



Análisis y Diseño de Sistemas

Fausto Panello 2025

¿Qué es un sistema?

Un sistema es todo el conjunto de elementos que intervienen para realizar un fin común



- Un cambio en un elemento afecta al conjunto de todos ellos.
- Los elementos operan en armonía o con un mismo propósito.
- Los sistemas pueden ser **materiales** o **conceptuales**.
- Estamos rodeados de sistemas por todos lados. (sistemismo)

Sistemas físicos

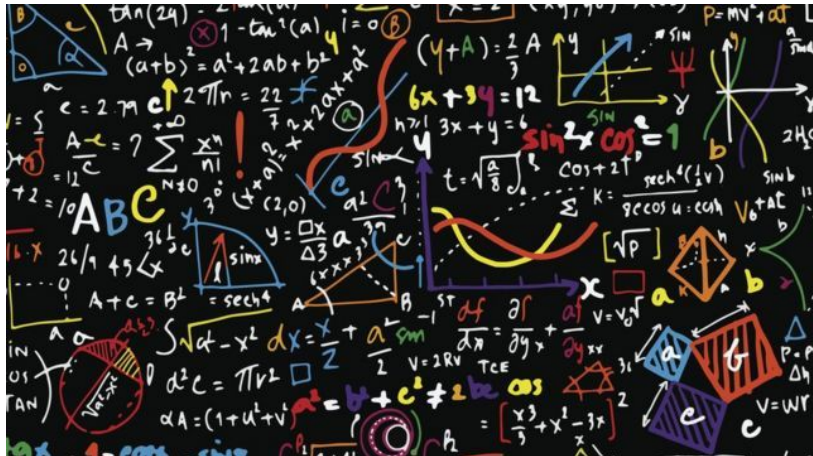
Son tangibles y concretos, y están compuestos de componentes físicos.



- Es, por definición, una “cosa” compuesta por dos o más “cosas” relacionadas. Se entiende por “cosa” a algo físico.
- Por más que no lo podamos ver, la historia, por ejemplo, es un sistema físico, ya que es algo concreto que ocurrió.
- El esfuerzo por encontrar leyes generales del comportamiento de los sistemas materiales funda la **teoría de sistemas**.

Sistemas conceptuales

Son conjuntos ordenados e interrelacionados de conceptos e ideas.



Conformados por definiciones, ideas, símbolos, conceptos u otros elementos que no tienen entidad material.

Si bien las matemáticas están presentes constantemente en nuestra vida, y podemos incluso escribir números, y ver que hay 2 “cosas” en vez de 1 (suma o resta), no es un sistema con propiedades tangibles inherentes, sino que son ideas y teorías.

Sistemas en informática

Un conjunto de datos ordenados conforme a una serie de instrucciones o algoritmos.



Si hay hardware (“cosas” físicas, tangibles y concretas), y software (instrucciones abstractas) en un sistema informático, ¿es un sistema físico o un sistema conceptual?

Sistema informático

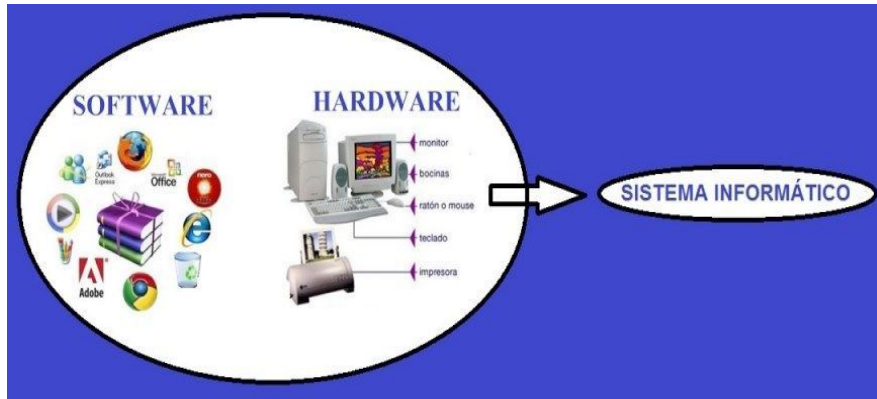
Un conjunto de datos ordenados conforme a una serie de instrucciones o algoritmos.



