Managing the Development of Large Software Systems

Pipe & Filter

Martín Fixman[1] Ignacio Harari[1]
Damián Alemán[1] Martín Arjovsky[1]

[1]Facultad de Ciencias Exactas y Naturales

Primer Cuatrimestre 2016

Introducción

Esta presentación muestra algunas observaciones sobre la administración de proyectos presentadas en el paper del Dr. Winston W. Royce[1].

En dicho paper, se presenta un proceso para mejorar los pasos a seguir durante la administración de un proyecto grande para prevenir errores y lograr bajar los costos de corregirlos cuando ocurren.

Winston W. Royce (1929 – 7 de junio de 1995)

- Fue un computólogo norteamericano, director en el Centro de Tecnología de Software Lockheed en Austin, Texas.
- Fue el primero en describir “modelo en cascada” para el desarrollo de software, aunque:
  - Royce no utilizo el termino “cascada” el paper
  - Ni lo propuso como metodología de trabajo.

Hay dos pasos esenciales que son comunes entre todos los desarrollos de programas de computadora: un paso de análisis, y uno de código. Si el programa es lo suficientemente pequeño y el producto final va a ser operado por los que lo construyeron (como suele suceder con los programas de uso interno) puede ser que esto sea todo lo requerido.

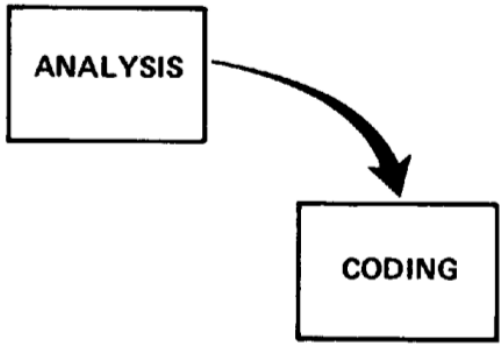
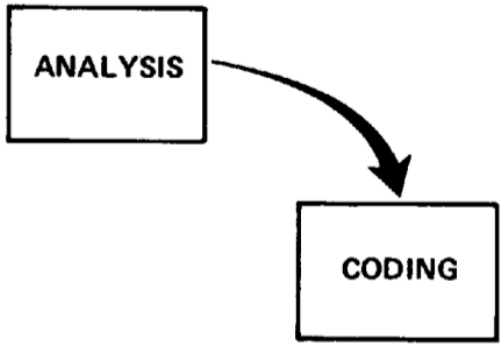


