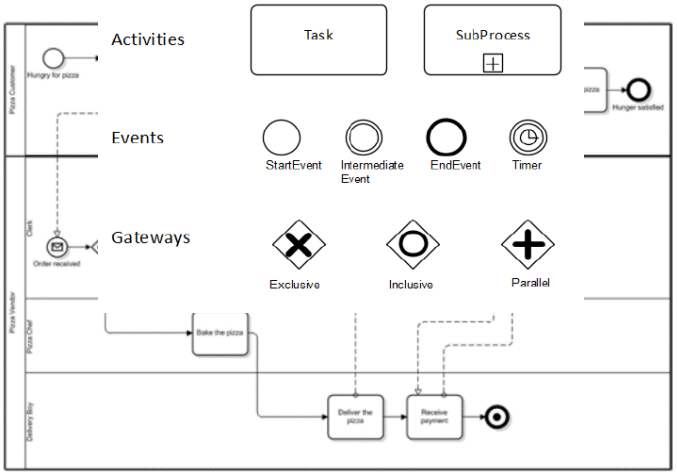
Diagramas de actividad

UML también nos ofrece un diagrama de actividad. Esto se puede utilizar para representar el flujo de actividades de un ciclo de trabajo, para representar lo que pasa dentro de una función, dentro de un método, tiene múltiples aplicaciones. Nosotros podemos utilizar, por ejemplo, para representar un proceso de negocio que hace falta automatizar mediante algún tipo de software. Es muy sencillo, simplemente lo que hay son andariveles que representan distintos roles; hay actividades, hay transiciones entre las actividades, hay puntos de decisión. Es una especie de cursograma en una notación un poco más moderna. Esto es muy útil, obviamente para representar ciertos aspectos relacionados con el análisis de requisitos.

