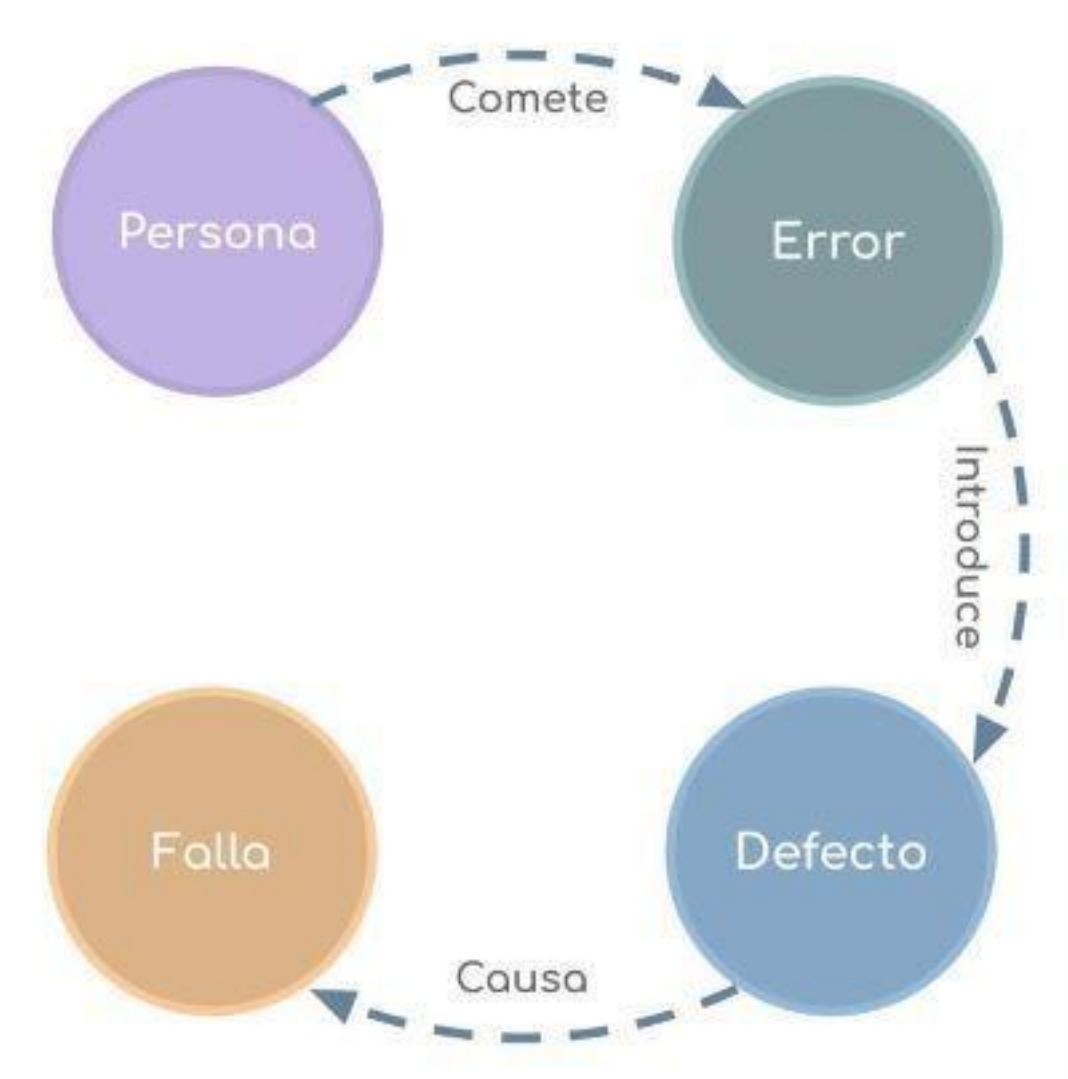
I. INTRODUCCIÓN AL PROCESO DE CALIDAD

“Testers don't break the code, they break your illusions about the code.”

1.1 El Rol de QA

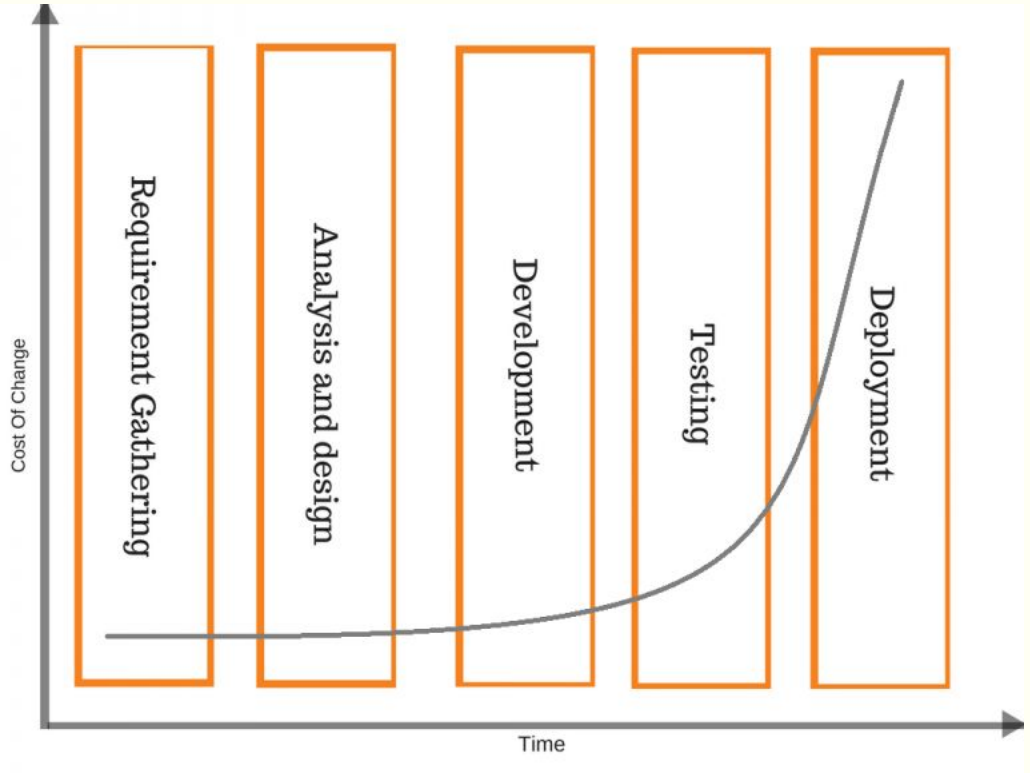
~~Causas de los errores del~~ Software

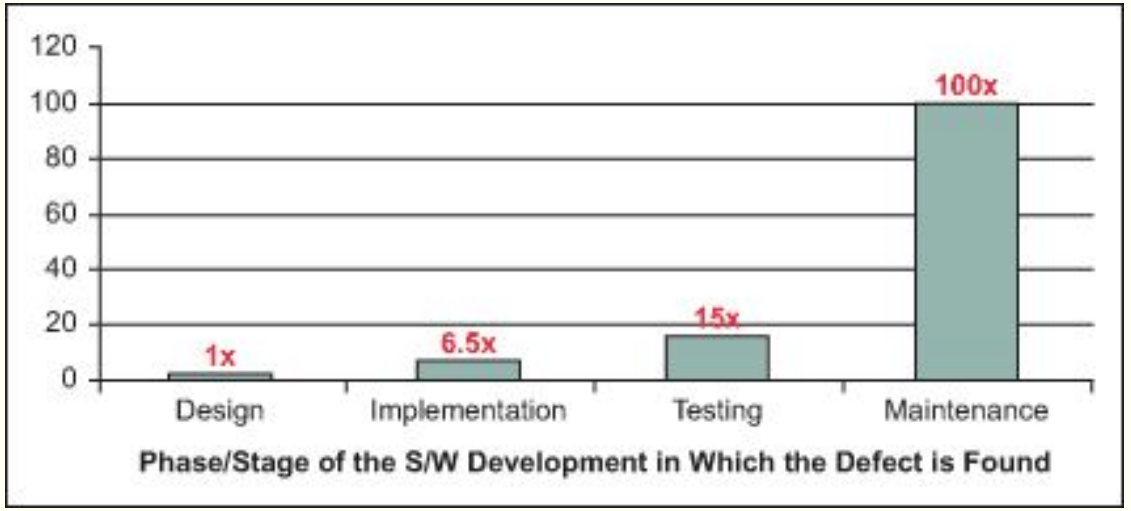
- Problemas de comunicación
- Complejidad del software
- Errores de programación
- Requisitos cambiantes
- Presión
- Exceso de confianza
- Código no documentado
- Herramientas
- Falta de *Testing* o habilidades



1.1 El Rol de QA

Coste de los errores del Software





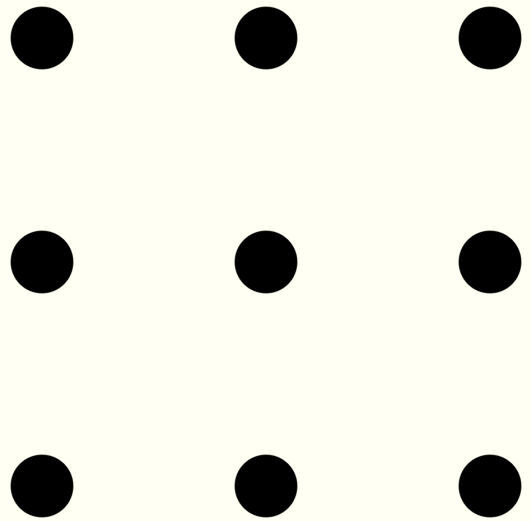
1.1 El Rol de QA

Habilidades en el Rol de QA

- Atención al detalle
- Gestión del tiempo y tareas
- Resolutivo
- Multi-tarea
- Excelente comunicación verbal y escrita
- Trabajo en equipo
- Creatividad (”*Out of the box*”)

EJEMPLO:

- https://www.youtube.com/watch?v=izJSSFNmnv0



1.1 El Rol de QA

QA vs. *Tester*

- Tester: persona encargada de probar un software durante su fase de desarrollo con el fin de detectar fallos e informar sobre ellos.

- Quality Assurance: persona que realiza un conjunto de actividades con el objetivo asegurar la calidad de un software durante todas sus fases.

En muchos casos, un QA realiza las tareas de un *Tester*, pero no podemos decir que un *Tester* realice las tareas de un QA

- Un Tester se encarga de encontrar fallos, pero un QA no solo los encuentra, sino que ayuda a prevenirlos".