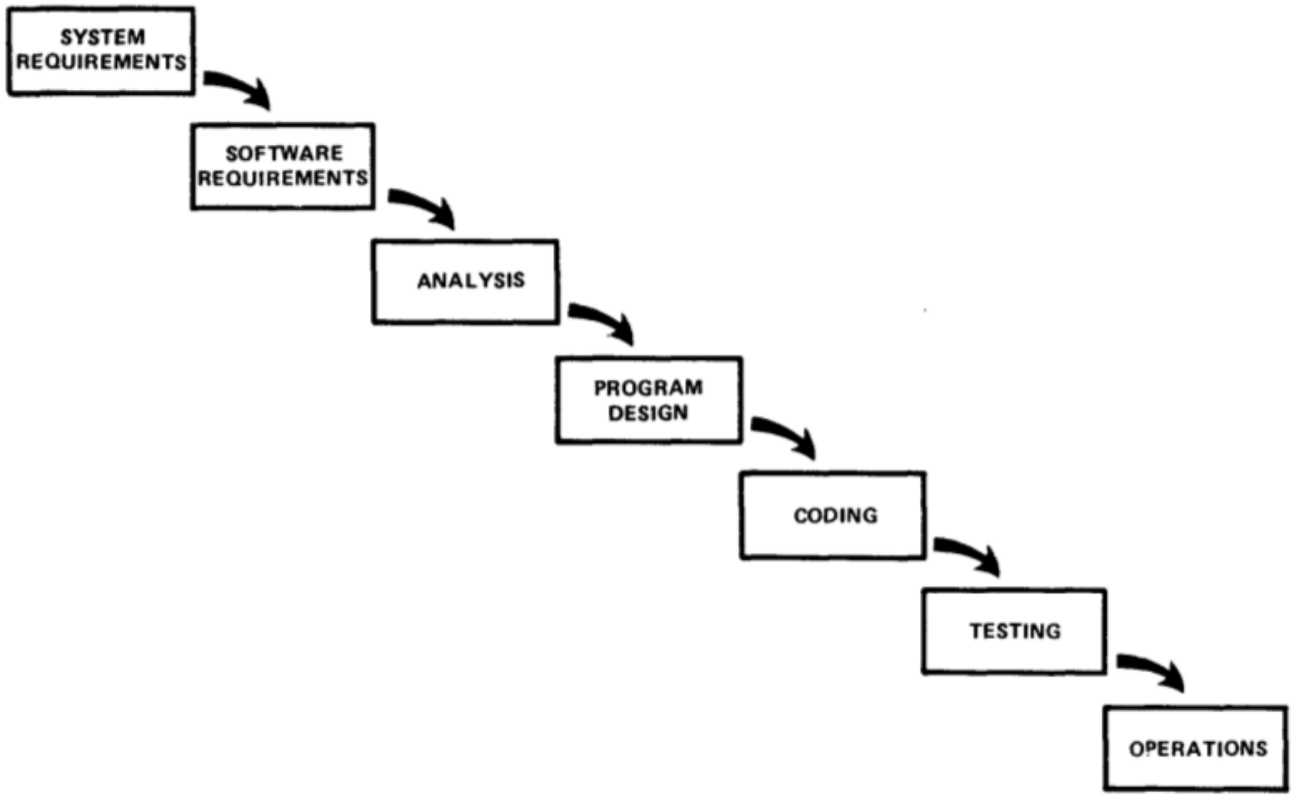
Figura 1: Implementación de un programa pequeño de operaciones internas

Hay dos pasos esenciales que son comunes entre todos los desarrollos de programas de computadora: un paso de análisis, y uno de código. Si el programa es lo suficientemente pequeño y el producto final va a ser operado por los que lo construyeron (como suele suceder con los programas de uso interno) puede ser que esto sea todo lo requerido.