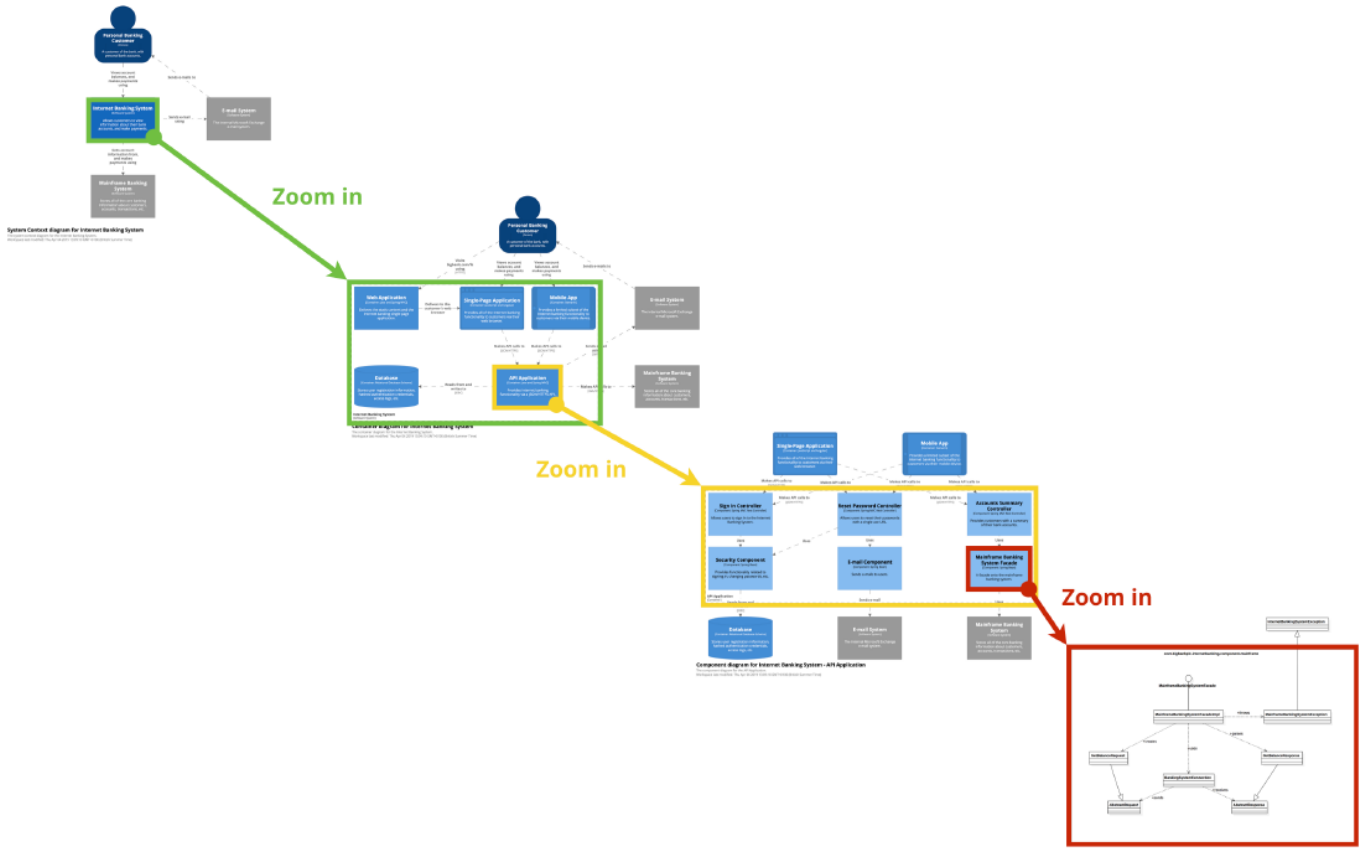


Business Process Modeling and Notation (BPMN)

Algo que también se puede utilizar y que tiene similares características es BPMN. BPMN son las siglas de Business Process Modeling and Notation. Es un estándar de la OMG desde 2005 también. Lo que busca representar es un proceso de negocio, un proceso de manera similar al diagrama de actividades de UML que representábamos recién. La notación es sumamente sencilla, hay un nodo de inicio, hay un nodo de fin, hay andariveles, hay tareas; las tareas tienen transiciones entre sí. Podríamos representar acá, por ejemplo, el proceso de ventas, el proceso de incorporación de un nuevo cliente a una aplicación, las aplicaciones son infinitas. Esto está muy pensado no sólo para describir procesos, sino también para automatizarlos mediante herramientas. También vamos a ver que es muy útil para entender. De repente cuando uno va a una organización y necesita entender cuál es el proceso antes de automatizarlo, usualmente puede utilizar este tipo de diagramas. Lo mismo con el diagrama de actividades de UML. Muy sencillo, muy útil, hay que tenerlo en la caja de herramientas.



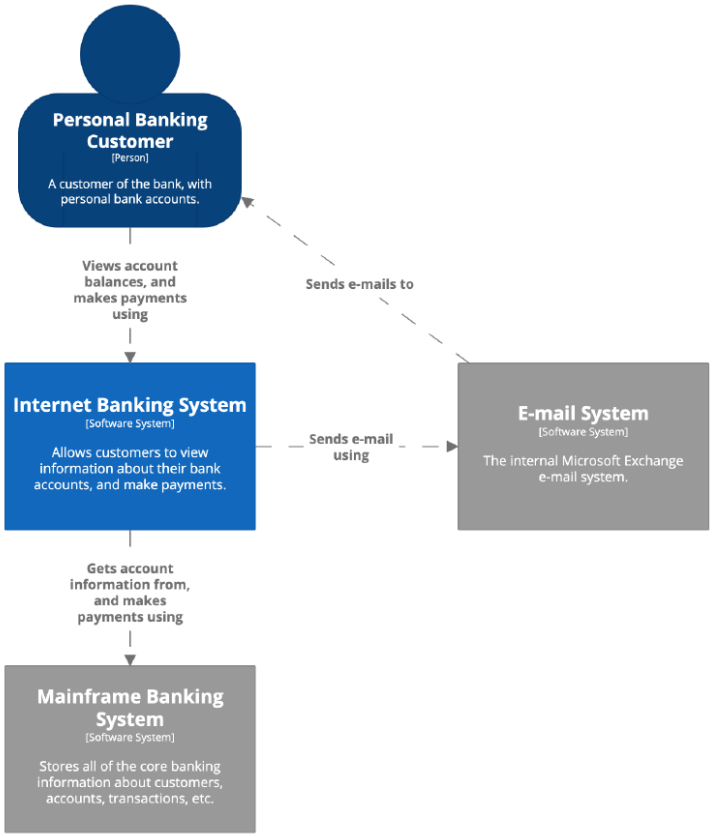
[**V** / F] BPMN está orientado a modelar procesos de negocio.