1.1 El Rol de QA

QA

El **QA** trabaja en:

1. Preparación de la estrategia de pruebas del Sprint
2. Preparación y mantenimiento del Entorno de Pruebas
3. Preparación de Casos de Pruebas de los requisitos del Sprint
4. Creación y mantenimiento de datos de prueba
5. Validación de “Definición de Hecho” de los requisitos
6. Registro y notificación de *Bugs*
7. Aceptación del desarrollo junto con el PO
8. Llevar las métricas de Calidad de *Software*

Además:

1.1 El Rol de QA

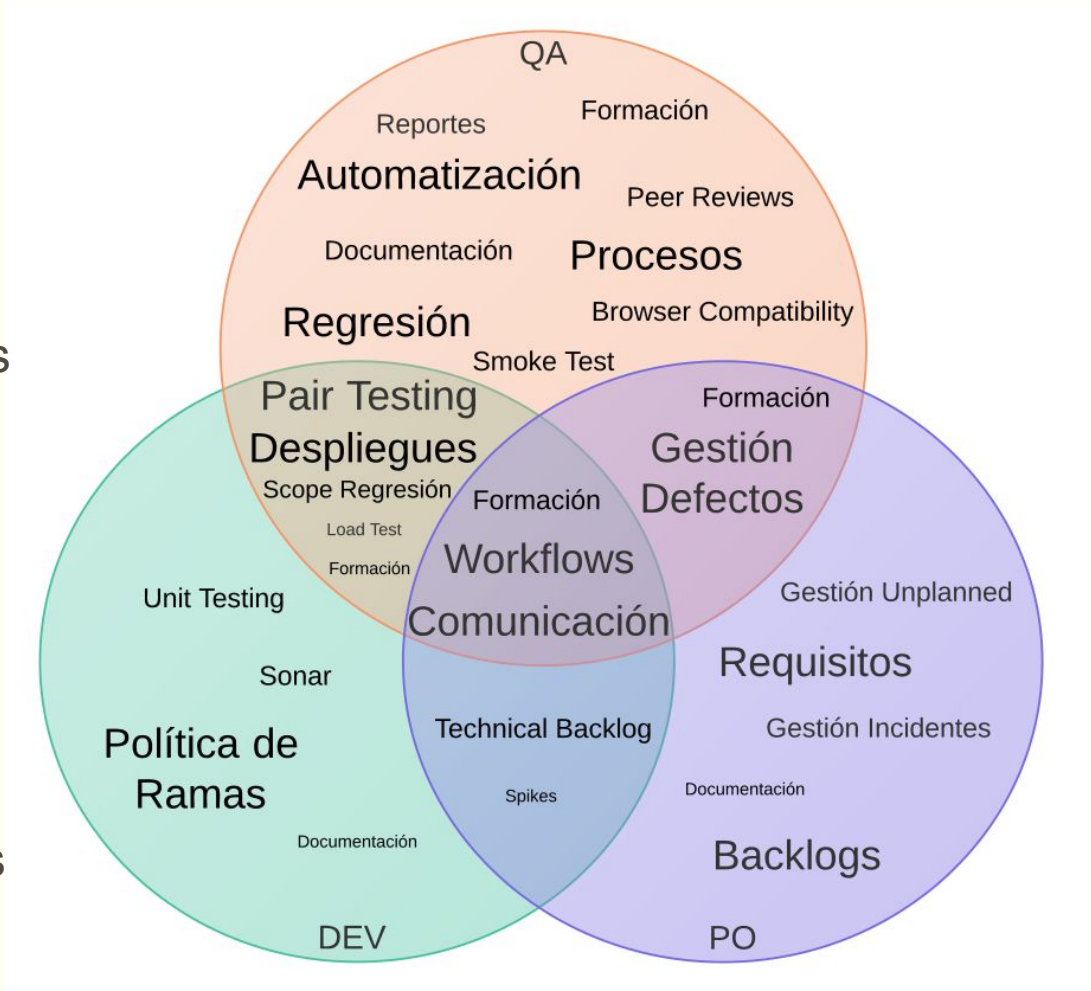
QA

Tareas junto con el ***Product Owner*** y/o ***Business Analyst***:

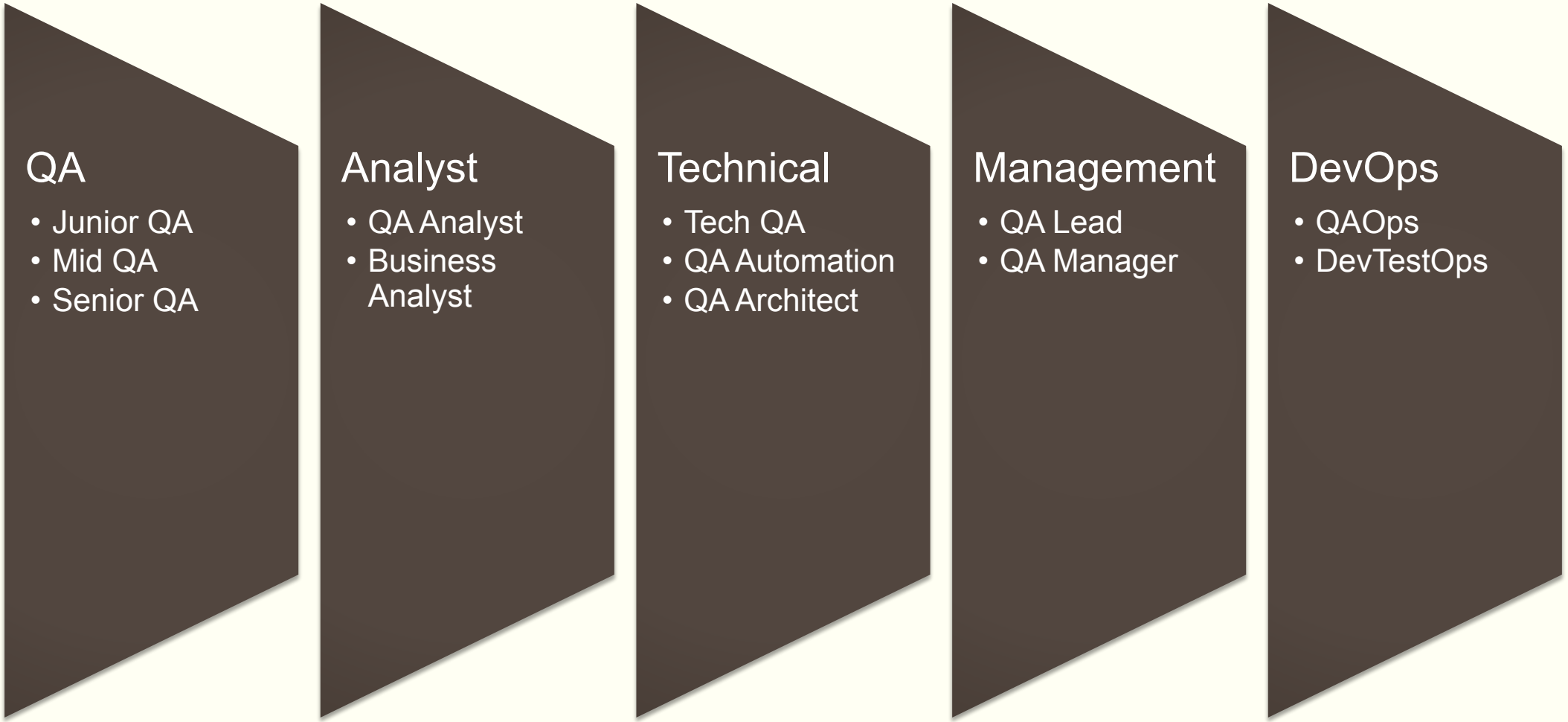
1. Mejor comprensión de las necesidades del cliente
2. Revisión de requisitos
3. Definición de Criterios de Aceptación (a partir de los cuales se pueden generar Casos de Prueba)
4. Identificar dependencias entre requisitos
5. Identificar requerimientos No funcionales

Tareas junto con los ***Developers***:

1. La definición de estrategias de prueba
2. Apoya en la automatización de pruebas funcionales
3. Detecta y notifica bugs
4. Sugiere mejoras funcionales y de usabilidad



1.1 El Rol de QA



1.2 Concepto de Calidad del Software: Principios

Calidad del Software, ¿estamos haciendo **correctamente** el producto **correcto**?

- Es la aptitud de un producto o servicio para satisfacer las necesidades del usuario.
- Además, lo consideramos de mejor calidad cuando el coste de su mantenimiento es bajo y la dificultad para introducir nuevos cambios (nuevos requisitos) también es baja o trivial
- En el desarrollo de software, la calidad del producto está ligada a la calidad de los requisitos, especificaciones y diseño del sistema.
- Si la implementación sigue al diseño, y el sistema resultante cumple con los objetivos de requisitos y de rendimiento, la calidad de es alta.

¿Qué tiene más calidad?

Bic Cristal Original…
1,89 €
eBay
Envío gratuito

Montblanc - Meisterstück 149…
795,00 €
Montblanc Spain

1.2 Concepto de Calidad del Software: Principios

Las pruebas de *Software* muestran la presencia de defectos

- Todo desarrollo es susceptible de tener defectos.
- Con el *testing*, buscamos reducir la presencia de estos.
- NO podemos asegurar que el software esté libre de defectos.
- Con nuevas modificaciones y adiciones de funcionalidad aumentos el riesgo de presencia de nuevos defectos.

1.2 Concepto de Calidad del Software: Principios

Las pruebas de exhaustivas son imposibles

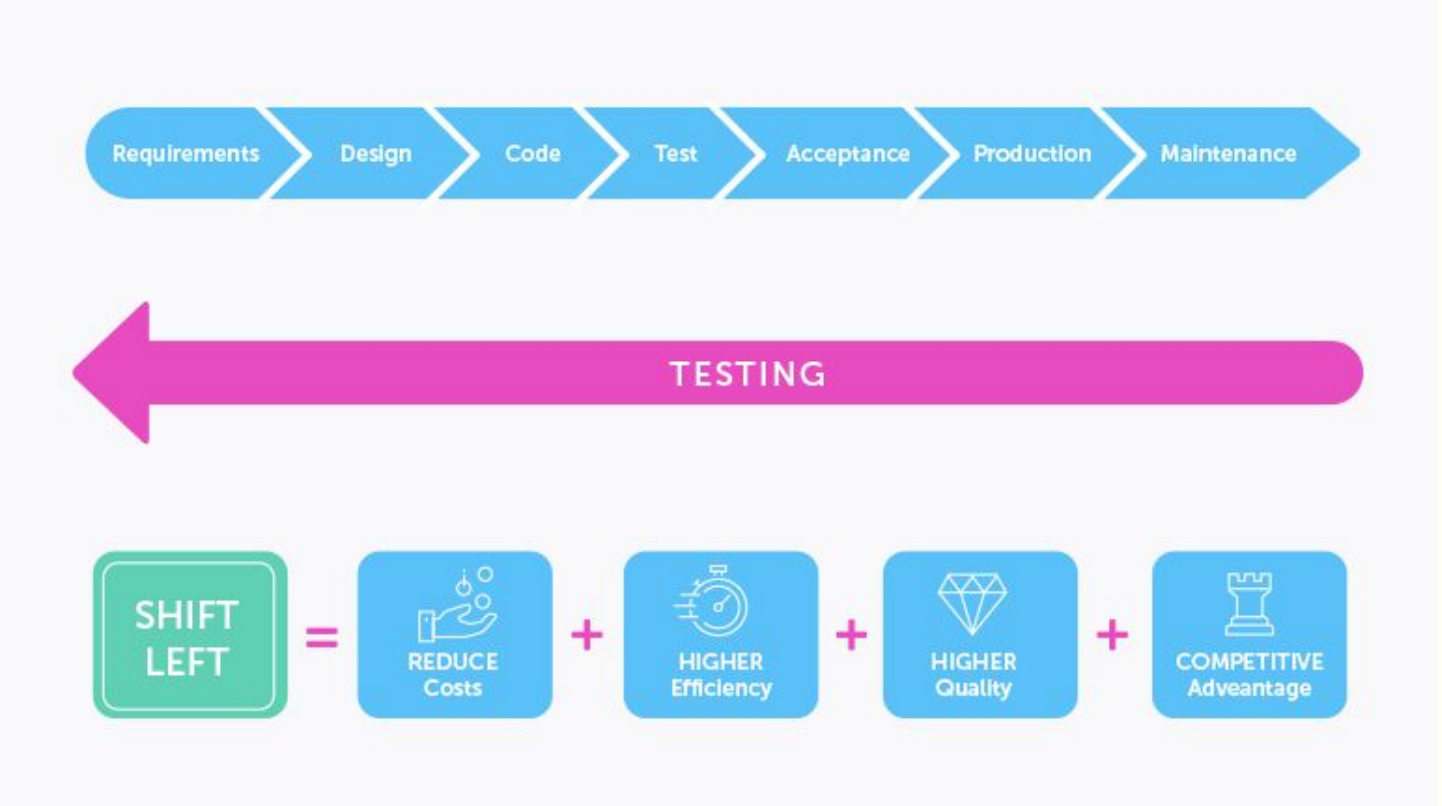
- Por sencillo que sea un software, es casi imposible cubrir todas las posible combinaciones.
- Tenemos que identificar funcionalidades prioritarias para su cobertura.

¿Podríamos hacer pruebas exhaustivas sobre este programa?