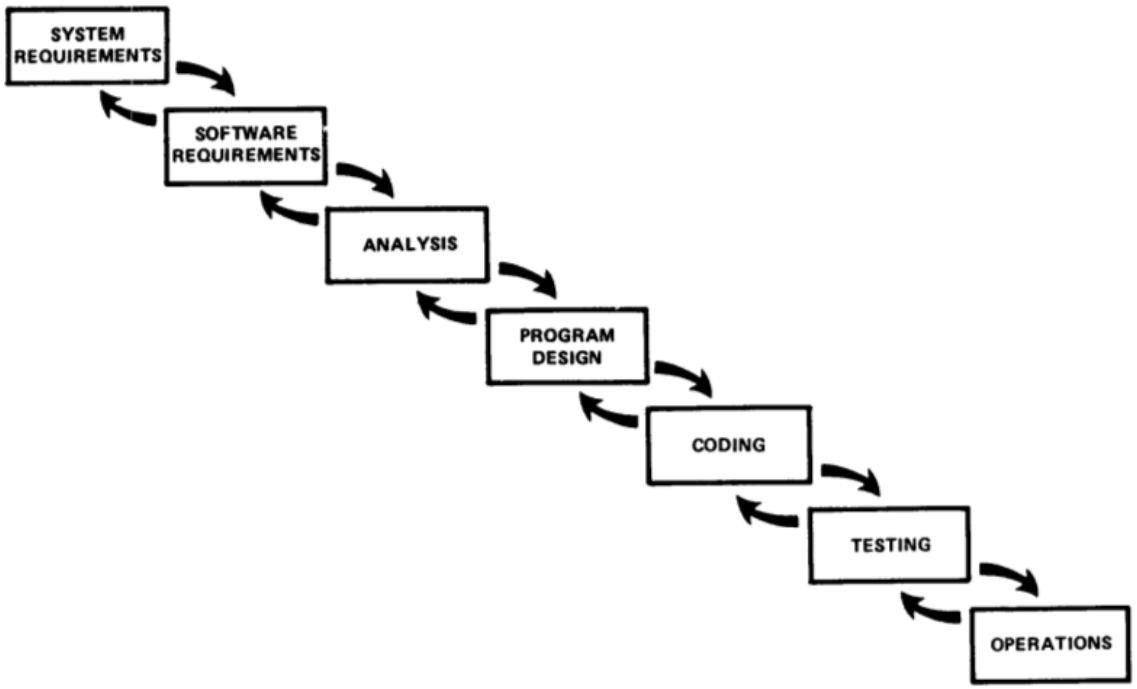


Un plan para hacer sistemas más grandes que se base en estos pasos está condenado al fracaso, ya que en dichos sistemas se requiere un desarrollo de software más detallado.

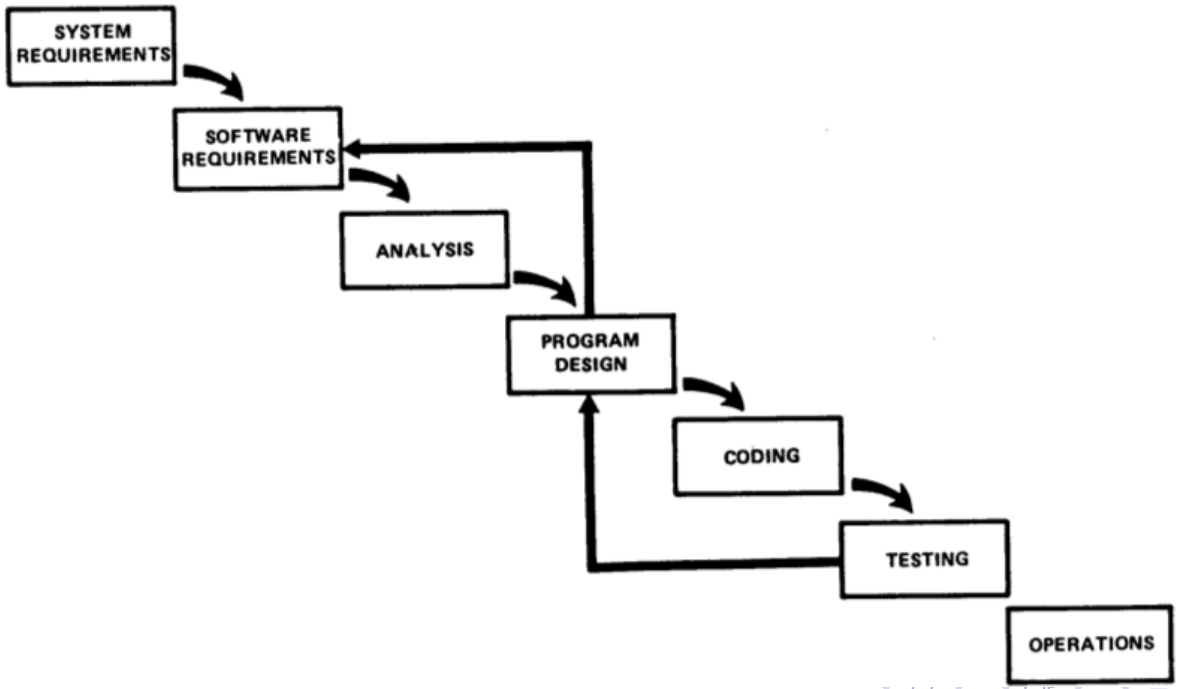
Un mejor enfoque para proyectos grandes es usar el famoso “diseño en cascada”. En este diseño los pasos de análisis y código están precedidos por dos niveles de análisis de requerimientos, separados por un paso de diseño y seguidos por un paso de testeo.



Aunque este concepto es bastante bueno, la implementación es riesgosa y suele fallar.
Idealmente, en cada paso el diseño está más detallado, y si hay un problema en alguna iteración se puede resolver en la iteración anterior.



Aunque este concepto es bastante bueno, la implementación es riesgosa y suele fallar.
En el mundo real suele haber errores de diseño que no se pueden analizar precisamente hasta la fase de testing y los cambios al diseño pueden violar los requerimientos de software y deben ser modificados.



