Universidad Nacional de Cajamarca
"Norte de la Universidad Peruana"

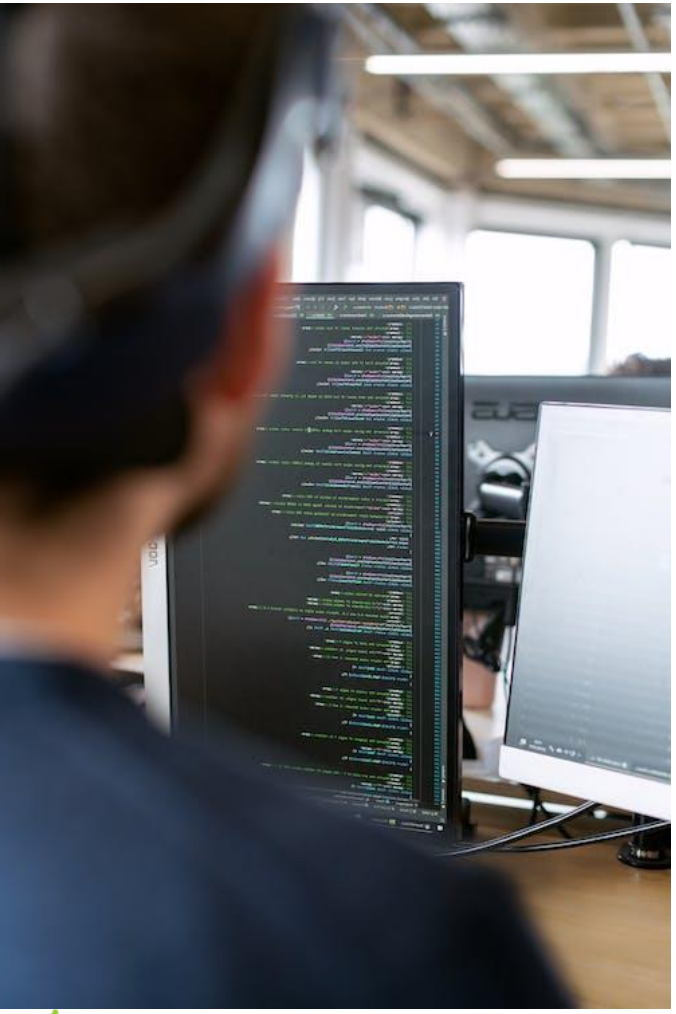
INGENIERÍA DE SOFTWARE I

Semana 4

LOGRO

- Al término de la sesión, el estudiante identifica los pasos del modelo de proceso recomendado y lo plasta en un diagrama de Gantt.

