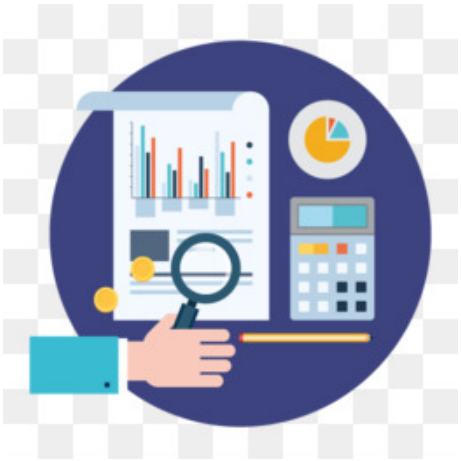
.UBAfiuba
FACULTAD DE INGENIERÍA



Apuntes de Análisis de la Información

[7509]
Curso González
1C 2021

Grassano, Bruno
bgrassano@fi.uba.ar

Índice

1. Introducción

El presente archivo contiene los apuntes que fueron tomados a lo largo de la cursada de la materia análisis de la información (7509) en el curso del profesor González - Turri.

Recomendaciones

La materia es muy tranquila y buena parte de los contenidos ya se tiene una idea de antes, sobretodo si se esta trabajando. (En mi opinión esta materia quedaría mejor al comienzo de la carrera)

Respecto a los trabajos prácticos, son 3 (este cuatrimestre en virtualidad) y se realizan en grupos de 5/6. En los tres se realiza el mismo proceso de análisis de requisitos y una propuesta de mejora con Lean UX, no se programan los sistemas. En los dos primeros se entrega el documento con todo lo realizado, y en el tercero se realiza una presentación como preparación para el final. No requieren mucho tiempo cada uno.

El final consiste en realizar otro análisis de requisitos y mejora pero sin el acompañamiento del profesor de la practica, este trabajo se expone en una presentación, y luego se realizan preguntas teóricas de forma individual.

Primera clase

2. Introducción a la Ing. Software

Definición: *'Es la aplicación de un enfoque sistemático disciplinado y cuantificable al desarrollo, operación y mantenimiento de productos de software.'* (IEEE 1993) (Enfoque ingenieril)

Otra definición: *'Comprende principios y **metodologías** para desarrollar, operar y mantener software de calidad.'*

2.1. Metodologías

Contiene 3 partes. Los modelos, procesos, y las herramientas de un desarrollo de software. Estos son específicos a una metodología.

- Los modelos necesitan de un lenguaje de modelado → UML.
- Los procesos son *XP, SCRUM, RUP, etc*
- Herramientas *Suite Rational*

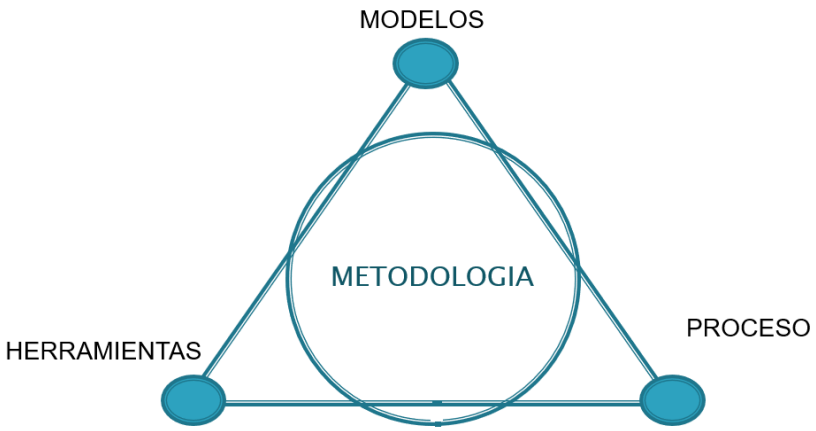


Figura 1: Composición de una metodología

2.2. Modelos

¿Que es un modelo?

Es una simplificación de la realidad. Como la realidad es compleja le aplicamos abstracción, para incluir elementos que tienen mucha influencia y omitir los relevantes.

¿Para que sirve un modelo?

- Para comprender mejor el sistema que estamos desarrollando.
- Para poder comunicarnos, con el cliente y el equipo de diseño.
- Visualizar y controlar rápidamente la arquitectura del sistema.
- Gestionar el riesgo de un proyecto. *Calidad, plazos, y presupuesto*

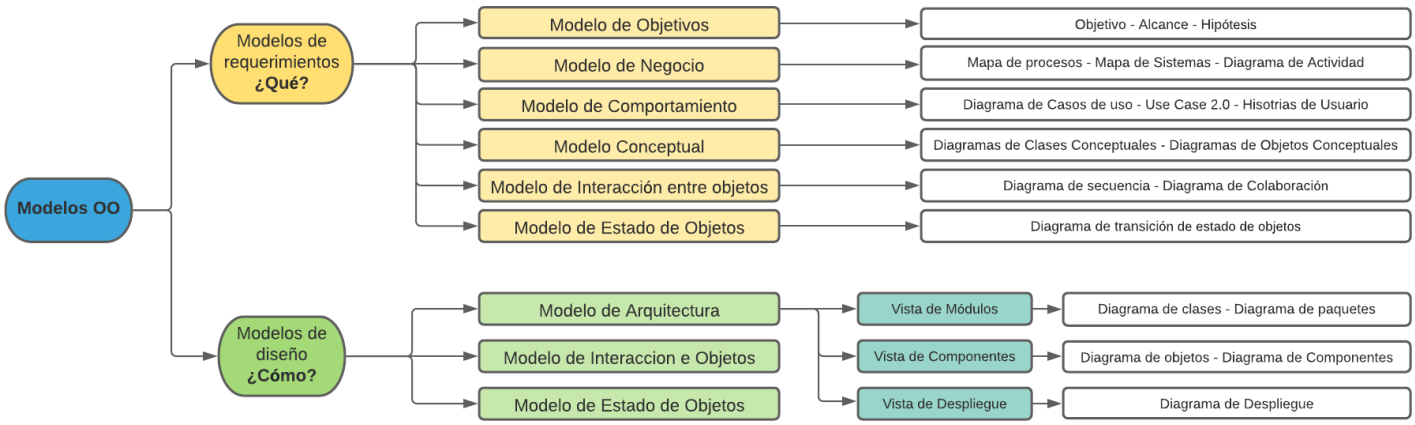


Figura 2: Modelos OO diferenciados en los tipos del QUE y del COMO

Etapas

- Captura de requerimientos. *Reuniones con cliente, cuestionarios, focus groups,...*
- Análisis de requerimientos: Es el ¿Que se necesita? *Modelos de casos de uso, historias de usuario,...*
- Diseño: Es el ¿Como voy a hacer? *Modelos de clases, de objetos, de despliegue,...*
- Construcción. *Programación, integración de componentes...*
- Pruebas. *Pruebas de sistema, unitarias, de usuario, de integración*
- Entrega. *Despliegue del producto a los usuarios*
- Mantenimiento. *Correctivo (Corregir defectos de diseño) y evolutivo (Adaptar a los nuevos procesos)*

Principios de la modelización

- La elección de que modelos crear tiene una gran influencia sobre como se aborda un problema y como se da la solución.
- Tienen distintos niveles de precisión.
- Los mejores modelos tienen que estar ligados a la realidad.
- Un único modelo no es suficiente.
- Tener un Objetivo y alcance claro.

Beneficios de la modelización

- Forma de visualizar necesidades y requerimientos contra costos reales antes de comenzar el desarrollo.
- Se trabaja en un alto nivel de abstracción

Enfoques

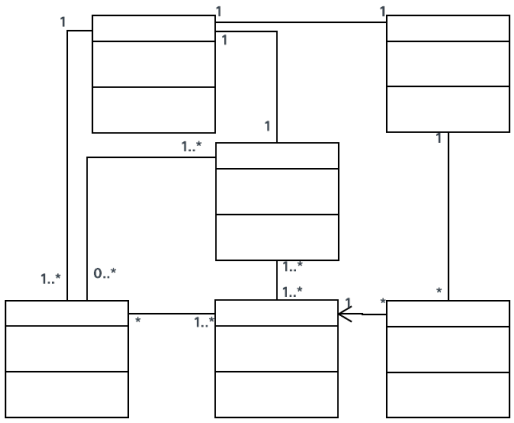
- Algorítmico
- Orientado a objetos

2.3. Paradigma orientado a objetos

- Esta basado en la creación de componentes reutilizables.
- Construcción de software mas rápida y dinámica.
- Facilita la creación de prototipos.

2.4. UML

Es un **lenguaje** estándar para escribir 'planos' de software. Lenguaje para visualizar, especificar, construir y documentar sistemas bajo el paradigma de orientación a objetos. Se utiliza desde el inicio hasta el fin del proyecto. **No** es una **metodología**.



Aspectos del modelado

- Modelado funcional o de comportamiento. *Historias de usuario*
- Modelado estático o estructural. *Diagrama de clases*
- Modelado dinámico. *Diagramas de interacción, secuencia*

2.5. Procesos de desarrollo de software

Tiene que responder a 4 preguntas: ¿Quien lo hace?, ¿Que lo hace?, ¿Como lo hace?, ¿Cuando lo hace? De forma tal de obtener un producto de calidad, dentro de plazos y presupuestos predecibles.

3. Modelo de Objetivos

Dentro de los modelos de requerimientos.

Macroproceso	Funcionalidades	Alcance
Nombre macroproceso	*Nombre funcionalidad*	*Si/No*
...	...	...

3.1. Objetivo del sistema

Define lo que el sistema va a hacer (objetivo). Es una descripción breve con alto nivel de abstracción, sobre **que** se requiere automatizar con el sistema, declarando los 'macroprocesos'.

Ej. El sistema esta dirigido a administrar las ventas de la empresa, incluyendo la gestión de los pedidos del stock y la facturación de las mismas'.

Macroprocesos involucrados: Gestión de pedidos, gestión de stock, gestión de facturación.