Sin embargo, este método sigue siendo fundamentalmente bueno. Para asegurarse que no ocurran problemas como el anterior, se pueden seguir 5 pasos que eliminan gran parte de los riesgos del desarrollo.

El paper presenta 5 pasos de diseño que puede seguir el desarrollo de software para prevenir problemas como el anterior.

El paper presenta 5 pasos de diseño que puede seguir el desarrollo de software para prevenir problemas como el anterior.

1. **Program Design Comes First** (Hacer el diseño primero)

El paper presenta 5 pasos de diseño que puede seguir el desarrollo de software para prevenir problemas como el anterior.

1. **Program Design Comes First** (Hacer el diseño primero)
2. **Document the Design** (Documentar el diseño)

El paper presenta 5 pasos de diseño que puede seguir el desarrollo de software para prevenir problemas como el anterior.

1. **Program Design Comes First** (Hacer el diseño primero)
2. **Document the Design** (Documentar el diseño)
3. **Do It Twice** (Desarrollar dos veces)

El paper presenta 5 pasos de diseño que puede seguir el desarrollo de software para prevenir problemas como el anterior.

1. **Program Design Comes First** (Hacer el diseño primero)
2. **Document the Design** (Documentar el diseño)
3. **Do It Twice** (Desarrollar dos veces)
4. **Plan, Control and Monitor Testing** (Planear, controlar y monitorear el testeo)

El paper presenta 5 pasos de diseño que puede seguir el desarrollo de software para prevenir problemas como el anterior.

1. **Program Design Comes First** (Hacer el diseño primero)
2. **Document the Design** (Documentar el diseño)
3. **Do It Twice** (Desarrollar dos veces)
4. **Plan, Control and Monitor Testing** (Planear, controlar y monitorear el testeo)
5. **Involve the Customer** (Involucrar al cliente)

Program Design Comes First

Se agrega una fase de diseño preliminar, entre la extracción de requerimientos y la fase de análisis.

- Empezar el diseño con los diseñadores (ni analistas ni programadores)
- Diseñar y definir los modelos de procesamiento de datos.
- Escribir un resumen entendible, informativo y actualizado.

Program Design Comes First

Se agrega una fase de diseño preliminar, entre la extracción de requerimientos y la fase de análisis.