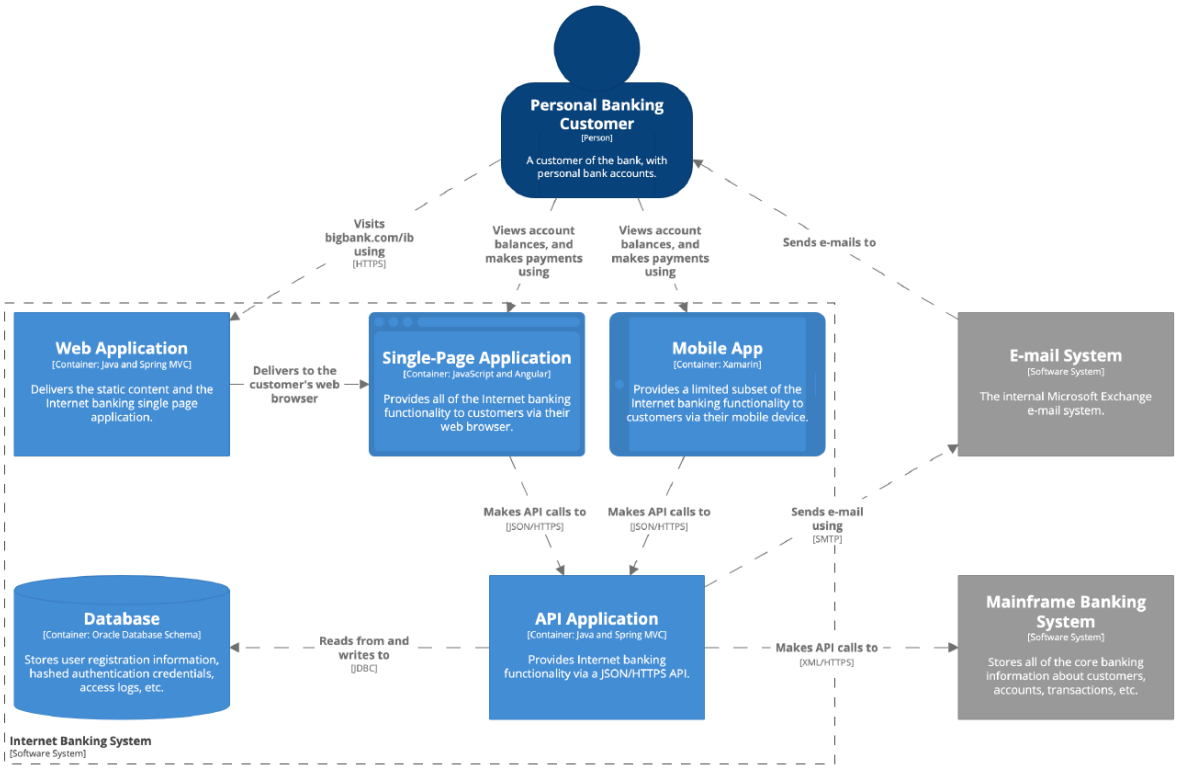
C4 tiene como propósito describir la arquitectura de software. ¿Qué es la arquitectura? La descripción de los grandes elementos que forman parte del software. Lo que propone C4 es una estructura de diagramas en niveles como muchas de las otras notaciones que hemos visto antes.