3.2. Alcance del sistema

Muestra los limites del objetivo(alcance). Para cada 'macroprocesos' hay que indicar las funcionalidades que el sistema va a contemplar, dejando bien en claro que funcionalidades están dentro/fuera del alcance

Formato típico de un análisis del alcance.

3.3. Hipótesis o supuestos funcionales

Establece premisas o restricciones a tener en cuenta. (hipótesis) Pueden ser establecidas por políticas de la empresa, normativas legales. Supuestos iniciales establecidos por el analista (deben ser verificados después por el cliente). No son requerimientos relevados, sino que complementan a los mismos.

Estos pueden darse por falta de relevamiento por determinadas circunstancias. *Ej. El cliente se lo olvido, es trivial, no lo sabe, lo oculta*

3.4. Caso de estudio: Airbus A320

Habilitar la aceleración en reversa. El avión al momento de frenar activa este mecanismo, abriendo unos alerones en las turbinas.

El software: el piloto acciona el comando, ocurre un servo-mecanismo que abre los alerones. Caso de error a evitar, si el piloto lo acciona volando, tiene que haber algún mecanismo que detecte que el avión este en el piso. Se agrego un sensor de giro de ruedas, que detecta cuando giran en el piso y habilita la reversa del sistema que acciono el piloto.

El software salio de esta forma, pero tenia un problema, el avión estaba aterrizando, pero en caso de que la pista tenga hielo se producía un bloqueo y la rueda patinaba en vez de girar. Esto pasó debido a que no se contemplo en el universo en estudio.

Se debería de haber tenido en cuenta el sensor de giro, de peso, y de altura.

Sistema de estudio: software, los 3 sensores, el servo-mecanismo reversa.

Segunda clase

4. Modelo de negocio

Dentro de los modelos de requerimientos.

4.1. Mapa de procesos

- Modeliza los procesos que una organización tiene para llevar adelante sus funciones.
- Permite entender la organización/negocio.
- Es un modelo matricial, modela los principales macroprocesos necesarios para el funcionamiento del negocio/empresa.
- Columnas: áreas del negocio. *Ej. Estrategia y planificación, operación, previsión, assurance, billing*
- Filas: visión por capas. *Ej. cliente, servicio, recurso, enterprise*

Conocer aquellos que vamos a automatizar, mas aquellos que interaccionan con el mismo.

4.2. Mapa de sistemas

- Modelo matricial: incluye los sistemas que dan soporte a los procesos de negocio.
- Columnas: áreas del negocio
- Filas: visión por capas

Mapea determinados macroprocesos, y va diciendo cuales son los sistemas que van a automatizar los macroprocesos (mapa de sistemas) Puede ser mapeo 1 a 1, 1 a N, N a 1.

4.3. Diagrama de actividad

- Modeliza una secuencia de actividades en un escenario determinado. Modelo gráfico.
- Esta compuesto por un grafo dirigido. Los nodos son actividades, y los arcos (dirigidos) representan la transición entre actividades.
- Actividad de inicio, es única (circulo relleno)
- Actividades de fin, puede haber varias (circulo con borde)
- El rombo (bifurcación) es de decisión. Indica caminos alternativos, elegidos según el valor de alguna expresión, surgida de un análisis que se lleva a cabo en una actividad.
- Barra de sincronización, agregar la barra indica que para que inicie otra actividad, tienen que haber finalizado todas las actividades que vayan a la barra.
- Dividiendo como columnas, agregar los responsables de cada actividad, quien las ejecuta. (Las 'Swimming lines', carriles, dividen las actividades')
- Estos diagramas no muestran temporalidad, esto esta en diagramas de secuencia.
- No muestran requisitos NO funcionales.
- Muy claro para comunicarse con el cliente y asegurarse de que se entendieron los requerimientos.

- Buen punto de partida para generar el modelo de comportamiento.
- Tipos de escenarios
  - Global, contiene todos los procesos de negocio del universo en estudio.
  - Parte del negocio, contiene algunos procesos de negocio del universo en estudio
  - Un solo proceso de negocio, tiene todas las actividades de ese proceso.
  - Regla de negocio, son las actividades de una regla de negocio.

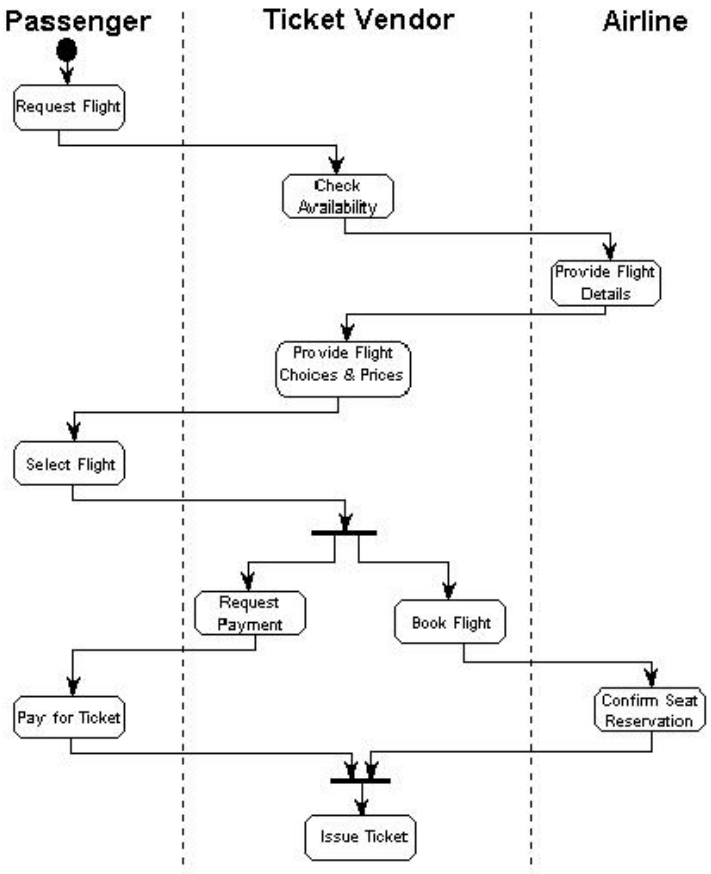


Figura 3: Diagrama de actividad. Notar las diferentes entidades: Actores, actividad de inicio (de fin falta), las secuencias, barras de sincronización.

Una actividad esta compuesta por un conjunto de tareas a ser realizadas. Verbo + objeto, *ej. verificar pedido*. No hay que asociarlas con actividades físicas, si con procesos y/o reglas de negocio que luego podrán ser (o no) automatizadas.

Tercera y cuarta clase

5. Modelo de comportamiento

Dentro de los modelos de requerimientos.

5.1. Casos de uso

Hay tres componentes en el modelo de comportamiento.

- Actores, representa algo o alguien (una persona, dispositivo, otro sistema) que interactúa con el sistema sin ser parte de el. Puede ingresar y/o recibir información del sistema. Se identifican fácilmente a partir de los roles representados por la swimline's. Se pueden identificar respondiendo a las siguientes preguntas: ¿Quien esta interesado en cierto requerimiento? ¿En que lugar de la organización se usa el sistema? ¿Quien se beneficiara con el uso del sistema? ¿Quien mantendrá el sistema? Etc.

  La clave para identificar buenos actores es pensar en el rol con el cual el actor interacciona con el sistema.

- Casos de uso, especifica el comportamiento de una parte del sistema. Representan un **requisito funcional** del sistema. Es un conjunto de acciones (variaciones incluidas) que ejecuta un sistema para producir un resultado para un actor. Gráficamente es una elipse con el nombre del caso de uso adentro.

  Son fácilmente identificables a partir del modelo de negocio (las actividades/tareas). Se pueden identificar con algunas preguntas también. ¿Cuales son las tareas de cada actor? ¿Que casos de uso crecerán, almacenaran, modificaran, consultaran, o eliminaran esta información? ¿Algún actor necesita informar al sistema sobre cambios externos? ¿Necesita algún actor ser informado de cambios en el sistema? Etc.

  Para identificar buenos y malos casos de uso, hay dos reglas.

  - Completitud temporal, un caso de uso debe ser completo de principio a fin. *Ej. Malos casos: seleccionar curso, registrar curso, informar registro (no son completos de principio a fin, cada uno esta en una parte, alumno, sistema)- Correcto: Registrar curso. (va del alumno al sistema)*
  - completitud funcional, cuando varios casos de uso tratan con la misma entidad y son iniciados por el mismo actor podemos resumirlo en un solo caso de uso.*Ej. Incorrecto: Departamento alumno: agregar curso, modificar curso, eliminar curso - Correcto: Mantener curso*

- Diagramas de casos de uso, muestrea un conjunto de casos de uso, actores y sus relaciones. Permite tener una visualización del comportamiento de un sistema. En los diagramas de caso de uso general, se tiene una visión de alto nivel de todo el sistema, hay que tener 'todos' los actores y 'todos' los casos de uso del sistema. Hay actores primarios (izquierda diagrama)(disparan casos de uso) y actores secundarios (derecha diagrama)(no disparan casos de uso). Puede darse que un actor sea principal y secundario (esta en ambos lados).

5.2. Documentación de casos de uso

Tienen una descripción del caso, y flujo de eventos. (pre-condiciones, flujo principal, subflujos, flujos de excepción, post-condiciones)

Las pre-condiciones son las acciones que deben haber sido ejecutadas previas a la ejecución del caso de uso.

Flujo de eventos, es una descripción de una secuencia de acciones necesarias para proveer la funcionalidad del sistema a un actor determinado.

Ej. Registrar curso a dictar.