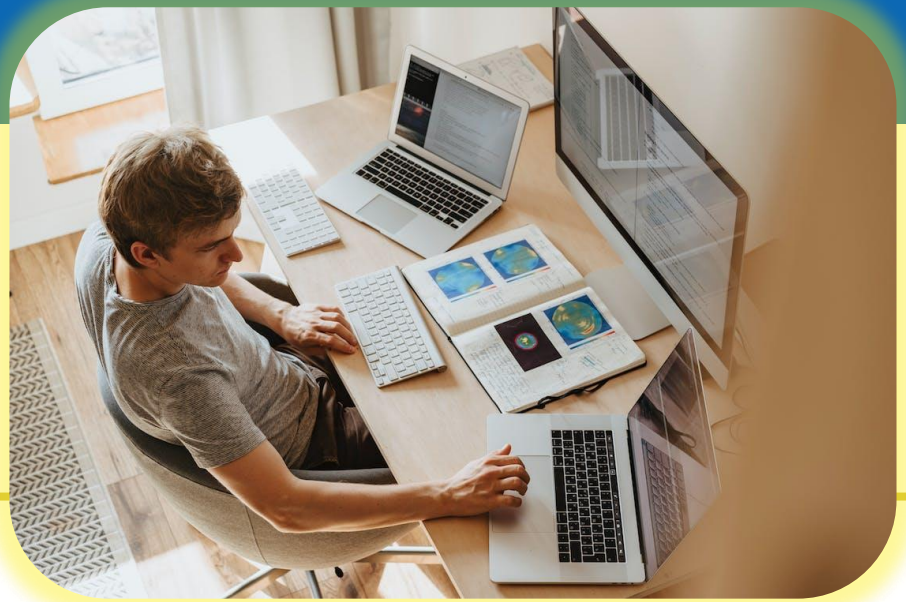
TEMARIO



Modelo de proceso recomendado

1. Definir requisitos
2. Diseño arquitectónico preliminar
3. Estimación de recursos
4. Construcción del primer prototipo
5. Evaluación del tipo de prototipo
6. Decisión de ir o no ir.
7. Evolución del tipo de prototipo
8. Lanzamiento del tipo de prototipo
9. Software de versión minimalista
10. Resumen

Modelo de proceso recomendado



Universidad Nacional de Cajamarca
"Norte de la Universidad Peruana"

Lectura

Ver material complementario

Introducción

- Cada proyecto es diferente y cada equipo es diferente. No existe un marco de ingeniería de software único que sea apropiado para cada producto de software. En este capítulo, compartiremos nuestras ideas sobre el uso de un proceso adaptable que se puede adaptar para satisfacer las necesidades de los desarrolladores de software que trabajan en muchos tipos de productos.

- Los enfoques prescriptivos del ciclo de vida del software (por ejemplo, el modelo en cascada) contienen varias debilidades por lo que se sugiere que deben considerarse al organizar un proyecto de desarrollo de software moderno.

1. Es arriesgado utilizar un modelo de proceso lineal sin suficiente retroalimentación.
2. Nunca es posible ni deseable planificar una gran recopilación de requisitos por adelantado.
3. La recopilación inicial de requisitos puede no reducir los costos ni evitar retrasos.
4. La gestión adecuada de proyectos es parte integral del desarrollo de software.
5. Los documentos deben evolucionar con el software y no deben retrasar el inicio de construcción.

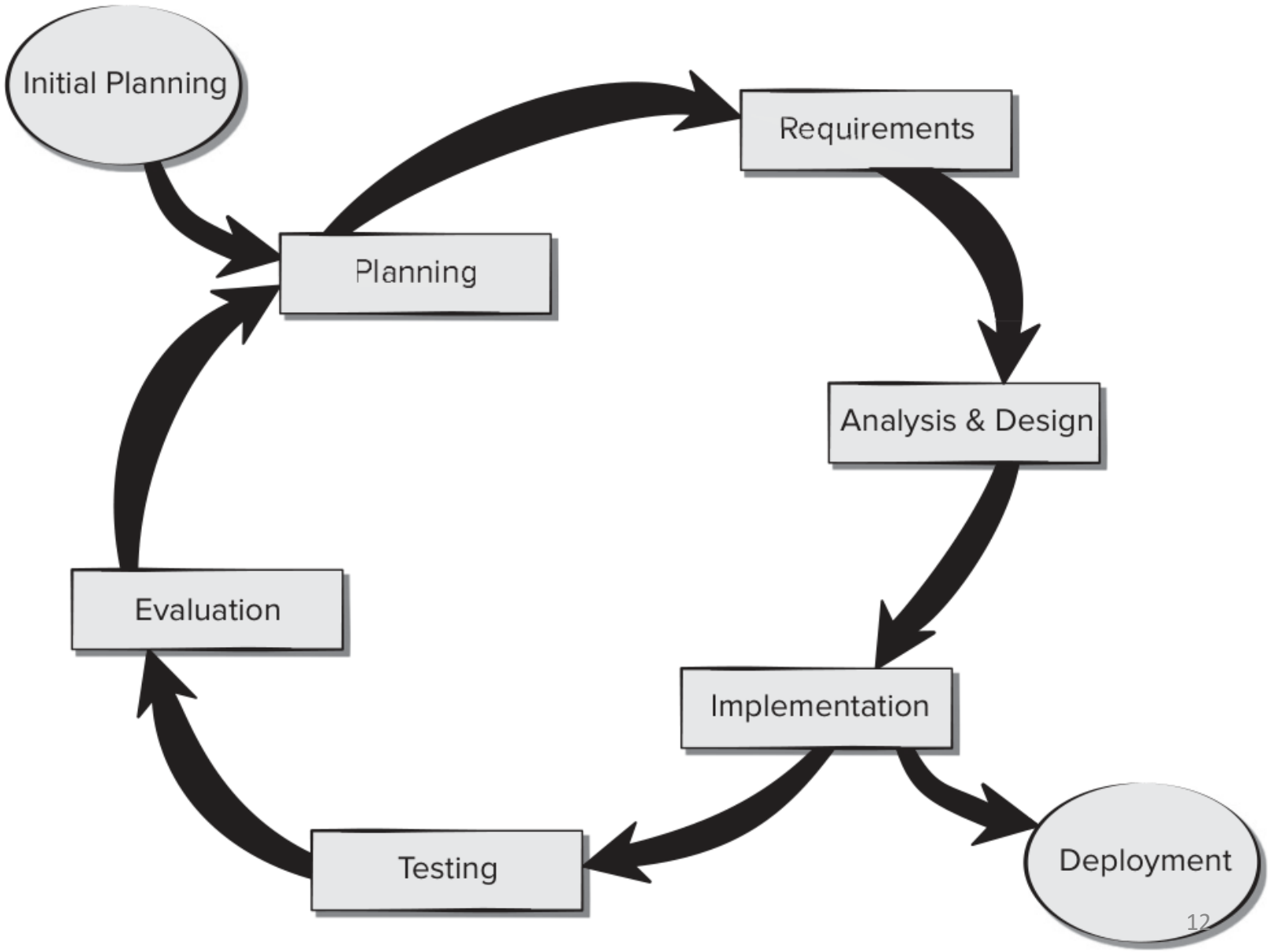
6. Involucrar a las partes interesadas temprana y frecuentemente en el proceso de desarrollo.