Nivel 1: Diagrama de contexto

Estos cuatro niveles arrancan con un diagrama de contexto que nos muestra el medio ambiente en el cual va a estar funcionando el sistema que queremos construir. Es un diagrama que tranquilamente se puede utilizar en el mundo de la ingeniería de requisitos. Muy sencillo, nos está mostrando (con una caja azul en este caso) el **sistema** en cuestión, el **usuario** (con ese ícono en una azul un poco más oscuro), con cajas grises los **sistemas que soportan** el sistema que estamos modelando y las relaciones entre ellos. Fíjense que es un diagrama muy sencillo, que tranquilamente uno podría incluir en un documento de visión o en una presentación ejecutiva para describir el contexto en el cual funciona el sistema.



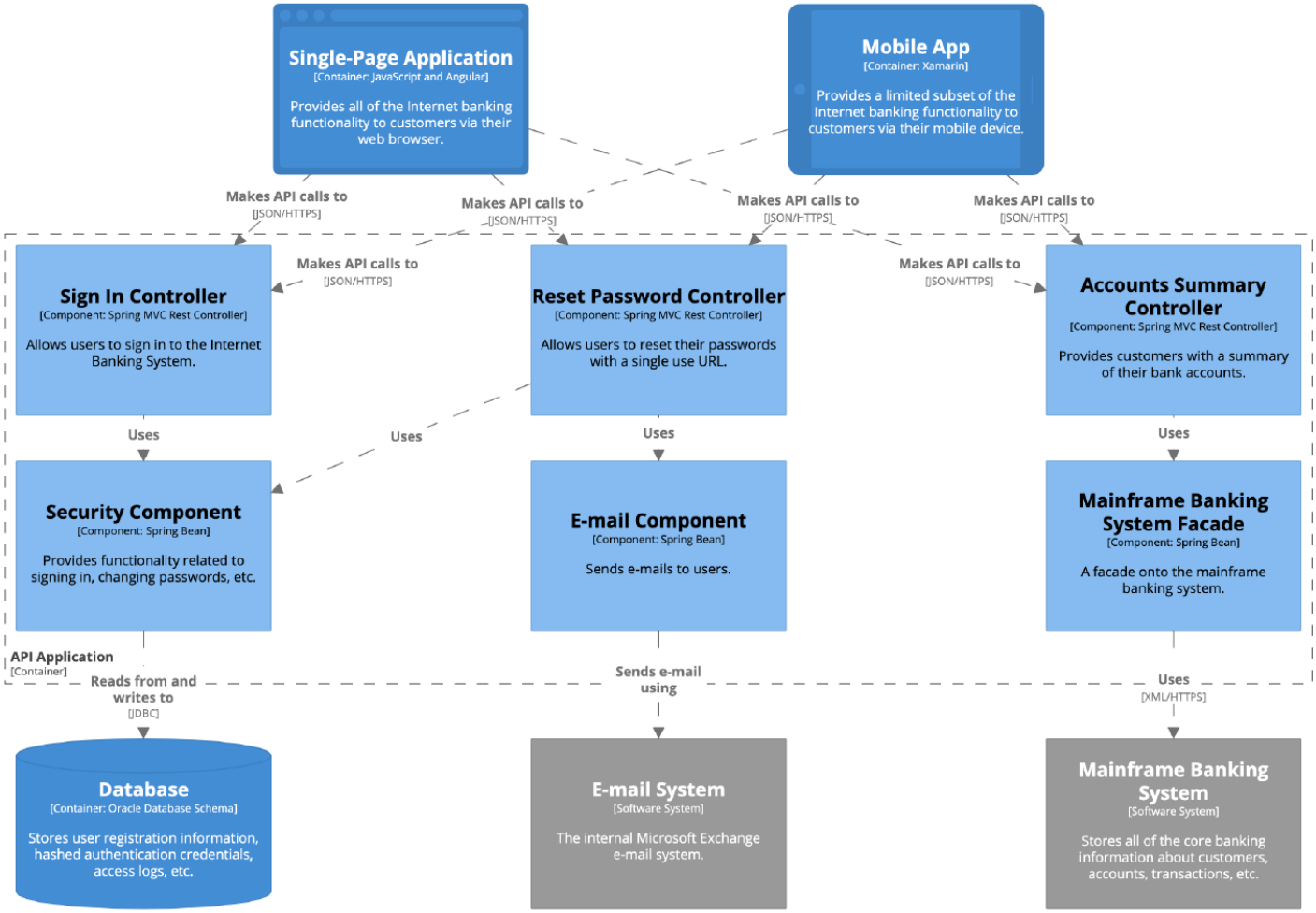
Nivel 2: Diagrama de contenedores



El siguiente nivel, el nivel 2, es el diagrama de contenedores. Lo que estamos haciendo en definitiva acá es un zoom, un doble clic en el sistema, qué es lo que hay ahí adentro. Y nos encontramos con esa abstracción que C4 llama "**Conteiners**", que no tienen absolutamente nada que ver con los conteiners de los que estamos hablando hoy. Simplemente es un mecanismo que lamentablemente tiene el mismo nombre, pero es un mecanismo de abstracción que lo que busca representar