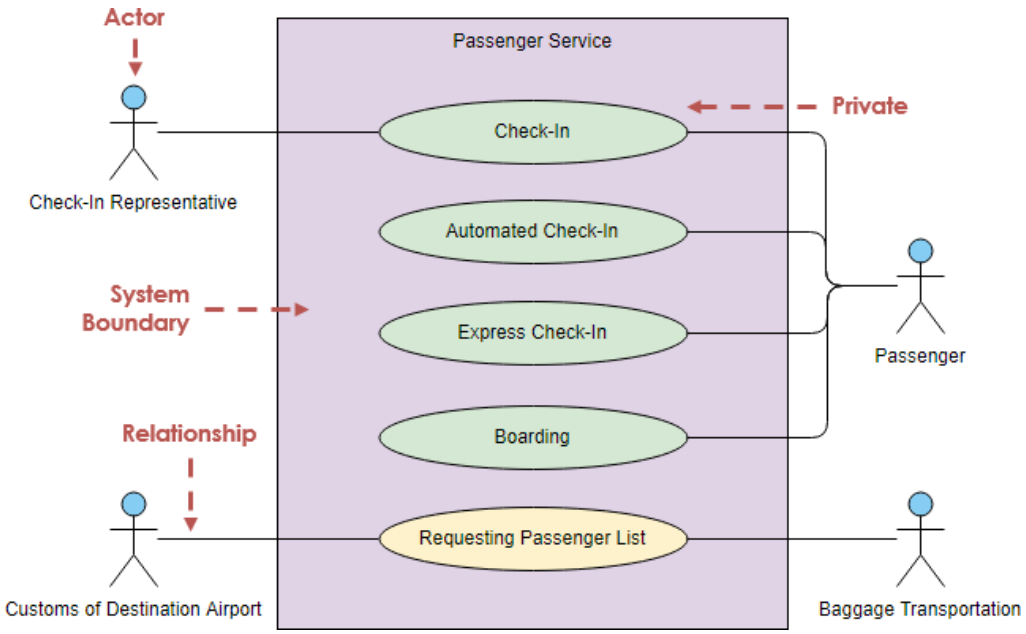


Figura 4: Ejemple de diagrama de casos de uso.

Flujo principal

1. Profesor inicia en el sistema
2. Sistema valida login (E1) y solicita ingresar cuatrimestre [Las excepciones (flujo) se ponen entre paréntesis]
3. Profesor ingresa cuatrimestre
4. Sistema solicita ingresar acción a ejecutar - Agregar, Modificar, Eliminar, Salir
5. Profesor ingresa acción [Comienzan los subflujos]
   - Agregar: Ejecutar subflujo S1: Agregar Curso a dictar
   - Modificar: Ejecutar sublfujo S2: Modificar curso a dictar ...

Subflujo S1 Agregar Curso a dictar

1. Sistema solicita ingresar materia
2. profesor ingresa materia (E2)
3. Sistema muestra los cursos correspondientes a la materia
4. Profesos selecciona los cursos a dictar
5. Sistema registra al profesor en los cursos seleccionados
6. El caso de uso comienza nuevamente en Sistema solicita ingresar acción a ejecutar [Se indica donde se sigue]

Excepción E1

1. Sistema solicita reingresar login o finalizar caso de uso.
   a) Ingresa login y continua en 2.
   b) Finaliza casos de uso.

5.3. Tipos de relaciones

- De asociación entre actores y casos de uso.
- De dependencia entre casos de uso. 2 tipos, de inclusión «INCLUDE» y de extensión «EXTEND»

  «INCLUDE» es para evitar repetir en varios casos de uso el mismo flujo de eventos. Llevo los casos comunes a un nuevo caso de uso común a ambos. Gráficamente es una linea punteada con «INCUDE». La llamada del flujo de eventos es ...
  n Include(Validar Usuario)
  n ...

  «EXTEND» se usa cuando en un caso de uso determinados eventos se dan solamente bajo ciertas condiciones. Gráficamente la flecha es punteada y va desde el caso de uso extendido al caso de uso base. El caso de uso base no se entera si se ejecuta el caso de uso extendido. Va acompañado por una etiqueta que tiene que coincidir en el flujo.
- De generalización/especialización entre actores/entre casos de uso.

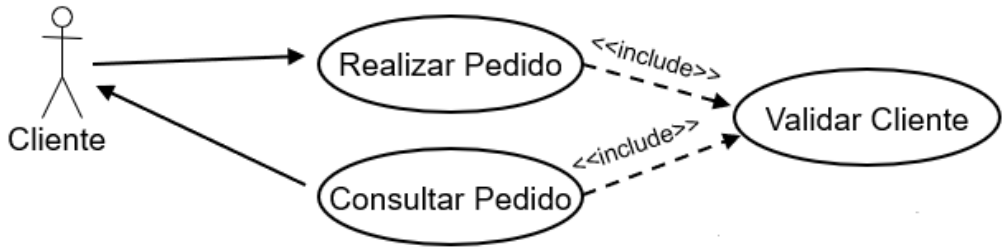


Figura 5: Ejemplo con «Include»

5.4. Caso de uso: Actor temporal

Indica el paso del tiempo, tiene una flecha punteada hacia el caso de uso. *Ej. Mensualmente, Diariamente, Semanalmente, Cada X tiempo*

Figura 6: Un actor temporal: Mensualmente

5.5. Caso de uso: Relaciones de especialización/generalización

Son actores que pueden tener una flecha hacia otro actor, para indicar que puede iniciar los casos de uso de ese otro actor.

5.6. Proceso de desarrollo iterativo e incremental

Voy realizando pequeñas partes del proyecto total. En cada iteración se van capturando requerimientos, análisis, diseño, construcción, testeo y despliegue. Es un mini ciclo de cascada. Cada iteración dura entre 2 y 6 semanas, esto permite que el usuario este probando mucho antes en comparación a cascada (interacción con el usuario constante). Ante cualquier error que ocurra, se esta a tiempo de arreglarlo.

5.7. Casos de Uso 2.0

Un slice es una porción de un caso de uso. Estos son hechos durante cada iteración del desarrollo iterativo e incremental.

Busca partir los casos de uso en porciones mas pequeñas, para que sean mas implementables en el lapso de 2 a 6 semanas. Proporciona mucha versatilidad.

5.8. Historias de usuario

Las historias de usuario son descripciones breves y simples de una funcionalidad, escritas desde la perspectiva que necesita una determinada capacidad del sistema. *Ej. Usuario, área de negocio, o cliente*

Componentes

- ID
- Titulo: Texto descriptivo de la historia
- Descripción: Como [rol], quiero [objetivo], para poder [beneficio]
- Estimación: Cuanto va a llevar implementarla (conocida también como puntos de historia)
- Valor: valor numérico que aporta importancia a la historia, generalmente de 1 a 5. El objetivo es maximizar el valor y la satisfacción percibida por el cliente o usuario en cada iteración.
- Dependencias: Procurar que no existe dependencia entre historias, los que no sea posible indicarla.
- Pruebas de integración: Clarifican el contexto en que ocurre la historia, y facilitan saber cuando están terminadas las historias.

Ej.

- COMO usuario QUIERO poder acceder al catalogo de cursos PARA poder tomar decisiones de los cursos en los cuales inscribirme.
- COMO usuario QUIERO poder buscar transacciones PARA poder detectar gastos innecesarios en mi cuenta durante cierto periodo de tiempo.

Buenas practicas

- Deben mapear a **una sola** funcionalidad del producto, software.
- Es una buena practica usar personajes en vez de roles. *Ej. Alumno en vez de usuario.*
- Se hacen en fichas.

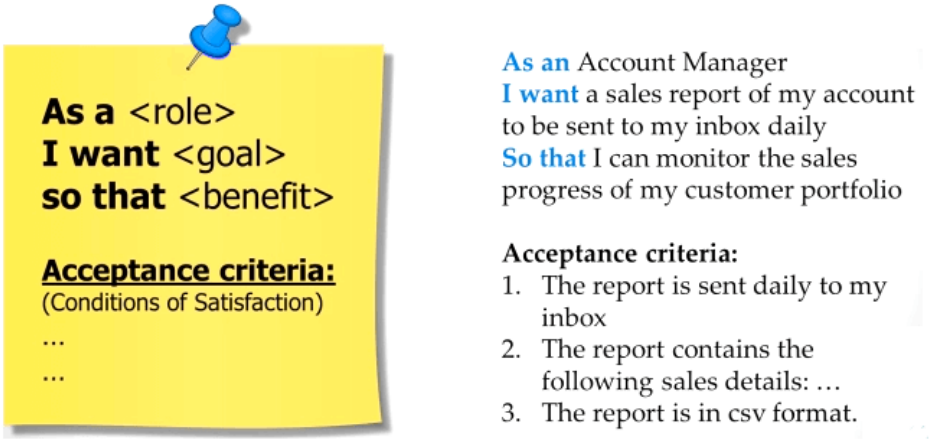


Figura 7: Modelo de una historia de usuario

INVEST

Es un metodo para escribir buenas historias de usuario.

- Independiente: No deben depender unas de otras, si ocurre tratar de dividir o combinar.
- Negociable: Deben ser negociables sus detalles entre cliente/usuario en la «conversación».
- Valor: Tiene que ser valiosa para el cliente.
- Estimable: Debe ser estimada con la precisión suficiente para ayudar al cliente o usuario a priorizar y planificar su implementación.
- Pequeña (Small): Como máximo una iteración.
- Testeable: Deben poder probarse y saber que la historia se completo con éxito.

Criterios de aceptación

Son una serie de preceptos que validan la implementación de la historia de usuario.

- Ayudan al desarrollador a implementarla de forma correcta.
- Se pueden mapear sobre los tests de funcionalidades.
- Ayudan a QA para dar el OK a la historia.
- Definen junto a la descripción la funcionalidad a implementar.
- Tienen solo 2 resultados: ÉXITO o FRACASO.
- Deben ser interpretados de una única manera por N personas, no deben ser ambiguos.
- Deben ser verificables por el cliente rápidamente.
- Tienen que ser completos, el grupo debe incluir todas las funcionalidades.

Ej.

- El nombre de usuario DEBE tener un valor, en caso contrario se mostrara el mensaje de error pertinente.
- Si el usuario y la contraseña son correctos, el usuario podrá acceder a la aplicación.

Conversaciones

En el mundo ideal los clientes tienen una idea muy clara de lo que quieren desde el principio, y nos lo cuentan con lujo de detalles sin dejarlo en el tintero. En la realidad esto no pasa.

Una historia de usuario abrirá normalmente una conversación entre el equipo de desarrollo y el cliente. Las historias de usuario no cambian a nivel conceptual, son las conversaciones las que las volverán mas detalladas.

