**Unidad I**

EL SOFTWARE ES MALEABLE (cantidad de energía necesaria para modificar el software es mínima, los clientes creen que es fácil y se de…)

El **software** es la parte intangible y lógica de miles de dispositivos.

**Es un colección de conocimiento modelado.**

* Lo primero que se hace es el análisis de requerimientos.

a) Todo inicia con una reunión informal, averiguar cosas importantes que no sabemos

b) El problema es definir el objetivo del software (suele ser lo primero que se dice en la charla)

**MODELO DE REQUERIMIENTOS**

Modelo de casos de uso: conjunto de símbolos, diagramas, actores, escenarios. Da origen, es la fuente por la cual nosotros u otros analizaran requerimientos y pensaran en formas de satisfacer esos requerimientos.

<aside> 📌 El actor iniciador es el que pone en marcha el sistema, el actor beneficiario es el que recibe la salida del sistema, generalmente es una máquina, siempre es controlado por el programador.

</aside>

Modelo de análisis: los casos de usos terminan en realizaciones.

El código fuente no puede hacer que una maquina funcione si no tiene la codificación, la compilación, el ensamblado.

*No tenemos la capacidad de la verificación a través de un modelo, nuestro producto nunca va a estar verificado. Se hacen pruebas para garantizar cada vez menos probabilidades de error. Los estándares de calidad no pueden ser dados en el producto sino en el proceso, un proceso de calidad aumenta la probabilidad de un producto de calidad.*

El software siempre es requerido por una persona, y la dificultad es entender qué es lo que quiere la persona.

Otra dificultad es la complejidad de un software, ***la complejidad es constante y proporcional a las herramientas que se han realizado para solucionar esa complejidad.***

Cada vez que lo vamos actualizando y satisfaciendo, pide más, por lo tanto aumenta la complejidad.

La teoría de la bala de plata de Broc: el hombre lobo solo puede matarse con una bala de plata.

Las tecnologías se transforman en el “monstruo”, se transforman en nuestros problemas, no existe una bala para matarlo.

**Cualidades deseables en el software**

* Corrección funcional
* Confiabilidad
* Robustez

El software por definición no se rompe, se deteriora. Cuando se actualiza y se va llenando de cosas, el sistema se deteriora, quedan cosas que ocupan espacio y no hacen nada, como una variable que haya quedado en alguna actualización que no se borró.

***El software es maleable, la cantidad de energía necesaria para producir un cambio es prácticamente nula. Es un problema.***

El cliente no llega a entender el esfuerzo que lleva modificar un software.

El software necesita tener calidad, que viene de las cualidades deseables

**Lista descriptiva no taxativa (conjunto de elementos pero no todos):**

(taxativa es todos los elementos)

* Hay cualidades que se aplican al producto, cualidades que se aplican al proceso y cualidades que se aplican a ambos.
* **Hay cualidades que solo las ve el equipo de desarrollo (internas) y otras que solo ve el usuario (externas, son imprescindibles), y otras que ven ambos (duales).**
* La satisfacción del cliente es el punta pie de un nuevo negocio, incluso si no es nuestro.
* El éxito de nuestro futuro depende de la opinión del cliente acerca del producto y que piensa el cliente de nosotros.

**Corrección fundamental: Importancia superlativa**

* El software es correcto si funciona de acuerdo a los requerimientos
* La corrección es necesaria pero no suficiente
* cualidad interna

**Corrección → Importancia superlativa**

* Propiedad interna, aplicable exclusivamente al producto, solo lo ven los desarrolladores.
* El software hace lo que los requerimientos plantean
* **Total y absoluta correspondencia entre funcionalidad y requerimientos**

**Software correcto**

* Hace lo que yo quiero que haga, pero hace lo que el cliente quiere que haga?
  + Requerimientos están expresados en lenguaje técnico
    - Interpreta una necesidad expresada sin ambigüedades
    - Glosario para los términos significativos →Cada ambiente tiene su vocabulario

“Nadie es como se piensa que es”

* Retroalimentación temprana para quitar los malos entendidos

**Robustez:**

**→ apoya la confiabilidad**

* cualidad externa
* Cuando el sistema se comporta de manera satisfactoria ante situaciones no previstas.
* Lo que separa la robustez de la corrección es la especificación funcional
* muy apreciada por los clientes

“La programación sencilla es la mejor, nadie se levanta minas programando” - Braca

A mayor simplicidad, menos probabilidad de error.

* Para generar software robusto hay que tener ***buenas prácticas profesionales:***

1. no extenderse más de 48 horas semanales, estar descansado a la hora de trabajar, no se puede vivir en contingencia.
2. usar buenos motores en la base de datos (sistema transaccional por ejemplo)
3. que los sistemas sean probados por segundos, un programador nunca prueba su propio código, un programador se equivoca porque *no sabe del tema.* Otra fuente de error es que el programador no conoce la plataforma (lenguaje por ejemplo). Tercera fuente de error es el estado de ánimo, una persona feliz tiene menos chance de equivocarse.

“cuando hago la primer línea cornudo, en la quinta línea sigo siendo cornudo” -Braca

* si desconozco del tema en la primer línea, en la quinta línea, sigo sin conocer el tema.
* el lugar mas propicio de encontrar un error es en la inmediaciones del primer error encontrado.
* es probable que si en la primer versión el sistema era correcta, en la segunda sea robusto.
* El código es lo último que se toca, primero se ve si se puede cambiar el hardware, luego la base de datos y por último el software, ya que hacerlo desde un inicio es más caro.

**Confiabilidad (la mas importante de todas) $$$**

* es una cualidad comercial, guita = felicidad, de esto vivimos
* cualidad externa
* Creo en los resultados del software

“Si un software es confiable es correcto” - Ghezzi

* el cliente debe confiar en el producto, para eso yo debo confiar en mi producto.
* La confiabilidad se construye, a través de la relación entre cliente, producto y productor.
* Puede tener errores pero si hay confianza el cliente queda contento.
* La confiabilidad no depende solamente del producto, existe cierta predisposición
  + Influye la relación
  + La forma en la que llevamos la relación con el cliente influye en la confianza en el producto
* También depende del cliente
  + Dos personas con el mismo producto pueden tener una relación diferencial
* tiene que ser robusto, funcionar bien ante situaciones no pensadas.

Tarea: 5 cualidades que debería tener un sistema:

1. Intuitivo: de fácil entendimiento
2. Ágil: que responda en el tiempo de reacción que se necesita o que se programó que responda y que lo haga de la manera correcta.
3. Eficiente: que el software haga lo que e supone que tiene que hacer, no que haga cualquier otra cosa o esté lleno de errores de funcionamiento.
4. Seguro: que el software y todos los datos e información proporcionados por el cliente no sean accesibles por personas o entidades no autorizadas.
5. Flexible: que se pueda modificar por varios desarrolladores sin tener que hacer malabares para entender el código.

**CUALIDADES DESEABLES**

* **performance:** es medible, indiscutible, objetiva, en términos de producto (*tiempo de corrida* (tiempo total que se tarda en obtener un resultado en un proceso bach), *tiempo de respuesta* (el tiempo que media entre el requerimiento y la satisfacción del requerimiento…), lo verifica el cliente; y *volumen de procesamiento* (cantidad de transacciones que un sistema realizará en una unidad de tiempo)) y en términos de proceso, en *velocidad, lo verifica el desarrollador. Es una cualidad dual.*
* Velocidad que puede alcanzar el proceso y el producto
* La performance se opone al control, y a la modificabilidad, y viceversa.
  + no puedo hacer un sistema altamente rápido, con alto control y alta modificabilidad al mismo tiempo
* Los usuarios se cansan cuando la performance es mala

“Hay dos cosas que no hay que dejar aburrido frente a una pantalla, un mono y un usuario” - Braca

* Es la unica medible
  + No es subjetiva, es objetiva
* Tiempo de respuesta
* es numéricamente demostrable
* **amigabilidad:** las cosas no están donde quiero que estén (por ejemplo la x de salida, arriba a la derecha)
* sensación de confort que uno siente frente al sistema
* a mayor cantidad de usuarios potenciables más difícil es lograr un sistema amigable

Puede ser deseable en ciertas situaciones

* Es una cuestión de costumbres
  + Depende del usuario
* si un sistema tiene letras chicas, y la gente no ve bien, es incómodo
* es marketing, carta de presentación del sistema
* **verificabilidad:** los sistemas tienen que ser correctos, para eso hay que estar en condiciones de verificarlo, es verificable si en cada una de las etapas de desarrollo se puede verificar que se corresponda en la etapa anterior y que finalmente se pueda verificar que las funcionalidades se corresponden con los requerimientos.