son aplicaciones, almacenamiento de datos, microservicios, base de datos. En este caso en particular, nos encontramos con que el sistema en cuestión está compuesto por una aplicación web, una aplicación mobile, una aplicación que tiene que ver con la API, interactúa con el sistema de email, eso está fuera del

del sistema, pero ahí se muestra interacción. Hay una base de datos. Es un diagrama más orientado a la implementación.

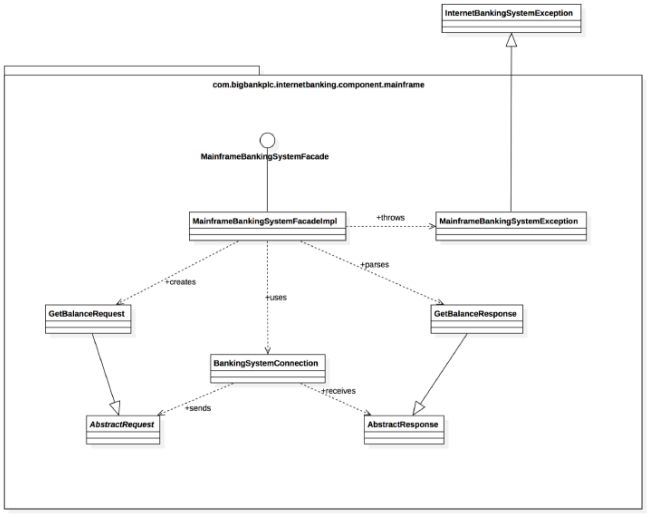
Nivel 3: Diagrama de componentes



Si hace falta podemos hacer un doble clic en un container y utilizar un diagrama de componentes para describir qué es lo que está pasando ahí adentro, qué es lo que hay adentro de ese contenedor. Aquí ya estamos empezando a hablar de **componentes**. Esos componentes tienen que mapear contra alguna cosa real del entorno de implementación. En este caso hay un par de controladores, hay componentes de seguridad, hay una interfaz con un sistema en un equipo mainframe. Acá ya estamos más cerca de la implementación, más cerca del código.

Nivel 4: Diagrama de código

Y si hace falta, podemos bajar un nivel más y tener un diagrama de código que no deja de ser más que un diagrama de clases o un diagrama de colaboración que lo que busca es representar el código que forma parte de ese componente.