- Por más que las instrucciones y el software sean abstractos e invisibles para el ojo humano, los sistemas informáticos son físicos.
- Cada bit que se prende y apaga, y cada instrucción que nosotros le damos a través del software a la computadora, crea un cambio físico que acarrea consecuencias dentro de la arquitectura de la misma.

Sistema informático

Un conjunto de datos ordenados conforme a una serie de instrucciones o algoritmos.

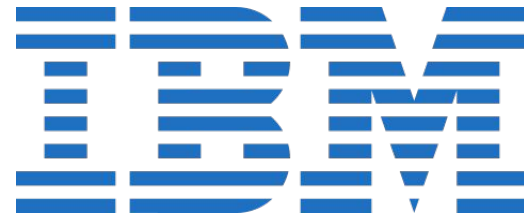


- A su vez, los sistemas informáticos se estructuran en subsistemas:
 - Sistema físico, asociado al hardware.
 - Sistema lógico, asociado al software.

¿Cómo se crea el sistema físico de un sistema informático?

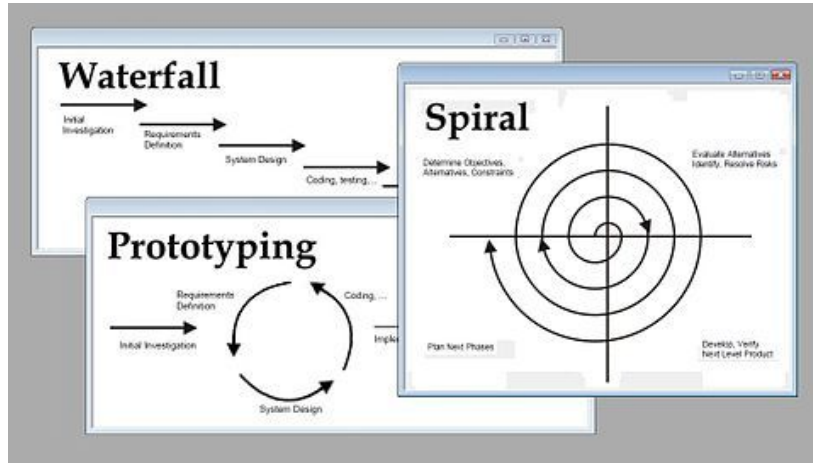


- Tiene que ver con la arquitectura de un sistema, y la evolución, desde la computadora más vieja de todas, hasta las supercomputadoras de hoy en día.
- Son creados por empresas tecnológicas.