EL HECHO DE PODER CONSTATAR QUE LO QUE QUERIAMOS HACER FUE LO QUE HICIMOS

* no es una condición tan absoluta
* Capacidad de verificar que cada una de las partes del software esté en concordancia con los otros modelos
* Forma en la cual los modelos están en concordancias
  + Lo que el software hace es lo que el software debería hacer según los requerimientos
  + Concordancia entre las clases de diseño y el código fuente
  + Etc.
* Para esto es necesario los demás modelos, ósea la visibilidad.
  + Hay metodologías ágiles que permiten hacerlo sin tantos modelos o diagramas
    - Visibilidad ciertamente frágil
  + Métodos ágiles: Basados en la comunicación oral
    - Poca documentación
    - La cantidad de canales de comunicación crece exponencialmente
      * Por lo que no debería aplicarse a grupos grandes
      * Transmitir el conocimiento entre tantos canales de comunicación es complejo
      * Se recomienda limites de 10 personas - Límite de aproximadamente 400 horas de trabajo semanales con 10 personas
* **mantenibilidad:** mantenibles, evolutibilidad y reparabilidad, debe ser fácil poder modificarlo, imprescindible para que la vida útil del sistema aumente. que sea adaptable a las nuevas circunstancias.
* depende de la vida útil estimada del software.
* depende de las necesidad
* el software generalmente común tiene altos requerimientos de mantenibilidad porque los requisitos cambian, el mundo cambia y en estos cambios generalmente se mete la pata, por eso el software debe ser evolutivo y reparable
* **reusabilidad:** poder hacer componentes sabiendo que la próxima vez voy a poder reutilizarlos.
* **portabilidad:** capacidad de un sistema de pasar de una plataforma a otra o dentro de la misma plataforma sin necesidad de instalación (medio portátil).
* complejas y demandantes de mucho dinero
* que un sistema no requiera instalación significa que es capaz de gestionar por si mismo todos sus recursos.

Capacidad de usar un sistema en multiples plataformas

* Puede ser bueno o puede ser malo
* Abre el horizonte a negocios mas importantes
* Increíblemente costosa

Se puede lograr:

* Maquina ejecutable entre el SO y nuestro sistema
  + caro en recursos
* Programar de forma particular para cada aparato
  + Programar en assembler
* Utilizar un compilador distinto para cada plataforma
  + Cada compilador tiene su propio set de errores
* Versiones distintas para cada plataforma y compiladores

Solamente es justificada cuando la magnitud del negocio lo hace

* es una cualidad deseable pero no exigible (la gente en linux no espera lo mismo que la gente en windows)
* **comprensibilidad:** desde el punto de vista del usuario: el individuo está en condiciones de entender que tiene que hacer para lograr un determinado resultado, que sea intuitivo.
* desde el punto de vista del desarrollador:
* prueba del mono:

Vinculada a la verificabilidad

Software “pequeñamente manejable” → Facilita la comprensibilidad

* Comentarios
* buscar personas que no estén capacitadas o familiarizadas
* “clickee el link para acceder a los servicios” “alta llantas” son incompatibles
* hacer una aplicación comprensible se vuelve difícil cuando aumenta el numero de usuarios porque aumenta el numero de alternativas
* **interoperatividad:**
* Generalmente imprescindible
* Capacidad de coexistir o cooperar con software existente
* Cada software consume los recursos de una forma distinta
  + Puede pasar que dos aplicaciones estén esperando que la otra libere un recurso
    - deadlock
* Las aplicaciones suelen tener fuertes conflictos entre si
* lo máximo es lograr un fluido intercambio de datos
* lo mínimo es coexistir
* encontrar un grado razonable de complejidad de cooperación
* todo producto que requiera de una version especifica de una tercera parte debe instalar la misma en un pack propietario en una paz particular, otra es decidir el directorio de destino. Esto es una buena practica.
* **productividad:** lo ven los desarrolladores y los clientes.
* lento no significa malo, sino no apropiado para un determinado tipo de negocio
* es una cualidad altamente deseable
* condición necesaria, la productividad está asociada al dinero
* beneficio intangible, cuanto clientes nuevos hago porque tengo una aplicación linda incluso si no es productiva
* para los desarrolladores, la productividad lo es todo, es relativa a la naturaleza del producto final
* punto de indiferencia: el objeto que tengo no genera el beneficio necesario que refleje el esfuerzo?

El punto de indiferencia indica el nivel de producción donde los costos de fabricar el producto son iguales a los de comprarlo. Por debajo de ese nivel es preferible comprar, mientras que por encima es preferible fabricar, cuando se da en una zona de ganancia.

el tiempo o la condición es la esencia del contrato, con la condición de que se cumpla.

* **oportunidad: l**a oportunidad es cuando un proceso genera un producto en tiempo (plazo estipulado con cliente) y forma (que tenga la calidad que el mercado demanda).

Desde el punto de vista del cliente: el producto opera oportunamente, es capaz de generar los resultados

la oportunidad es la cualidad que mide o valora si hay una correspondencia entre lo esperado en términos de tiempo y lo obtenido entre la necesidad y lo provisto.

Cualidad interna y externa

Cualidad únicamente externa desde el punto de vista del producto

Relación que existe entre el momento de la demanda y el momento de la satisfacción de la demanda

* Si estos tiempos coinciden, el software es oportuno

El software debe estar en tiempo y forma, para hacer plata

* Tanto el proceso software como el producto software debe ser oportuno
* El proceso software debe realizar las tareas en el tiempo necesario

Vinculada con la performance

* **visibilidad:** hacer visibles nuestras acciones y pensamientos, imprescindible para la mantenibilidad. consiste en hacer visible el contenido de nuestro sitio web para el motor de búsqueda. Es decir, que el motor de búsqueda pueda acceder e interpretar todo el contenido que deseamos posicionar sin problemas.
* documentar lo hace mucho más visible
* Comentarios
* Logica visible
  + Se puede entender aun desconociendo dicha lógica
* Se encuentra asociada a la documentación
  + Un software es mas visible cuanto mas documentado está
  + Es software es un todo, desde los diagramas hasta el código
  + Cuanto mas vistas del modelo tenemos, mas visible es dicho modelo
* No es una cualidad absoluta, aun no siendo visible el software hace lo que tiene que hacer
  + No siempre necesitamos que sea visible
  + Hay casos que no queremos o no podemos hacerlo visible

Se derivan cualidades siempre deseables → “Cascada”

Ángela Merkel, César Augustus, Grigori Rasputín - Gestión efectiva de proyectos

* liderazgo basado en el conocimiento
* fortalezas y debilidades, comunicación efectiva, negociación.

El spike (clavo), es una tarea no percibida por el cliente. Si hacemos una pantalla el cliente ve la tarea , ve la pantalla, si hacemos el diseño de la base de datos, el cliente no percibe todas las horas y trabajo que hay detrás de eso.

El software se degrada, es maleable, es una cualidad, es términos de descripción, pero no quiere decir que sea bueno.