Quinta clase

6. Modelo Conceptual

Dentro de los modelos de requerimientos.

6.1. Diagramas de clases conceptuales

Las clases representan los bloques de construcción mas importantes de cualquier sistema.

- Nombre
- Atributos: propiedades del elemento que estamos modelando
- Operaciones (Métodos): implementación de un servicio
- Responsabilidades: lo que le corresponde realizar a la clase,las obligaciones que deben realizar las clases para cumplir con los requerimientos. *Ej. determinar el riesgo de un pedido de un cliente - manejar criterios de fraude específicos del cliente*

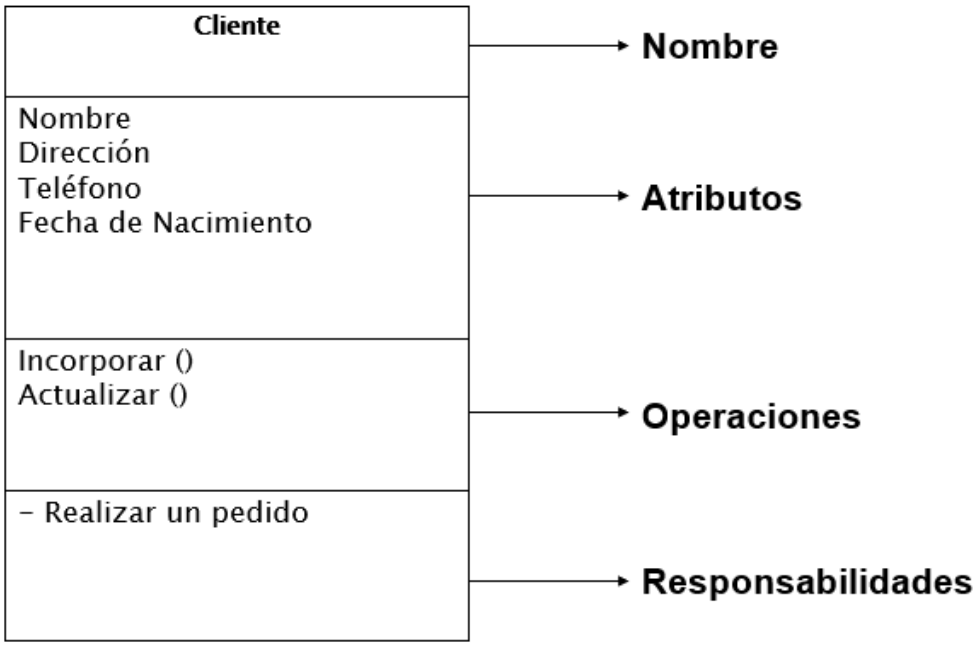


Figura 8: Modelo de una clase para el diagrama.

Es buena practica iniciar el modelaje especificando las responsabilidades, dejando para después los atributos y operaciones. Deben de contener al menos 'una' responsabilidad. (tratar de respetar el Single Responsability Principle)

Tipos de clase: MVC

- Las clases conceptuales (de entidad), son las necesarias para cumplir con las responsabilidades asignadas al sistema.
- Clase de entorno, maneja el dialogo entre el sistema y el medio externo. Modelan las interfaces del sistema.
- Clases de control, coordinan los eventos necesarios para realizar el comportamiento especificado en un caso de uso (Representan su dinámica).

Identificación

Para identificar clases, se puede usar la regla practica, usar sustantivos o frases sustantivas que describen responsabilidades del sistema. La otra forma es con tarjetas CRC (Colaboration Responsability Card), que es una técnica de modelado para refinamiento de clases.

De acá se tiene una lista inicial de clases, y con esto se hace un diagrama de clases participantes (sin colaboraciones).

Relaciones entre clases

- De asociación, es entre instancias que se encuentran en un mismo nivel. Tiene nombre de asociación, rol, multiplicidad. Existen la simple, de agregación, de composición y clase de asociación
  - Simple, relación de asociación entre objetos de una clase. *Ej, Nombre de asociación: Emplea* Recta que une las dos clases. El rol puede omitirse si es trivial. La multiplicidad es obligatoria.
  - Agregación, se tiene una clase que representa un todo, y otra una parte. Si se elimina una instancia del todo, las partes siguen existiendo. Tiene un rombo vacío del lado del todo.
  - Composición, fuerte relación de pertenencia y vida entre un todo y las partes. Si se elimina e todo, las partes asociadas mueren. Rombo negro del lado del todo. Un todo puede tener muchas partes, pero no puede haber una parte de muchos todos.
  - Clase de asociación, es una nueva clase que surge de la asociación de dos clases, contiene elementos que no le son propios a las clases relacionadas. Existen una sola vez, si se repite se convierte asociación simple.
- De generalización, especialización: Es una relación que conecta una clase general con una mas especializada. (Subclase) La clase hijo siempre es instanciable, la padre puede o no serlo. Si no lo es, se dice abstracta. el discriminador indica en base a que especializo.
- De uso, dependencia, conoce a determinada clase ya que la usa en algún momento, puede ser porque se la pasan por parámetro. No guarda una referencia a ese objeto.
- De realización, indica la implementación de interfaces.

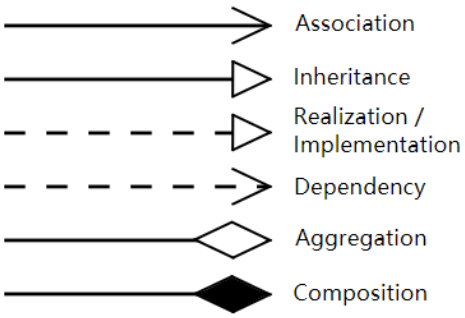


Figura 9: Los tipos de flechas

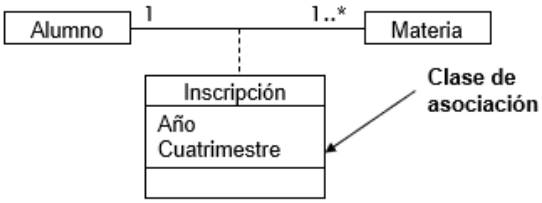


Figura 10: Ejemplo de una clase de asociación.

Sexta clase

7. Diagramas de interacción entre objetos

La interacción entre objetos comprende el conjunto de mensajes que son intercambiados entre objetos.

Estos diagramas describen la manera en que colaboran grupos de objetos para cierto comportamiento. Se usan cuando se quiere ver el comportamiento en un escenario predefinido.

Ambos tipos de diagramas son equivalentes. De cada uno puedo llegar al otro. Se enumeran los mensajes en ambos.

7.1. Diagrama de secuencia

Muestran el orden temporal de los mensajes y el paso de mensajes. 'Perspectiva cronológica de las interacciones'.

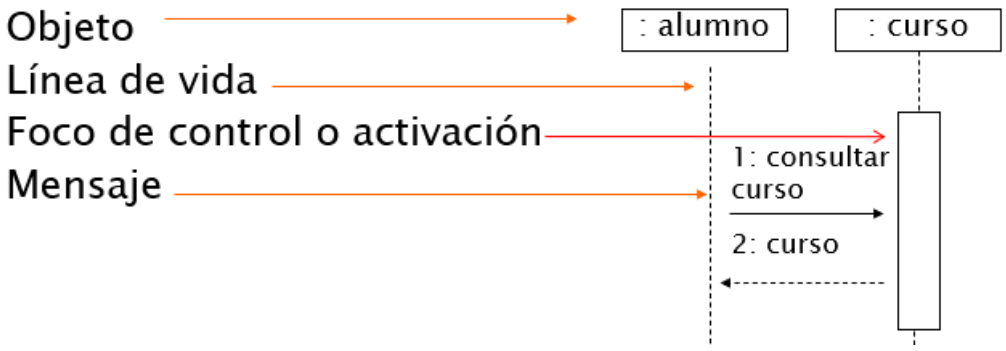


Figura 11: Elementos de un diagrama de secuencia

7.2. Diagrama de colaboración

Muestra la organización estructural de los objetos que envían y reciben mensajes. 'Perspectiva espacial de las interacciones'

Linea entre objetos que intercambian mensajes y otra linea indicando la dirección.

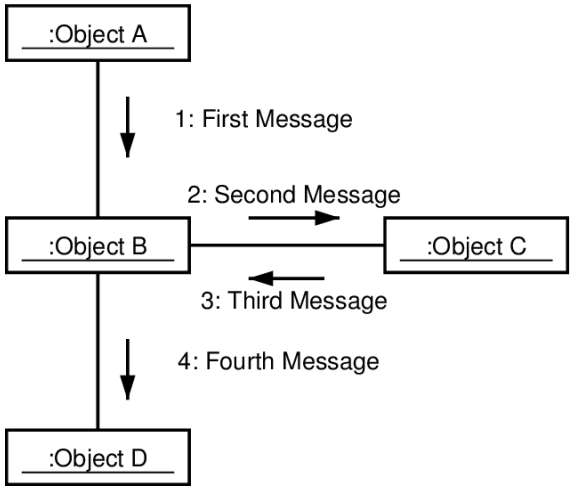


Figura 12: Modelo de un diagrama de colaboración

Séptima clase

8. Modelo de Estado de Objetos

Esta dentro de los modelos de requerimientos.

8.1. Estado

Es una condición o una situación en la vida de un objeto durante la cual satisface una condición, realiza alguna actividad o espera algún evento.

Tienen varias partes: nombre, transiciones, condiciones de transición, acciones, transiciones temporizadas, sub-estados, super-estados.

8.2. Diagrama de estados

Muestra los cambios de estado existentes en un objeto durante su ciclo de vida. Muestra el comportamiento que especifica las secuencias de estados por las que pasa un objeto. También muestran como se reflejan los cambios que se producen.

Estos diagramas son grafos dirigidos, tienen un estado inicial y final (pueden ser varios estados finales) diferenciados del resto. La transición entre estados es instantánea y se debe a la ocurrencia de un evento. (La definición de autómata básicamente.)