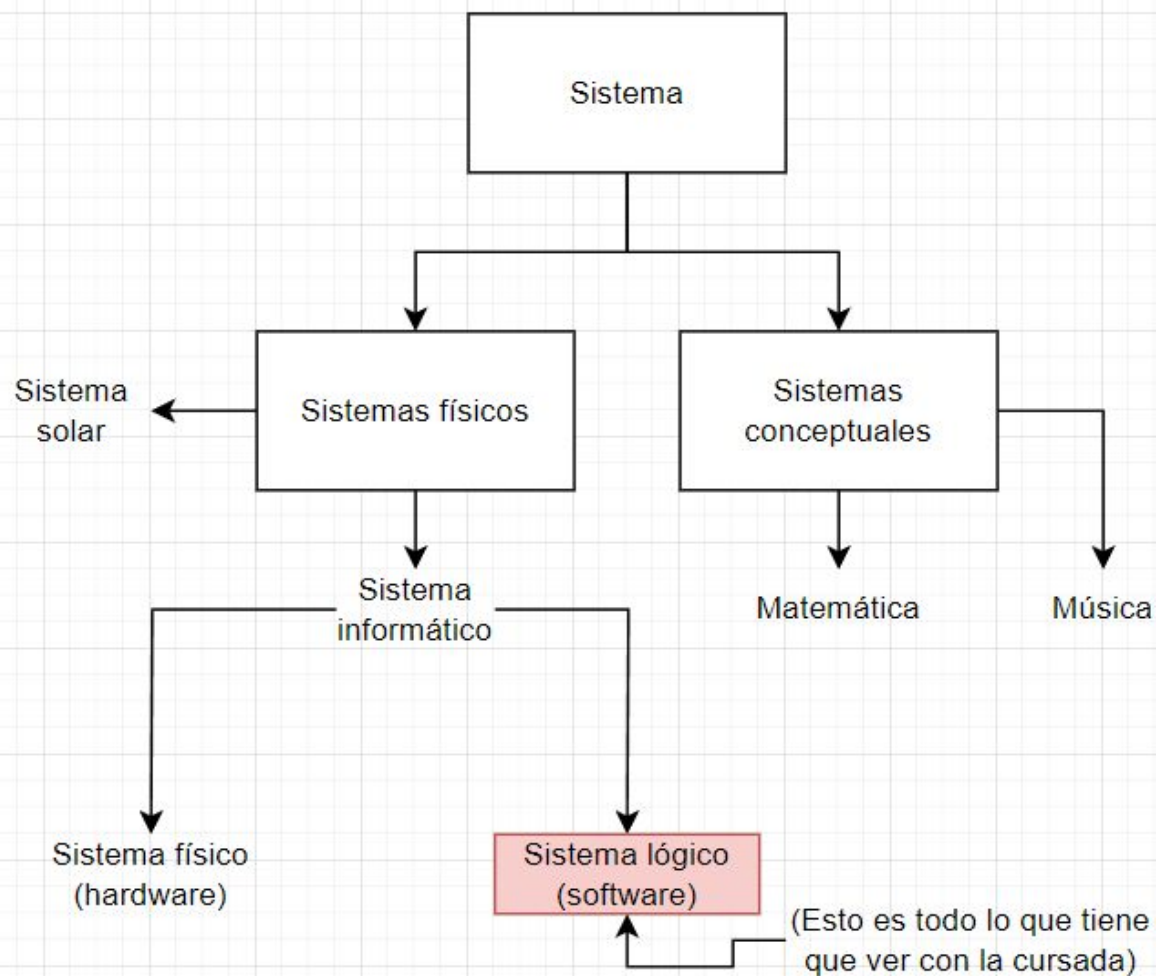


¿Cómo se crea el sistema lógico de un sistema informático?

Hay muchas **metodologías** de desarrollo de software.



- Son frameworks (estructuras) que sirven para estructurar, planear y controlar el **proceso de desarrollo**.
- A lo largo de la historia, una gran cantidad de métodos han sido desarrollados, cada uno con su fortaleza y debilidad.
- Cascada, espiral, iterativo, evolutivo, SCRUM, UML, son algunos ejemplos.



Metodologías de software a lo largo del tiempo

1969: Programación estructurada SOL (todo comportamiento puede secuenciarse por medio de un autómata)

1980: SSADM*, un enfoque de sistemas para el análisis y diseño de sistemas de información. (**cascada**)
1980: SADT**, describe sistemas como una jerarquía de funciones.
1981: Ingeniería de la Información

1975: Programación estructurada de Jackson; utilizaba pasos para capturar la estructura existente de las entradas y salidas de un programa en la estructura del programa en sí.

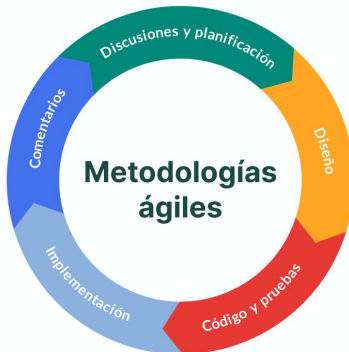
2002: EUP(6), una extensión o variante del **RUP**, para afrontar la falta de producción de un software.
2004: CDM(7), hecho para el diseño y la implementación de inteligencias interactivas.
2005: AUP(8), describe de una manera simple y fácil de entender la forma de desarrollar software de negocio, utilizando conceptos del RUP.