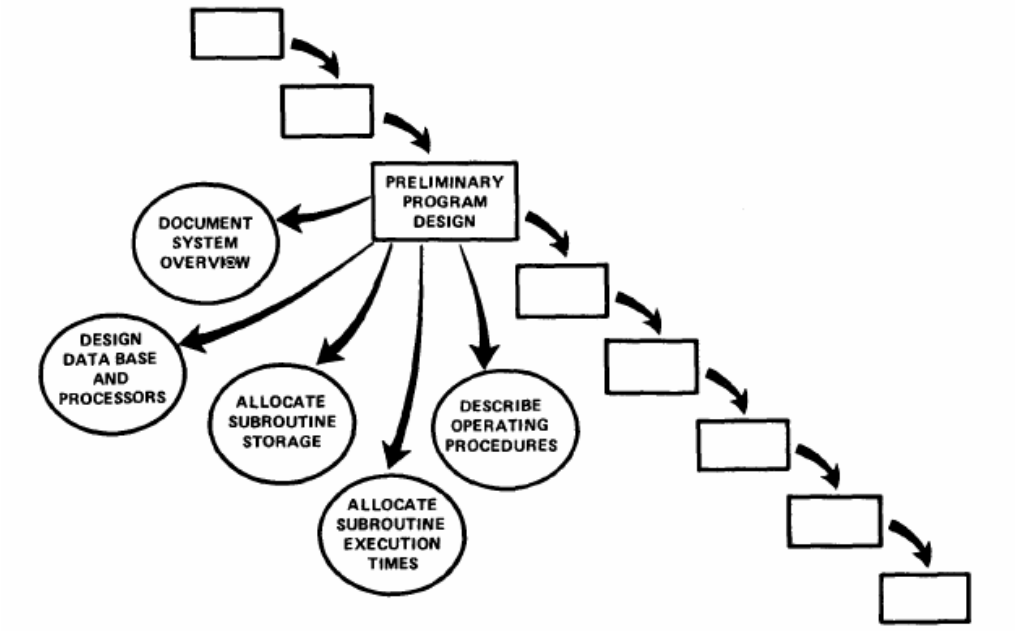


Figura 2: Diagrama con diseño preliminar

Document The Design

- Cuanta documentación: MUCHA
- Más de lo que harían la mayoría de los programadores, analistas, o diseñadores por su cuenta
- La primer regla del manejo del desarrollo de software es la orden estricta de documetar los requerimientos.

Do It Twice

Para detectar y atacar los principales riesgos tempranamente, es beneficioso implementar una corta iteración de las fases de diseño, análisis y desarrollo obviando los detalles menos complejos.

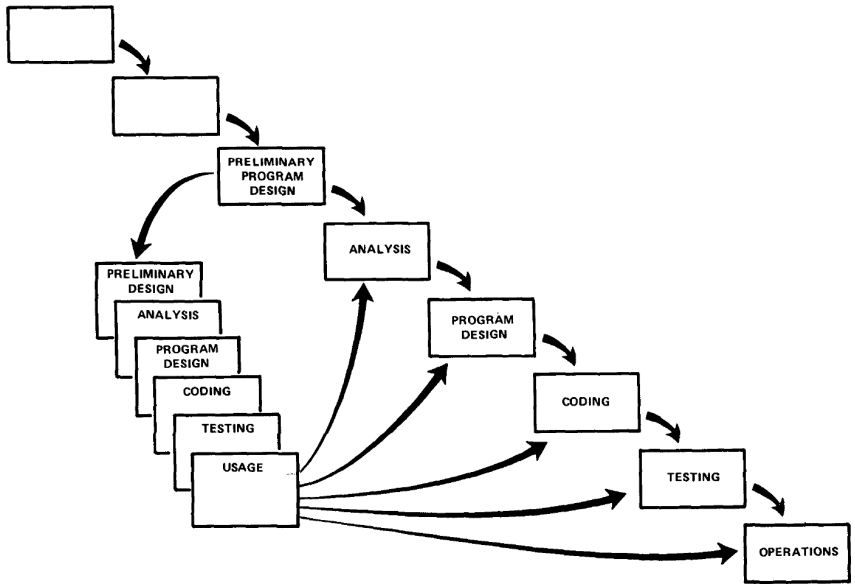


Figura 3: Nuevo mini-desarrollo en el desarrollo

Do It Twice

Esto nos permite testear las hipótesis clave, para acotar el error humano en las estimaciones iniciales, que suelen ser invariablemente optimistas.