1.2 Concepto de Calidad del Software: Principios

Pruebas tempranas

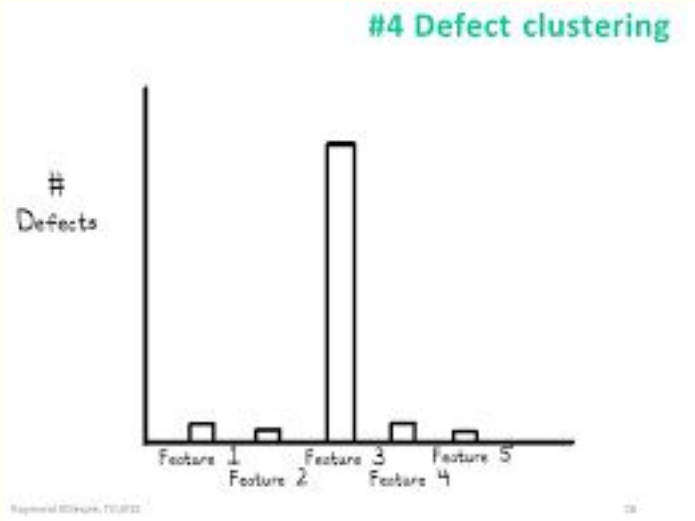
- Las pruebas deben realizarse cuanto antes mejor.
- La efectividad de las pruebas será mayor en fases tempranas del *testing*, incluso desde los requisitos.
- Mientras más tarde se realicen las pruebas, mayor será el coste de los defectos.

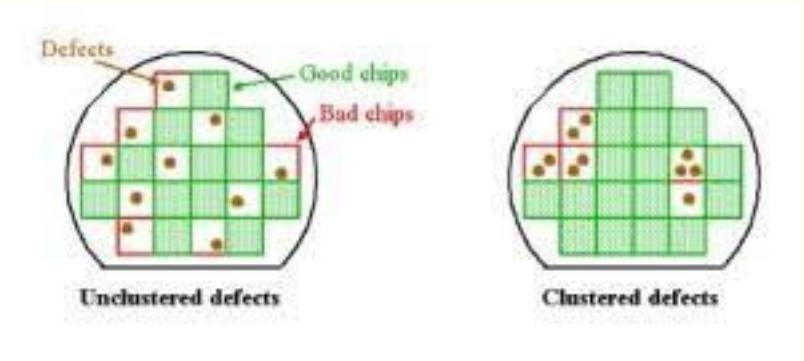


1.2 Concepto de Calidad del Software: Principios

Agrupación de defectos

- La mayoría de los defectos se encuentran en una parte específica del *software*.
- Es necesario priorizar tempranamente áreas críticas.
- Las pruebas se enfocarán en zonas sensibles del *software*.





1.2 Concepto de Calidad del Software: Principios

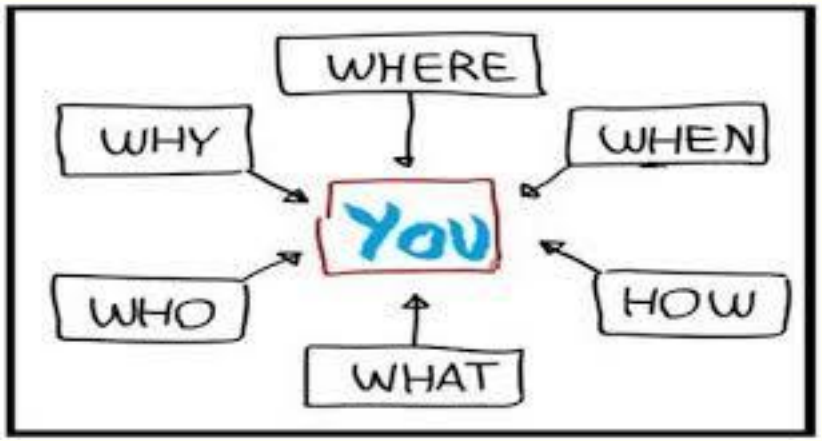
La paradoja del “Pesticida”

- Similar al hecho de que los bichos o insectos se vuelven inmunes al pesticida.
- Los mismos casos de pruebas dejarán de ser efectivos y no encontrarán nuevos fallos.
- Es necesario el mantenimiento y refactorización.
- La adición de nuevos casos, ayudará a encontrar nuevos defectos.

1.2 Concepto de Calidad del Software: Principios

Las pruebas son dependientes del contexto

- Las pruebas tienen que adecuarse al tipo de software que se está probando.
- No hay que ejecutar el mismo tiempo de pruebas sobre un software médico que un software bancario o una app móvil.
- El nivel de las pruebas también depende del contexto entre otros factores.

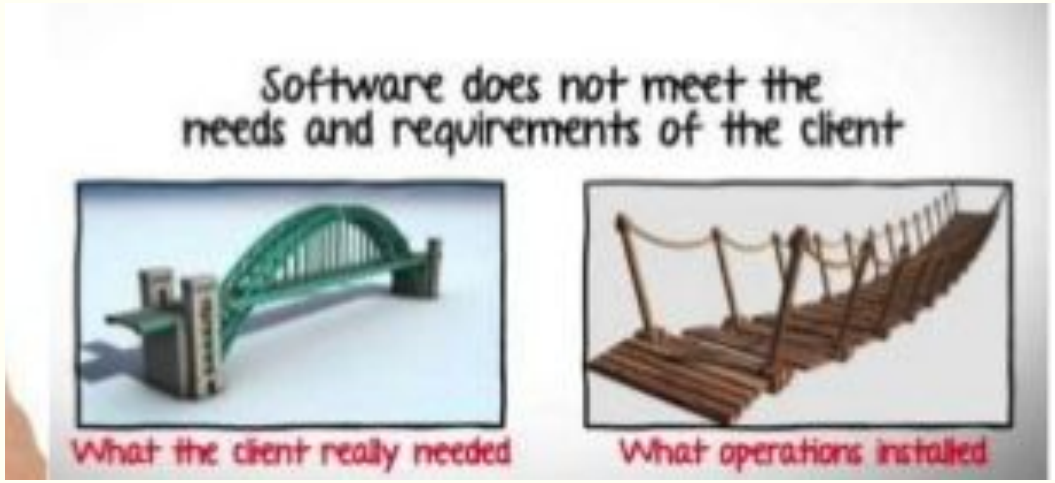


1.2 Concepto de Calidad del Software: Principios

Falacia de ausencia de errores

- Un software ha podido pasar satisfactoriamente todas las fases de prueba, pero esto no indica que no pueda tener defectos que aún no se han logrado identificar.

- El hecho de que no se hayan encontrado errores, tampoco garantiza que el producto entregado sea el esperado.



1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Equipo de Desarrollo Tradicional

- JEFE DE PROYECTO:
  Es el encargado de hacer el seguimiento diario de las tareas y de resolver cualquier problema de comunicación con otros equipos si los hubiera. Es el responsable de la agenda del proyecto y la implementación de los requerimientos.

- LÍDER DE EQUIPO: Es el programador líder. Se encarga de escribir las especificaciones técnicas y crear las tareas, asignándolas a los desarrolladores de su equipo. Sus tareas de programación se focaliza en la arquitectura. Aparte de esto, tiene que revisar el trabajo de los programadores a su cargo para asegurar la calidad del código escrito.

- DESARROLLADOR: Es un programador, que se encarga de ejecutar el trabajo asignado por el líder del equipo. Su trabajo es utilizar la arquitectura para completar los requerimientos.

1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Equipo de Desarrollo Tradicional

- DISEÑADOR GRÁFICO Y UX: Este rol consiste en, a nivel de UX, realizar los flujos de trabajo dentro de una aplicación a nivel de mockups. Posteriormente, es el encargado de realizar el diseño gráfico de las pantallas que compone la aplicación, atendiendo a las reglas de UX que determinen la posición de los elementos, los esquemas de colores, tipografías, etc.

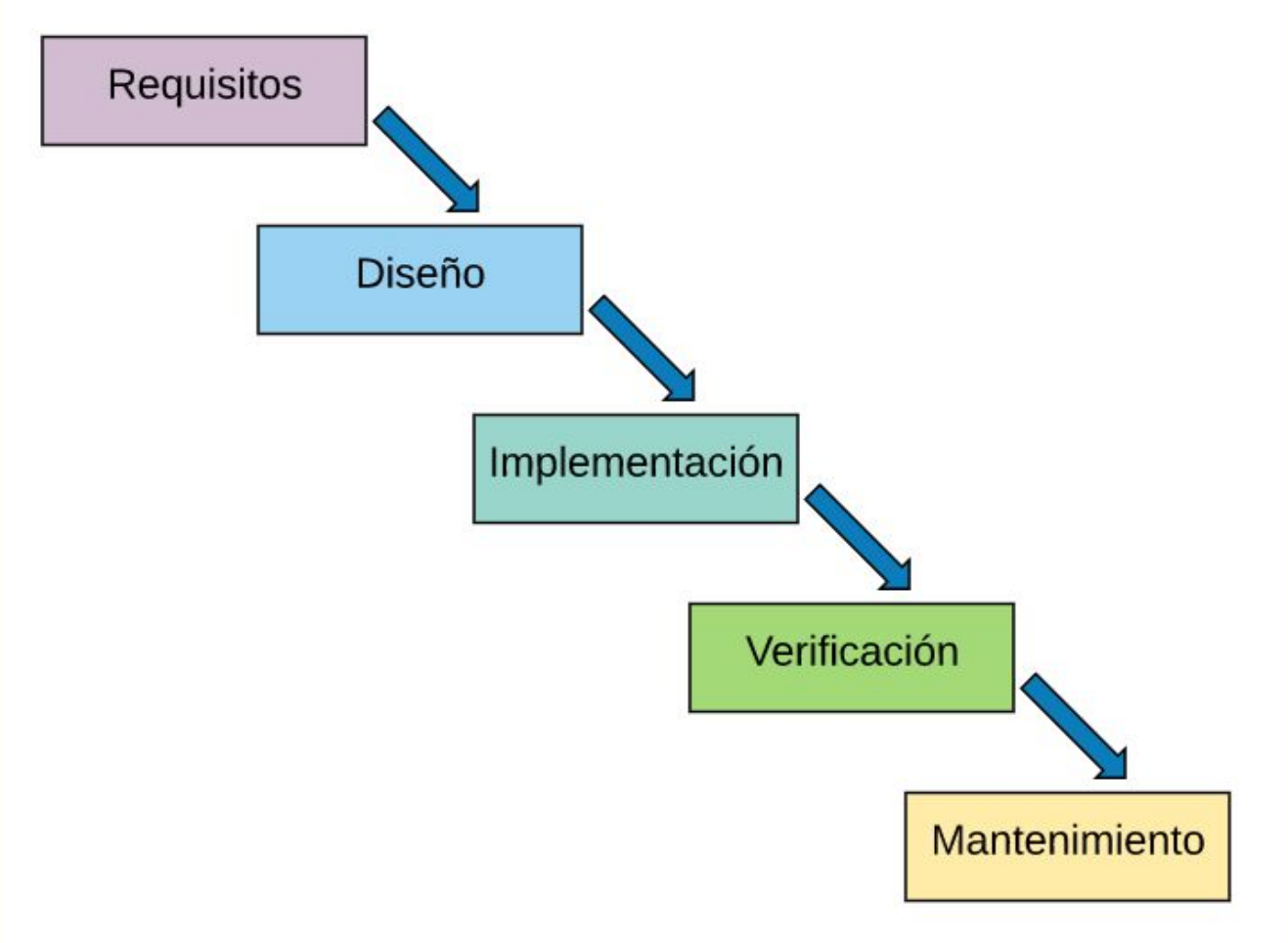
- LÍDER DE CALIDAD: Debe ser capaz, utilizando los requerimientos, de desarrollar una suite de *tests* que verifiquen que el software cumple con los requerimientos. El líder de calidad es el último responsable de que las características funcionan tal y como se han especificado en los requerimientos.

- ANALISTA DE NEGOCIO O DUEÑO DEL PRODUCTO: Es la persona que traslada las necesidades del cliente a requisitos de desarrollo.