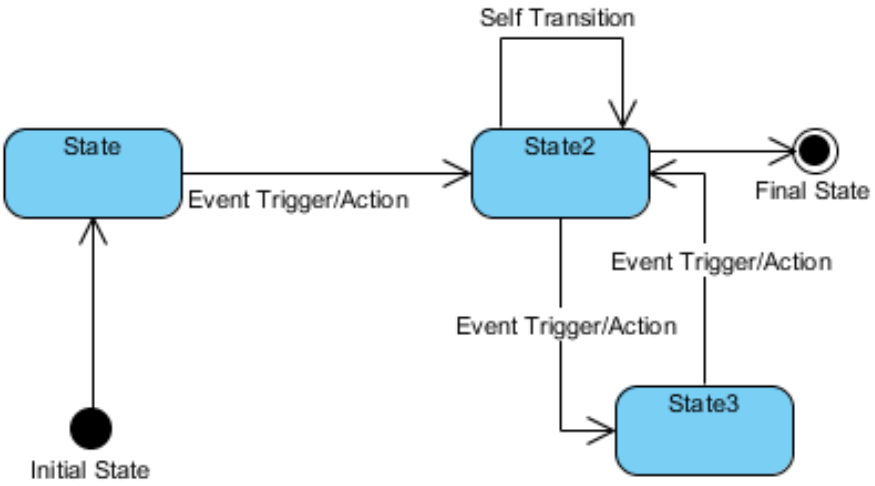


Figura 13: Diagrama de estados generico

Describe el comportamiento de un objeto a través de varios casos de uso. Es útil combinarlo con otros diagramas, *Ej. Diagramas de secuencia.*

- Acciones: es la solicitud de un servicio a otro objeto. *Ej. Evento[condición]/Acción - La acción podría ser otroObjeto.operacion()* Pueden tener palabras claves, Entry como una acción a ejecutar al ingresar al estado, un Exit como operación al salir, y un Do que es una acción a realizar mientras esta en el estado. Estas palabras clave están dentro del grafo, el estado. Pueden tener mas de uno.
- Transiciones Temporizada: indican un tiempo para el cambio de estado si no ocurre el evento que debía de ocurrir *Ej. "después de 30 segundos", puede ir si se anula una transacción porque no se pago.*

- Relaciones de Jerarquía - Superestados - Subestados: Buscan reducir la complejidad de los diagramas generalizando los estados. la entrada puede hacia estas generalizaciones puede ir a los estados específicos con una flecha directo, o un nuevo circulo indicando el comienzo (es preferible el circulo para que no se sepa desde los niveles superiores.).

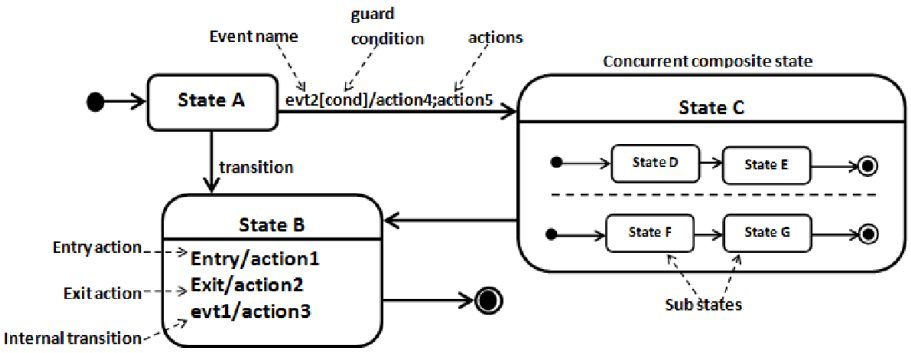


Figura 14: Diagrama de estados con acciones en State B

Antes de hacer el diagrama hay que contextualizar sobre el objeto al que le vamos a hacer el diagrama.

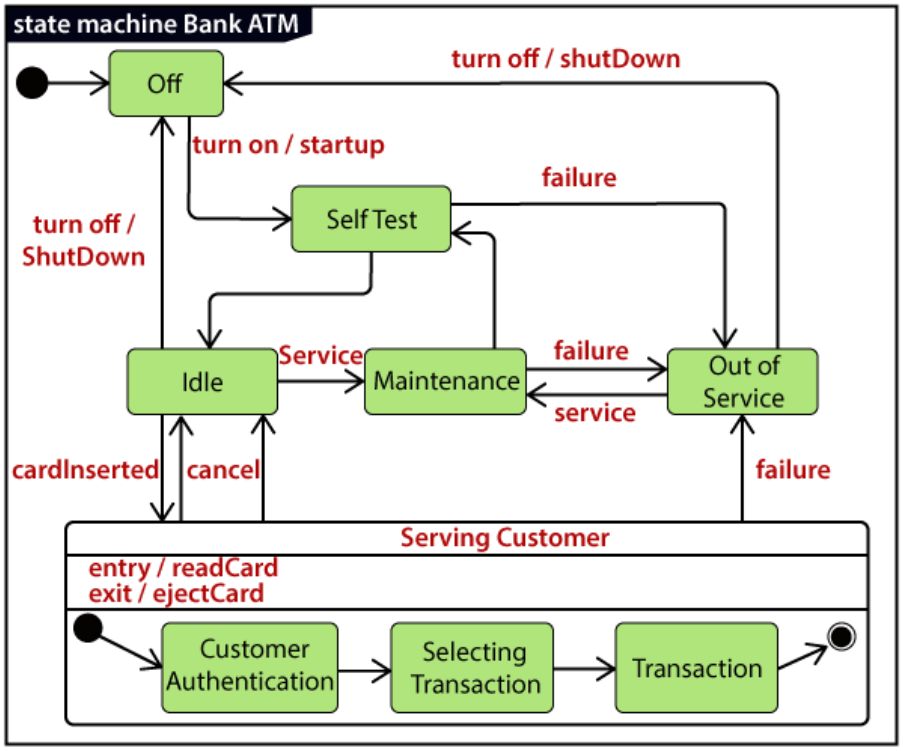


Figura 15: Ejemplo de un diagrama de estados

Octava y novena clase

9. Lean UX

Es 'dar luz a la verdadera naturaleza de nuestro trabajo y con *menos enfasis en los entregables* y con mayor foco en la experiencia que esta siendo diseñada (la solucion)'.

Se remonta al método Lean Startup

9.1. Lean Startup

Tiene cinco postulados.

1. Hay emprendedores en todos lados.
2. Emprender es administrar. (En su sentido mas amplio)
3. Generar conocimiento valido.
4. Innovación medible. (Si no podemos medir, no podemos controlar el éxito del proyecto)
5. Proceso Build-Measure-Learn (Construir-Medir-Aprender).

9.2. Lean UX vs Diseño Centrado en el Usuario (DCU)

Lean UX	DCU
Generar Conocimiento Validado	Hacer pruebas con usuarios
Innovacion Medible	Tomar metricas
Pivotear	Iterar
MVP(Minium Viable Product)	Prototipado iterativo
Costumer Development	Trabajo de campo

9.3. Build-Measure-Learn

Es un ciclo iterativo en el cual se parte de determinadas ideas, se construye rapidamente (code faster) un prototipo, se hacen mediciones (measure faster), y con estas mediciones se puede aprender (learn faster). Una vez completada la vuelta se repite.

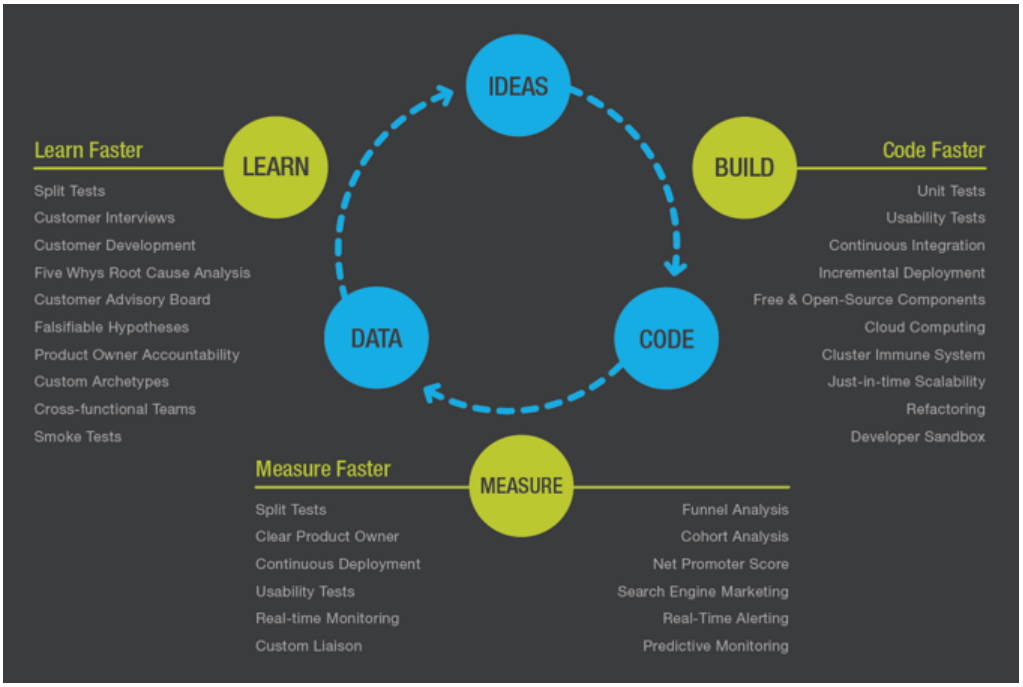


Figura 16: Ciclo que se sigue

Consideraciones al emprender un proyecto

- Hay que saber distinguir entre lo que quiere el emprendedor y lo que quieren los usuarios.
- Los emprendedores son los padres de la 'criatura'. La capacidad de juicio critico esta sesgado. El que tiene que opinar del producto son los usuarios, tienen que dar el juicio.
- Las personas 'dicen' es diferente a lo que las personas 'hacen'. (Tenerlo en cuenta cuando se hacen las pruebas, se mejora cuando se tiene en cuenta lo que hacen) (Observar las conductas de las personas) *Ej. Focus Group*
- La capacidad de 'resiliencia' de los emprendedores, muy necesaria. Se necesita una alta capacidad de 'resiliencia'.
- Hay que poder medir el 'valor' que perciben los clientes a nuestro producto. No es agregar *features* al producto continuamente si es que los clientes no lo perciben.
- Hay que poder identificar 'quienes' van a ser nuestros futuros usuarios.
- Siempre se va a tener un alto grado de incertidumbre.
- Nosotros como emprendedores, no somos los usuarios pero decidimos como si lo fuéramos, esto se debe de evitar.