1991: RAD***, que comprende un desarrollo interactivo.
90s: Programación orientada a objetos y VFMS**** (Java).
1995: Dynamic Systems Development Method.
90s: **SCRUM**, que es lo que más aparece en el mercado hoy en día.
1999: **RUP******* (IBM), junto con **UML** constituye la metodología estándar más utilizada para el desarrollo de sistemas orientados a objetos.
1999: **XP** (Extreme Programming), pone más énfasis en la adaptabilidad que en la previsibilidad; cambiar sobre la marcha es natural, inevitable y deseable.

Desarrollo de software ágil o Agile

El software debe ser fundamental en un contexto en el que debe formalizar y automatizar procesos de ingeniería.

GANTTPRO



- Sería de gran importancia contar con un proceso de Arquitectura de Software capaz de acoplarse a cualquier metodología de desarrollo de software.
- Tiene 4 valores:
 - Las interacciones individuales son más importantes que los procesos y las herramientas.
 - Enfoque en el software de trabajo en lugar de una documentación completa.
 - Colaboración en lugar de negociaciones contractuales.
 - Enfoque en responder al cambio.

Cómo encarar una metodología de desarrollo de software

Variables a tener en mente



- Primero tengo que entender qué es lo que quiere el cliente.
- Prueba de aceptación por parte del cliente
- Posibilidad de inversión económica del cliente
- Qué diferencias hay en caso de que haya más de un usuario en el mismo proyecto, y cómo encarar el gusto de cada uno de ellos.

Cómo encarar una metodología de **desarrollo** de software

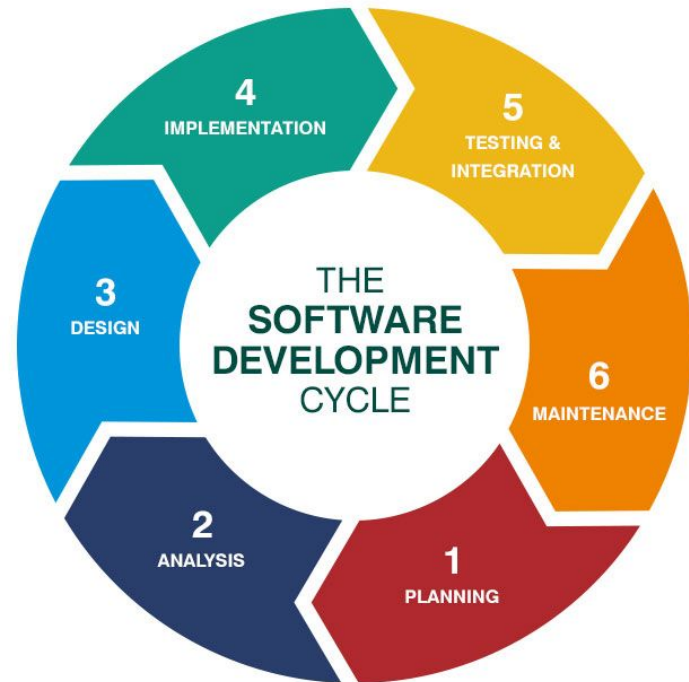
Variables a tener en mente



- Cómo tener herramientas, y obtener un **modelo** conceptual para plasmar un diseño o construir planos para programar el sistema que nos piden.
- Cómo llevar a la programación el diseño construido.
- Tener en cuenta las distintas restricciones a la hora de crear un sistema; operativas o económicas.

¿A qué nos referimos con desarrollo de software?