1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Modelo en Cascada (Waterfall)

- El modelo en cascada es un proceso de desarrollo secuencial, en el que el desarrollo de software se concibe como un conjunto de etapas que se ejecutan una tras otra. Se le denomina así por las posiciones que ocupan las diferentes fases que componen el proyecto, colocadas una encima de otra, y siguiendo un flujo de ejecución de arriba hacia abajo, como una cascada.

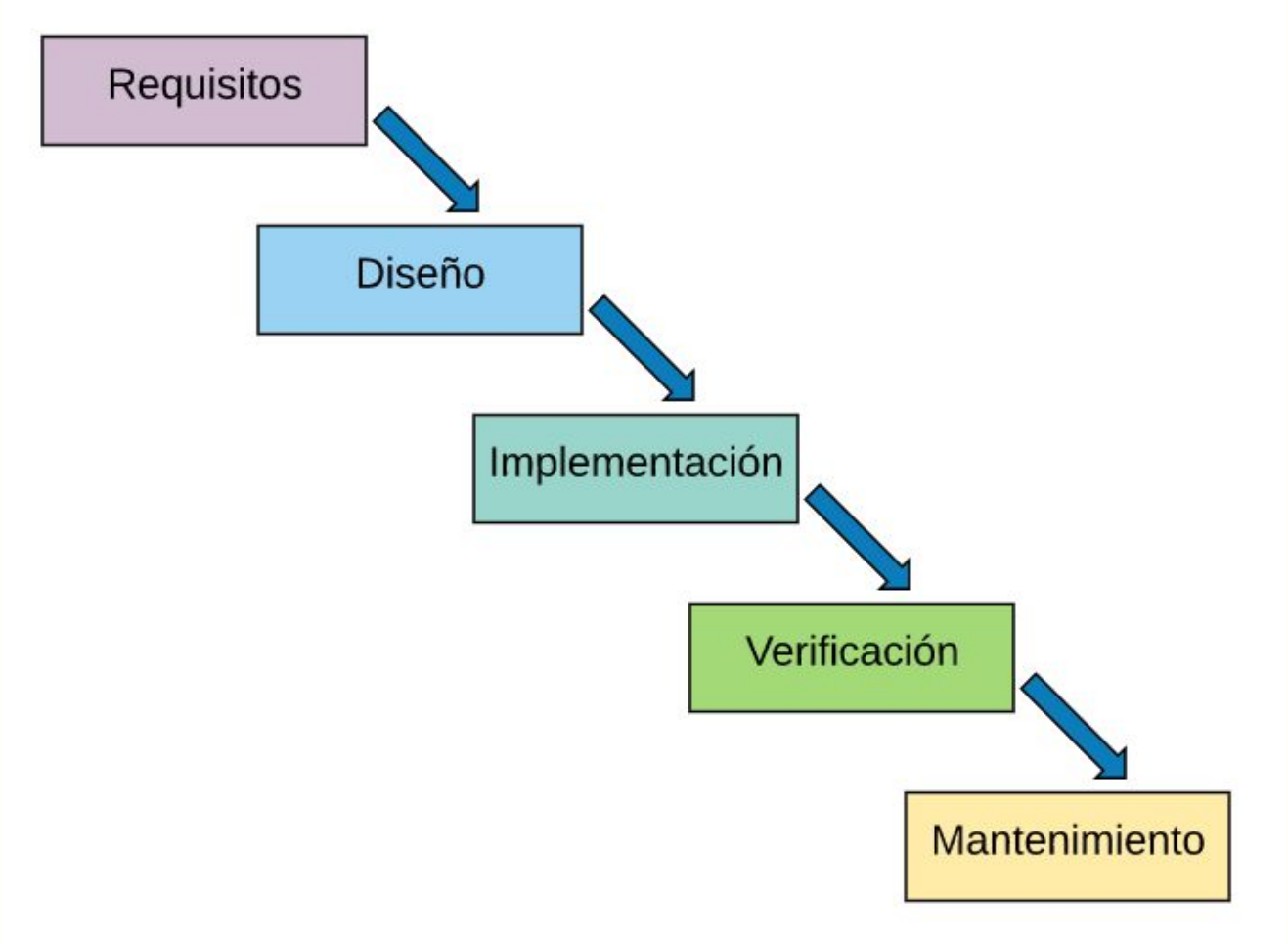


1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Modelo en Cascada (Waterfall)

Requisitos del software

- Se hace un análisis de las necesidades del cliente para determinar las características del software a desarrollar.
- Se especifica todo lo que debe hacer el sistema sin entrar en detalles técnicos.
- No se pueden añadir nuevos requisitos en mitad del proceso de desarrollo.
- Nos permite estimar de forma rigurosa las el coste del producto, los riesgos y los plazos.

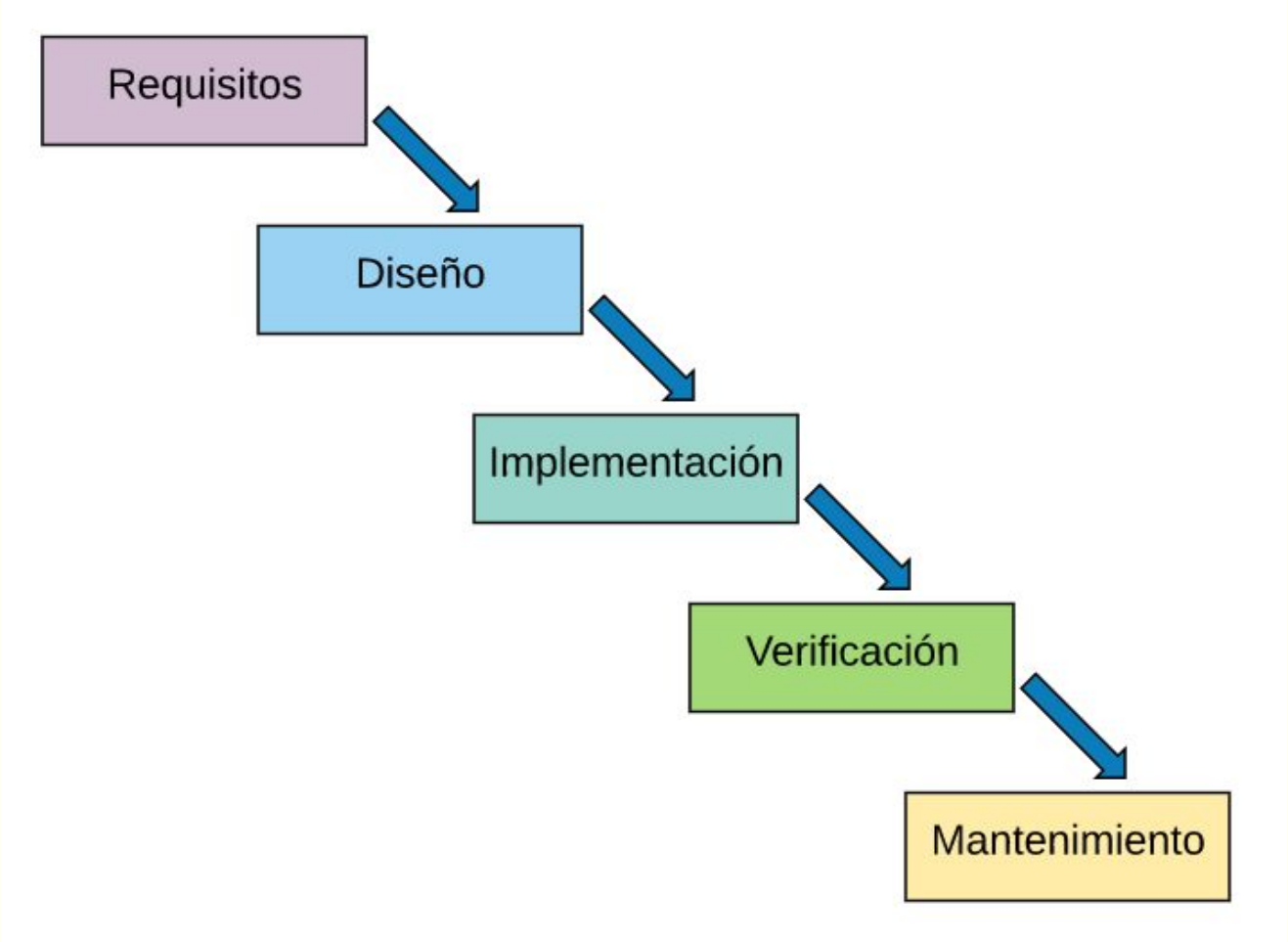


1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Modelo en Cascada (Waterfall)

Diseño

- Se describe la estructura interna del software, y las relaciones entre las entidades que lo componen.
- Se detalla la especificación de lo que debe hacer cada una de sus partes, así como la manera en que se combinan unas con otras.
- Se define la arquitectura de la solución elegida.

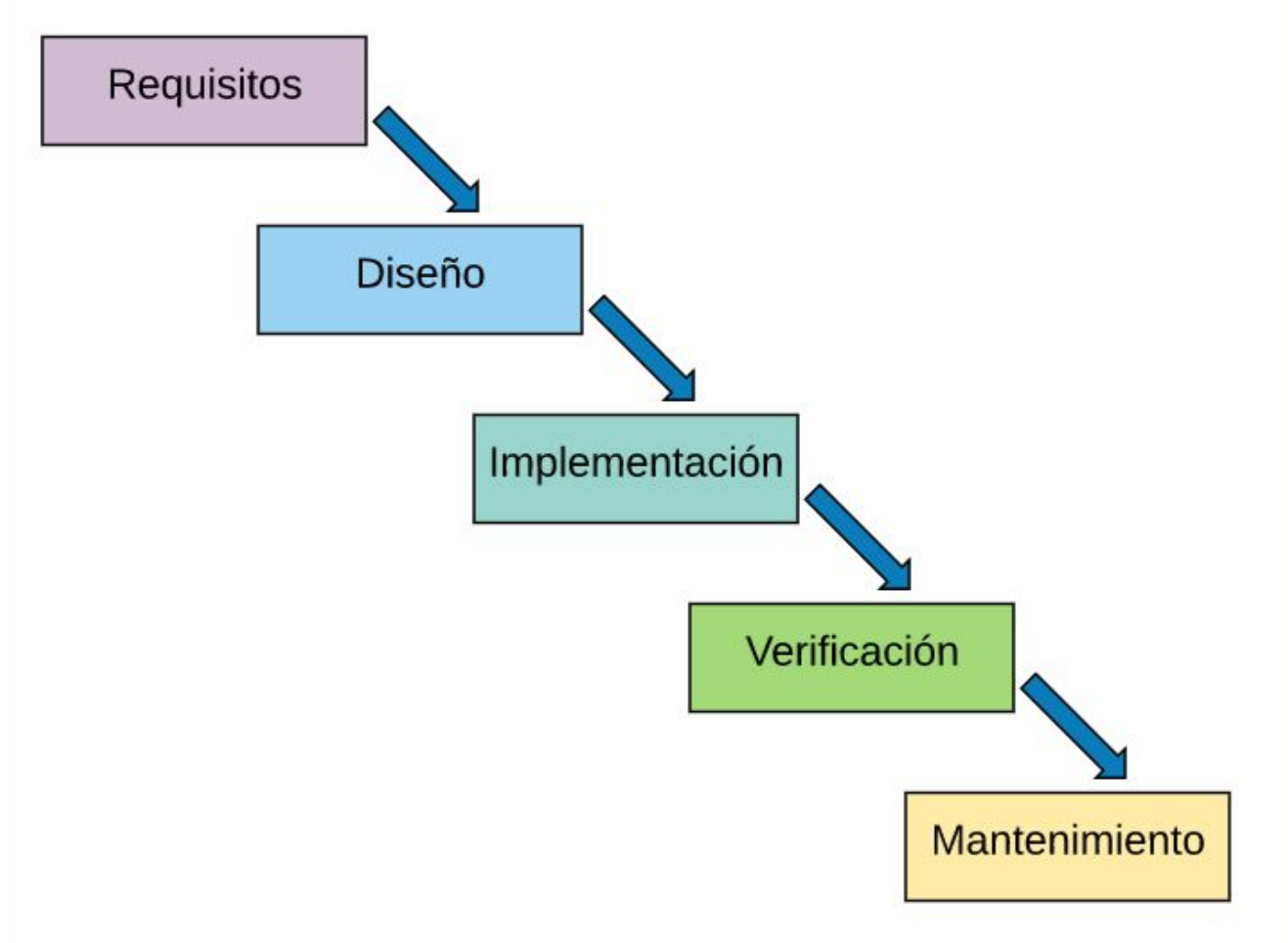


1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Modelo en Cascada (Waterfall)

Implementación

- Se programan los requisitos especificados.
- La programación es el proceso de creación de algoritmos, y la implementación de éstos en un lenguaje de programación específico.
- Un algoritmo es un conjunto de instrucciones o reglas bien definidas y ordenadas que permiten llevar a cabo una actividad mediante pasos sucesivos.