9.4. Declaración de supuestos

El método Lean UX no comienza con requerimientos, empieza con supuestos. A partir de estos se crean y validan diferentes hipótesis. De estas validaciones se miden los resultados, comparándolos con los esperados.

En el procesos suponemos que el producto a desarrollar esta orientado a personajes (arquetipos de personas).

9.5. MVP

Un MVP es un producto mínimo viable que se usa para validar las hipótesis. Esta formado por artefactos que crea el estudio de diseño. Cada etapa de diseño tiene un tiempo estimado de iteración.

'Un producto mínimo viable es la versión de un nuevo producto que permite a un equipo obtener la máxima cantidad de aprendizaje validado sobre los clientes, con el menor esfuerzo posible' - Eric Ries

El objetivo del MVP es obtener aprendizaje y generar valor en el producto. Los MVP se pueden clasificar en prototipos, descripción, y producto con funciones mínimas. La calidad de un MVP incrementa a medida que se van haciendo las iteraciones en el ciclo de vida.

La calidad de los MVP debe ser: *Keep. It. Simple. Stupid.* Fallar rápido y barato, debido a que es una inversión en un prototipo que después puede cambiarse.

Al comenzar la planificación de un MVP hay que pensar: ¿Existe una necesidad por parte de un usuario para la funcionalidad a implementar? ¿Existe valor en la funcionalidad a implementar? ¿Existe usabilidad en la funcionalidad a implementar?

Prototipos

Se presenta a clientes potenciales una simulación del producto con suficiente nivel de detalla para comprender la funcionalidad a validar. No es un producto.

Descripciones

Se capta la atención de clientes potenciales con una descripción e información general sobre el producto/servicio. Son los mas baratos y simples. *Landing page, correos, vídeos*

Producto

Es código funcionando, no todo el desarrollo. Es el mas costoso. Solo se realiza la funcionalidad necesaria para validar la hipótesis.

A quien se dirige

- Innovator: obtienen el producto recién sale
- Early adopters: adoptan el producto de forma temprana
- Early majority:
- Late majority
- Laggards: La inversa a los innovadores.

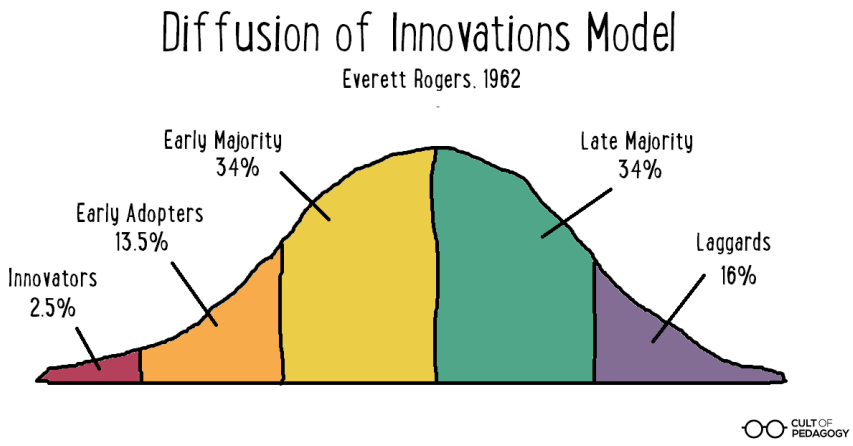


Figura 17: curva de adopción de la innovación

Obstáculos al desarrollo

- Tiempo
- Alcance
- Recursos
- Legales, patentes
- Miedo a la competencia
- Miedo al riesgo
- Miedo a la frustración

Experimento

Es el comienzo del proceso de validación. Se utiliza para probar el MVP con usuarios, para comprobar las hipótesis.

Se debe mostrar o probar con clientes potenciales. Estos deben utilizar y explorar el prototipo lo mas libremente posible, para obtener mas feedback.

Prueba de estabilidad, identifican problemas de usabilidad en el producto (prototipo). Permite reunir información cuantitativa y cualitativa del desempeño.

Participa el facilitador (conduce la prueba), los observadores (toman notas), el wizard of Oz (simula la interacción en el prototipo si es necesario), y los usuarios (personas con el perfil necesario, protopersonas)

Es necesario definir el perfil del usuario, el contexto de uso, y la meta.

1. El facilitador pide comenzar la prueba
2. El facilitador lee el primer escenario y tarea, el usuario lo inicia
3. El usuario dice que piensa, facilitador y observador toman notas de lo que dice y hace el usuario
4. Continua la prueba a través de los escenarios y tareas
5. Se logra la meta o el tiempo, el facilitador termina la prueba

6. El facilitador agradece al usuario por su participación

Se evalúa el producto, no al usuario en las pruebas de usabilidad.
Se realiza Lean User Research, se realiza un rápido registro de los aspectos a considerar.

- Stoppers: Dudas y problemas que tuvieron los participantes, cuando se quedan trabados
- Values:
- Feedback: Continuo (calendario permanente de actividades, simplificar el protocolo y entorno de experimentación, simplificar reclutamiento de usuarios, el equipo trabaja todo el tiempo, se reúne el equipo rápidamente luego de la prueba) y colaborativo (todo el equipo trabaja en conjunto sobre la declaración de hipótesis y construcción del MVP)
- Recommendations: Sugerencias del moderador desde la perspectiva del usuario para corregir problemas de usabilidad y mejorar la experiencia.

Herramientas

- Business Model Canvas, gestiona el modelo de negocio
- Validation Board, gestiona el ciclo de vida del producto
- Scope Canvas, centrado en la información relevante del producto, ver en 10.

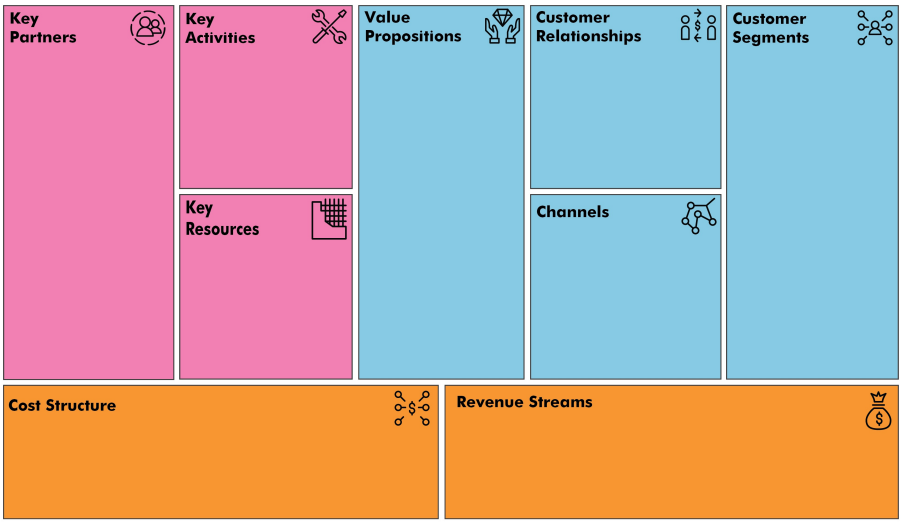


Figura 18: Plantilla Business Model Canvas

Figura 19: Plantilla Validation Board

10. Scope Canvas

El Scope Canvas combina elementos que ya se venían trabajando en proyectos de lean UX de forma separada. Nos permite establecer formas de observar el éxito del proyecto. Es una foto del momento.

El Scope Canvas no nos dice como vamos a trabajar.

10.1. Plantilla

Figura 20: Plantilla de Scope Canvas

- Hay que definir todas las partes que aparecen en la imagen.
- Con Scope Canvas se logran equipos de trabajo mas alineados en torno a las metas comunes. Que se tenga una cultura mas explicita.
- La visión de los actores clave desde el principio del producto que se esta ideando.

El propósito

- ¿que hace que esto valga la pena? ¿Por que hacemos esto? ¿Que nos mueve?
- Nos mueve hacer esto porque...
- El propósito debe servir a la organización y a los usuarios por igual, muy ambicioso y muy estimulante. Hay que ser sincero con las motivaciones personales.
- *Ej. Que todos puedan ahorrar energía - Conectar al mundo en un instante - Una sociedad mas respetuosa - Organizar la información del mundo (Google)*

Necesidades

- Esta en la parte de los usuarios. ¿Que problemas y/u oportunidades insatisfechas tenemos el potencial de atender?
- Nuestros usuarios necesitan/desean...
- Puede haber varios segmentos de usuarios. Para proyectos multi-sided se defina al menos una necesidad por cada segmento de usuario o cliente.
- Hay que evitar mencionar posibles soluciones al declarar necesidades. (Evitar el ¿Como?)
- Los emprendedores deben detectar estas oportunidades. Son valor agregado para los usuarios que ellos aun no visualizan. Para validar las necesidades hay que investigar, crear historias (storytelling), y sintetizar.
- *Ej. Sacar su certificado sin ir a las oficinas - Ahorrar energía - Aprender un nuevo idioma*

Objetivos

- Son del negocio a corto plazo. ¿Cuales son los objetivos del proyecto? ¿Que esperamos obtener? ¿cual es la prioridad?
- El objetivo es...
- Los objetivos deben ser cuantificables y medibles. No necesariamente hay que determinar uno con numero. Hay que evitar los objetivos vagos (no se pueden medir). Los objetivos deben relacionarse con la adopcion de usuarios, no son solo internos.
- En este momento hay que preguntarse como se sustentara el proyecto. Puede ser útil realizar el Buisness Model Canvas antes de llenar esa sección.
- *Ej. Reducir el tiempo de compra en un 50 % - Crecer en un 25 % anual*

Motivadores

- Se tienen motivaciones de mediano y largo plazo. ¿Que mejora entregamos a los usuarios para satisfacer sus necesidades?
- Los usuarios adoptaran el producto si les entregamos...
- Las necesidades y los motivadores están relacionados.
- Los futuros usuarios ya están intentado satisfacer sus necesidades. Antes de definir los motivadores debemos investigar cuales son las maneras actuales de satisfacer dichas necesidades, y de que forma es mejor nuestra idea.
- La innovación debe reflejarse en los motivadores.
- Los motivadores los validamos a través de las acciones. Si el motivador es correcto, los usuarios adoptan el producto realizando acciones.