7. Los evaluadores deben involucrarse en el proceso antes de la construcción del software.

- El modelo en cascada no admite cambios que puedan ser necesarios una vez que los desarrolladores comiencen a codificar. Por lo tanto, la retroalimentación de las partes interesadas se limita al principio y al final del proyecto. Parte de la razón de esto es que el modelo en cascada sugiere que todos los productos del trabajo de análisis y diseño se completen antes de que se realice cualquier programación o prueba. Esto dificulta la adaptación a proyectos con requisitos en evolución

- Una opción es cambiar a un modelo incremental, como el modelo de creación de prototipos o Scrum. Los modelos de procesos incrementales involucran a los clientes desde el principio y con frecuencia y, por lo tanto, reducen el riesgo de crear productos que no sean aceptados por los clientes.
- Existe la tendencia de fomentar muchos cambios a medida que las partes interesadas ven cada prototipo y se dan cuenta de que faltan funciones y características que ahora saben que necesitan. A menudo, los desarrolladores no planifican la evolución de los prototipos y crean prototipos desechables.

Modelo incremental para el desarrollo de prototipos.



- Recuerde que el objetivo de la ingeniería de software es reducir el esfuerzo innecesario, por lo que los prototipos deben diseñarse teniendo en cuenta la reutilización. Los modelos incrementales proporcionan una mejor base para crear un proceso adaptable si los cambios se pueden gestionar sabiamente.
- Los modelos de procesos ágiles son muy buenos para acomodar el conocimiento incierto sobre las necesidades y problemas reales de las partes interesadas.

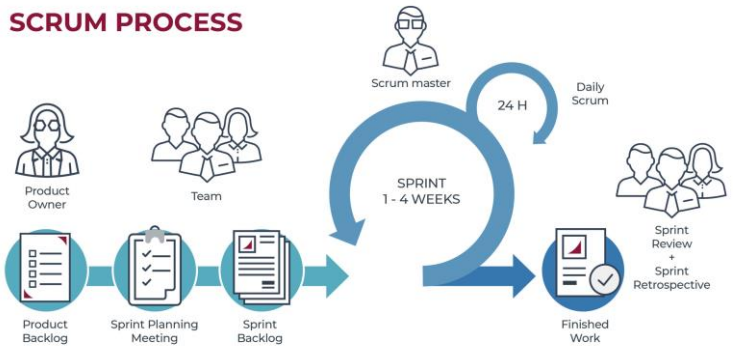
Características clave de los modelos de procesos ágiles son:

- Los prototipos creados están diseñados para ampliarse en futuros incrementos de software.
- Las partes interesadas participan durante todo el proceso de desarrollo.
- Los requisitos de documentación son livianos y la documentación debe evolucionar junto con el software
- Las pruebas se planifican y ejecutan con antelación.

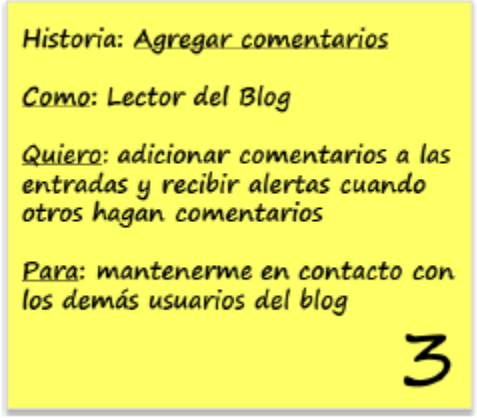
Características clave de los modelos de procesos ágiles son:

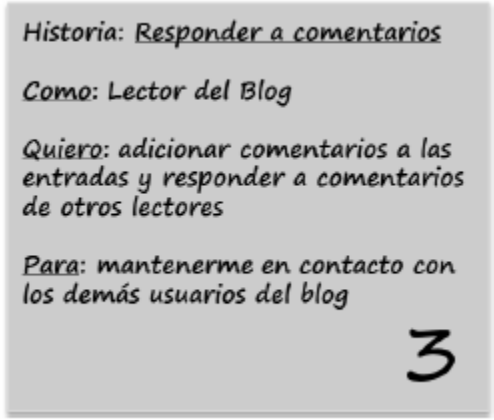
Universidad Nacional de Cajamarca
"Norte de la Universidad Peruana"

Scrum y Kanban amplían estas características. A veces se critica a Scrum por requerir demasiadas reuniones. Pero las reuniones diarias dificultan que los desarrolladores se desvíen demasiado de la creación de productos que las partes interesadas encuentren útiles. Kanban proporciona un buen sistema de seguimiento ligero para gestionar el estado y las prioridades de las historias de los usuarios.



- Tanto Scrum como Kanban permiten la introducción controlada de nuevos requisitos (historias de usuario). Los equipos ágiles son pequeños por diseño y pueden no ser adecuados para proyectos que requieren una gran cantidad de desarrolladores, a menos que el proyecto pueda dividirse en componentes pequeños y asignables de forma independiente. Aun así, los modelos de procesos ágiles ofrecen muchas características buenas que pueden incorporarse a un modelo de procesos adaptable.