[V / **F**] A medida que se avanza de niveles en C4 los diagramas se acercan menos a la implementación.[2]

Resumen

Hemos visto distintas notaciones. UML probablemente sea la más conocida, la más importante; BPMN no tanto; y C4 probablemente sea la menos conocida de todas. UML tiene la ventaja de que se enseña en los cursos de programación, con lo cual, muchos de ustedes probablemente estén muy familiarizados, al menos con los diagramas de clase, los diagramas de secuencia. Hay que tener mucho cuidado cuando utilizamos UML fuera de lo que es diseño, cuando queremos utilizar UML más para los que son los modelos abstractos, conceptuales del análisis de requisitos, ahí hay que tomar en cuenta algunas consideraciones. BPMN es muy bueno para modelar procesos de negocio, con lo cual, si necesitamos entender un proceso antes de pensar en automatizarlo, es una muy buena herramienta. Y C4 nos ofrece una muy buena alternativa porque combina una notación muy sencilla con UML en el último nivel. C4 surge como alternativa a los diagramas de arquitectura que propone UML, que tienden a ser muy complejos. Entonces, con esos diagramas muy, muy elementales, muy sencillos, nos podemos comunicar y evitamos ese modelado informal que, como muchos autores llaman de “cajitas y flechitas” para representar arquitecturas, para representar las grandes abstracciones que describen a al sistema que estamos pretendiendo modelar.

Por supuesto que el tema no se agota acá, simplemente es un panorama general. Wiegers en su libro de ingeniería de requisitos nos propone una descripción muy permeabilizada de los tipos de modelos y diagramas que se pueden utilizar en el ámbito de la ingeniería de requisitos. Los libros de Booch o de Larman nos plantean los diagramas que se pueden utilizar más para los aspectos de diseño. Ahí uno tiene un gran abanico de alternativas de opciones para elegir.

Escenarios de utilización

Lo que siempre hay que tener presente es que hay que tener mucho cuidado con el escenario que nos toque, y elegir correctamente las herramientas y las notaciones en función de esos escenarios. Si vamos a construir una casa para una mascota, probablemente no haga falta hacer un diagrama, probablemente el modelo lo tengamos en la cabeza y simplemente lo que hacemos es ir, conseguir un poco de madera, un par de clavos, algunas herramientas y trasladar ese diseño que tenemos en la mente a la cosa concreta. Si lo que vamos a construir es una casa, probablemente necesitemos modelos un poco más detallados, porque vamos a necesitar emplear materiales más caros, vamos a tener que hacer cálculos de estructuras, vamos a tener que compartir la información con distintos especialistas, el proceso de construcción tiene que estar bien definido para optimizar costos, de manera que en ingeniería de software nos vamos a encontrar con la misma situación. Quizás, hay casos en donde el modelado puede ser muy informal, alcanza con dibujar un diagrama de clases en un pizarrón, en otros casos probablemente, en vez de dejar ese diagrama en un pizarrón, quizás sea conveniente hacerlo en una herramienta. Lo que siempre tenemos que sopesar es el costo de construir el modelo y mantenerlo versus la utilidad que nos provee ese diagrama. Siempre vamos a tener que balancear entre esos 2 aspectos. En algunos segmentos de la industria del desarrollo de software, hay algo llamado Modeling Driven Design. En ese entorno, los modelos son muy, muy formales y se pueden ir construyendo y transformando en modelos de menor nivel hasta llegar al código. Las herramientas Case de los años 90 prometían eso y, de hecho muchas lo consiguieron. Hay algunos artículos muy interesantes al respecto, que podemos compartir con ustedes, en donde en segmentos muy particulares de la industria de la ingeniería de software ese modelado, esa estrategia de modelado, tiene mucho éxito. En otro tipo de aplicaciones, el modelado es muy muy informal, de hecho, hay toda una escuela de Agile Modeling que provee unas perspectivas sumamente interesantes de cómo enfocar el ejercicio de modelado.

En definitiva y, como siempre, qué herramientas utilizar, que notación utilizar, qué modelos construir y que modelos no, va a depender de nuestro criterio profesional.

[2] A medida que se avanza de niveles en C4 los diagramas se acercan MÁS a la implementación

El diagrama de clases es un diagrama…

- ☑ **estático**
- ☐ dinámico
- ☐ de contexto