Acciones (Comportamiento)

- ¿Que acciones especificas de los usuarios implican conversión? ¿Como sabremos si estamos entregando los motivadores correctos?
- Esperamos que los usuarios...
- Las acciones tienen que ser cruciales y observables.
- *Ej. Comprar- Transportarse en bicicleta en lugar de en auto - Registrarse*

Impacto

- El impacto es a mediano y largo plazo. ¿cual es el potencial del proyecto si nos va bien? ¿En que esperamos convertirnos si nos va bien?
- Nuestras metas de largo plazo son...
- Es el impacto en el negocio. Nos permite visualizar el éxito de largo plazo del proyecto. Nos acerca al propósito. Siempre es observable por terceros. tiene un efecto poderoso en la motivación.
- No hay que mezclar impactos con objetivos. Impacto es observable, el objetivo es medible.
- El impacto debe reflejas un éxito masivo, a gran escala.
- Hay que prestar atención a los impactos no esperados y corregirlos.

Métricas

- ¿Como sabemos que los objetivos se están cumpliendo>¿Donde medimos? ¿Como medimos?
- Mediremos ... para saber si estamos logrando el objetivo
- Tienen que estar expresadas en datos. Nos cuentan la historia de como le fue a nuestros usuarios. No hay que engañarse.
- Las métricas deben responder a las acciones de los usuarios y a los objetivos que estamos siguiendo.
- *Ej. Recurrencia de uso - tiempo para completar una tarea - SUS (system Usability Scale) - NPS (Net Promoter Score)*

10.2. Reglas

- Se usan *post-its* y lapices grandes. Se debe de ser simple y breve. Hay que debatir, no juzgar.
- Dura típicamente 2 horas. Máximo 15 personas.
- El debate permite simplificar todos los *post-its*.

Décima y Onceava clase

11. Desarrollo Iterativo e Incremental

Procesos de desarrollo de software, es una forma disciplinada de asignar tareas y recursos en un proyecto de software con el objetivo de tener un producto de calidad dentro de plazos y presupuestos predecibles.

Permite comprender de forma creciente los requerimientos a medida que se va haciendo crecer el sistema. Se logra reducir los riesgos del proyecto

Se vio en 5.6.

12. Manifiesto Ágil

12.1. Valores

Son 4, hay que mirarlos como una balanza, quien tiene mayor peso.

- *Valorar: al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas.* La gente es el principal factor de éxito de un proyecto. Significa darle mayor preponderancia a las personas. Sobre los procesos y herramientas no quiere decir de no darles importancia, dice de no ponerlas sobre las personas.
- *Desarrollar software que funciona mas que conseguir una buena documentación.* No producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante.
- *La colaboración con el cliente mas que la negociación de un contrato.* Tiene que haber una interacción constante entre el cliente y el equipo de desarrollo. Llevar al pie de la letra al contrato (con plazos y costos preestablecidos) puede llevar al fracaso.
- *Responder a los cambios mas que seguir estrictamente un plan.* La habilidad de responder a los cambios determina el éxito o fracaso del proyecto.

12.2. Principios

Son 12, se derivan de los valores.
Principios generales

1. *La prioridad es satisfacer al cliente mediante entregas tempranas y continuas de software que le aporten valor.* Un proceso es ágil si a las pocas semanas de empezar ya estoy entregando software que funcione (aunque sea rudimentario). El cliente decide si pone en marcha el software con la funcionalidad que funciona o no.

2. *Dar la bienvenida a los cambios.* Deben verse como algo positivo. La estructura debe ser flexible para poder incorporarlos sin demasiado coste.

   Relativos al Proceso de desarrollo

3. *Entregar frecuentemente software que funcione desde un par de semanas a un par de meses.* (2 semanas a un mes y medio)

4. *La gente del negocio y los desarrolladores deben trabajar juntos a lo largo de todo el proyecto.* Necesita ser guiado por el cliente.

5. *Construir el proyecto en torno a individuos motivados.* La gente es el principal factor de éxito, todo lo demás queda en segundo plano (proceso, entorno, gestión, etc). Los equipos motivados tiene muchísimo mejor desempeño.

6. *El dialogo cara a cara es el método mas eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.* Los miembros de equipo deben hablar entre ellos, la principal forma de comunicación.

7. *El software que funciona es la medida principal de progreso.* Se mide en el estado por el código generado que este funcional. *Proyecto en 50 % si el 50 % de los requisitos esta cubierto*

8. *Los procesos ágiles promueven un desarrollo sostenible en el tiempo.* No se trata de desarrollar lo mas rápido posible, si no de mantener el ritmo durante la duración del proyecto, asegurando que la calidad de lo producido es máxima.

   Relativos al Equipo de desarrollo

9. *La atención continua a la calidad técnica y al buen diseño mejora la agilidad.*

10. *La simplicidad es esencial.*

11. *Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por si mismos.* Todo el equipo es informado de las responsabilidades, estas recaen sobre todos sus miembros. El propio equipo decide la forma de organizarse.

12. *En intervalos regulares, el equipo reflexiona respecto a como llegar a ser mas efectivo, y según esto ajusta su comportamiento.*

13. SCRUM

Es un proceso iterativo e incremental utilizado para la construcción de productos de Software. Se compone de iteraciones llamadas Sprints, estas son fijas en el tiempo. Se recomienda una duración de 1 a 4 semanas. El objetivo de las Sprints es construir un incremento del producto que potencialmente pueda ser usado por los clientes. No nos serviría algo que no pudiéramos utilizar al final de la iteración.

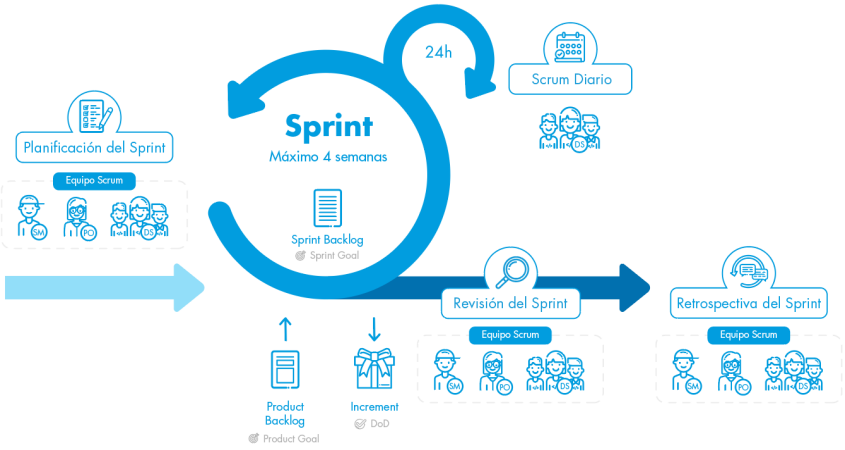


Figura 21: Proceso Scrum

13.1. Artefactos

Pila de producto

Una vez que tengamos claro que personas son a las que vamos a aportar valor, es necesario recopilar en un único sitio todas las ideas, funcionalidades y elementos que van a componer el producto. Estos van ordenados por valor de negocio (arriba los que aportan mas). Esto se llama Pila de Producto o Product Backlog. Los elementos se conocen como PBIs (Product Backlog Items), estos contienen como mínimo una descripción, la prioridad que tengan, la estimación en tiempo y costo, y el valor que aporta. Los PBIs podrían ser historias de usuario.

Las características de las pilas de producto deben de ser DEEP (Detallado, estimado, emergente, priorizado)

- Detallado: los PBIs próximos (arriba de la pila) deben de tener el mayor detalle.
- Estimado: Cada uno nos tiene que servir para planificar, mas arriba, estimaciones mas precisas.
- Emergente: Una pila de producto no es estática, va cambiando con el tiempo, se agregan/eliminan elementos.
- Priorizado: Mas importantes mas arriba de la pila, menos importantes abajo. Para priorizar hay distintos métodos. *Ej. Método MoSCoW*[1] *, Método Kano, Método ROI.*

[1] M:must have - S:should have - C:could have - W:won't have

PBIs

Contienen como mínimo una descripción, la prioridad que tengan, la estimación en tiempo y costo, y el valor que aporta. Los PBIs podrían ser historias de usuario (Regla 3Cs - Card - Conversation - Confirmation). (mencionados arriba)

Pila del Sprint, Sprint Backlog

El conjunto de PBIs y las tareas se llama Pila del Sprint (seleccionados por el Product Owner como mas importantes). Para llegar acá de la pila de producto, se hace una reunión de planificación.

Se tiene Pendiente, En progreso, y Terminado. Las tareas mas arriba tienen mayor valor.

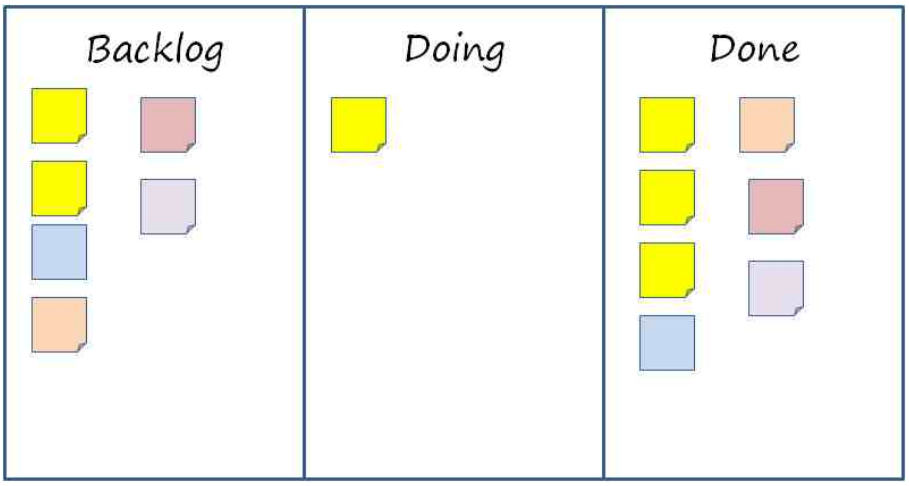


Figura 22: Sprint Backlog

Incremento del producto

Es el resultado de cada Sprint. Buscan que potencialmente se pueda poner en Producción, aportar valor al cliente.