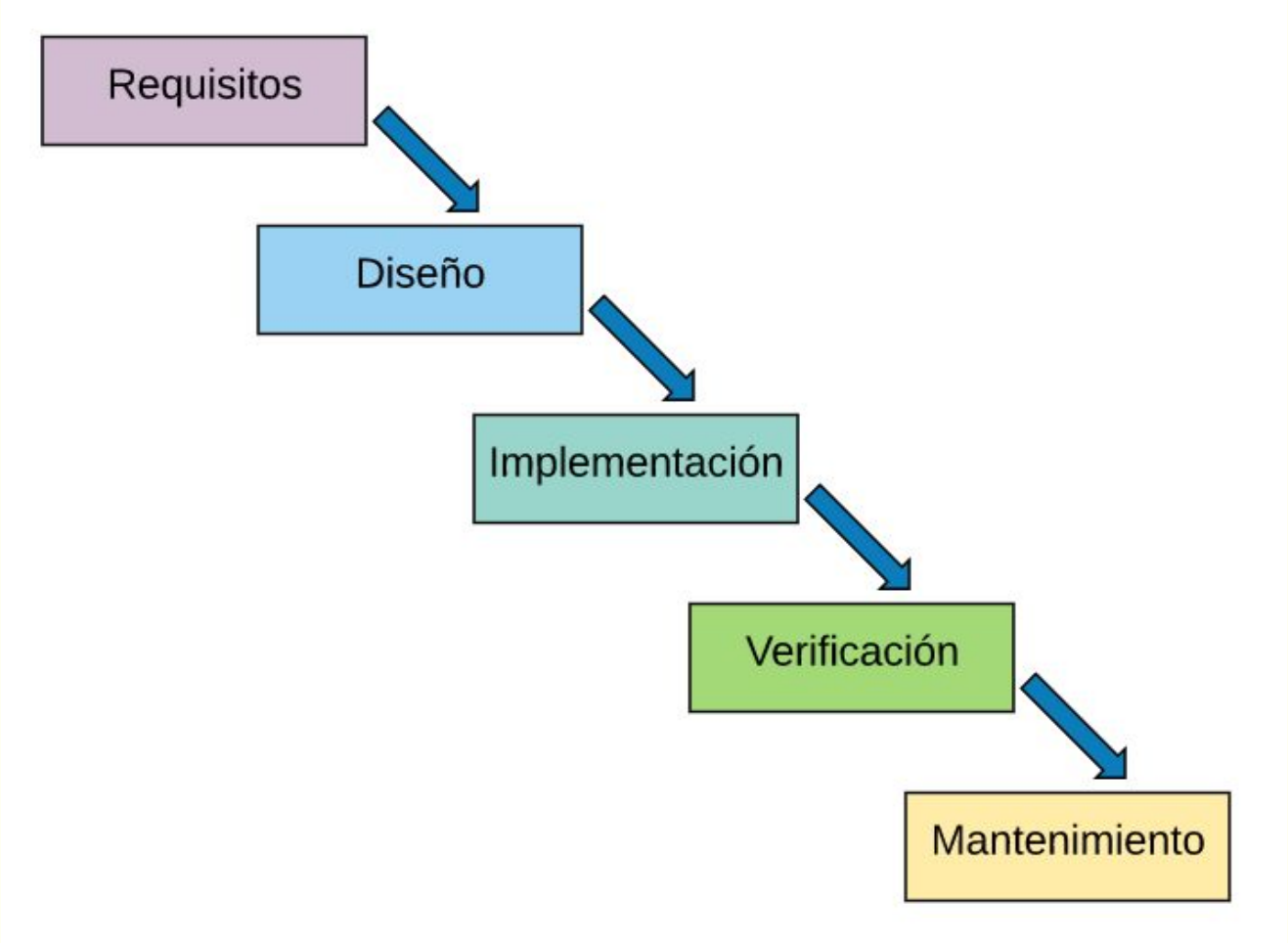


1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Modelo en Cascada (Waterfall)

Verificación

- Se verifica que todos los componentes del sistema funcionen correctamente y cumplen con los requisitos.
- Las pruebas sirven para: encontrar defectos o bugs, aumentar la calidad del software, refinar el código previamente escrito sin miedo a romperlo o introducir nuevos bugs, etc.

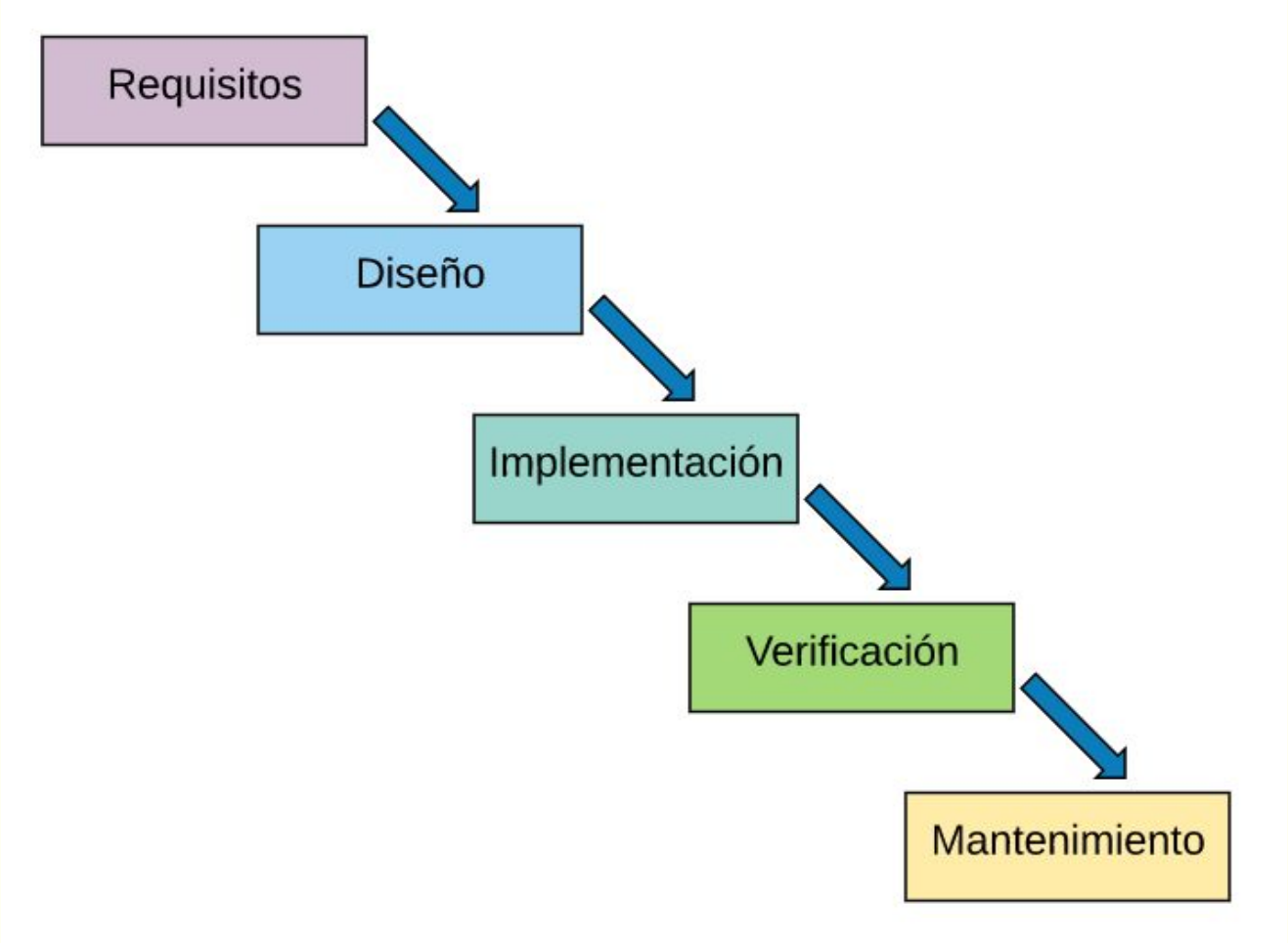


1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Modelo en Cascada (Waterfall)

Mantenimiento

- Se instala la aplicación en el sistema y se comprueba que funcione correctamente.
- El mantenimiento del software consiste en la modificación del producto después de haber sido entregado al cliente.



1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Modelo en Cascada (Waterfall)

Ventajas

- El tiempo que se pasa en diseñar el producto en las primeras fases del proceso puede evitar problemas que serían más costosos cuando el proyecto ya estuviese en fase de desarrollo.
- Al ser un proyecto muy estructurado, con fases bien definidas, es fácil entender el proyecto.
- Ideal para proyectos estables, donde los requisitos son claros y no van a cambiar a lo largo del proceso de desarrollo.

Inconvenientes

- Cualquier cambio de requisito supondrá volver a realizar fases ya superadas y provocará un incremento del coste.
- No se va mostrando al cliente el producto a medida que se va desarrollando, si no que se ve el resultado una vez ha terminado todo el proceso.
- Los diseñadores pueden no tener en cuenta todas las dificultades que se encontrarán cuando estén diseñando un software, lo que conllevará rediseñar el proyecto para solventar el problema.

1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Modelo en Cascada (Waterfall)

Ventajas

- El tiempo que se pasa en diseñar el producto en las primeras fases del proceso puede evitar problemas que serían más costosos cuando el proyecto ya estuviese en fase de desarrollo.
- Al ser un proyecto muy estructurado, con fa ien definidas, es fácil entender el proyecto.
- Ideal para proyectos estables, donde los r son claros y no van a cambiar a lo largo del proceso de desarrollo.