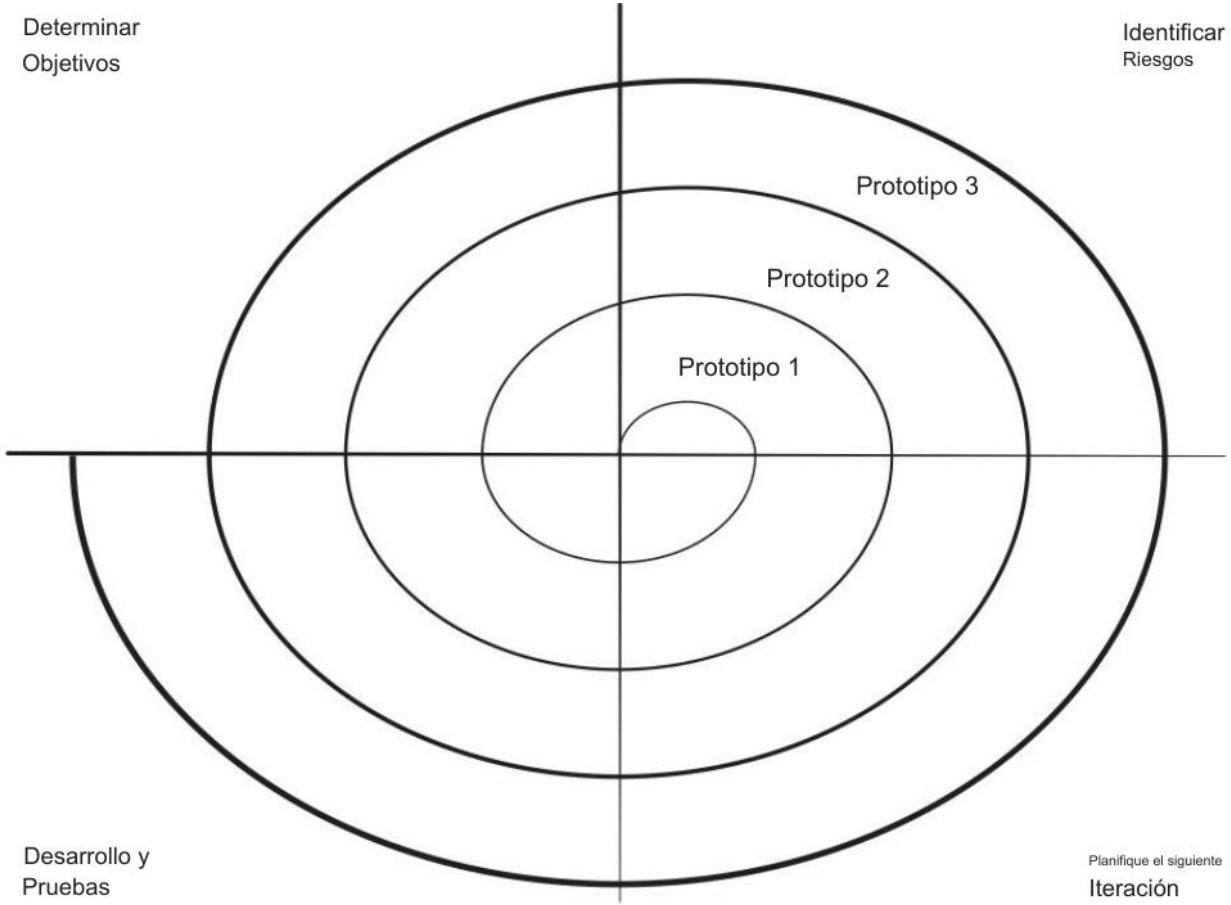




Modelo en espiral para desarrollo de prototipos.

Universidad Nacional de Cajamarca
"Norte de la Universidad Peruana"

- El modelo en espiral puede considerarse como un modelo de creación de prototipos evolutivos con un elemento de evaluación de riesgos. El modelo espiral se basa en actores moderados.



- Participación y fue diseñado para grandes equipos y grandes proyectos. Su objetivo es crear prototipos extensibles cada vez que se itera el proceso. Las pruebas tempranas son esenciales. La documentación evoluciona con la creación de cada nuevo prototipo. El modelo en espiral es algo único en el sentido de que se incorpora una evaluación formal de riesgos y se utiliza como base para decidir si se invierten los recursos necesarios para crear el siguiente prototipo. Algunas personas argumentan que puede ser difícil gestionar un proyecto utilizando el modelo en espiral, porque es posible que el alcance del proyecto no se conozca al inicio. Esto es típico de la mayoría de los modelos de procesos incrementales. La espiral es una buena base para construir un modelo de proceso adaptable.