* Tener un punto de estabilidad en tanto caos
* el software tiene un conjunto de dificultades esenciales que son constantes en el tiempo
* la tecnología no las soluciona sino que cambia la dificultad
* la suma de los problemas de los seres humanos es una constante
* utilizamos la tecnología para simplificar los problemas iniciales, sino que se hace un problema proporcional con esa nueva tecnología y a su vez aumenta la dificultad de lo que los clientes piden.
* Los usuarios son la razon de ser de los sistemas, pero son la principal oposición de los mismos, y cuando hablamos, los hacemos en un lenguaje natural.
* El lenguaje tiene sus dificultades, el entendimiento entre dos personas depende de las costumbres de cada uno, hay palabras que pueden activar malos momentos con el usuario.
* Hay que saber que quiere el usuario y eso es lo más difícil.
* la corrección funcional (interna, el cliente no tiene la menor idea de lo que es un requerimiento), la confiabilidad (dual, y la robustez son la condición necesaria para que el software sea un éxito, no garantizan nada pero sin ellas es un fracaso.
* el sistema tiene que hacer mínimamente lo que nosotros queremos aunque este mal

[Ghezzi Cualidades del software.pdf](https://prod-files-secure.s3.us-west-2.amazonaws.com/5356f21f-19a1-45d5-995c-032a5d363ea0/0ba25bc7-cec3-477e-9502-9a3bed56852f/Ghezzi_Cualidades_del_software.pdf)

[No hay balas de plata. Brooks.pdf](https://prod-files-secure.s3.us-west-2.amazonaws.com/5356f21f-19a1-45d5-995c-032a5d363ea0/36ac5f32-25b0-46a3-83cd-b1bb1b088075/No_hay_balas_de_plata._Brooks.pdf)

**2 tipos de dificultades**

* Accidentales: → corregibles con tecnología
  + Utilizar un software con el que no estoy familiarizado
* Esenciales: → hacen a la esencia de las cosas
  + Comunicación
  + Malos entendidos
  + Se hace hablando con el cliente
  + Complejidad en base a lo que la gente espera

Hay que vencer a las dificultades accidentales, para las esenciales no hay bala de plata

“Un caso de uso no va a mejorar tu comunicación con el cliente”

**El significado de las palabras en una lengua adquirida es menos ambiguo, se suelen aprender fuera de las emociones, suele ser mas util cuando se necesita mas precisión**

<aside> 💡 Sistemántica - Jhon Gall

</aside>

* Principio de conservación de la “anergia”

*Todo es una herramienta para hacer guita - Braca*

* Los aspectos teoricos son deseables pero se ajustan al tiempo, recursos, etc

**MODELOS DE DISEÑOS**

* modelos de desarrollo de un producto que son beneficiosos para el cliente y para el desarrollador
* no apasionarse por las metodologías, ver que tienen para ofrecer y utilizarlo
* tenemos que tener algo que nos sirva como guía
* la repetición del proceso hace la perfección
* sistematización de la cadena productiva que permita mejorarla vez a vez,

**MODELO DE CASCADA (WATERFALL)**

necesidades (cliente) ⇒ requisitos (analistas) ⇒ diseño (diseñadores) ⇒ codificación (programadores) ⇒ pruebas (testers) ⇒ despliegues (implementadores) ⇒ mantenimiento (todos)

/ es imposible hacer un sistema rápido y totalmente cambiable, es uno o lo otro

/ el control y la performance son contrapuestos

* Derrame de información de una etapa a la otra
* Al terminar una etapa vamos a la otra
* Tenemos “protagonistas” que elaboran cada etapa del modelo
* **Cliente →Necesidades**
* expresión de un deseo en un lenguaje natural
* es fácil entenderse en un lenguaje neutro
  + Confusas
  + Mal expresadas
  + Incompletas
  + Expresadas en un lenguaje coloquial, propio de su profesión
  + Palabras con significados ambiguos
  + No necesitamos comprender el significado específico, sino coincidir en el significado social
  + Glosario = “Gran arma”
    - Capacidad de expresarnos y entendernos
    - Es imprescindible ser cultos

“La diferencia entre un idiota y un estúpido no está clara, ni falta que hace” - Braca

* **Analista → Requisitos**
  + Necesidad expresada en un lenguaje no ambiguo
  + En lo posible representar gráficamente o mediante una simbología
    - Estos se los llama requisitos funcionales
  + Los requisitos que no se pueden representar simbólicamente se los llama no funcionales
  + Capacidad de tomar las frases de un lenguaje “muy parecido al que uno habla” y convertirlas en requisitos no ambiguos
    - Las palabras se aprenden en ámbitos
    - Existen palabras que son validas solamente en un contexto específico, e inentendibles para terceros
  + “Arte de quitar ambiguedades y redactarlos de forma inequívoca a los demás”
    - Elicitar requisitos
    - Un requisito debe significar lo mismo para todos los interesados en el producto
  + Glosario: Pone orden en el caos, nos pone a todos de acuerdo en el significado de una palabra
    - Herramienta para desambiguar
  + Documentos de requisitos entregados a diseñadores
* **Diseñadores →Diseño**
  + Plantean de que forma mas eficiente en que se van a satisfacer los requisitos
  + Especificación funcional
  + bajan costos de producción para llegar al objetivo
* **Programadores → Codificación**
  + Programan y corrigen
  + Testean codigo en pruebas de escritorio
  + arranca el loop, codificar, probar
* **Testers → Pruebas**
  + Intercambio de codigo con los programadores, para crear un plan de pruebas sistemáticas al sistema o modulos programados
  + Formas y teorías para generar los datos de prueba
  + Pruebas alfa:
    - Pruebas en conjunto con los usuarios
  + Pruebas beta:
    - Pruebas hechas solamente por el usuario
* **Implementadores → Despliegue**
  + Hacer que un sistema pase de desarrollo al estado de régimen
  + Está trabajando con los datos para los que fue diseñado
  + Está implementado cuando es capaz de manejar con una cantidad considerable de los datos para los que fue pensado
  + Es importante limitar el plazo de implementación por cuestiones de “responsabilidad”
    - Estamos obligados a reparar errores sin costo en esta etapa
  + Contablemente un sistema se deberia amortizar en no mas de 5 años
    - El periodo de responsabilidad del profesional es de 1/5 de esta etapa
    - Este periodo se lo denomina como periodo de implementación
    - Luego de esa etapa se considera mantenimiento
  + Trabajo duro
    - Bastante mal pago
* **Todos → Mantenimiento → la muerte**
  + “Alabado sea el clliente”
  + El analista es el proveedor → Son los mejores pagos en toda la cadena informatica
    - Son productores de riqueza
    - Es quien está mas cerca del cliente

No es ni mejor ni peor que otros

* Existe un gran conjunto de problemas que puede ser resuelto con este modelo
  + a pesar de ser una porquería atómica

Analiza etapas de forma exhaustivas y genera subproductos que pasan a la siguiente etapa

* el principal problema de este modelo es la lejanía del cliente, no hay retroalimentación, es un modelo peligroso, se puede usar cuando el producto tiene poca probabilidad de ser sometido a nuevos requisitos durante la gestión.
* Ventajas: me permite tener un conjunto de puntos de control fijos

**Por qué se usa?**

* Porque es un modelo ordenado
  + Hay insituciones donde lo ordenado prima
    - Ejemplo: El gobierno
  + El control de la evolución del trabajo es claramente sencillo
    - “puedo saber exactamente por donde ando”
  + Para poder realizar este modelo necesitamos que las necesidades se mantengan inmutables
    - Esto acota el campo de acción de este modelo
      * En la gran mayoria de los campos las necesidades varían o se refinan con el tiempo
  + **Un sistema es un modelo de la realidad, lo unico que hace es apoyar a un modelo existente en la realidad**
    - Es una componente informática de un sistema real mas grande
      * Que suele cambiar
* En las primeras reuniones se suele obtener el objetivo del sistema
  + A qué estará orientado el software

Modelo teórico

* permite un enfoque sistemico para el analisis del problema en la educación
* No suele ser una alternativa razonable para los problemas de la vida moderna

**MODELO WIN-WIN DE BOEHM**

Define actividades de negociación al principio de cada ciclo

* Identificación del sistema o subsistemas clave de los directivos
* Determinación de las condiciones de victoria de los directivos
* Negociación de las condiciones de victoria con los Stackholders

La condición de victoria de mi cliente es poder satisfacer la pregunta que el 99% de los clientes le hace cuando entra.