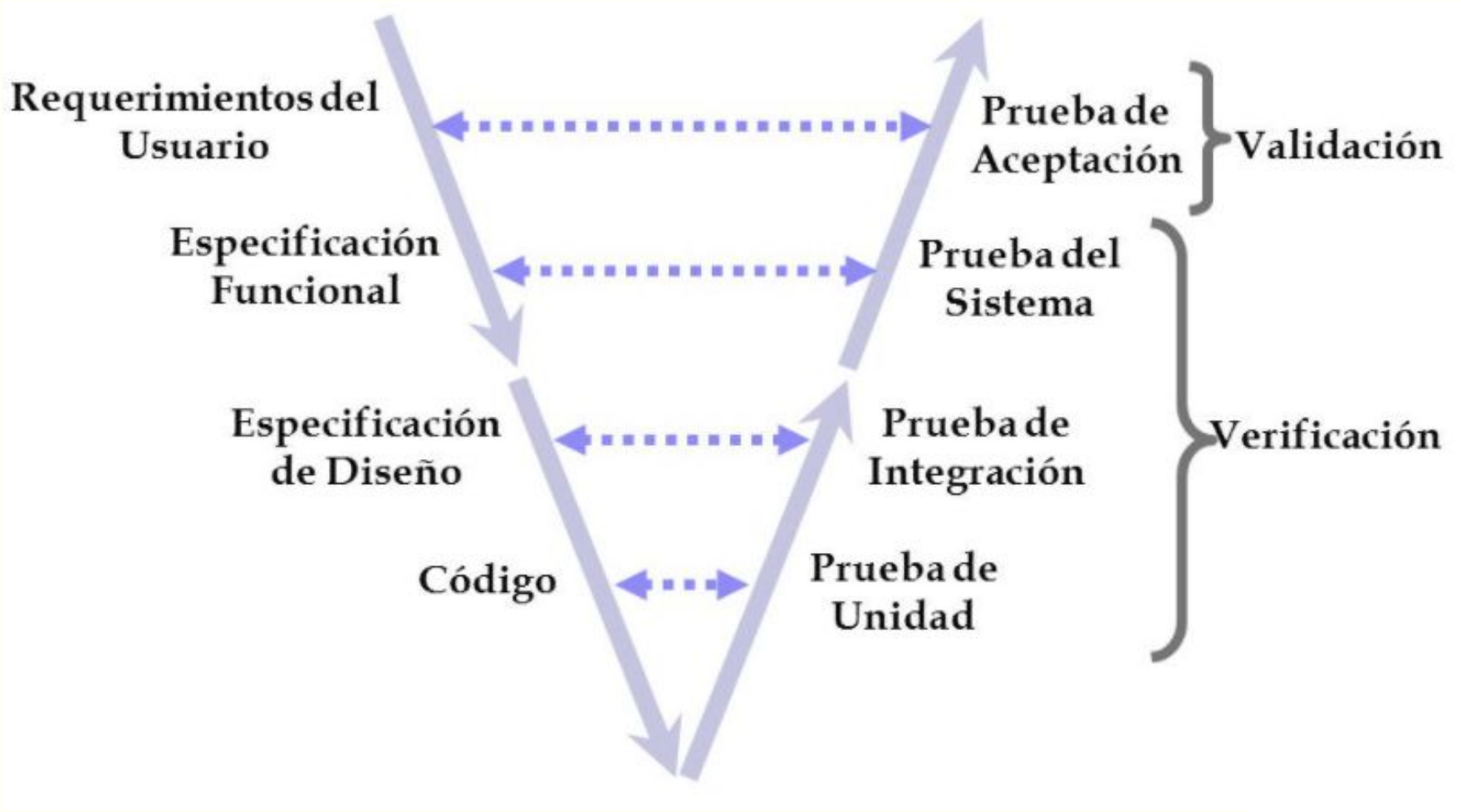
Inconvenientes

- Cualquier cambio de requisito supo fases ya superadas y provocará un incremento del coste.
- No se va mostrando al cliente el a desarrollando, si no que se ve el resultado una vez ha termin
- Los diseñadores pueden no s que se encontrarán cuando estén diseñando un softwa cto para solventar el problema.

¿Qué principio del *Testing* viola?

1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Modelo en V



1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Modelo en V

En los niveles lógicos del 1 al 4, para cada fase del desarrollo, existe una fase correspondiente o paralela de verificación o validación.

- El **nivel 1** está orientado al “cliente”. El inicio del proyecto y el fin del proyecto constituyen los dos extremos del ciclo. Se compone del **análisis de requisitos y especificaciones**, se traduce en un documento de requisitos y especificaciones.
- El **nivel 2** se dedica a las características funcionales del sistema propuesto. Puede considerarse el sistema como una caja negra, y caracterizarla únicamente con aquellas funciones que son directa o indirectamente visibles por el usuario final, se traduce en un documento de **análisis funcional**.
- El **nivel 3** define los componentes hardware y software del sistema final, a cuyo conjunto se denomina **arquitectura del sistema**.
- El **nivel 4** es la fase de **implementación**, en la que se desarrollan los elementos unitarios o módulos del programa.

1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Modelo en V - Verificación VS. Validación

"La **verificación** está más encaminada a comprobar los requisitos de cada una de las fases. La **validación** la debemos entender cómo una actividad de evaluar un resultado final, comprobar que cumple con lo inicialmente acordado y que el producto o servicio es aceptado por el usuario o no."

- La **verificación** normalmente incluye:
  - Revisiones y reuniones para evaluar documentos, planes, códigos, requisitos y especificaciones.
  - Los test de verificación pueden ser: pruebas de verificación, test unitarios, retrospectivas, regresiones.
  - La verificación es un proceso continuo que normalmente ocurre a lo largo del ciclo de desarrollo, y es importante habilitar muchas iteraciones, por lo que automatizar sería muy útil en procesos de verificación.
  - Normalmente quien las hace son el equipo técnico, y especialmente el departamento de calidad.

1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Modelo en V - Verificación VS. Validación

"La **verificación** está más encaminada a comprobar los requisitos de cada una de las fases. La **validación** la debemos entender cómo una actividad de evaluar un resultado final, comprobar que cumple con lo inicialmente acordado y que el producto o servicio es aceptado por el usuario o no."

- La **validación** por el contrario:
  - Requiere interacción con el mundo real.
  - Normalmente tiene lugar después de que se completan las verificaciones.
  - Los test que se ejecutan son los de UAT (*User Acceptance Test*) y son los *Product Owners* o en su defecto los Jefes de Proyectos quienes los ejecutan.
  - Los entornos son lo más parecidos a producción, como son los de *Staging*.

1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Modelo en V

Ventajas

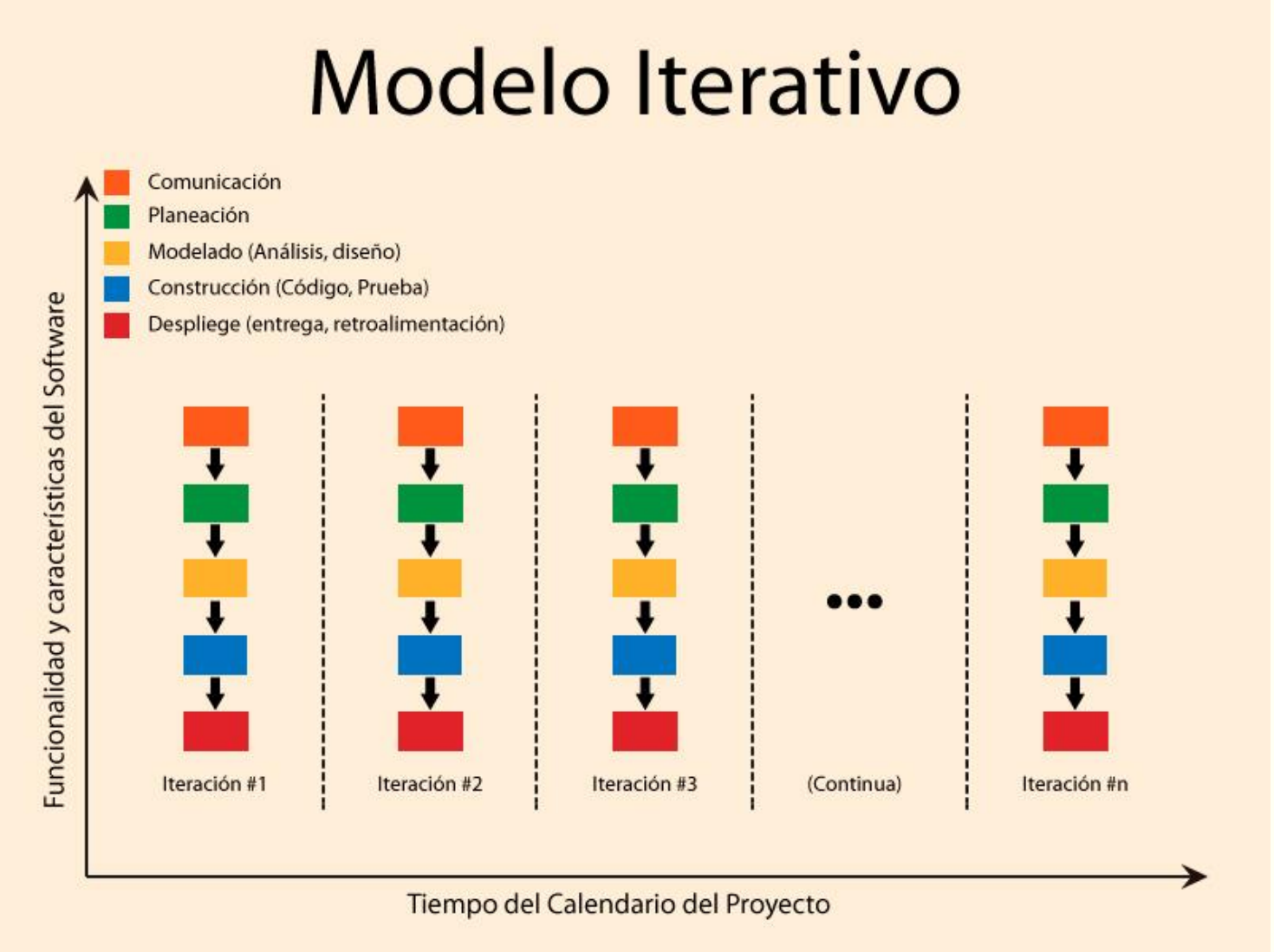
- Optimización de la comunicación entre las partes involucradas a través de términos y responsabilidades claramente definidos.
- Minimización de riesgos y mejor planificación a través de roles, estructuras y resultados fijos y predeterminados.
- Mejora de la calidad del producto gracias a medidas de control de la calidad firmemente integradas.
- Ahorro de costes gracias al procesamiento transparente a lo largo de todo el ciclo de vida del producto.

Inconvenientes

- Demasiado simple para mapear todo el proceso de desarrollo desde el punto de vista de los desarrolladores.
- Está sobre todo centrado en la **gestión de proyectos.**
- Su estructura relativamente rígida permite una respuesta **poco flexible** a los cambios durante el desarrollo, y, por lo tanto, promueve un curso lineal del proyecto.

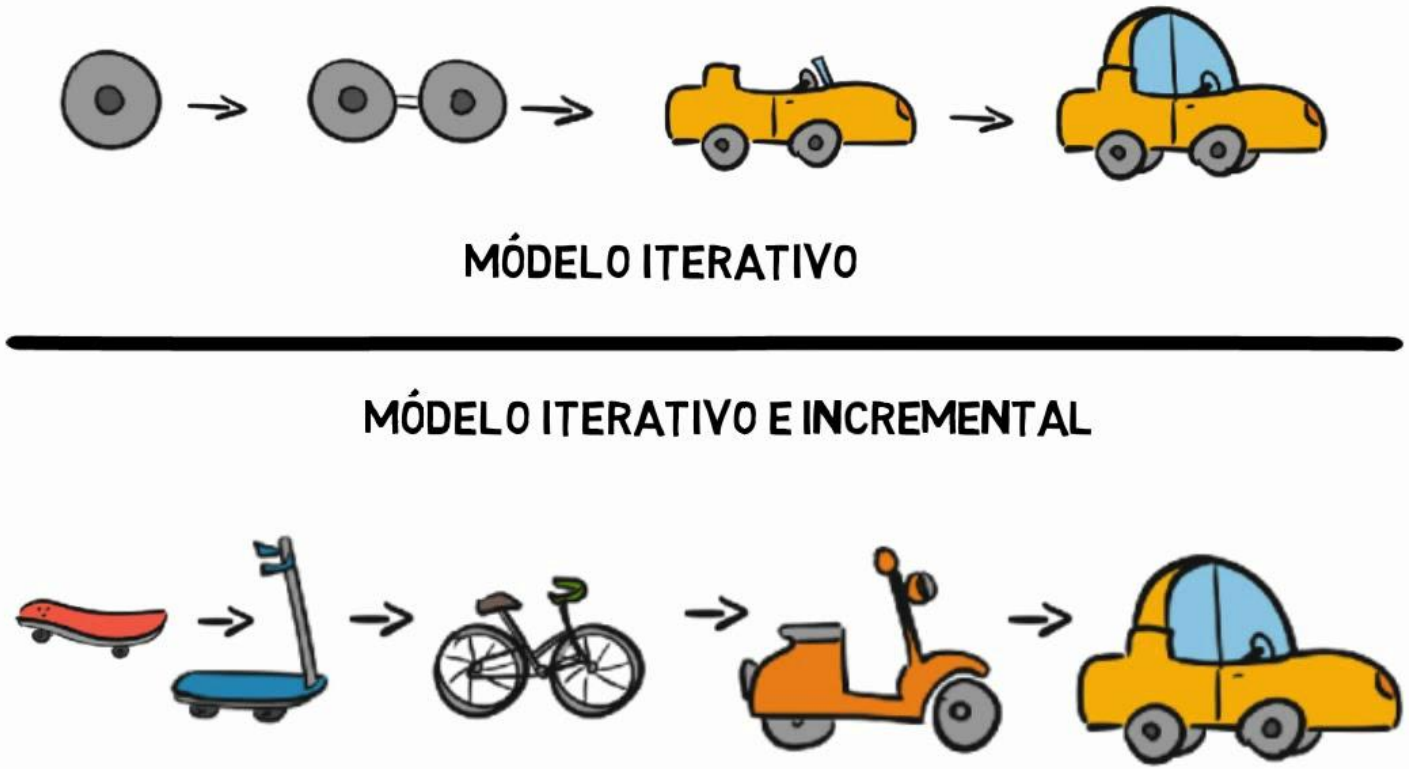
1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Modelo Iterativo



1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Modelo Iterativo e incremental



1.3 Metodologías y Modelos de desarrollo de Software y Calidad

Modelo iterativo e incremental

Ventajas

- Los usuarios no tienen que esperar hasta que el sistema completo se entregue para hacer uso de él. El primer incremento cumple los requerimientos más importantes de tal forma que pueden utilizar el software al instante.
- Los usuarios pueden utilizar los incrementos iniciales como prototipos y obtener experiencia sobre los requerimientos de los incrementos posteriores del sistema.
- Existe muy pocas probabilidades de riesgo en el sistema. Aunque se pueden encontrar problemas en algunos incrementos, lo normal es que el sistema se entregue sin inconvenientes al usuario.
- Se da la retroalimentación muy temprano a los usuarios.
- Permite separar la complejidad del proyecto, gracias a su desarrollo por parte de cada iteración o bloque.