Características de los modelos ágiles

Universidad Nacional de Cajamarca
"Norte de la Universidad Peruana"

Ágil

1. No apto para proyectos grandes de alto riesgo o de misión crítica.
2. Reglas mínimas y documentación mínima
3. Participación continua de los evaluadores
4. Fácil de adaptar a los cambios de producto
5. Depende en gran medida de la interacción de las partes interesadas
6. Fácil de gestionar
7. Entrega temprana de soluciones parciales
8. Gestión informal de riesgos
9. Mejora continua del proceso incorporada

Espiral

1. No apto para proyectos pequeños y de bajo riesgo.
2. Se requieren varios pasos, junto con la documentación realizada por adelantado.
3. Participación temprana de los evaluadores (puede ser realizada por un equipo externo)
4. Es difícil adaptarse a los cambios del producto hasta que se complete el prototipo.
5. Participación continua de las partes interesadas en planificación y evaluación de riesgos
6. Requiere gestión formal del proyecto y coordinación
7. El final del proyecto no siempre es obvio
8. Buena gestión de riesgos
9. Mejora del proceso manejada al final del proyecto.

- Personas creativas y conocedoras realizan ingeniería de software. Adaptan los procesos de software para hacerlos apropiados para los productos que construyen y para satisfacer las demandas del mercado.
- Creemos que utilizar un enfoque en espiral que incluya agilidad en cada ciclo es un buen punto de partida para muchos proyectos de software.
- Los desarrolladores aprenden muchas cosas a medida que avanzan en el proceso de desarrollo. Por eso es importante que los desarrolladores puedan adaptar su proceso lo más rápido posible para adaptarse a este nuevo conocimiento.

1- Definir requisitos

- Cada proyecto de software comienza cuando el equipo intenta comprender el problema a resolver y determinar qué resultados son importantes para las partes interesadas. Esto incluye comprender las necesidades comerciales que motivan el proyecto y los problemas técnicos que lo limitan. Este proceso se llama ingeniería de requisitos Los equipos que no dedican una cantidad de tiempo razonable a esta tarea encontrarán que su proyecto contiene retrabajos costosos, sobrecostos, mala calidad del producto, retrasos en los tiempos de entrega, clientes insatisfechos y baja moral del equipo.

- No se puede descuidar la ingeniería de requisitos, ni se puede permitir que se repita sin cesar antes de proceder a la construcción del producto.
- Scott Ambler [Amb12] sugiere varias mejores prácticas para la definición de requisitos ágiles:

1. Fomentar la participación activa de las partes interesadas haciendo coincidir su disponibilidad y valorando sus aportaciones.
2. Utilice modelos simples (p. ej., notas adhesivas, bocetos rápidos, historias de usuarios) para reducir barreras a la participación
3. Tómese el tiempo para explicar sus técnicas de representación de requisitos antes de usarlas a ellos.
4. Adoptar terminología de las partes interesadas y evitar la jerga técnica siempre que sea posible.
5. Utilice un enfoque amplio para tener una visión general del proyecto antes de atascarse en los detalles
6. Permitir que el equipo de desarrollo refine (con aportes de las partes interesadas) los detalles de los requisitos "justo a tiempo" a medida que se programa la implementación de las historias de los usuarios.

- Trate la lista de funciones a implementar como una lista priorizada e implemente y Mencione primero las historias de usuario más importantes.
- Colabore estrechamente con sus partes interesadas y solo documente los requisitos en un nivel que es útil para todos a la hora de crear el próximo prototipo.
- Cuestiona la necesidad de mantener modelos y documentos que no serán remitidos a en el futuro
- Asegúrese de contar con apoyo administrativo para garantizar que las partes interesadas y los recursos disponibilidad durante la definición de requisitos.

- Es importante reconocer dos realidades: (1) es imposible para las partes interesadas describan un sistema completo antes de ver el software en funcionamiento, y (2) es difícil para las partes interesadas describan los requisitos de calidad necesarios para el software antes de verlo en acción.
- Los desarrolladores deben reconocer que los requisitos se agregarán y perfeccionarán a medida que se creen los incrementos del software.
- Un buen punto de partida es capturar las descripciones de las partes interesadas sobre lo que el sistema debe hacer en sus propias palabras en una historia de usuario.