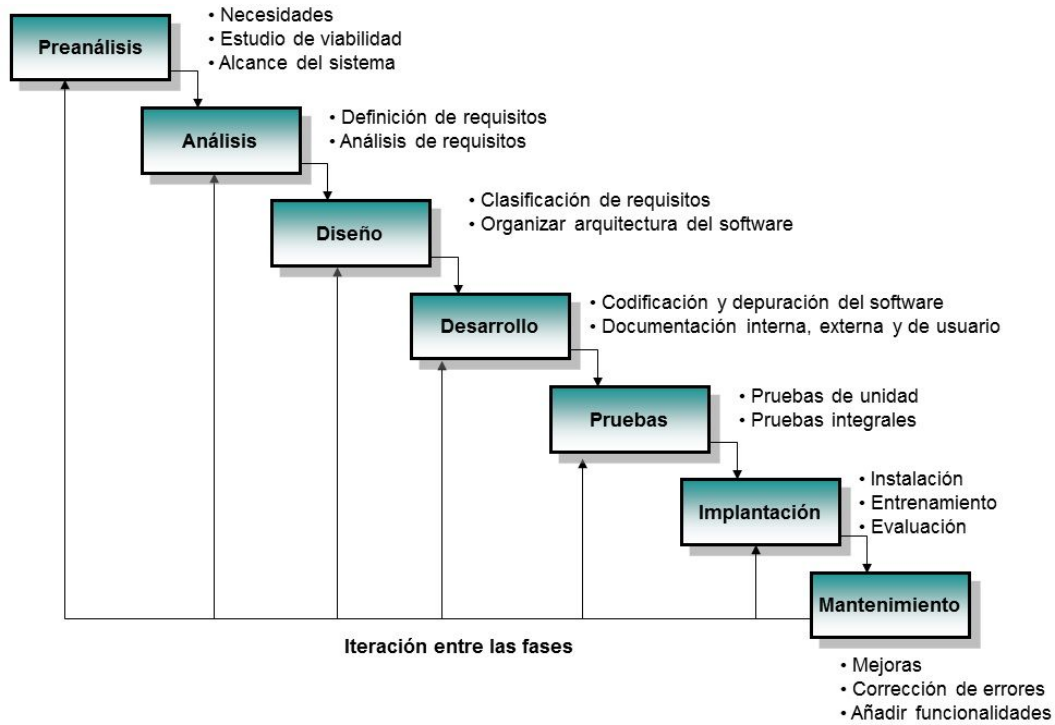
Desarrollo es todo: desde antes de codear, hasta el producto completo.



- El desarrollo de software no es solamente escribir código, sino que significa buscar los requisitos, analizarlos con las distintas alternativas de la mejor propuesta que se le puede hacer al usuario, diseñarlo, construir el código, probarlo e instalarlo. Finalmente, los programas, al ser utilizados por usuarios, terminan requiriendo de un cierto mantenimiento y una actualización constante del mismo.

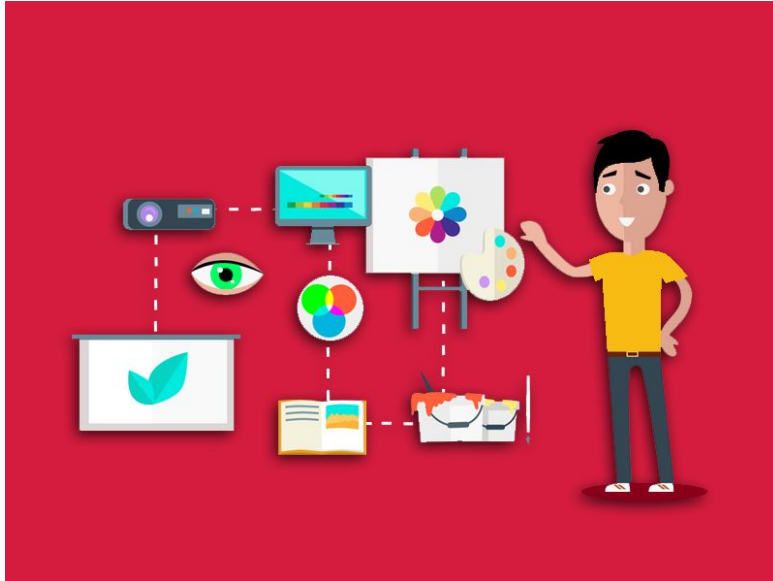
Ciclo de Vida

También podemos llamar como “ciclo de vida” al desarrollo de software.



Modelo

Un modelo es una abstracción de la realidad, que pone énfasis en un determinado aspecto de la misma, mientras que se desentiende de los otros aspectos.