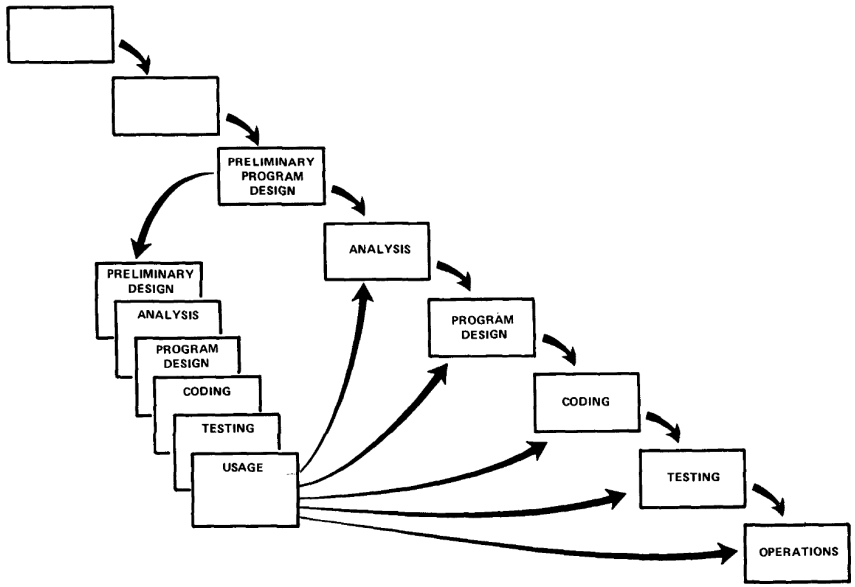


Figura 3: Nuevo mini-desarrollo en el desarrollo

Plan, Control, and Monitor Testing

En cualquier proyecto de software, el testing es la parte más larga, pesada y donde los errores suelen ser más catastróficos. Para que esta fase sea leve, se puede empezar a planear durante la fase de diseño de programa así el código ya se crea teniendo en cuenta los tests.
También se deberían seguir 4 heurísticas de testeo.

1. Hay partes del testing que se hacen idealmente por especialistas que no contribuyen al diseño original. Por lo tanto, se necesita buena documentación en el diseño para que el testeados pueda colaborar independientemente del resto del equipo.

Plan, Control, and Monitor Testing

En cualquier proyecto de software, el testing es la parte más larga, pesada y donde los errores suelen ser más catastróficos. Para que esta fase sea leve, se puede empezar a planear durante la fase de diseño de programa así el código ya se crea teniendo en cuenta los tests.
También se deberían seguir 4 heurísticas de testeo.

2. La mayoría de los errores son obvios y pueden ser encontrados a ojo. Todo el análisis y el código debería ser visto por una persona que no esté trabajando en este.

Plan, Control, and Monitor Testing

En cualquier proyecto de software, el testing es la parte más larga, pesada y donde los errores suelen ser más catastróficos. Para que esta fase sea leve, se puede empezar a planear durante la fase de diseño de programa así el código ya se crea teniendo en cuenta los tests.
También se deberían seguir 4 heurísticas de testeo.

2. La mayoría de los errores son obvios y pueden ser encontrados a ojo. Todo el análisis y el código debería ser visto por una persona que no esté trabajando en este.[1]

[1] El paper también recomienda **no** hacer análisis estático por software, ya que este es muy caro en los recursos del software. Aunque esto era verdad en el 1970, en el año 2016 suele ser una buena idea usarlo para el testeo.

Plan, Control, and Monitor Testing

En cualquier proyecto de software, el testing es la parte más larga, pesada y donde los errores suelen ser más catastróficos. Para que esta fase sea leve, se puede empezar a planear durante la fase de diseño de programa así el código ya se crea teniendo en cuenta los tests.
También se deberían seguir 4 heurísticas de testeo.

3. Testear todo camino lógico del código con algún test numérico, aún si el programa es tan largo y complejo que esto es difícil.

Plan, Control, and Monitor Testing

En cualquier proyecto de software, el testing es la parte más larga, pesada y donde los errores suelen ser más catastróficos. Para que esta fase sea leve, se puede empezar a planear durante la fase de diseño de programa así el código ya se crea teniendo en cuenta los tests.
También se deberían seguir 4 heurísticas de testeo.

4. Una vez que se resuelven todos los errores "obvios", se deben hacer tests finales de temas más complicados por personas que se dediquen particularmente a esto. La administración se debe ocupar de quién y cuándo hace el último checkout.