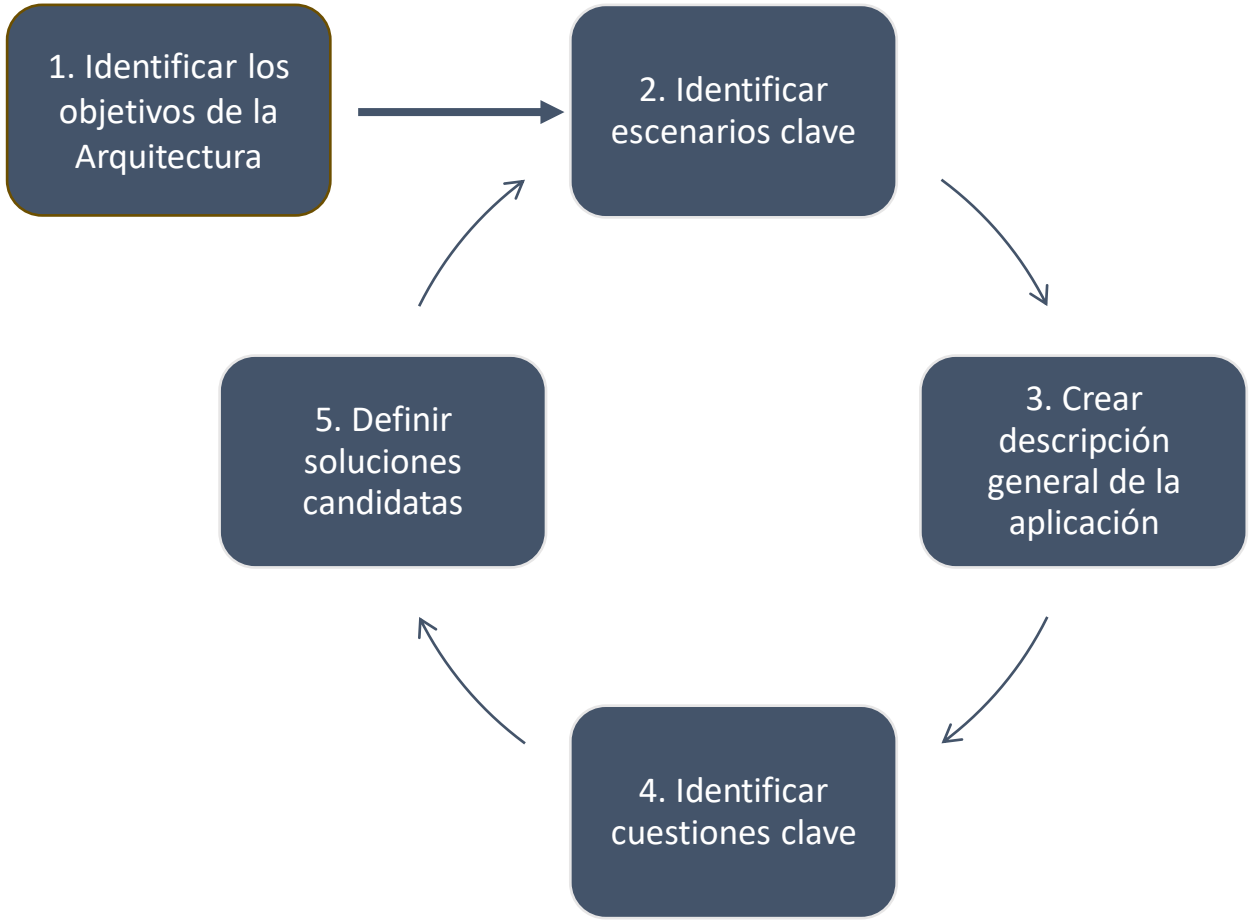
- Si puede lograr que las partes interesadas definan criterios de aceptación para cada historia de usuario, su equipo habrá tenido un gran comienzo.
- Es probable que las partes interesadas necesiten ver una historia de usuario codificada y en ejecución para saber si se ha implementado correctamente o no. Por lo tanto, la definición de requisitos debe realizarse de forma iterativa e incluir el desarrollo de prototipos para la revisión de las partes interesadas.