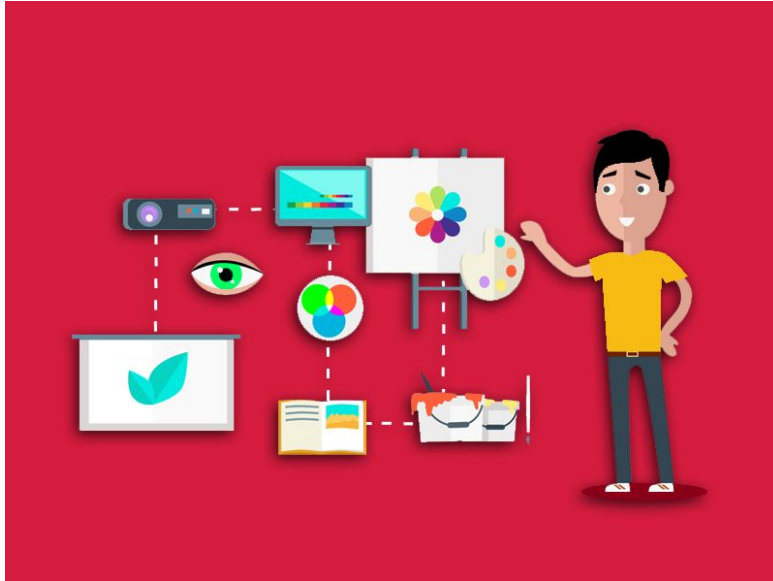
- Deben ser **gráficos** con detalles textuales de apoyo apropiados.
- Deben permitir que el sistema sea visto en segmentos, en forma **descendiente**
- Deben tener redundancia mínima.
- Deben ayudar al lector a predecir el comportamiento del sistema.
- Deben ser transparentes al lector.

Algunos ejemplos de modelos son:

- Mapas
- Notación musical (pentagrama)
- Gráfico de X e Y

¿Para qué usamos modelos? - ¿Por qué no construimos el sistema mismo?

Un modelo/método es una abstracción de la realidad, que pone énfasis en un determinado aspecto de la misma, mientras que se desentiende de los otros aspectos.



- Podemos construir modelos de manera tal que analizamos ciertas propiedades críticas del sistema, mientras que simultáneamente desacentuamos otros de sus aspectos.
- Esto nos permite comunicarnos con los usuarios de manera enfocada, sin distracciones con otros asuntos y características ajenas al sistema.
- Si no comprendimos correctamente los requerimientos de los usuarios, podemos actualizar el modelo, a bajo costo y con riesgo mínimo.
- También podemos verificar que realmente entendimos el ámbito del usuario, y que se ha documentado de tal manera que se puede construir un sistema.

Prácticas útiles para ser utilizadas en procesos de desarrollo de software

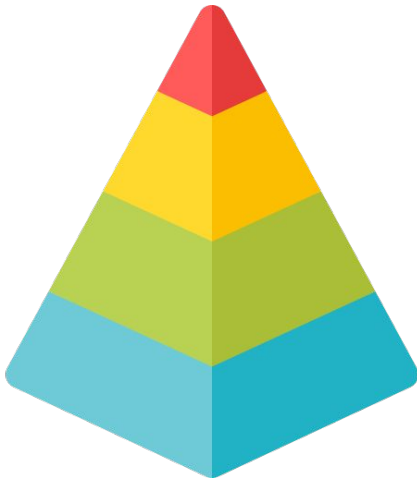
(independientemente de la metodología utilizada)



- Es elemental contar con un proceso de software capaz de acoplarse a cualquier tipo de **metodología** de desarrollo de software, en el que se distingan claramente los beneficios de aplicarlo en cada etapa, y las consecuencias de no tenerlo en cuenta.

No jerarquizar el rol

Prácticas útiles para ser utilizadas en procesos de desarrollo de software



Ser arquitecto es sólo un rol. Este rol puede ser desempeñado por cualquiera que tenga las habilidades necesarias. Habría que considerar como muy importante no darle una concepción divina al rol. Quien lo desempeña no se encuentra en una posición superior, en el organigrama, a ningún otro miembro del equipo.