13.2. Roles

Product Owner

Para gestionar toda esta comunicación y gestionar la pila de producto existe un rol especifico llamado Dueño de producto (Product Owner), es el responsable de mantener la visión del producto que se va a construir. Tiene que maximizar el valor entregado al final de la Sprint. Asiste a las reuniones de planificación y revisión (DEMO) del Sprint con el equipo de planificación para transmitir la visión del producto.

Scrum Master

Facilita todas las reuniones que se realicen, acompaña al equipo a resolver las problemáticas que aparezcan en el desarrollo. Es el vigía del proceso, vela para que se lleve a cabo sin olvidar que esta pendiente de las personas que conforman el equipo. Debe de tener una buena visión de la organización, buena comunicación y gestión, ser un líder.

Clientes, Stakeholders

Para empezar a construir un producto, antes debe haber una idea o necesidad. Las personas a las que construimos el producto se les llama Interesados o Stakeholders. Hay que saber distinguir a un usuario (cualquiera que use la aplicación) de los clientes (los que pagan por la aplicación).

Equipo de Construcción

El equipo de construcción es el encargado de la construcción del producto en cada sprint. El equipo debe estar enfocado sobre una parte pequeña del producto. Esta formado por todas las personas que desarrollan, desarrolladores, diseñadores, arquitectos, testers, etc. El equipo es multidisciplinario (entre todos puedan dar servicio al proyecto), tienen que poder auto-organizarse (asignar tareas y estimarlas).

El equipo no debe ser de mas de 9 personas.

13.3. Reuniones

El Sprint conviene que tenga una duración fija, busca un ritmo sostenido.

De planificación

La primer reunión se llama Reunión de Planificación o Sprint Planning, se selecciona de los PBIs los mas importantes como el desgranado de estos tareas técnicas. Acuden el dueño de producto, el equipo de construcción, el Scrum Master. El resultado es la Pila del Sprint.

El Product Owner explica al equipo de construcción la funcionalidad a construir.

Daily

La Daily, para realizar una gestión de riesgos adecuada y fomentar la comunicación y sincronización entre los miembros del equipo. Se busca detectas problemas e impedimentos que afecten al desarrollo del Sprint. Es una reunión corta de no mas de 15 minutos. Se realiza todos los días. Busca delinear el plan para el día.

De revisión, Demo

Al finalizar el Sprint, se realiza la Reunión de Revisión (Demo) donde se inspecciona todo lo realizado por el equipo de construcción. Acuden el Product Owner, el equipo de construcción y el Scrum Master. Puede ir cualquier persona que quiera ver lo entregado.

De retrospectiva, Retro

Después de la Demo, el equipo se reúne para inspeccionar el proceso y la forma en que trabajaron. El objetivo es detectar posibles problemas y dar soluciones (Foco en las personas y proceso). Se llama Retro. Acuden todos los miembros del equipo de construcción, el Scrum Master y el dueño del producto. La Retro la facilita el Scrum Master

Se pone el foco en las personas y el proceso dejando de lado el producto, acuden todos los miembros del proyecto.

De refinamiento

Existe otra reunión cuyo objetivo es trabajar sobre los elementos futuros que entraran en el Sprint. Se llama reunión de refinamiento, se realiza cuando lo decide el equipo SCRUM. Busca añadir detalle, estimaciones y orden a los PBIs.

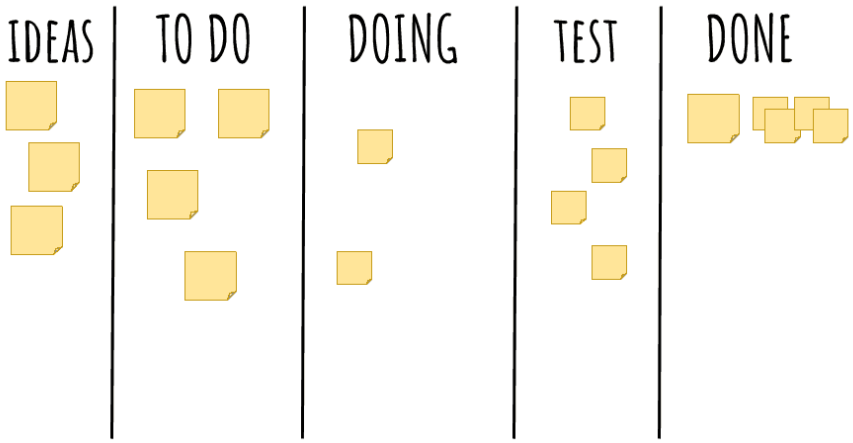


Figura 23: Tablero Kanban

13.4. Tablero Kanban

Se enfoca en la entrega de servicios de una organización, una o mas personas colaboran para la entrega.

Doceava clase

14. Proceso de Desarrollo Unificado - RUP

Es una forma disciplinada de asignar tareas y responsabilidades en un proyecto de desarrollo de software, con el objetivo de obtener un producto de calidad. RUP (Rational Unified Process), no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización. Divide el proceso de desarrollo en ciclos, teniendo un producto al final de cada ciclo.

Mejores practicas

- Desarrollo iterativo e incremental del software, lo visto en las secciones anteriores. (N iteraciones, cada una con su mini-ciclo en cascada) en las primeras iteraciones hay mas esfuerzo en captura de requerimientos y modelo de negocio, a medida que se avanza en el proyecto el tiempo se dedica mas a producción, testeo y despliegue.
- Proceso dirigido por casos de uso - RUP contribuye a obtener, organizar, documentar, rastrear, captar y comunicar los requerimientos. Los casos de uso probaron que son una buena forma de obtener los requisitos. Los casos de uso se integrarían a lo largo de todo el proceso.
- Uso de arquitecturas basadas en componentes (reusabilidad) - Se basa en poder diseñar tempranamente una arquitectura base ejecutable, esta debe de ser flexible, fácil de modificar, intuitivamente comprensible, y promover la reusabilidad.
- Trabajar con modelos visuales (UML)
- Verificar la calidad del software - RUP ayuda a planificar, diseñar, implementar, ejecutar y evaluar pruebas que verifiquen las cualidades. Esto tiene que ser parte del proceso de desarrollo, no de un grupo independiente.

- Control de cambios - RUP indica como controlar, rastrear y monitorear los cambios dentro del proceso iterativo de desarrollo. Ante un cambio se evalúa la relación costo/beneficio de introducirlo en la versión actual.

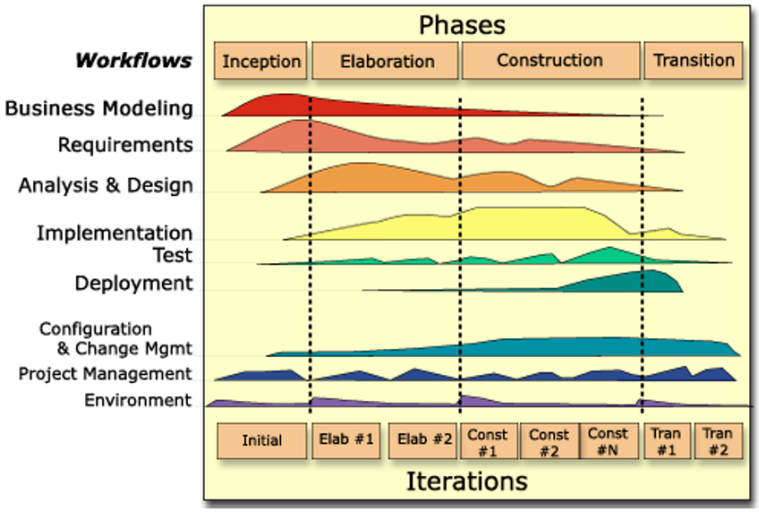


Figura 24: Distribución del tiempo en las fases.

Fases de RUP

Cada ciclo tiene estas fases.

1. Inicio (o concepción): Se define el ámbito y los objetivos del proyecto. Se definen las funcionalidades y se identifican las entidades externas con las que se trata.
2. Elaboración: Busca una arquitectura del sistema estable (el hito). El dominio del problema busca finalizar los casos de uso. Tanto la funcionalidad como el dominio del problema se estudian en profundidad.
3. Construcción: Busca tener el producto beta (capacidad operacional inicial - hito). Se tiene el producto de software integrado y corriendo en la plataforma adecuada. Se pone mucho énfasis en la producción eficiente.
4. Transición: Es la release del producto, busca pasar el software desarrollado a la comunidad de usuarios. Hay que tener autosuficiencia por parte de los usuarios.

Componentes del proceso

- Buissness modeling
- Requirements
- Analysis & design
- Implementation
- Test
- Deployment

Componentes de soporte al proceso

- Environment
- Project Managment
- Configuration & change managment

Elementos

- Trabajador: Identifica un rol especifico en el proyecto de software. *Ej. diseñador de objetos, de casos de uso, etc*
- Actividad: Es una tarea que es responsabilidad de un trabajador. *Ej. Diseñar objetos, Especificar casos de uso*
- Artefacto: Resultado parcial o final que es producido y usado durante el proyecto. *Ej. un documento, pruebas, requisitos*

15. Bibliografía de la materia

- The Unified Modeling Language Reference Manual, Rumbauch, Jacobson, Booch
- The Unified Software Development Process, Rumbauch, Jacobson, Booch
- UML distilled, Third Edition, Fowler
- Leax Ux: Applying Lean Principles to improve user experience, Jeff Gothelf, Josh Seiden