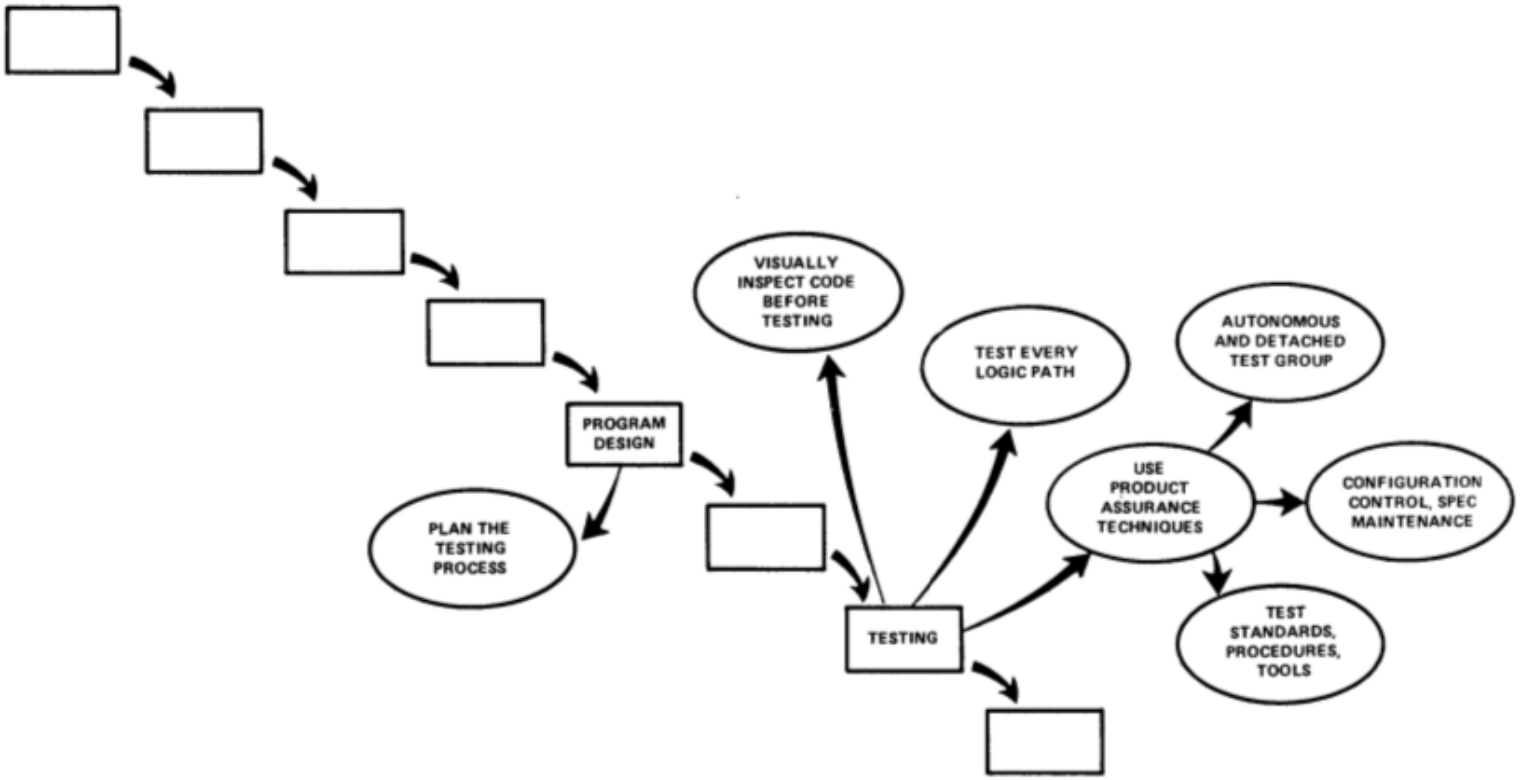


Figura 4: Testing bien hecho

Involve the Customer

No es recomendable limitar la interacción con el cliente a las fases de requerimientos y operación, ya que la interpretación de las funcionalidades suele estar tintada de subjetividades.