Un solo arquitecto para definir la arquitectura

Prácticas útiles para ser utilizadas en procesos de desarrollo de software



La arquitectura debe ser el producto de un arquitecto solamente, o de un pequeño grupo con un único líder claramente identificado, pero macro y en una concepción abstracta debe, en lo posible, ser realizada por un solo arquitecto, para brindar un componente ágil, evitando discusiones. El arquitecto debe compartir, delegar, y comunicar cuestiones de menor abstracción y puede apoyarse en otros integrantes del equipo de desarrollo para tomar las decisiones. A éste vamos a llamarle **arquitecto de software**.

Requerimientos de calidad

Prácticas útiles para ser utilizadas en procesos de desarrollo de software







Los arquitectos deben tener una lista de requerimientos funcionales para el sistema y una lista priorizada de atributos de calidad (modularidad, modificabilidad, etc.). Los atributos de calidad pueden estar categorizados y no deberían limitarse solamente a la calidad del producto final, sino también a los requerimientos de calidad de diseño, código, testing, performance, etc..

Riesgos

Prácticas útiles para ser utilizadas en procesos de desarrollo de software

Evaluación de riesgos Análisis de impacto con probabilidad

This slide is 100% editable. Adapt it to your needs and capture your audience's attention.

					
SEVERITY	PROBABILITY	Catastrophic (1)	Critical (2)	Marginal (3)	Negligible (4)
Frequent (A)		High	High	Serious	Medium
Probable (B)		High	High	Serious	Medium
Occasional (C)		High	Serious	Medium	Low
Remote (D)		Serious	Medium	Medium	Low
Improbable (E)		Medium	Medium	Medium	Low
Eliminated (F)		Eliminated			

Muchos requerimientos de calidad pueden significar riesgos; si uno de ellos no se cumple, implica un riesgo. Se suelen usar plantillas en las que se detalla el impacto, costo de mitigación, probabilidad, etc., del riesgo. En la plantilla, siempre debe indicar la arquitectura seleccionada a la hora de crear el sistema.

No intentar predecir el futuro

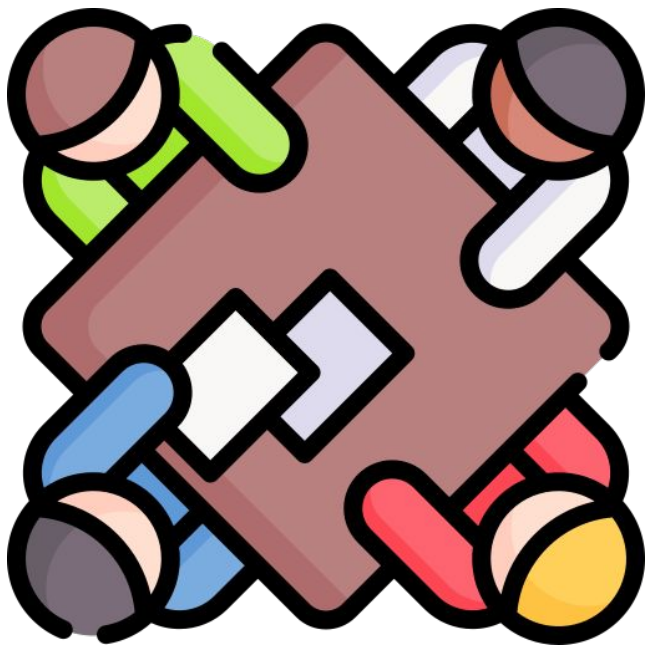
Prácticas útiles para ser utilizadas en procesos de desarrollo de software



Realizar solamente la arquitectura de lo que se sabe no va a cambiar en el corto plazo. Hoy día la creación de arquitecturas de aplicación está evolucionando hacia la especificación de servicios y la creación de arquitecturas transversales. Esto plantea la creación de arquitecturas transversales a los servicios y evolutivas en sí mismas. Querer predecir el futuro de la compañía y cómo será su negocio en el mediano plazo no es tarea del arquitecto ni de los desarrolladores.