1.3 Metodologías y Modelos de desarrollo de Software y Calidad

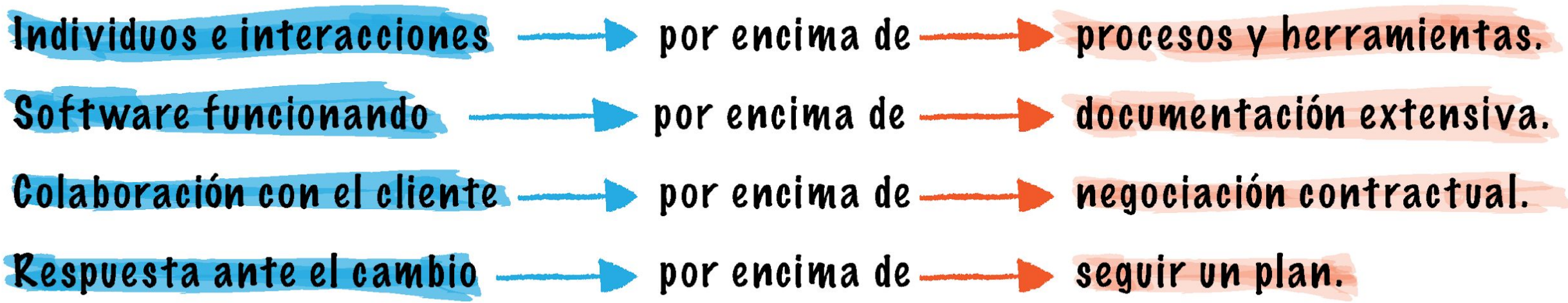
Modelo iterativo e incremental

Inconvenientes

- La entrega temprana de los proyectos produce la creación de sistemas demasiados simples.
- Los incrementos no deben constar de muchas líneas de código ya que la idea de los incrementos es agregar accesorios al programa principal (o funcional).
- Requiere de un cliente involucrado durante todo el curso del proyecto. Hay clientes que simplemente no estarán dispuestos a invertir el tiempo necesario.
- El trato con el cliente debe basarse en principios éticos y colaboración mutua, más que trabajar cada parte independientemente, defendiendo sólo su propio beneficio.