* poder atender más rápido a los clientes

Es un ciclo ITERATIVO, da todas las vueltas que necesita y puede seguir creciendo

* en lugar de intentar de una sola vez con toda la info desde el principio, voy a hacer un conjunto de iteraciones, ir cumpliendo objetivos a corto plazo y tarde o temprano voy a llegar al final casi sin errores y al cliente le va a gustar.

RUP (PROCESO UNIFICADO DE DESARROLLO RACIONAL)

* ingeniería del software I
* ir en pequeñas entregas ayuda a que el cliente no esté decepcionado.
* requerimientos, análisis e implementación, prueba, (cada iteración produce un producto ejecutable) evaluación (no es un juicio de valor sobre la calidad del trabajador sino un juicio de valor sobre la calidad del proceso.

Métodos ágiles

Individuos e interacciones sobre procesos y herramientas, es más importante hablar con la gente que hacer un dibujito.

Software funcionando sobre documentación extensiva, es más importante el software funcionando que un toda la documentación

Colaboración con el cliente sobre negociación contractual

Respuesta ante el cambio sobre seguir un plan

No se desprecia lo otro pero se prefiere antes que eso. De vez en cuando hace falta un plan.

* XP como método ágil
* libro para apuntes Pressman

resumen lautaro:

Pretencioso

* Intenta prometer que todos ganan
* tanto el cliente porque en teoria todas sus necesidades van a poder ser resueltas y por el costo

Destinado a generar ganancias en ambar partes

**Tres etapas**

Identificación:

* que hace falta
* Entran muchas metodologías dentro de este modelo

Determinación:

* Identificar las condiciones de victoria en ambas partes
  + Qué significa ganar para el cliente
* Dejar claro quien es el encargado de cargar los datos viejos en el sistema

Negociación:

* de las condiciones de victoria con los Stackholders

Es mucho mas facil cobrar contra entrega de un producto, en lugar de cobrar contra entrega de una promesa

Un poquito de analisis, un poquito de diseño, un poquito de codificación, un poquito de implementación

* Cobro por las distintas partes, en vez de cobrar por las etapas del producto y entregarlo finalizado
* Lo importante es el concepto del modelo

Donde termina?

* depende
* “Hay que sacarle la sangre a la vaca sin matarla”
  + Adecuar la velocidad de producción a la velocidad de pago del cliente
* Cada parte del sistema donde lo dividamos es una vuelta de la espiral
* Cuando cambia un requisito no lo vemos como problema
  + La funcionalidad pasada ya la cobramos, una funcionalidad nueva es un trabajo nuevo que consiste en modificar
  + El cliente puede cambiar los requisitos cuantas veces quiera

Ejemplo de metodologías que pueden estar en win-win:

* Rup
  + Basado en conocer los objetivos globales del sistema y diagramar el proceso con mayor o menor grado de exactitud
  + Cosas separadas en etapas
    - De forma iterativa e incremental
      * Cada etapa se repite y crea incremento en funcionalidades
  + No es del todo util para el desarrollo
  + Es una metodologia “Visagra” entre los metodos clasicos y los metodos agiles
    - Es el mas pesado de los metodos agiles, o el mas liviano de los metodos clasicos
  + Otorga agilidad y otorga documentacion *“para las almas sensibles que la requieren”*
  + Permite la trazabilidad de un producto si es requerida
    - Para poder acreditar calidad de producto hay que acreditar el proceso, osea, documentarlo

Producción de software temparana pero incompleta

* Cada vuelta es una parte funcional del software
* si la velocidad de iteracion crece, los metodos se acercan a los metodos ágiles

**DISEÑO DE ENTRADAS**

¿Qué es una entrada? El continente del producto, es lo que vemos.

¿Por qué compramos algo? Un cliente acepta un producto porque:

* Le gusta lo que *ve*
* Le gusta lo que *recibe*
* Le conviene el *precio por lo que ve y recibe*
* Le gusta lo que los *demás perciben* de él al tener producto
* Le gusta como él se *autopercibe*.

Un iphone, un BMW, Audi, dan prestigio.

<aside> 📌 ***UN SISTEMA ES UN PRODUCTO, para un sistema valen todas las reglas de marketing***

</aside>

Cuando hacemos un sistema, tenemos que pensar en como lo va a ver el cliente. Si le gusta lo que ve y percibe, si lo ve conveniente, etc.

Si convenzo al cliente de que mis entradas son maravillosas, el va a quedar re contento, no va a juzgar si el código está en x lenguaje.

**Hay que cuidar la interfaz.**

***Los sistemas funcionan buen cuando se adecuan a la gente.***

<aside> 📌 \*No se puede vender caramelos envuelto con diarios.

* Bracalenti\*

No importa que tan rico sea el caramelo, la apariencia lo cambia todo.

</aside>

**Poner el foco en:**

* **Corrección.**
* los datos que ingresamos en el sistema sean los datos que necesitábamos
  + Para probarla o intentar asegurarla, ***validando*** lo que entra. Minimizar la probabilidad de que entre algo mal. Validar que la cantidad de errores es mínima.
  + La ***codificación*** la utilizamos para que no haya ambigüedades.
* **Funcionalidad.**

Que las funcionalidades que el deseaba estén y sean fluidas, amigables

* **Estética**.

Tiene que ser lindo. Es muy subjetivo

* **Adaptabilidad**.

Que funcione en todos los dispositivos y que se vea bien

Responsive

***Clasificación: Los datos de entrada podrían ser:***

* Inválidos. Por tipo o valor.
* Validos

Optimización de datos de prueba

* Negativos
* Cero
* Reales
* Positivos

LOS OBJETOS QUE TIENEN MUCHOS ATRIBUTOS SON DIFÍCILES DE PROBAR Y SE TRANSFORMAN EN LOS ESLABONES DELGADOS DEL SISTEMA, NO HAY FORMAS DE PROBARLOS SISTEMÁTICAMENTE.

* hay un problema en la arbitrariedad

***TIPS A RECORDAR***

Las **particiones de equivalencia** dependen del problema.

El salario no es el Paraná (se requiere experticia).

El análisis de **valores límites** es bueno si se conocen los límites de las clases.

otras formas de probar:

La c**onjetura de error** suele ser necesaria pero no alcanza.

Las pruebas por **FUERZA BRUTA** son sólo complementarias.

La calidad de los datos es ajena a nosotros.

A714 es un código único y siempre será A714 El asunto es ¿Cómo recordar “A714”? • De memoria si son pocos • Con una búsqueda blanda con una lista desplegable • Con una lista • Otros.

**Corrección**

Condición necesaria para que funcione el sistema, que las entradas sean correctas

Validación y codificación.

Vamos a hacer todo lo posible para minimizar la probabilidad para que los datos sean correctos.

Para que lo que entre no sea una porquería debemos codificar y validar

**1. Codificación**

Establecer una relación entre una - y un código

Requiere que la codificación sea biunívoca, que vaya en los dos sentidos.

“el tipo metió la pata, pero el sistema no debió permitir que la pata fuera metida”

CODIFICO Y VALIDO

Codificar todo lo que se pueda codificar, así quitamos la ambigüedad.

Se genera ambigüedad en el ingreso de datos

* Codificación soluciona este problema
  + De memoria si son pocos
  + Búsqueda blanda
  + Lista desplegable
  + Etcétera

*Eficacia y eficiencia:*

* **Eficacia:** Es único, cada código implica una entidad y sólo una entidad, unicidad
* **Eficiencia:** Es corto
  + Mas rápido de escribir
  + Entran mas en la caché

**Se codifica para:**

1. Mantener el seguimiento de algo
   1. Tiene en si mismo información
2. Clasificación de la información
3. Ocultar información
4. Revelar Información
5. Solicitar acciones apropiadas

**Principios del código:**

1. **Unicidad**: único para cada entidad
2. **Existencia**: existe para todas

* Podemos dedicarle menos tiempo cuando el consumidor es un profesional que no se deja engañar por una pantalla bonita.