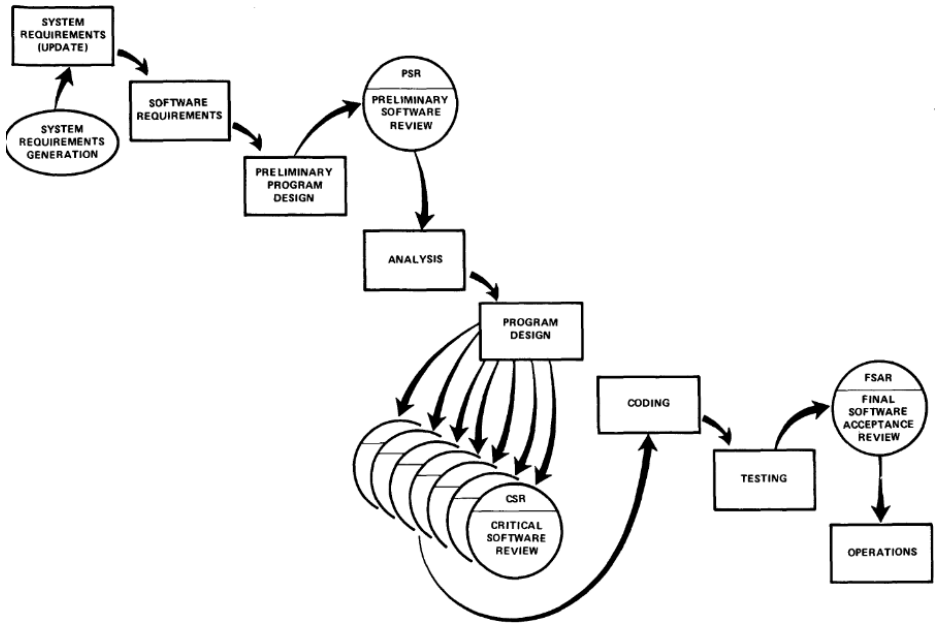


Figura 5: Dónde involucrar al cliente

Involve the Customer

En cambio es mucho más beneficioso involucrarlo en toda etapa en la que su juicio, su visión de la aplicación y su compromiso con lo decidido alimente el proceso de desarrollo.

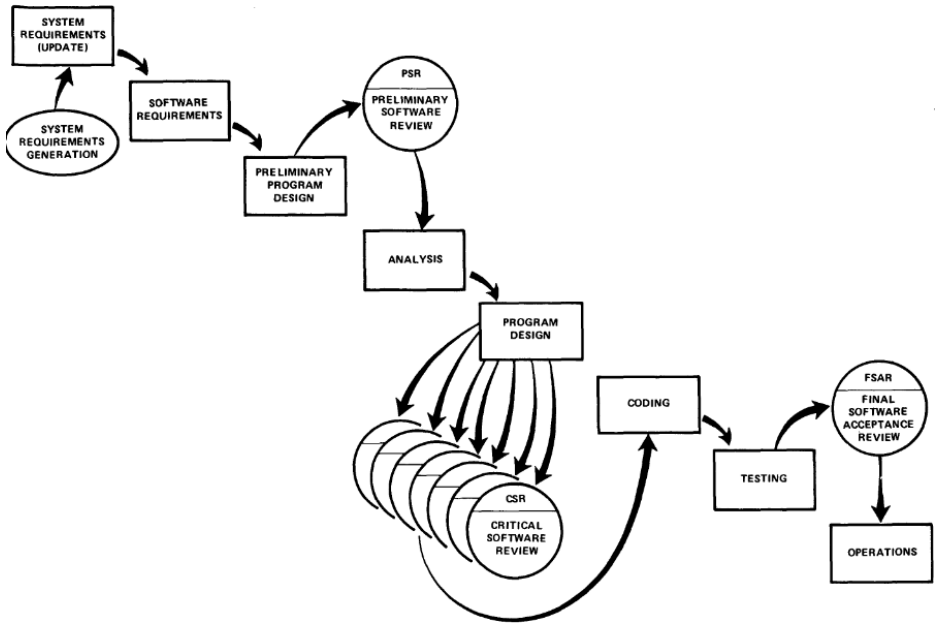


Figura 5: Dónde involucrar al cliente

Conclusiones

El principal objetivo de las pautas y procesos expuestos es conseguir un marco de trabajo en donde los errores eventuales y los riesgos puedan ser previstos y recuperados en la etapa más temprana posible, y con la menor cantidad de trabajo perdido.

En los desarrollos de software de gran escala es casi imposible prevenir el surgimiento de errores de alguna índole, o el cambio de requerimientos. Por lo tanto tener un buen trabajo de fondo sobre el cual poder recaer (por ejemplo, la documentación) en esos casos permite recuperarse con un costo de recursos y dinero mucho menor que la versión simplificada del proceso que sólo contempla el análisis y el desarrollo del código.

Dr. Winston W. Royce
Managing the Development of Large Software Systems
IEEE Wescon, August 1970.
Copyright ®1970 The Institute of Electrical and Electronics Engineers
https://cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf