GESTIÓN DEL DESARROLLO DE SISTEMAS DE SOFTWARE DE GRAN TAMAÑO

Dr. Winston W. Royce

INTRODUCCIÓN

Voy a describir mi punto de vista personal sobre la gestión del desarrollo de sistemas de software de gran tamaño. En los últimos nueve años he desempeñado diversas tareas, la mayoría relacionadas con el desarrollo de paquetes de software para la planificación de misiones espaciales, el mando y el análisis posterior al vuelo. En estas tareas he experimentado diferentes grados de éxito en lo que respecta a llegar a un estado operativo, a tiempo y dentro de los costes. Mis experiencias me han creado prejuicios y voy a relatar algunos de ellos en esta presentación.

FUNCIONES DE DESARROLLO DE PROGRAMAS INFORMÁTICOS

Hay dos pasos esenciales comunes para todos los desarrollos de programas informáticos, sin importar su tamaño o complejidad. En primer lugar, hay un paso de análisis, seguido de un paso de codificación, como se muestra en la Figura 1. Este tipo de concepto de implementación muy simple es, de hecho, todo lo que se requiere si el esfuerzo es lo suficientemente pequeño y si el producto final va a ser manejado por quienes lo construyeron, como suele hacerse con los programas informáticos para uso interno. También es el tipo de esfuerzo de desarrollo por el que la mayoría de los clientes están dispuestos a pagar, ya que ambos pasos implican un trabajo genuinamente creativo que contribuye directamente a la utilidad del producto final. Sin embargo, un plan de implementación para fabricar sistemas de software más grandes, y centrado únicamente en estos pasos, está condenado al fracaso. Se requieren muchos pasos de desarrollo adicionales, ninguno contribuye tan directamente al producto final como el análisis y la codificación, y todos encarecen los costes de desarrollo. El personal del cliente suele preferir no pagarlos y el personal de desarrollo prefiere no ponerlos en práctica. La función primordial de la dirección es vender estos conceptos a ambos grupos y luego imponer su cumplimiento por parte del personal de desarrollo.

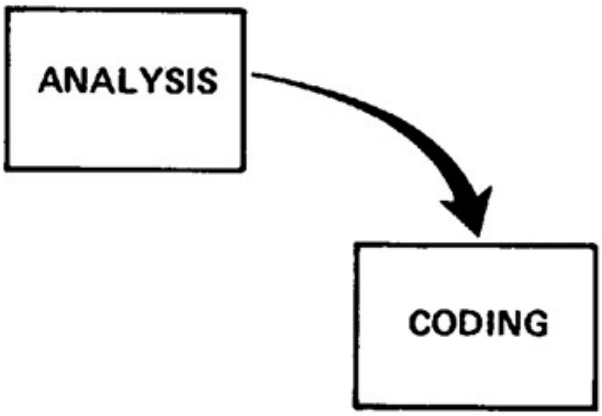

Figura 1. Etapas para desarrollar un programa informático simple para operaciones internas.

En la Figura 2 se ilustra un enfoque más grandioso del desarrollo de software. Las fases de análisis y codificación siguen presentes, pero van precedidas de dos niveles de análisis de requisitos, separadas por una fase de diseño del programa y seguidas de una fase de pruebas. Estas adiciones se tratan por separado del análisis y la codificación porque son claramente diferentes en la forma en que se ejecutan. Deben planificarse y dotarse de personal de forma diferente para aprovechar al máximo los recursos del programa.

La figura 3 muestra la relación iterativa entre las sucesivas fases de desarrollo de este esquema. El orden de los pasos se basa en el siguiente concepto: a medida que avanza cada paso y se detalla más el diseño, se produce una iteración con los pasos precedentes y sucesivos, pero rara vez con los pasos más alejados en la secuencia. La virtud de todo esto es que, a medida que avanza el diseño, el proceso de cambio se reduce a límites manejables. En cualquier punto del proceso de diseño, una vez completado el análisis de requisitos, existe una línea de base firme y cercana a la que volver en caso de dificultades imprevistas en el diseño. Lo que tenemos es una posición de repliegue eficaz que tiende a maximizar el alcance del trabajo inicial que se puede salvar y preservar.

Reprinted from *Proceedings, IEEE WESCON,* August 1970, pages 1-9.

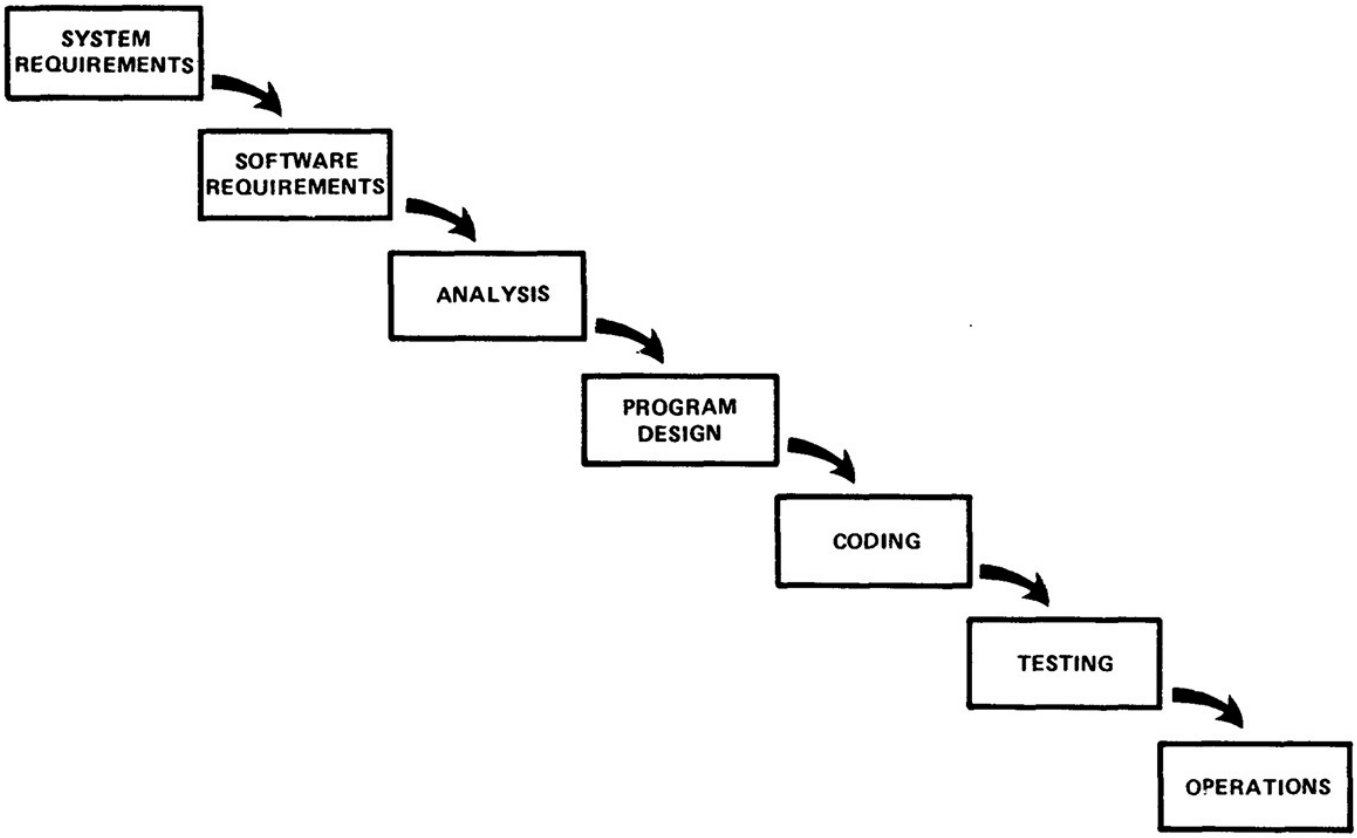

Figura 2. Etapas para desarrollar un programa informático de gran tamaño.

Creo en este concepto, pero la implementación descrita anteriormente es arriesgada y propicia el fracaso. El problema se ilustra en la Figura 4. La fase de prueba, que ocurre al final del ciclo de desarrollo, es el primer evento en el que la sincronización, el almacenamiento, las transferencias de entrada/salida, etc., se experimentan, en lugar de analizarse. Estos fenómenos no son analizables con precisión. No son las soluciones a las ecuaciones diferenciales parciales estándar de la física matemática, por ejemplo. Sin embargo, si estos fenómenos no satisfacen las diversas restricciones externas, invariablemente se requiere un rediseño importante. Un simple parche octal o la repetición de algún código aislado no solucionará este tipo de dificultades. Es probable que los cambios de diseño requeridos sean tan disruptivos que se incumplan los requisitos de software en los que se basa el diseño y que justifican todo. O bien deben modificarse los requisitos, o bien se requiere un cambio sustancial en el diseño. En efecto, el proceso de desarrollo ha vuelto al origen y se puede esperar un sobrecosto de hasta el 100 % en plazos o costes.

Cabe destacar que se han omitido las fases de análisis y código. Por supuesto, no se puede producir software sin estos pasos, pero generalmente estas fases se gestionan con relativa facilidad y tienen poco impacto en los requisitos, el diseño y las pruebas. En mi experiencia, hay departamentos enteros dedicados al análisis de la mecánica orbital, la determinación de la actitud de la nave espacial, la optimización matemática de la actividad de la carga útil, etc., pero cuando estos departamentos han completado su difícil y complejo trabajo, los pasos resultantes del programa implican unas pocas líneas de código aritmético serial. Si en la ejecución de su difícil y complejo trabajo los analistas han cometido un error, la corrección se implementa invariablemente mediante un pequeño cambio en el código sin retroalimentación disruptiva en las demás bases de desarrollo.

Sin embargo, creo que el enfoque ilustrado es fundamentalmente sólido. El resto de esta discusión presenta cinco características adicionales que deben añadirse a este enfoque básico para eliminar la mayoría de los riesgos del desarrollo.

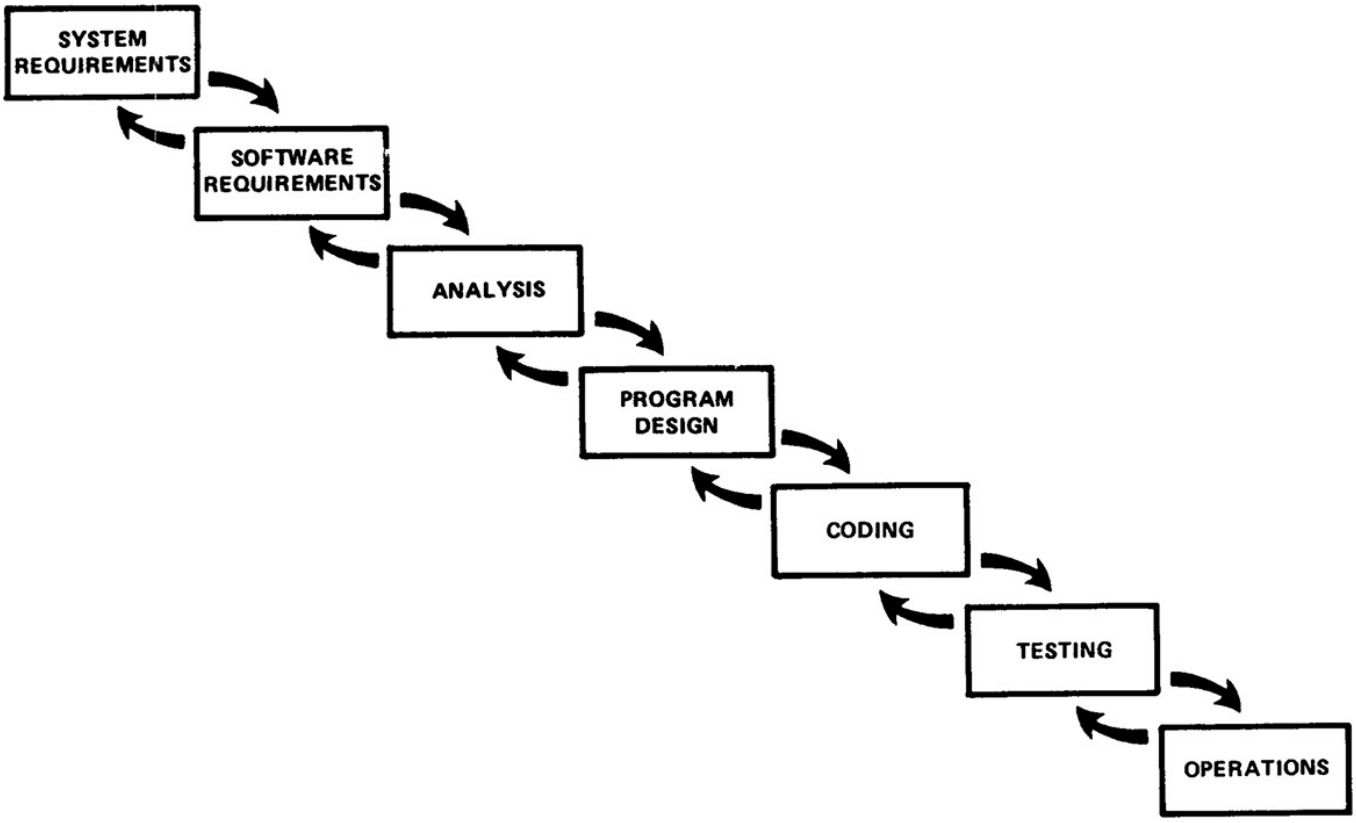

Figura 3. Es de esperar que la interacción iterativa entre las distintas etapas se limite a pasos sucesivos.

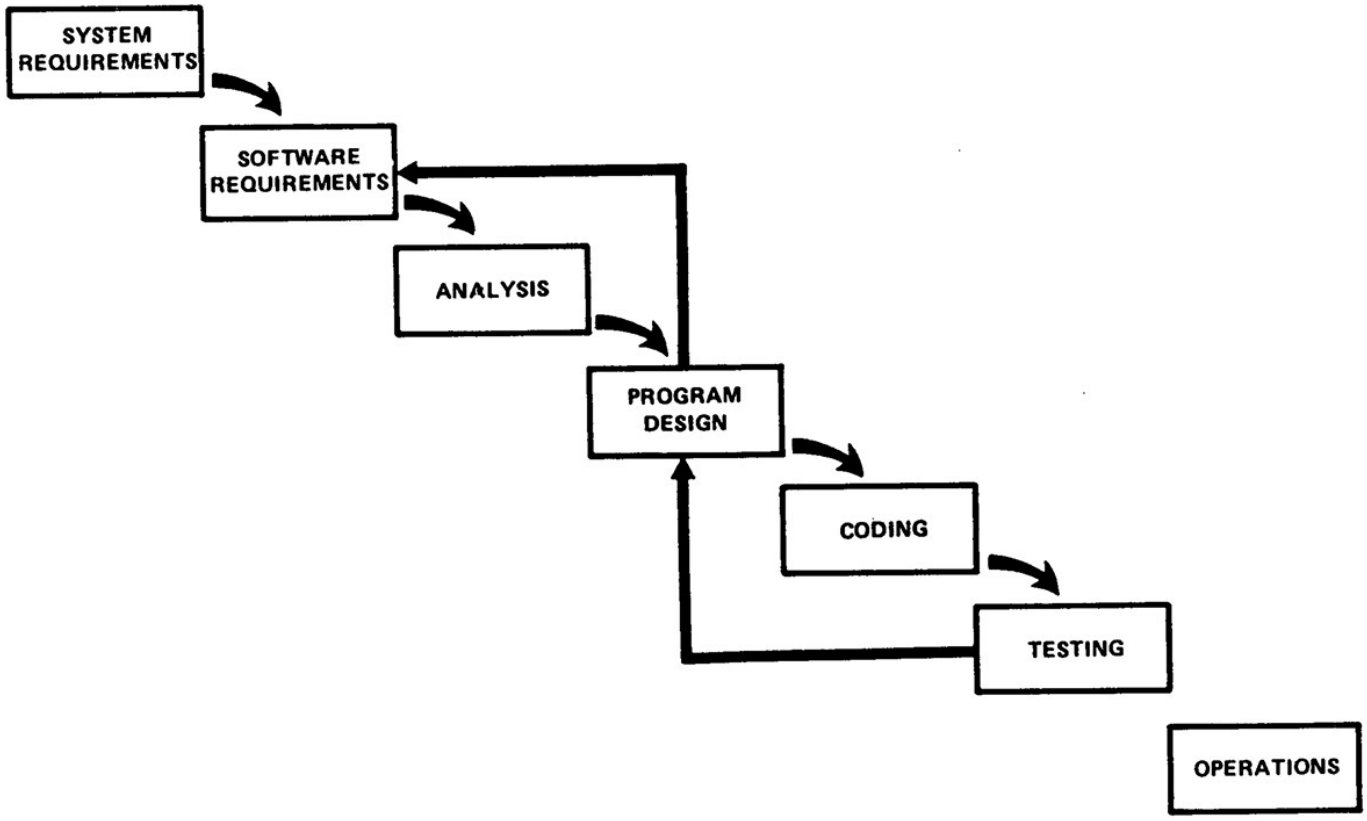

Figura 4. Desgraciadamente, para el proceso ilustrado, las iteraciones de diseño nunca se limitan a los pasos sucesivos.

PASO 1: EL DISEÑO DEL PROGRAMA ES LO PRIMERO

El primer paso hacia una solución se ilustra en la Figura 5. Se ha insertado una fase preliminar de diseño del programa entre la fase de generación de requisitos de software y la fase de análisis. Este procedimiento puede criticarse, ya que el diseñador del programa se ve obligado a diseñar en el relativo vacío de requisitos iniciales de software, sin ningún análisis previo. Como resultado, su diseño preliminar podría ser sustancialmente erróneo en comparación con el diseño previo si esperara hasta que se completara el análisis. Esta crítica es correcta, pero no es pertinente. Mediante esta técnica, el diseñador del programa garantiza que el software no fallará por razones de almacenamiento, tiempo y flujo de datos. A medida que avanza el análisis en la fase siguiente, el diseñador del programa debe imponer al analista las restricciones de almacenamiento, tiempo y operativas de tal manera que este prevea las consecuencias. Cuando justifique la necesidad de más recursos de este tipo para implementar sus ecuaciones, debe arrebatárselos simultáneamente a sus colegas analistas. De esta manera, todos los analistas y diseñadores del programa contribuirán a un proceso de diseño significativo que culminará en la asignación adecuada de tiempo de ejecución y recursos de almacenamiento. Si los recursos totales a aplicar son insuficientes o si el diseño operativo embrionario es incorrecto, se reconocerá en esta etapa anterior y se podrá rehacer la iteración con requisitos y diseño preliminar antes de que comience el diseño final, la codificación y las pruebas.

¿Cómo se implementa este procedimiento? Se requieren los siguientes pasos:

1) **Iniciar el proceso de diseño con diseñadores de programas**, no con analistas ni programadores.

2) **Diseñar, definir y asignar los modos de procesamiento de datos**, incluso a riesgo de equivocarse. Asignar procesamiento y funciones, diseñar la base de datos, definir el procesamiento de la base de datos, asignar el tiempo de ejecución, definir interfaces y modos de procesamiento con el sistema operativo, describir el procesamiento de entrada y salida, y definir procedimientos operativos preliminares.

3) **Redactar un documento general** comprensible, informativo y actualizado. Todos los trabajadores deben tener un conocimiento básico del sistema. Al menos una persona debe tener un conocimiento profundo del sistema, lo cual se debe en parte a su experiencia en la redacción de dicho documento.

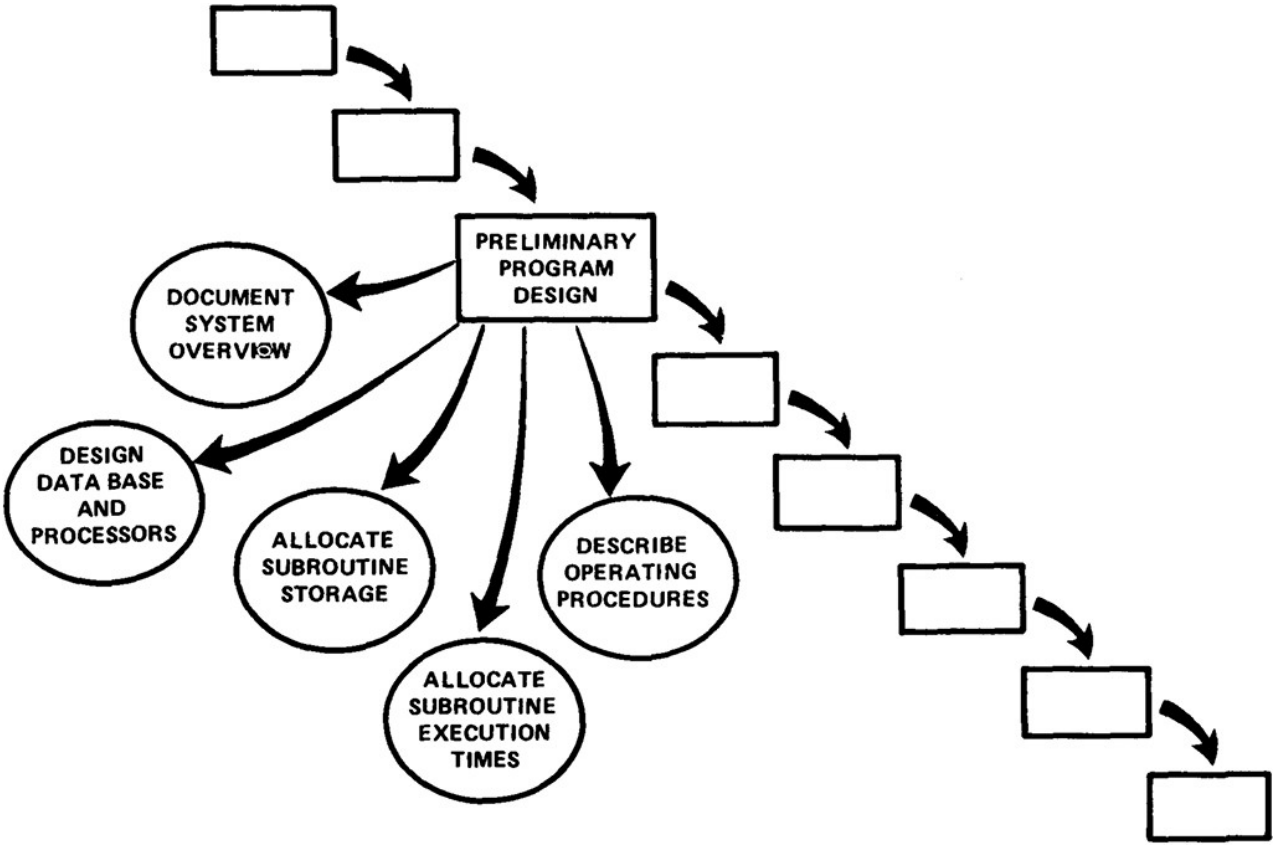

Figura 5. Paso 1: Asegúrese de que el diseño preliminar del programa está completo antes de iniciar el análisis.

PASO 2: DOCUMENTAR EL DISEÑO

En este punto, conviene plantear la pregunta: "¿Cuánta documentación?". En mi opinión, "bastante"; sin duda, más de la que la mayoría de programadores, analistas o diseñadores de programas están dispuestos a aceptar si se les deja a su suerte. La primera regla para gestionar el desarrollo de software es la aplicación rigurosa de los requisitos de documentación.

En ocasiones, me piden que revise el progreso de otros proyectos de diseño de software. Mi primer paso es investigar el estado de la documentación. Si la documentación presenta un incumplimiento grave, mi primera recomendación es simple: sustituir la gestión de proyectos. Detener todas las actividades no relacionadas con la documentación. Ajustar la documentación a estándares aceptables. La gestión de software es simplemente imposible sin un alto grado de documentación. A modo de ejemplo, permítanme ofrecer las siguientes estimaciones para comparar. Para adquirir un dispositivo de hardware de 5 millones de dólares, esperaría que una especificación de 30 páginas proporcionara los detalles suficientes para controlar la adquisición. Para adquirir software de 5 millones de dólares, estimaría que una especificación de 1500 páginas es suficiente para lograr un control comparable.

¿Por qué tanta documentación?

1) Cada diseñador debe comunicarse con los diseñadores de interfaz, con su gerencia y, posiblemente, con el cliente. Un registro verbal es demasiado intangible para proporcionar una base adecuada para una decisión de interfaz o gerencia. Una descripción escrita aceptable obliga al diseñador a adoptar una postura inequívoca y a proporcionar evidencia tangible de finalización. Evita que el diseñador se escude en el síndrome de "He terminado al 90 %" mes tras mes.

2) Durante la fase inicial del desarrollo de software, la documentación es la especificación y el diseño. Hasta que comienza la codificación, estos tres sustantivos (documentación, especificación, diseño) denotan una sola cosa. Si la documentación es deficiente, el diseño es deficiente. Si la documentación aún no existe, aún no hay diseño, solo personas que piensan y hablan sobre el diseño, lo cual tiene cierto valor, pero no mucho.

3) El verdadero valor monetario de una buena documentación comienza en las fases finales del proceso de desarrollo, durante la fase de pruebas, y continúa durante las operaciones y el rediseño. El valor de la documentación puede describirse en tres situaciones concretas y tangibles que todo gerente de programa enfrenta:

a) **Durante la fase de pruebas**, con una buena documentación, el gerente puede concentrar al personal en los errores del programa. Sin una buena documentación, cada error, grande o pequeño, es analizado por la persona que probablemente lo cometió en primer lugar, ya que es la única que comprende el área del programa.

b) **Durante la fase operativa**, con una buena documentación, el gerente puede emplear personal especializado en operaciones para operar el programa y realizar un mejor trabajo a un menor costo. Sin una buena documentación, el software debe ser operado por quienes lo desarrollaron. Generalmente, estas personas muestran un desinterés relativo a las operaciones y no realizan un trabajo tan efectivo como el personal especializado en operaciones. Cabe señalar a este respecto que, en una situación operativa, si surge algún problema, siempre se culpa primero al software. Para justificar el error o para corregirlo, la documentación del software debe ser clara.

c) **Tras las operaciones iniciales**, cuando las mejoras del sistema son necesarias, una buena documentación permite un rediseño, actualización y modernización efectivos en campo. Si no existe documentación, generalmente se debe descartar todo el marco operativo del software, incluso para cambios relativamente modestos. La Figura 6 muestra un plan de documentación que se basa en los pasos mostrados anteriormente. Se hace constar que se producen seis documentos, y al momento de la entrega del producto final, los Documentos No. 1, No. 3, No. 4, No. 5 y No. 6 se encuentran actualizados y vigentes.

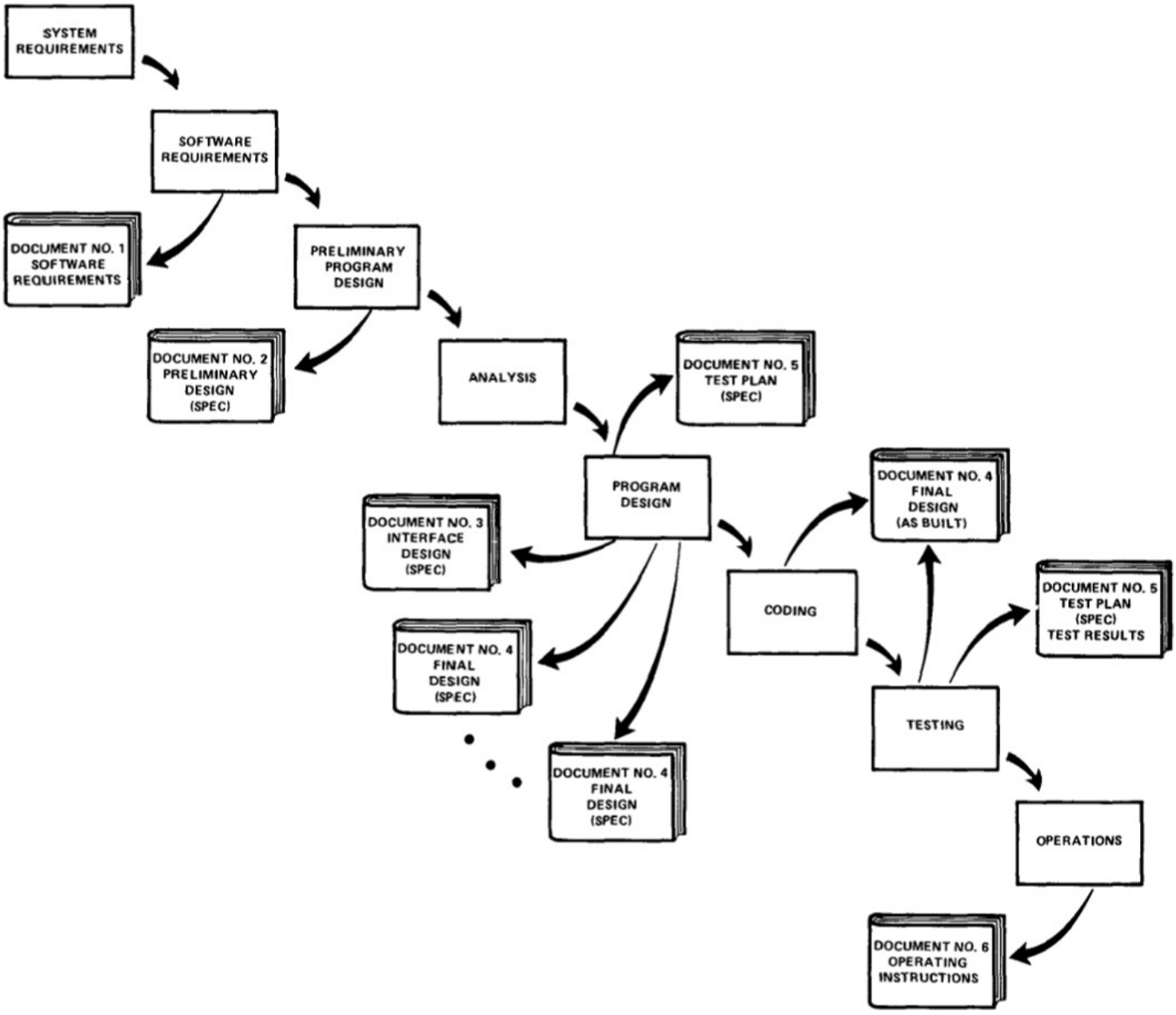

Figura 6. Paso 2: Asegúrese de que la documentación esté actualizada y completa: se requieren al menos seis documentos únicos y diferentes.

PASO 3: HAZLO DOS VECES

Después de la documentación, el segundo criterio más importante para el éxito reside en la originalidad del producto. Si el programa informático en cuestión se desarrolla por primera vez, es necesario organizar la operación para que la versión entregada al cliente para su despliegue operativo sea, en realidad, la segunda versión en lo que respecta a las áreas críticas de diseño y operaciones. La Figura 7 ilustra cómo se podría llevar a cabo esto mediante una simulación. Cabe destacar que se trata simplemente de todo el proceso realizado en miniatura, en un plazo relativamente pequeño en relación con el esfuerzo total. La naturaleza de este esfuerzo puede variar considerablemente, dependiendo principalmente del plazo total y de la naturaleza de las áreas problemáticas críticas que se modelarán. Si el esfuerzo dura 30 meses, el desarrollo inicial de un modelo piloto podría programarse para 10 meses. Para este cronograma, se pueden utilizar controles formales, procedimientos de documentación, etc. Sin embargo, si el esfuerzo total se redujera a 12 meses, el esfuerzo piloto podría reducirse a quizás tres meses para aprovechar al máximo el desarrollo principal. En este caso, se requiere una amplia competencia por parte del personal involucrado. Deben tener una intuición para el análisis, la codificación y el diseño de programas. Deben detectar rápidamente los puntos problemáticos del diseño, modelarlos, modelar sus alternativas, olvidar los aspectos sencillos del diseño que no vale la pena estudiar en esta etapa inicial y, finalmente, llegar a un programa sin errores. En cualquier caso, el objetivo de todo esto, al igual que con una simulación, es que las cuestiones de tiempo, almacenamiento, etc., que de otro modo serían cuestiones de juicio, ahora se puedan estudiar con precisión. Sin esta simulación, el director del proyecto está a merced del juicio humano. Con la simulación, al menos puede realizar pruebas experimentales de algunas hipótesis clave y delimitar lo que queda para el juicio humano, que en el área del diseño de programas informáticos (como en la estimación del peso bruto de despegue, los costos de finalización o el doble diario) es invariable y seriamente optimista.

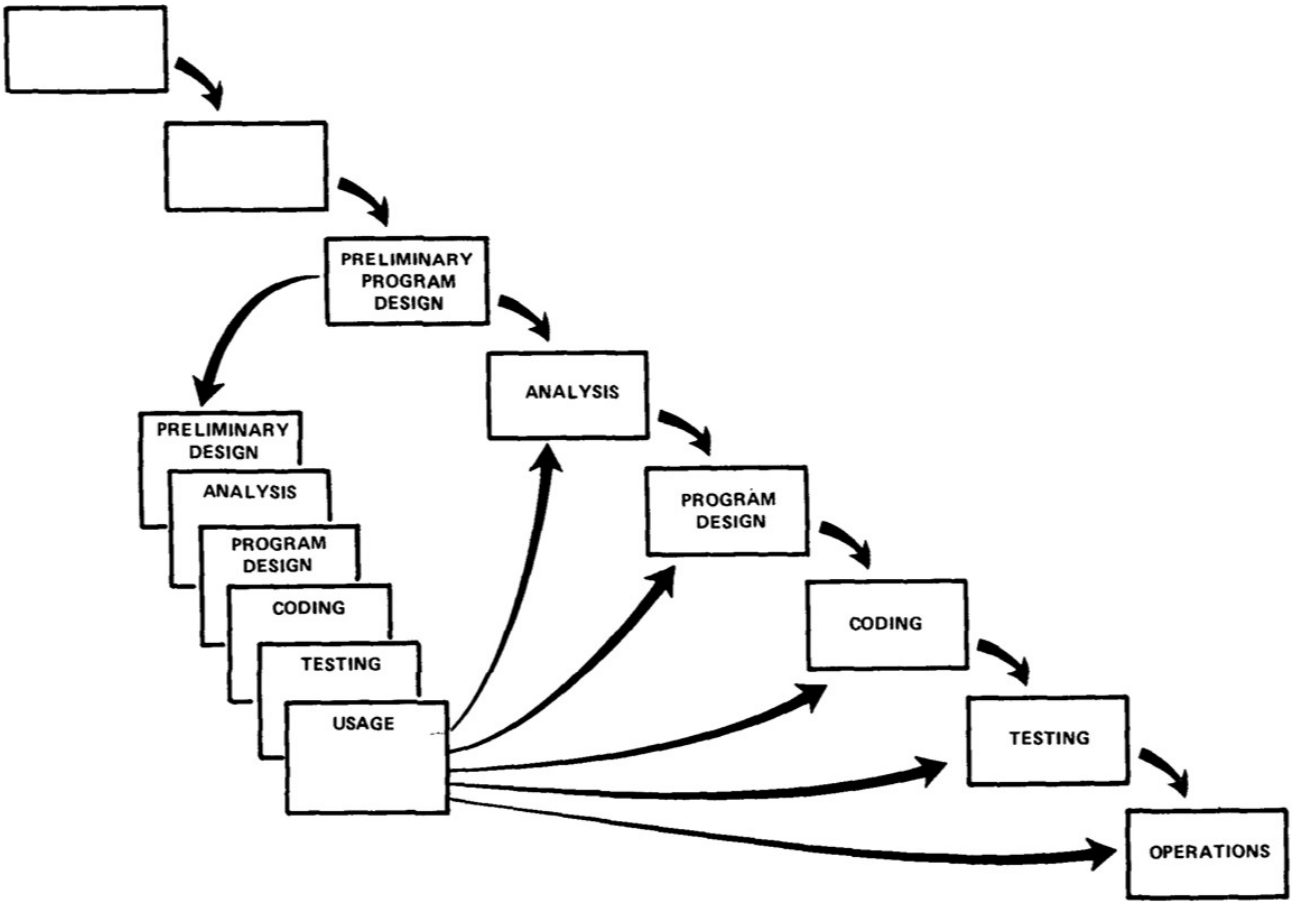

Figura 7. Paso 3: Intente realizar el trabajo dos veces: el primer resultado proporciona una simulación temprana del producto final.

PASO 4: PLANIFICAR, CONTROLAR Y SUPERVISAR LAS PRUEBAS

Sin duda, la fase de prueba es la que más recursos del proyecto consume, ya sea mano de obra, tiempo de computación o criterio de la gerencia. Es la fase de mayor riesgo en términos de dinero y planificación. Ocurre en el último punto de la planificación, cuando las alternativas de respaldo son menos disponibles, si es que existen.

Las tres recomendaciones anteriores (diseñar el programa antes de comenzar el análisis y la codificación, documentarlo completamente y construir un modelo piloto) buscan identificar y resolver problemas antes de entrar en la fase de pruebas. Sin embargo, incluso después de realizar estas acciones, aún queda una fase de pruebas y quedan tareas importantes por hacer. La Figura 8 enumera algunos aspectos adicionales de las pruebas. Al planificar las pruebas, sugiero considerar lo siguiente:

1) Muchas partes del proceso de prueba son mejor gestionadas por especialistas en pruebas que no necesariamente contribuyeron al diseño original. Si se argumenta que solo el diseñador puede realizar una prueba exhaustiva porque solo él comprende el área que construyó, esto es una clara señal de una documentación deficiente. Con una buena documentación, es posible recurrir a especialistas en aseguramiento de productos de software que, en mi opinión, realizarán un mejor trabajo de pruebas que el diseñador.

2) La mayoría de los errores son obvios y se detectan fácilmente mediante una inspección visual. Cada parte del análisis y cada fragmento de código debe someterse a un simple análisis visual por parte de una segunda persona que no realizó el análisis ni el código original, pero que detectaría errores como la omisión de signos menos, la ausencia de factores de dos, saltos a direcciones incorrectas, etc., que forman parte de la corrección del análisis y el código. No utilice la computadora para detectar este tipo de errores; es demasiado costoso.

3) Pruebe cada ruta lógica del programa informático al menos una vez con algún tipo de verificación numérica. Si yo fuera cliente, no aceptaría la entrega hasta que este procedimiento estuviera completado y certificado. Este paso detectará la mayoría de los errores de codificación. Si bien este procedimiento de prueba parece simple, para un programa informático grande y complejo es relativamente difícil analizar cada ruta lógica con valores de entrada controlados. De hecho, hay quienes argumentan que es casi imposible. A pesar de esto, insisto en mi recomendación de que cada ruta lógica se someta al menos a una verificación auténtica.

4) Una vez eliminados los errores simples (que son la mayoría y ocultan los grandes), llega el momento de entregar el software al área de pruebas para su verificación. En el momento oportuno durante el desarrollo y en manos de la persona adecuada, la computadora es el mejor dispositivo para la verificación. Las decisiones clave de gestión son: ¿cuándo es el momento y quién es la persona indicada para realizar la verificación final?

PASO 5: INVOLUCRAR AL CLIENTE

Por alguna razón, el diseño de un software está sujeto a una amplia interpretación, incluso tras un acuerdo previo. Es importante involucrar formalmente al cliente para que se comprometa con anterioridad, antes de la entrega final. Darle al contratista total libertad entre la definición de requisitos y la operación es una invitación a los problemas. La Figura 9 indica tres puntos posteriores a la definición de requisitos donde la comprensión, el criterio y el compromiso del cliente pueden impulsar el desarrollo.

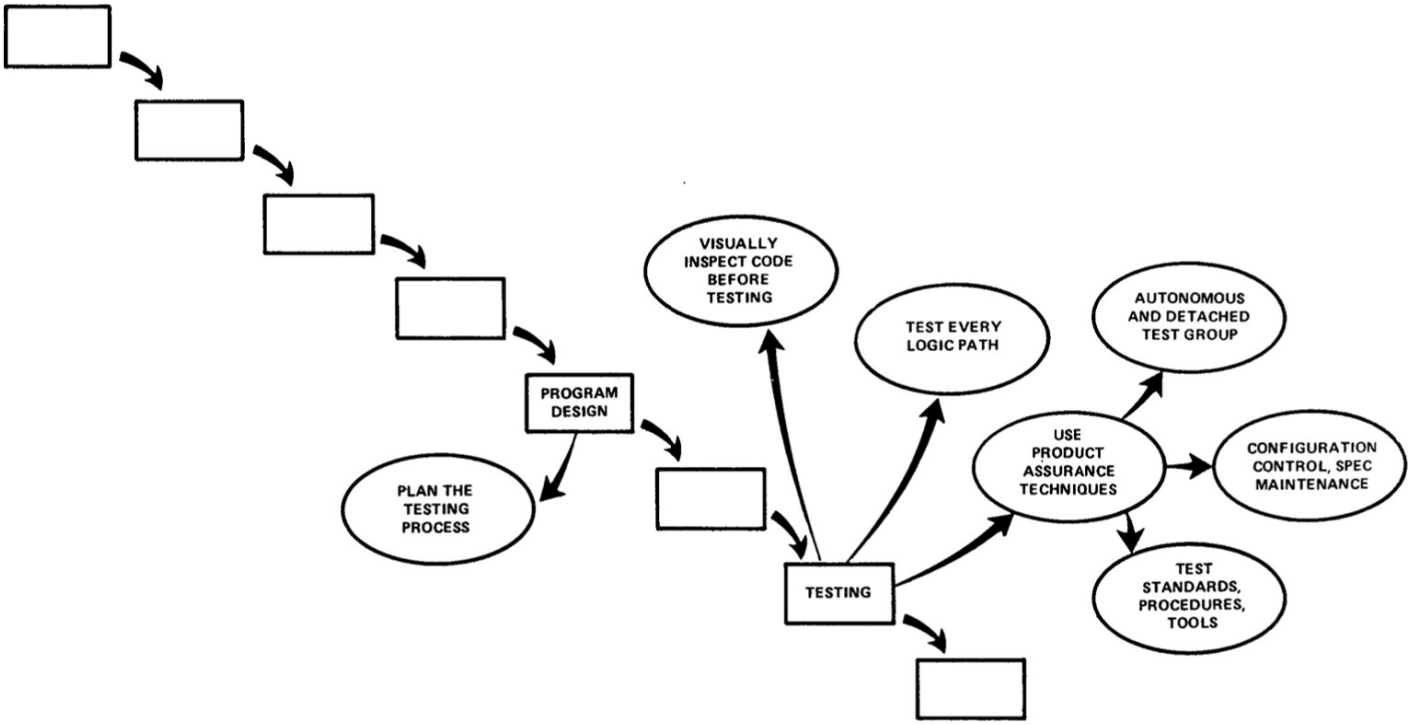

Figura 8. Paso 4: Planificar, controlar y supervisar las pruebas de los programas informáticos.

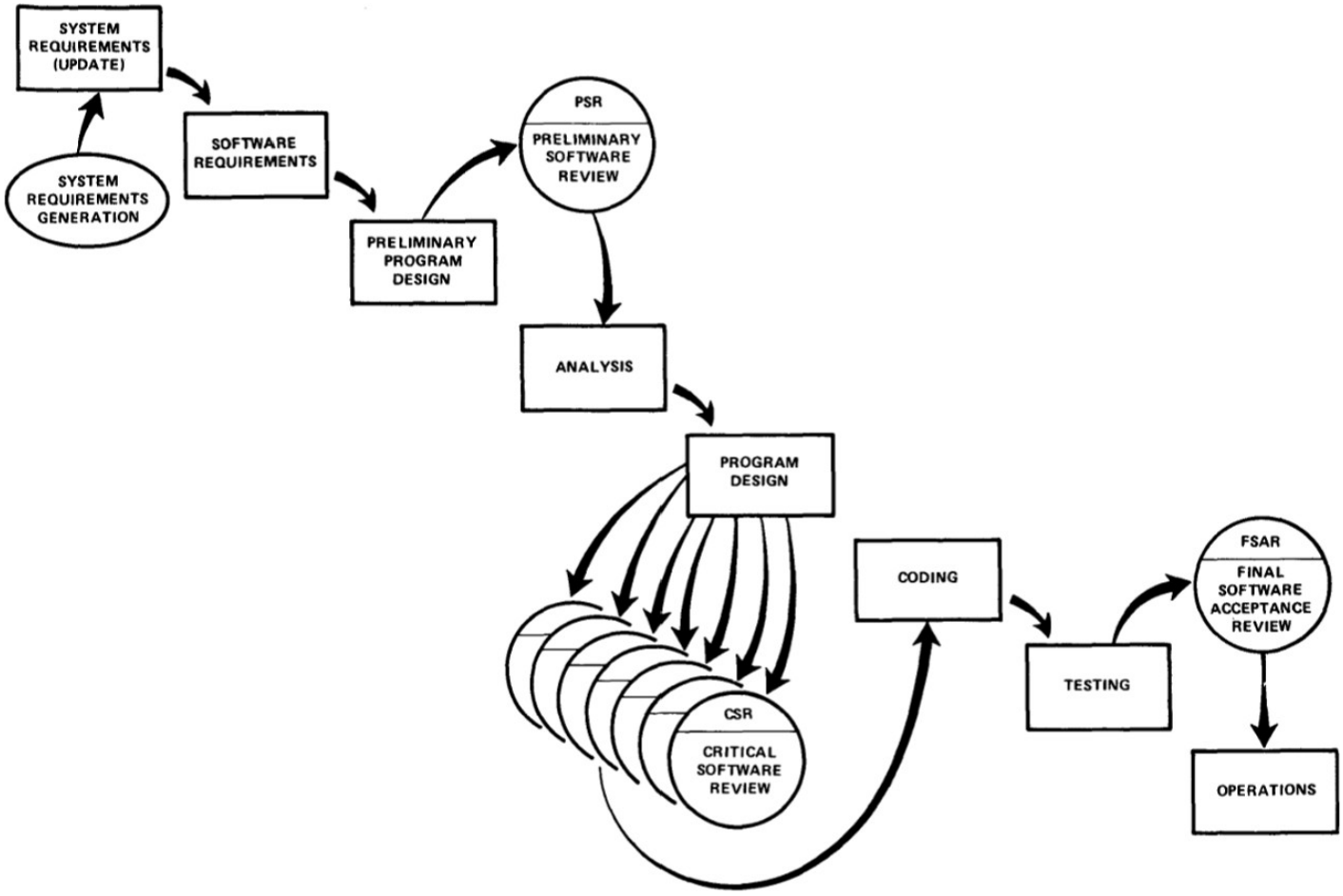

Figura 9. Paso 5: Involucrar al cliente: El involucramiento debe ser formal, profundo y continuo.

RESUMEN

La Figura 10 resume los cinco pasos que considero necesarios para transformar un proceso de desarrollo arriesgado en uno que proporcione el producto deseado. Cabe destacar que cada elemento tiene un costo adicional. Si el proceso relativamente más simple, sin las cinco complejidades descritas aquí, funcionara correctamente, entonces, por supuesto, el dinero adicional no estaría bien invertido. Sin embargo, en mi experiencia, el método más simple nunca ha funcionado en grandes proyectos de desarrollo de software, y los costos de recuperación superaron con creces los necesarios para financiar el proceso de cinco pasos mencionado.

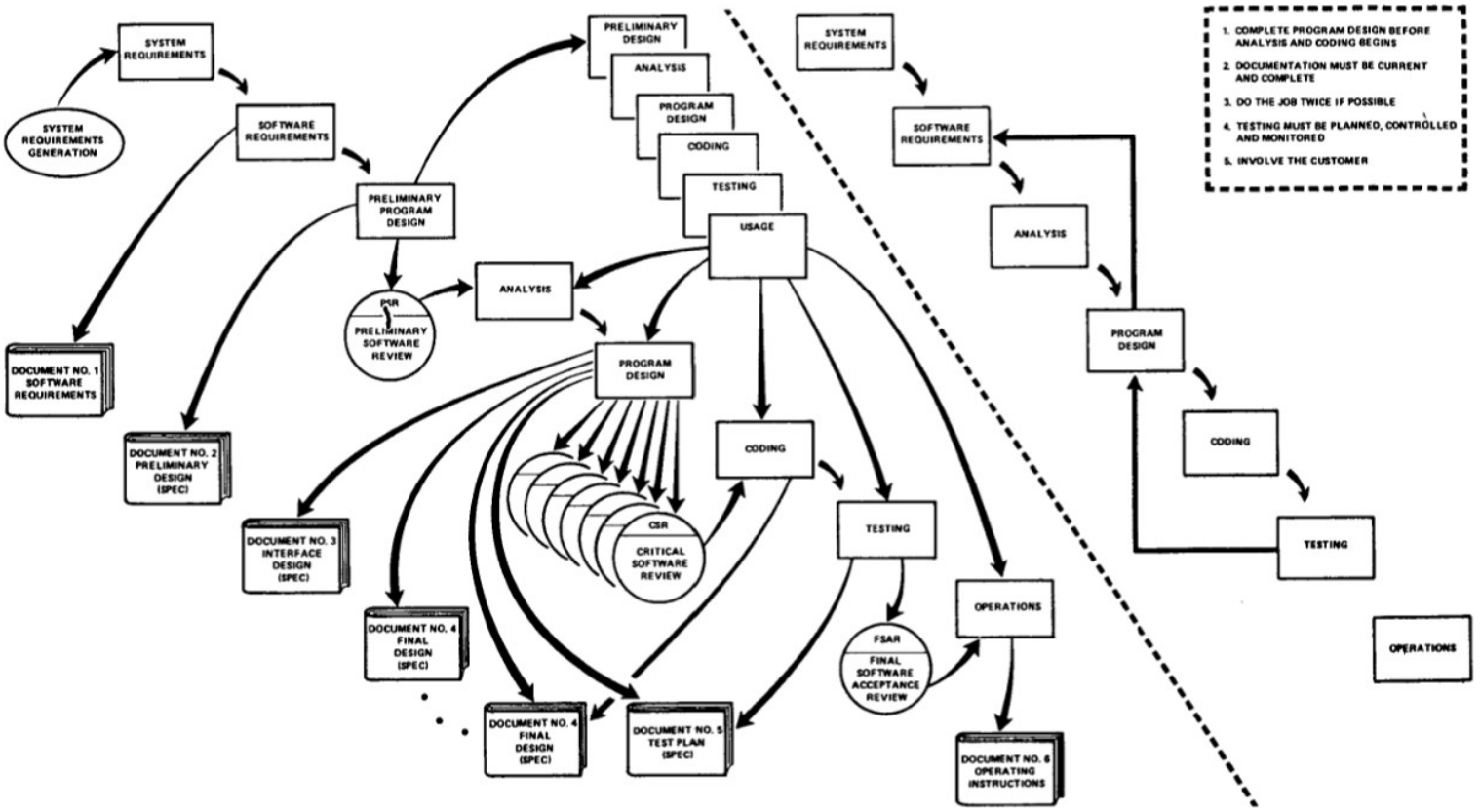

Figura 10. Resumen