Lista descriptiva y no taxativa

* códigos secuenciales simples
  + códigos muy sencillos de generar
  + le precede un antecesor y le sucede un sucesor. La distancia entre el antecesor y el elemento en si y entre el sucesor es la misma (la distancia entre 1 y 2 es la misma que entre 2 y 3)
  + es muy lindo pero tiene sus límites, se pueden usar códigos secuenciales cuando las entidades pueden conocer el código y lo saben
  + se usa cuando es legalmente requerido, por ejemplo, la afip que pide las facturas, para seguirlas.
  + cuando la cantidad a codificar es pequeña, el sistema secuencial es recomendado, ocupa poco espacio, simple de codificar
  + cuando no es necesario que sea conocido, sino que es una característica del sistema, el sistema es el único que tiene que conocerlo
* códigos de origen alfabético
  + es importante el orden de los caracteres, los números y las letras son parecidos, conviene usar letras porque la base numérica (62) es mayor que la base numérica de los números.
  + la base siempre es 1 0
  + bajo la cantidad de posibilidades para mejorar la calidad del dato en cuestión, por eso separo números y letras y les asigno una posición fija
  + la mente humana recuerda mejor con dos tipos de datos distintos.
* códigos de clasificación
  + dice el estado o la acción a realizar de o con alguna cosa
  + siempre hacer una tabla para que en caso de desborde solo debamos agregar cositas a la tabla y no modificar el código fuente.
  + los desbordes del código son peligrosos de acuerdo a la calidad del software
* códigos de bloques secuenciales
  + simple
    - cada parte de la secuencia/conjunto representa algo. (de 1-5 es River de 5-10 es Boca de 11-15 de otro) si aparece alguien mas se produce un desborde de rango ya que no quedan lugares.
    - Si esta “hardocodeado” es un drama de recodificación, si no simplemente se modifica solo la tabla (se agregan nuevas entradas en la tabla ej: de 16-20 es River).
  + multiple
    - es prácticamente infinito desde un punto de vista práctico
    - no tengo que tocar ni modificar el almacenamiento
    - pongo la secuencia en una misma línea donde cada número después del punto representa algo distinto (xx.xx.xxxx -> 08.08.2004 esto podría significar día.mes.año en que alguien nació).
    - Es un sistema muy flexible y la codificación tiene muchísima potencia.
* **códigos cifrados**
* **códigos de subconjuntos de dígitos significativos**
* **códigos nemotécnicos**
* **códigos funcionales**

**2. Validación**

Asegurarme que el tipo y rango sea correcto, no puedo asegurarme del valor, no importa cuantas validaciones hagamos, el sistema puede fallar. Vamos a hacer todo lo posible para minimizar la probabilidad de que los datos sean incorrectos (la gente no se pone a hacer cosas imposibles).

* La fuerza bruta no sirve como técnica de prueba, salvo como complemento.
* Maneras de validación

*Particiones de equivalencia:* Encontrar grupos representativos, poner en la misma bolsa todos los datos que sean parecidos y probar con un solo dato. Lo más probable es que funcione bien para el otro dato. Si funciona, entonces tengo una confianza razonable.

*Pruebas por valores límites:* pruebo por arriba, pruebo por abajo (max y min)

*Conjeturas:* pruebas basadas en las experiencias del programador, son formas asistemáticas de encontrar datos de prueba, no al azar, sino basadas en la experiencia. Conjeturar sobre posibles errores.

*Fuerza bruta:* los dos anteriores, si tengo archivos lo pruebo también.

apunte de nuria:

* Códigos de bloques secuenciales - Simple: cada parte de la secuencia/conjunto representa algo. (de 1-5 es River de 5-10 es Boca de 11-15 de otro) si aparece alguien mas se produce un desborde de rango ya que no quedan lugares. o Si esta “hardocodeado” es un drama de recodificación, si no simplemente se modifica solo la tabla (se agregan nuevas entradas en la tabla ej: de 16-20 es River). o Los desbordes del código son peligrosos de acuerdo a la calidad del software. • Múltiple: pongo la secuencia en una misma línea donde cada número después del punto representa algo distinto (xx.xx.xxxx -> 08.08.2004 esto podría significar día.mes.año en que alguien nació). o Un código de bloque es prácticamente infinito desde un punto de vista práctico. o No tengo que tocar ni modificar el almacenamiento. o Es un sistema muy flexible y la codificación tiene muchísima potencia.

**Funcionalidad**

* En todas las partes del sistema todo debe funcionar igual
* Dejar que el usuario decida mucho, incluso de las funciones estéticas
* Si es en una computadora el usuario no debe memorizar mucho

El software debe ser ***usable*** (la facilidad de uso suele ser la base de un juicio de valor)

1. **Dejar el control al usuario**
2. **Reducir la carga de memoria del usuario**
3. **Hacer que la interfaz sea consistente**

***DEJAR EL CONTROL AL USUARIO***

* no obligar al usuario a realizar acciones que no quiere
* interacción flexible, adaptable
* interacción interrumpible y reversible
  + si vuelvo atrás no tengo que empezar de cero, poder hacer y deshacer.
* facilitar la interacción a medida que aumenta la habilidad y permitir que aquella se personalice
* ocultar los tecnicismos internos al usuario
* diseñar la interacción directa con objetos de la pantalla

***REDUCIR LA NECESIDAD DE QUE EL USUARIO MEMORICE***

* reducir la demanda de memoria a corto plazo
  + todo lo que pueda ser recordado por una computadora, que lo recuerde, no lo debemos hacer nosotros
* hacer que lo preestablecido sea significativo
  + que muestre lo que queremos ver, si entro a la tarjeta quiero ver el dinero que tengo disponible y lo que tengo gastado. Lo primero que debiera mostrarse en una pantalla
* definir atajos que sean intuitivos
  + usar atajos que la gente ya tenga interiorizada, si quiero salir aprieto escape
* la interfaz debe basarse en una metáfora del mundo real
* revelar información de manera progresiva
  + no atiborrarse con info para no hacer una pantalla para idiotas

***HACER CONSISTENTE LA INTERFAZ***

* asociar la tarea en curso a un contexto significativo para el usuario
* mantener la consistencia en toda la familia de aplicaciones
  + es una gestión comercial, se quedan con lo que ya conocen y saben usar
* si los modelos interactivos anteriores han creado expectativas en el usuario, no haga cambios a menos de que haya una razon ineludible para ello (casi, casi “…si funciona no lo toque”)
  + cada uno se hace una idea del modelo
    - se establece un modelo de usuario (lo que nos pensamos que el usuario quiere)
    - se crea un modelo de diseño (que acción hace cada tecla, etc, teórico)
    - el usuario se forma un modelo mental (percepción del sistema)
    - los desarrolladores hacen un modelo de implementación
    - alguien tiene que conciliar todos estos modelos, yo
  + en el mundo de los negocios, ser bueno es rentable, si cliente = feliz, nosotros $$$
  + conciliar lo posible con lo deseable

***Conocer al usuario***

Todos son usuarios, y el problema es que es muy difícil hacer una aplicación que satisfaga a todos, a todas las necesidades y problemáticas.

El modelo mental depende del tipo de usuario.

1. no sabe nada de nada
2. …
3. …
4. sabe más que vos

*Cuanto menos genérico es el software, más son las probabilidades de éxito*

Crear las metáforas del mundo real, por ejemplo, los íconos pueden ser representaciones de acciones del mundo. El zoom lo representamos con una lupa.

Cuando vamos a crear una aplicación mas genérica hay que tener el foco en el consumidor

Las aplicaciones tienen un destinatario natural y deben ser ajustadas a ese destinatario

***Proceso de diseño de una interfaz***

* en espiral, pienso que hacer, lo esbozo, lo programo y antes de implementar le muestro al cliente.
* si el cliente no interviene en el proceso de desarrollo en la interfaz, la interfaz es un fracaso
* si no recibimos retroalimentación pensamos que el manejo de la interfaz es razonable, cuando en realidad puede no ser así. Para nosotros cualquier interfaz nos resulta intuitiva.

***Entender al usuario y su percepción***

* ¿Quién es el usuario? ¿Cuánto sabe? ¿Qué hace?
* ¿Dónde se encontrará físicamente la interfaz?
* ¿El usuario estará sentado, de pie o haciendo otras relacionadas con la interfaz?
* ¿El hardware de la interfaz cumple las restricciones de espacio, iluminación o ruido?
* ¿Hay consideraciones especiales de factores humanos generadas por los factores ambientales

La componente más importante de todo este negocio es la componente social, cuando entendemos para quien estamos trabajando, cuando entendemos sus expectativas.

**A mayor complicación física, mayor simpleza de la interfaz (ejemplo interfaz del colectivero)**

***Diseño de la interfaz***

**Definir un conjunto de objetos y acciones**

* definir objetos y acciones de la interfaz (operaciones)
* definir eventos (acciones del usuario) que cambien el estado de la interfaz
* ilustrar cada estado de la interfaz como lo vería en la realidad el usuario final
* indicar cómo interpreta el usuario cada estado del sistema

El involucramiento del usuario en la línea productiva, es comercialmente rentable, ya que el usuario siente orgullo de un proyecto en el que estuvo trabajando.

Una interfaz consistente es aquella que tiene la misma estética en todas las pantallas, guarda correspondencia en sus acciones y en sus teclas. Podemos lograr la consistencia teniendo un patrón de diseño (diseñando todas de la misma forma) y teniendo comunicación con el usuario.

* tiempo de respuesta
  + inmediatez de las acciones
  + es determinante
  + mantener al usuario interesado en lo que ve
* herramientas de ayuda
  + manual, la gente no las lee
  + la ayuda debe ser en línea y rápida
* manejo de errores
  + como va a ser vista nuestra aplicación
  + que hacemos con el error, como se va a notificar, que hay que notificar
* accesibilidad de la aplicación
  + importante
  + que colores poner, tamaño de letra, que va a ser cómodo para los usuarios. Que la aplicaciones sea configurable para que se adecue a las posibles limitaciones de los usuarios.
  + el informe de error no puede ser un error en sí mismo (double panic)
  + debe mostrar la naturaleza del error, no es un error fatal, es un error, donde está ese error, ayudar al usuario, no hacer que piense que tiene que formatear una computadora
* internalización
  + Internacionalizar una aplicación es mucho mas complejo que traducirla, hay que tener en cuenta con que tipo de idiomas vamos a hacerla compatible.
    - En general todas las pantallas comienzan a dibujarse con modelo de alambres.
    - Al final te termina construyendo la maqueta de una pantalla, es la que nos permite una primera interacción efectiva con el cliente, porque el cliente entiende de eso, no podemos definir la belleza, pero todos la entendemos, todos podemos saber si algo es lindo o no. Esta muestra es determinante para el éxito del proyecto, el cliente tiene que quedarse con ganas de más, tiene que quedar fascinado. Apps para hacer modelos: • Figma (colaborativo): es el que se usa en la UAP • Framer X (para diseñadores muy experimentados) • Principio (especial para móviles) • Atomic • UXPin • ProtoPie Diseño adaptativo (“responsive” = adaptativa): CSS (Cascading Style Sheets), en español “Hojas de estilo en cascada”. La aplicación se adapta a cualquier tipo de pantalla.
    - Nunca la adaptación supera al original. En lugar de hacer una pagina adaptable, es mejor hacer una app para esa página. Porque la adaptabilidad es muy linda, pero es muy difícil de lograr con calidad.

**El sesgo:**

Es la malinterpretación de la salida, es el matiz con el que una cosa se interpreta. El usuario entendemos la información de una pantalla de cierta forma, que puede llegar a ser distinta a la forma en la que se quería que fuera entendida la cuestión.

es la percepción distorsionada de la salida por parte del consumidor

* El usuario tiene una predisposición a entender los datos de cierta forma.
* grafico o escrito, intencional o involuntario
* la posición, los colores, nos hacen tener prejuicios, los prejuicios nos hacen tomar acciones rápidas sin necesidad de cuestionarlas, pero cuando desaparece el peligro, los prejuicios quedan.
* Siempre va a haber sesgo y un buen profesional va a intentar minimizarlo.
* Encontramos sesgo en:
  + colores:
    - verde, azul (ejemplo windows), un medico viste de blanco por lo inmaculado, el blanco es bueno, por lo tanto, el negro es malo. Las cosas que están en amarillo, significan que hay algún tipo de alerta. Rojo, asociado a lo malo, a la sangre. Semáforo, trabaja sobre una predisposición de la gente.
  + orden de la información
    - aviso clasificado de los diarios, orden alfabético, los primeros son los mas perjudicados o beneficiados.
  + establecimiento de los límites aceptables
    - determinar que es normal y que no, la elección de los límites razonables es la elección de un mínimo y un máximo razonable y donde tiene que ir el centro de la imagen. Alterar una gráfica es muy común y tramposo. Los medios de información manipulan la opinión pública
  + elección de las gráficas
    - la manera en que presento las cosas logra un impacto
    - las gráficas de torta son comunes para torcer la opinion del usuario. Si pongo personitas el usuario empatiza porque se ve reflejado
    - si no explico bien las gráficas se puede entender cualquier idea
      * grafica de torta:
        + NO DEBE HABER INFORMACIÓN DE DISTINTA MAGNITUD EN LA MISMA PÁGINA
        + proporcionalidad de los números y de las gráficas si son más de una
        + al ponerle perspectiva, lo que está en frente crece visualmente, la perspectiva distorsiona
      * grafica lineal y grafica de barra:
        + aplanar una escala es la mejor forma de sesgar si yo quiero mostrar que no existen diferencias, estirar la escala es una forma de sesgar si yo quiero magnificar las diferencias
        + usar colores y perspectivas, truncamiento de escala, desplazamiento de la base, son los trucos para manipular estas gráficas
      * íconos:
        + dan información y emoción
  + propia predisposición del usuario
    - si yo se como piensa, lo tengo, esto es la base.
    - nuestros padres nos transfieren su cosmovisión
    - cada uno tiene una forma de ver el mundo, al principio de como ha sido criado, y luego de las reacciones al como fuimos criados.
    - percepciones diferentes respecto a la misma cosa, lo podemos utilizar a favor
    - si conocemos esa percepción podemos utilizarla a favor para crear un sesgo determinado.
    - la forma en la que cada uno ve el universo es determinante para hacer el proyecto y para saber como lo va a ver el cliente.
    - ver el usuario, ver el público y determinar sesgo.

**DISEÑO DE SALIDAS**

“Te juzgan por eso”

* Nadie cree que es estúpido, si no entiende algo es culpa de la salida

***Es lo que se entrega a los usuarios por medio del sistema a través de dispositivo o redes***

El usuario en general no entiende de algoritmos pero está altamente motivado para evaluar lo que recibe

Objetivos: → Tiene que servir a un propósito, si hacemos una salida que nadie pidió estamos mal

* Diseñar la salida para servir a propósito previsto
* Diseñar la salida para ajustarla al usuario
* Entregar la cantidad apropiada de información
  + más no es mejor
* Asegurarse que la salida esté donde se necesite
  + Ej: en un sistema de sueldos, los recibos deberán estar en el correo oficial del empleado 48hrs después de emitido
* Proveer la salida en forma oportuna
  + Solo cuando hace falta
* Elegir el método de salida correcto

***Diseñar la salida para servir al propósito previsto***

* ***Puede servir para entretener***
  + Pantalla de un videojuego
  + Películas
* Si la salida no es agradable para el propósito de entretenimiento, la gente lo deja, se aburre
* Flow
* Mueve guita 🤑
* ***Para informar (opuesto al entretenimiento)***
* Pasiva para el receptor
  + Tablas
  + Cartel, colores que llaman la atención
* Títulos impactantes →rápida, sintética, duradera
* Tiene que tener golpe (capacidad para llamar la atención)
  + Más impacto visual ayuda cuando más apurado estás
* ***Operar (salidas operativas)***
* Se espera una reentrada en el sistema, voy a tomar decisiones, voy a hacer una operación
  + Pagar la luz (la epe tiene un color verde arriba para dar la idea de un servicio positivo, es un mensaje subliminal para las personas)
* Una parte de la salida vuelve a la organización →Salida reentrante
  + Tiene que tener un nivel de calidad interesante
* El cliente recibe dicha salida y tiene que hacer algo
  + Pagar, imprimir
  + Tomar una decision
* Claridad necesaria para que el receptor pueda realizar la operación (fácil de entender)
  + ejemplo de mala salida operativa: Instagram y el bloqueo
* *Una salida operativa tiene que dar información clara, precisa e indubitable.*
* En una salida auditiva, hay que trabajar con el dialecto local o con el idioma más neutro posible
  + Salidas auditivas en dialectos extraños le quita seriedad a la persona
* ***Tomar decisiones***
  + Graficas
* Usualmente hay mucho color
* Son salidas destinadas a personas que tienen más poder que nosotros (vamos a trabajar mucho mejor, más prolijo, etc)
* Ayudar al decisor a tomar una decisión con un prejuicio sobre la decisión a tomar
* Tiene tendencia a influir en la decisión
  + Ya sea por maquiavélico o por estúpido
  + Los gráficos de torta inclinados son mentirosos
    - Si los hacen sean conscientes que lo están haciendo

***Diseñar la salida para ajustarla al usuario***

* Preguntar al usuario, no nos conformamos
  + “los usuarios siempre mienten”, no siempre te cuentan todo
  + no vamos a suponer nada, vamos a preguntar
* Observar
  + no alcanza con preguntar, te van a mentir por vergüenza
  + la gente quiere parecer mejor de lo que es
  + algunos usuarios son simplemente idiotas
* Calcular
  + “Por qué decís que no tense guita si tienes un escritorio de roble?”
* Diseñar → Pregunte

Preguntar, no creer ni la mitad de las cosas, calcular, hacer la pantalla y volver a preguntar

Una mala actitud es pensar que lo hicimos bien, hay que volver a preguntar para verificar o corregir

<aside> 🚫 La cantidad de consumidores dificulta la realización de una salida

</aside>

***Entregar la cantidad apropiada de salida***

* Granularidad de la información
* A una misma pregunta, cada consumidor de la salida necesita una granularidad distinta de la información, depende que quiere saber la persona
  + La información que tenemos que presentar depende del consumidor
  + “Si sobre informamos, nuestra información no sirve para nada”
  + Una base de datos podría tener 20 dimensiones, para cada tipo de dato una consulta distinta
* Que el usuario resuelva que quiere ver

***Que la salida esté donde se necesite***

* En tiempo y forma
* Salida por pantalla
* Salida por impresora (tienden a ser más duraderas)
  + Pueden implicar un reparto a domicilio (factura en cada caso, si el proceso tiene algún eror, el impuesto no se paga)
  + Retiro en dependencia (van a buscar a la municipalidad el recibo)
  + Envío por correo
* Envío digital
* Auto impresión (papel por tu cuenta)
* Almacenamiento (celular)

No todo es pantalla o papelito

Sistema editorial

* No imprimimos todo y distribuimos, distribuimos la información necesaria para que se imprima donde es necesario

<aside> 🚫 La cosas no termina en la programación, el ingeniero tiene que ir mas allá, sino solamente es programador. Entender la ingeniería como un todo

</aside>

***Proveer la salida de forma oportuna***

En tiempo y forma (el diario de ayer no le sirve a nadie)

* en un determinado formato y en un determinado lugar
  + Ej: Si el tipo que va a viajar no recibe el pasaje en tiempo oportuno, pierde el avión o el tren
* hay cuestiones que no aceptan la mas mínima dilación
  + Ej: Impuesto que no se distribuye a tiempo es impuesto que no existe
* guita que no va a llegar para pavimentar una calle y tener un voto

***Elegir el método de salida correcto***

* Papel
  + Tipos de impresoras
* Pantallas
  + LCD
  + Panel Led
  + E- INK
    - Tinta real → Polímero
    - Los colores se modifican electrónicamente
    - No es luz, el color de la tinta se le da eléctricamente, pero es tinta real

**Tipos de salidas**

**Internas**

* + Operativas: Seguramente la vea nuestro jefe
  + Toma de decisiones
  + Extrema prolijidad

**Externas**

* + La cara visible de la organización
  + Las compañías eléctricas necesitan imagen ecologista para corregir la mala prensa de la quema de combustible
  + CV

**Reentrantes**

* + Tiene que ser de calidad para que dure el proceso de salida y reentrada
  + Imprimen, doblan, ensobran, distribución, sufre, vuelve

***Topología:***

* + Duras
    - Papel
    - Plástico
      * Tarjetas
      * Objetos
    - Impresoras de viviendas
  + Blandas (no duran en el tiempo)
    - Pantallas
    - Sonidos
    - Olores
    - Movimiento
  + “Clasificación intermedia”
    - impresiones de temperatura

<aside> 🚫 Todas las salidas están sesgadas

</aside>

**Métodos Ágiles para el desarrollo de software**

No hay métodos mejores que otros

Es mejor el método que te permite tener resultados mejores con menos costos

**Manifiesto Ágil**

Es un documento de consenso, se confirmo todo lo que estábamos de acuerdo

La palabra ágil es más para marketing que otra cosa. Por marketing caen las empresas

* Un valor es algo que es percibido como bueno por un grupo o comunidad

**Individuos e interacciones** sobre procesos y herramientas

* *Lo importante de estos métodos es que todo lo que se pueda decir hablando, se dice hablando. Preferimos la colaboración con el cliente, esto hace el trabajo más flexible.*

**Software funcionando** sobre documentación extensiva

* Es mejor tener el software andando que todos los otros papelitos al pedo

**Colaboración con el cliente** sobre negociación contractual

* La inexistencia de un contrato escrito no implica que no haya un contrato, existe un contrato tácito de compra-venta
* No pensar que siempre vamos a entender al cliente porque somos unos genios, el cliente siempre va a tener una base de conocimiento diferente a la nuestra

**Respuesta ante el cambio** sobre seguir un plan

* Tiene que haber un sistema que soporte el cambio y un equipo preparado para cambiarlo
* *Esto es, aunque valoramos los elementos de la derecha, valoramos más a los de la izquierda. Pero cada vez que haga falta, cuando sea rentable, vamos a darle valor a los de la derecha. El fanatismo de uno u otro no es bueno.*
* De todos los métodos que existen, es importante que verifiquemos que cumple con esos 4 valores, no importa cuál usemos.
* Guardan en común todos esos métodos que no hay grandes investigadores, libros, etc detrás de ellos.

eso de arriba va si o si a un examen

**Características comunes**

* Surge en libros com impactos en la industria y en el público, no en papers
* Metodología simple y fácil de aprender
  + valores, principios, prácticas, roles, artefactos
* Equipos no jerárquicos y auto-organizados
* Comunicación intensiva
* Minimalismo
  + prescindencia de modelado
* Proceso iterativo e incremental
  + entregas rápidas
* Capacidad adaptativa
  + rápida respuesta al cambio