Grupos de arquitectura transversales

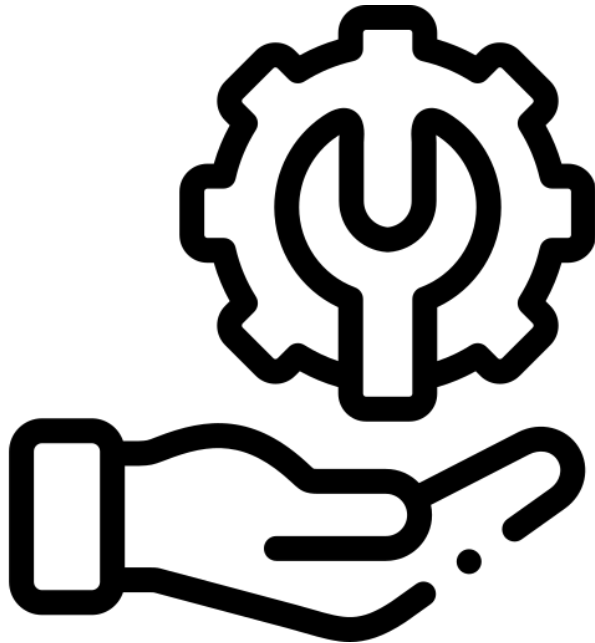
Prácticas útiles para ser utilizadas en procesos de desarrollo de software



Consiste en grupos de arquitectura con gente de distintos módulos, que se desempeñen en tiempo parcial, haciendo tareas de arquitectura y compartiendo su opinión respecto a esta. Este enfoque brinda la oportunidad de dar a conocer mejor la arquitectura a todo el grupo de desarrollo e involucrar a todos, mejorando la arquitectura gracias al feedback de los involucrados.

Plantear la arquitectura como un servicio

Prácticas útiles para ser utilizadas en procesos de desarrollo de software



La arquitectura puede verse como un servicio que da soporte a los procesos de negocios. Bajo esta concepción, la arquitectura debería dar soporte al negocio y debería poder evolucionar con éste. Desde este punto de vista, el carácter evolutivo o no de una arquitectura puede verse como un requerimiento de calidad.

Definir un lenguaje de alto nivel profesional

Prácticas útiles para ser utilizadas en procesos de desarrollo de software



Utilizar un lenguaje altamente profesional, mediante el conocimiento común de las técnicas y herramientas utilizadas por el grupo de profesionales, agiliza el proceso de comunicación entre las partes.

Si bien es cierto que existe un proceso de aprendizaje del lenguaje, se la puede considerar como una inversión a corto plazo.

Testear los requerimientos de calidad

Prácticas útiles para ser utilizadas en procesos de desarrollo de software



La arquitectura debe velar por los requerimientos de calidad. Es así que existen tests de unidad, se pueden definir baterías de test (conjunto de instrumentos para la evaluación y el seguimiento del desarrollo), entre otros, que validen los requerimientos de calidad, por ejemplo, el tiempo de respuesta promedio, cantidad de usuarios concurrentes, etc.. Los tests pueden automatizarse e incluirse dentro del proceso normal del desarrollo de software.

Puntos de control

Prácticas útiles para ser utilizadas en procesos de desarrollo de software



Auditar constantemente la arquitectura y su uso hará que se tengan en cuenta los nuevos requerimientos. Puede resultar bastante útil y práctico definir puntos de control en los tests de integración.

Cada test de integración plantea la necesidad de correr la batería de test de arquitectura que valida, y certifica los requerimientos de calidad.

No transferir responsabilidades

Prácticas útiles para ser utilizadas en procesos de desarrollo de software



El desempeño de un profesional en un rol determinado le confiere derechos y obligaciones. El rol de arquitecto tiene la obligación de no transferir la responsabilidad por las decisiones tomadas y por el resultado final de la arquitectura. Si la arquitectura no cumple con los requerimientos especificados, es responsabilidad del arquitecto y del grupo.

Releases incrementales

Prácticas útiles para ser utilizadas en procesos de desarrollo de software



Implementar releases incrementales que sean certificadas por el grupo de arquitectura. Cada release debe ser arquitecturalmente válida, es decir, debe respetar la arquitectura actual.

Es posible que la arquitectura evolucione y cambie, pero siempre hay que respetar los requerimientos de calidad.

Gracias!!!!!!