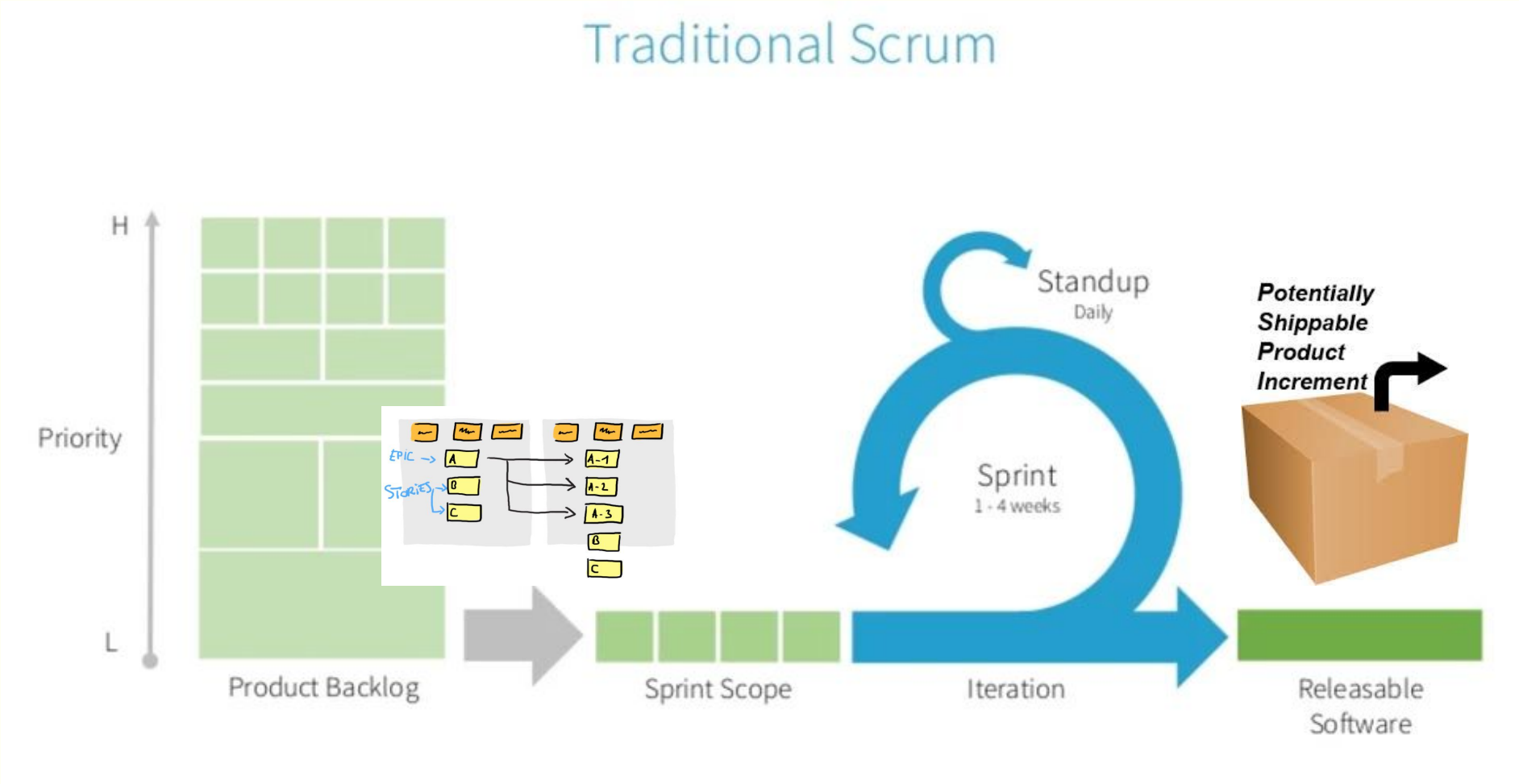
1.4 Metodologías Ágiles

Manifiesto Ágil



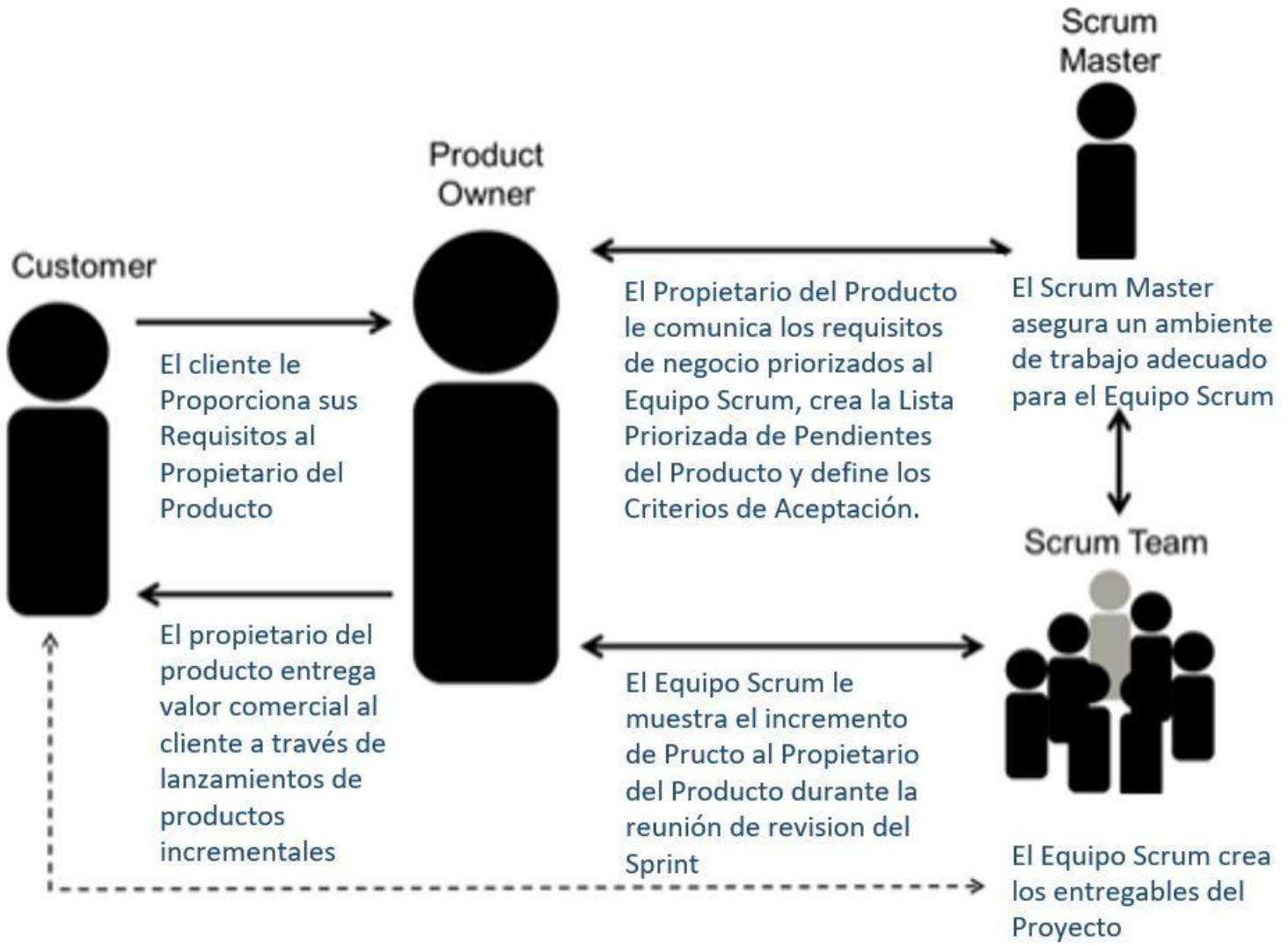
1.4 Metodologías Ágiles

Scrum Iteration



1.4 Metodologías Ágiles

Scrum Roles

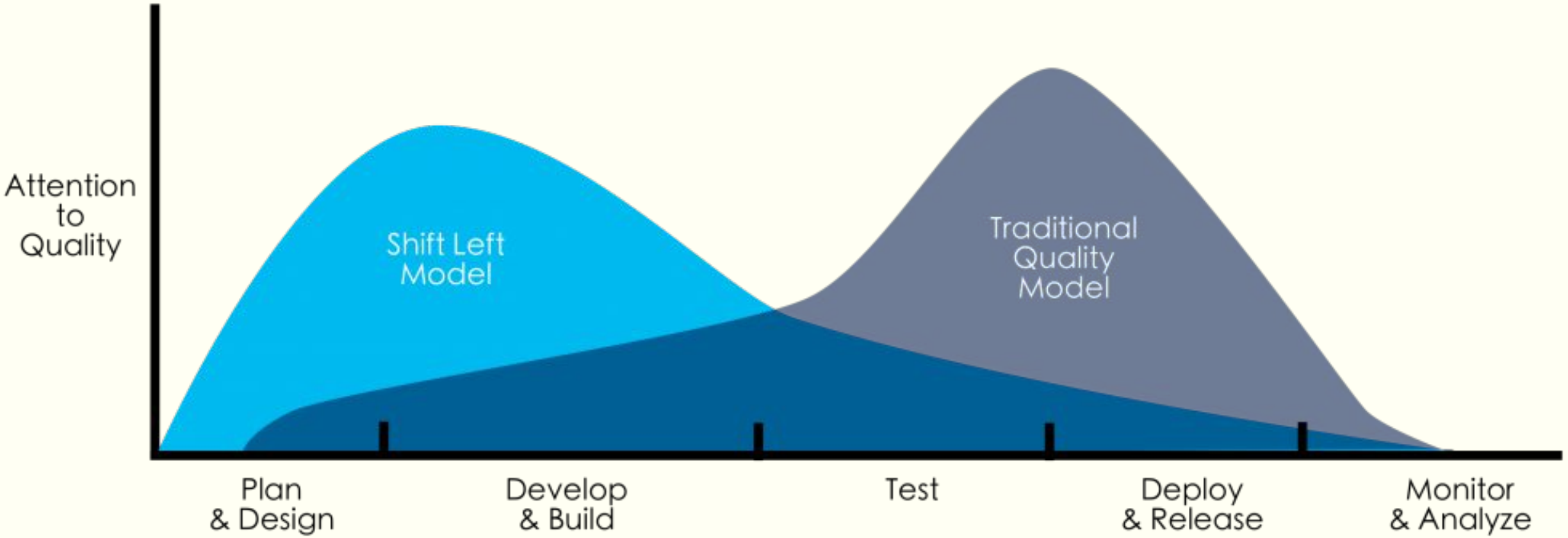


1.4 Metodologías Ágiles

Testing Ágil

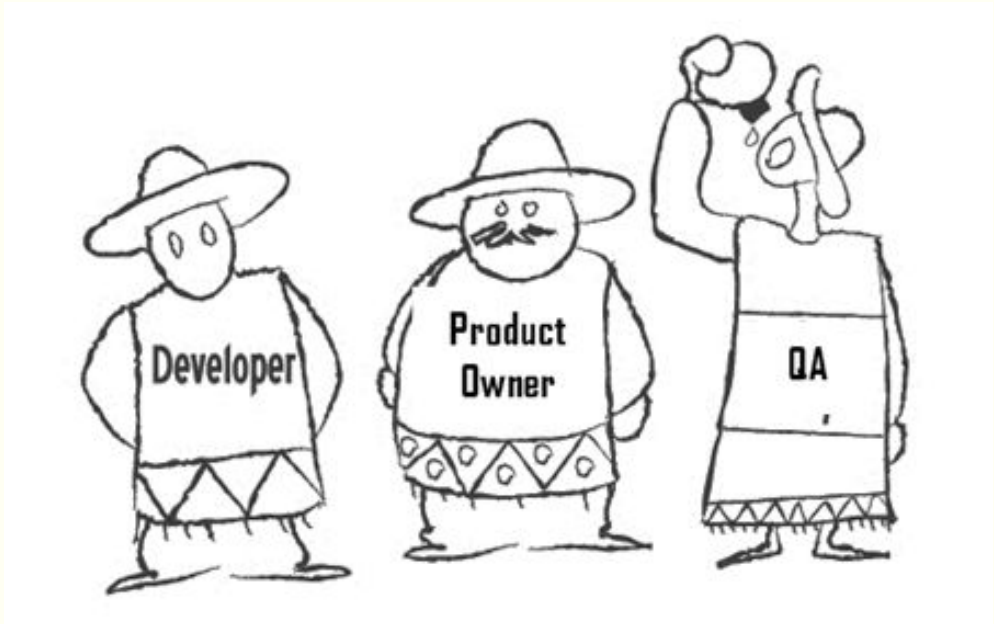
1.4 Metodologías Ágiles

Testing Ágil – *Shift Left Testing*



1.4 Metodologías Ágiles

Testing Ágil – Los 3 Amigos



Los tres amigos se refieren a las tres perspectivas principales al examinar el incremento de producto antes, durante y después del desarrollo. Estas perspectivas son:

- **Línea de negocio – ¿Qué problema intentamos resolver?**
- **Desarrollo – ¿Cómo vamos a crear la solución que resuelva ese problema?**
- **Testing – Si hacemos ésto, ¿Qué es lo que pueda ocurrir?**

Las personas responsables de tener estas perspectivas deben de colaborar en definir las acciones a realizar, y acordar cuando es que estas acciones estarán hechas correctamente. El resultado final de esta colaboración es una descripción clara del incremento de producto, generalmente en forma de ejemplos, y termina siendo un entendimiento compartido con el resto del equipo.

1.4 Metodologías Ágiles

Testing Ágil

- **El *Testing* no es una fase:** El testing continuo es la única forma de garantizar avance continuo, por esto, el testing se realiza continuamente junto con el desarrollo de software y demás actividades.
- **El *Testing* hace avanzar el proyecto:** Bajo métodos convencionales, el *testing* es un cuello de botella, en cambio en *Agile Testing* se proporciona retroalimentación continua, permitiendo corregir el rumbo continuamente durante el desarrollo de software.
- **Todo el equipo realiza pruebas:** en *Agile Testing*, los Analistas de negocio y Desarrolladores de *software* también ejecutan pruebas, no sólo los *testers* como en métodos convencionales.
- **Reducir el tiempo para recibir retroalimentación:** En *Agile Testing*, los equipos del área de negocio (el cliente) están involucrados en cada iteración, no solo al final durante la fase de aceptación, como resultado, el tiempo de retroalimentación se reduce y el costo de correcciones también es menor.
- **Código limpio:** Los defectos en el código se corrigen en la misma iteración, por lo que se mantiene el código limpio.

1.4 Metodologías Ágiles

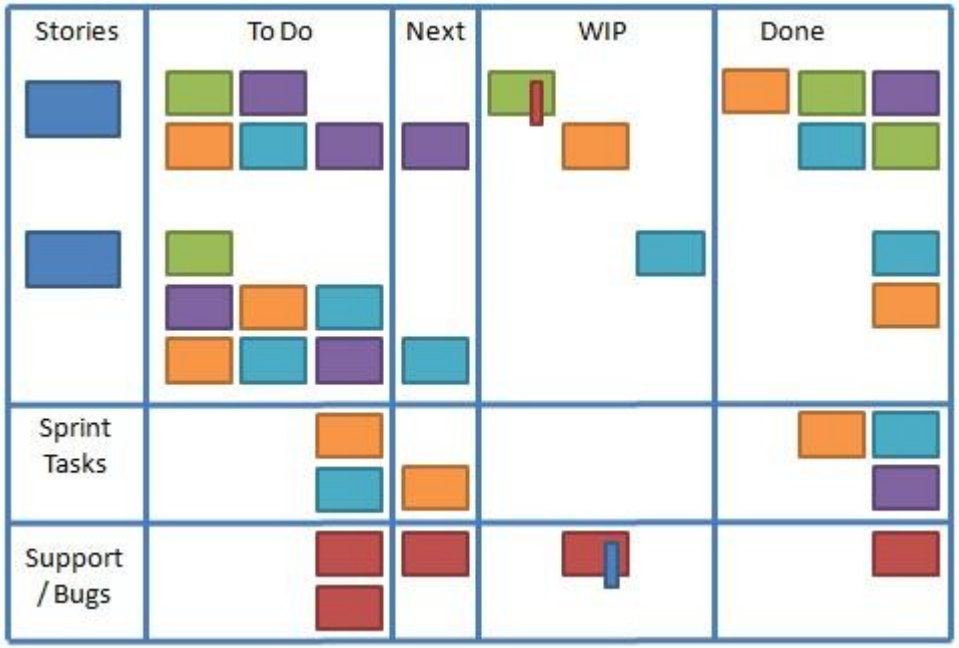
Testing Ágil

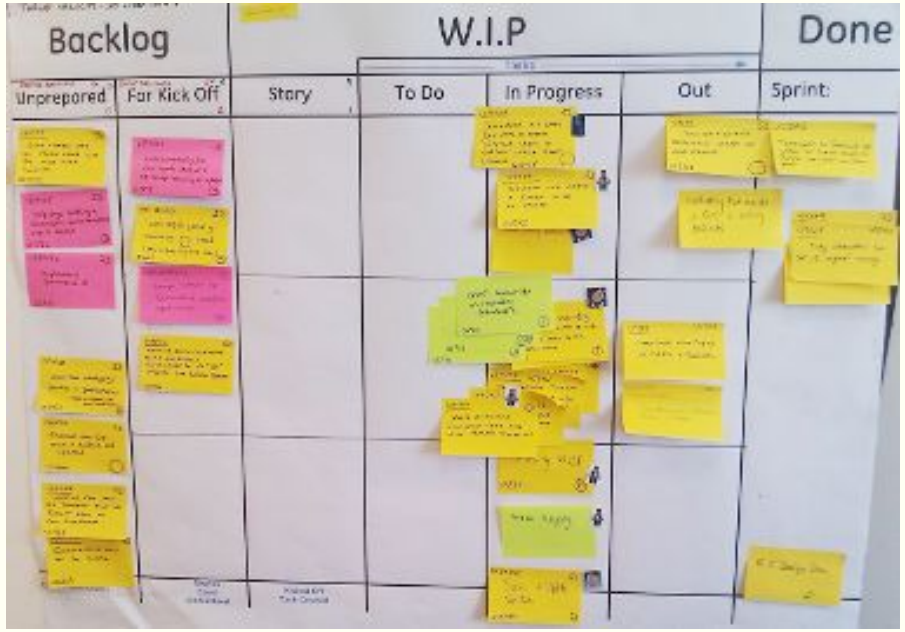
- **Reducir la documentación de pruebas:** Los *Agile Testers* usan listas de chequeo reusables en lugar de documentación extensa, se enfocan en la esencia de la prueba en lugar de detalles. Siguiendo principios ágiles estas listas de chequeo son el inicio de las definiciones de las pruebas y no el final, y el *tester* cuenta con libertad para aportar valor.

- **Guiado por pruebas:** El *Agile Testing*, las pruebas se hacen “durante” el desarrollo y no después del desarrollo como en métodos convencionales: algunas p´rácticas son:
  - **Test Driven Development (TDD)**
  - **Acceptance Test Driven Development (ATDD)**
  - **Behaviour Driven Development (BDD)**
  - **Testing Exploratorio**
  - **Automatización de pruebas de regresión**
  - **Automatización de pruebas unitarias**

1.4 Metodologías Ágiles

Visual Management





Fuentes

- **https://ingenieriadesoftware.es/grandes-errores-historia-software-informatico/**
- https://medium.com/@rodrosalazar/el-rol-de-qa-658795b64691
- https://www.softwaretestinghelp.com/why-does-software-have-bugs/
- https://www.javiergarzas.com/2019/06/las-curvas-del-coste-del-cambio.html
- https://www.paradigmadigital.com/dev/tester-vs-quality-assurance/
- https://es.wikipedia.org/wiki/Calidad_de_software
- https://www.rafablanes.com/blog/elpdpa-que-es-la-calidad-del-software
- https://www.testingcolombia.com/principios-del-testing-segun-istqb-international-software-testing-qualifications-board/
- https://www.sistel.es/equipos-desarrollo-software
- http://www.pmoinformatica.com/2015/03/que-es-el-agile-testing.html