**Principios del Manifiesto Ágil**

* **Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.**
  + mínimamente entregarle algo, arrancar rápido, empezar a producir, no entrego promesas, entrego laburo, cuánto más rápido entreguemos algo, más rápido empieza a circular plata.
* **Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.**
  + cada cambio es guita, es la esencia del negocio, si todo cambia todo el tiempo, todo es ganancia para mi, facturo hasta el día final. Se implora el cambio.
* **Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al período de tiempo más corto posible.**
* **Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.**
  + Esto funciona en cuanto sean organizaciones distintas
  + Los otros ejecutivos son cabezas a pisar, tengo que ser mejor que ellos
  + El ambiente idílico no existe, nos odiamos todos, cada uno quiere triunfar a costa del otro, por más que vayamos todos juntos al culto.
  + Va a funcionar bien si es software prosumido, software para consumo
* **Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.**
  + recompensas, la motivación es fundamental para que el equipo de trabajo se mantenga unido y produzca.
  + ser bueno es rentable, no es un juicio de valor moral, es comercial, si sos un buen jefe, los empleados van a estar contentos
* **El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.**
  + es la forma más rápida de ponerse de acuerdo
  + la cantidad de información que se transmite es exponencialmente más rápida que cualquier otro método
  + lo mínimo que se debe hacer es una videollamada, los tonos no viajan por mensajes o correos
  + face to face mejor que cualquieeeer otra cosa
* **El software funcionando es la medida principal de progreso.**
  + “una vez que entro no me voy” diría el sabio de Braca
* **Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.**
  + hacer que el negocio genere los recursos para sostener el negocio, hacer ver que gracias a vos la empresa está generando ganancias, después de eso no te vas más. Al principio sos gratis.
* **La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.**
* **La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.**
  + buscar siempre la solución más simple.
  + si funciona no lo toque
  + preguntarse siempre ¿hace falta el cambio? ¿tengo que hacerlo? ¿hacerlo 3% más rápido justifica todo el trabajo que tengo que hacer?
* **Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.**
  + surgen de una discusión entre 10 profesionales le gana siempre a un arquitecto
* **A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.**
  + atacar los problemas con dureza y ser blando con las personas
  + no sirve de nada rajar a una persona que metió la pata, con eso adquirió experiencia
  + resolver el problema en términos honestos, sin herir aunque es difícil
  + mantener conversaciones a nivel personal, diciendo solamente la verdad y sin herir es muy difícil
  + ¿Estoy diciendo la verdad? ¿Hace falta que la diga?

La simplicidad o el arte de maximizar la cantidad de trabajo realizado, es esencial. Debo tender a no hacer NADA de lo que no me hayan pedido.

Al principio lo que tenemos, es un estado provisional de funcionalidades.

Todos los trabajos que el cliente no ve se los llama clavos. Porque se los hace muy difícil lograr la cobrabilidad de un clavo.

Dividir el costo total por cantidad de funcionalidades y eso da el costo particular de cada funcionalidad

[XP (extreme programming)](https://www.notion.so/XP-extreme-programming-bd3dd1c7ac044b7c8a2ed45d977257f2?pvs=21)

**GESTIÓN DE RIESGO**

Uno está sujeto a un conjunto de riesgo por el simple hecho de levantarse o no.

Los ingenieros no pueden dejar que la existencia de un riesgo nos impida actuar. Siempre estamos sujetos a riesgo, tenemos que impedir que eso nos conduzca a la inmovilidad total porque los proyectos no avanzan o directamente no hay proyectos. Tampoco podemos actuar a ciegas y no considerar lo que hay. Lo importante es conseguir un balance, identificar el riesgo y luego intentar gestionarlo (tomar acciones de algún tipo que permita sobrellevarlo).

Existen 2 tipos de preocupaciones, las que le tenemos que dar bola y las que no.

***probabilidad de ocurrencia x daño $$ = un numerito que es la base de la decisión objetiva.***

**Tenemos que preocuparnos por cuál es la probabilidad de ocurrencia del evento y cual es el daño potencial asociado.**

| **Evento** | **Probabilidad** | **Costo** | **Producto** |
| --- | --- | --- | --- |
|  |  |  |  |

Ordeno la tabla de riesgos mayores a menores por la columna de producto, luego trazo una recta que dicte de donde a donde debo preocuparme y de donde a donde no debería preocuparme porque es un caso menor.

**La probabilidad de ocurrencia podemos sacarla de bases de datos de la intuición. Números aproximados que podemos inferir sin mucho margen de error. La intuición es la sistematización de la experiencia. Nuestra mente llena los agujeritos con los conocimientos anteriores.**

La línea de cosas a considerar depende de la criticidad del problema.

RIESGOS **CONSTITUTIVOS**: La esencia de todo negocio es el riesgo. El riesgo es la base del beneficio, si no hay riesgo no hay beneficio, no pain no gain. Hay un conjunto de riesgo sin los cuales no tendría razón de ser el negocio.

RIESGOS **INFERENTES**: Hay riesgos que se oponen al negocio (que se enferme un empleado, que caiga un meteorito, no son la base del negocio)

***A los riesgos hay que contenerlos:***

1. Contener la probabilidad de que se materialice
2. Contener el daño que cause de materializarse
3. Contener ambos (dividir probabilidad y morigerar(atenuar) el daño)
4. Ignorar, conocerlo pero no tenerlo en cuenta (cuando el producto entre la probabilidad y el daño es muy bajo).
5. No conocer un riesgo. Si ignoro la existencia de un riesgo, me voy a meter en problemas

El problema de los riesgos es que no hay soluciones iguales a problemas iguales, la naturaleza de la solución suele ser diferente, no podemos tener una base de datos para actuar de igual forma frente a cada riesgo. Cada proyecto suele ser una circunstancia diferente.

No es lo mismo ausencia de personal en un momento crítico de la empresa a un momento tranquilo.

Es muy poco probable que una solución pueda ser copiada.

Una de las cuestiones mas difíciles de determinar es encontrar una forma temprana de predecir cuando un riesgo se va a manifestar, cuando un riesgo se materializa estamos en presencia de una contingencia.

* Entonces debemos saber que se nos viene una contingencia con la debida anticipación.

***Reglas disparadores o gatillos:*** eventos perceptibles que alertan en forma temprana de la potencial ocurrencia de una contingencia.

Indicadores de problemas. Disparador de alertas tempranas: tengo que estar alerta de que se me viene una contingencia.

Estos disparadores tienen que ser predichos.

* Un casamiento es indicador de que se viene un hijo.
* Cambiar el director de recursos humanos y hay muchos jardineros. eso es un disparador de que posiblemente hayan despidos.
* La llegada de la primavera es un disparador de que podríamos tener ausentismo por dengue.
* Un país en el cual no se puede ahorrar va a un shock financiero.

Después de las probabilidades y el producto debemos determinar si existen disparadores de ese riesgo.

Entender el ambiente y poder determinar si lo que está ocurriendo nos puede perjudicar a nosotros.

Los disparadores sirven para ***tomar acciones*** que minimicen la probabilidad o minimicen el impacto.

***Medidas preventivas:*** Es lo que hacemos para minimizar la probabilidad de ocurrencia. En general son mucho mas baratas que las correctivas.

***Medidas correctivas:*** Decisiones que se toman para corregir lo que ocurre.

* Determinar medidas preventivas y determinar las medidas para corregir.
* En general la tabla no suele hacerse sobre valores reales, no se trabaja con números, de manera cuantitativa. Se suele trabajar en forma cualitativa.
  + Los impactos pueden ser:
    - Catastróficos (en riesgo existencia de la organización) 8
    - Altos (ponen en riesgo al proyecto) 4
    - Moderados (producen enlentecimiento en el proyecto) 2
    - Bajos (no afectan demasiado) 1
  + Los riesgos pueden ser:
    - Muy altos 4
    - Altos 3
    - Medios 2
    - Bajos 1
  + El impacto puede ser catastrófico (8) + el riesgo puede ser alto (4) = 32 (la tabla se tiene en cuenta el límite mínimo como 32, de 32 para arriba la tengo en cuenta, lo demás no.

El analista de riesgo es el trabajador más pesimista, se encarga de ver todas las posibles cosas que pueden salir mal.