- Los prototipos son realizaciones tangibles de los planes del proyecto a las que las partes interesadas pueden hacer referencia fácilmente cuando intentan describir los cambios deseados. Las partes interesadas están motivadas para discutir los cambios de requisitos en términos más concretos, lo que mejora la comunicación. Es importante reconocer que los prototipos permiten a los desarrolladores centrarse en objetivos a corto plazo centrándose únicamente en los comportamientos visibles de los usuarios. Será importante revisar prototipos teniendo en cuenta la calidad.

- Los desarrolladores deben ser conscientes de que el uso de prototipos puede aumentar la volatilidad de los requisitos si las partes interesadas no se centran en hacer las cosas bien la primera vez. También existe el riesgo de que la creación de prototipos antes de que se comprendan bien los requisitos de la arquitectura del software pueda dar como resultado prototipos que deban descartarse, lo que desperdiciará tiempo y recursos.

2. Diseño arquitectónico preliminar

- En algún momento, las decisiones arquitectónicas deberán asignarse a incrementos de producto. Según Bellomo y sus colegas, la comprensión temprana de los requisitos y las opciones de arquitectura es clave para gestionar el desarrollo de productos de software grandes o complejos.



- Los requisitos se pueden utilizar para informar el diseño de arquitectura. Explorar la arquitectura a medida que se desarrolla el prototipo facilita el proceso de detallar los requisitos. Es mejor realizar estas actividades simultáneamente para lograr el equilibrio adecuado.

Hay cuatro elementos clave para un diseño arquitectónico ágil:

1. Centrarse en los atributos clave de calidad e incorporarlos en los prototipos a medida que vayan surgiendo están construidos.
2. Al planificar prototipos, tenga en cuenta que los productos de software exitosos combinan características visibles para el cliente y la infraestructura para habilitarlas.

3. Reconocer que una arquitectura ágil permite el mantenimiento y la capacidad de evolución del código si se presta suficiente atención a las decisiones arquitectónicas y los problemas de calidad relacionados

4. Es necesario gestionar y sincronizar continuamente las dependencias entre los requisitos funcionales y arquitectónicos para garantizar que la base arquitectónica en evolución esté lista justo a tiempo para futuros incrementos.

- La toma de decisiones sobre la arquitectura de software es fundamental para el éxito de un sistema de software. Dasanayake descubrió que los arquitectos de software son propensos a cometer errores cuando sus decisiones se toman bajo niveles de incertidumbre. Los arquitectos toman menos malas decisiones si pueden reducir esta incertidumbre mediante una mejor gestión del conocimiento arquitectónico.
- A pesar de que los enfoques ágiles desalientan la documentación pesada, no registrar las decisiones de diseño y su justificación en las primeras etapas del proceso de diseño hace que sea difícil revisarlas al crear futuros prototipos.
- Documentar las cosas correctas puede ayudar con las actividades de mejora del proceso. Documentar las lecciones aprendidas es una de las razones por las que se deben realizar retrospectivas después de evaluar el prototipo entregado y antes de comenzar el siguiente incremento del programa. La reutilización de soluciones previamente exitosas a problemas arquitectónicos también es útil

4.3 Estimación de recursos

- Uno de los aspectos más controvertidos del uso de prototipos en espiral o ágiles es estimar el tiempo que llevará completar un proyecto cuando no se puede definir por completo.
- Es importante comprender antes de comenzar si tiene posibilidades razonables de entregar productos de software a tiempo y con costos aceptables antes de aceptar asumir el proyecto. Las estimaciones iniciales corren el riesgo de ser incorrectas porque el alcance del proyecto no está bien definido y es probable que cambie una vez que comience el desarrollo.
- Las estimaciones realizadas cuando el proyecto está casi terminado no proporcionan ninguna orientación para la gestión del proyecto. El truco consiste en estimar tempranamente el tiempo de desarrollo del software basándose en lo que se sabe en ese momento y revisar sus estimaciones periódicamente a medida que se agregan requisitos o después de que se entregan incrementos de software. Examinemos cómo un gerente de proyectos de software experimentado podría estimar un proyecto utilizando el modelo ágil en espiral que hemos propuesto. Las estimaciones producidas por este

- El método debería ajustarse a la cantidad de desarrolladores y la cantidad de historias de usuarios que se pueden completar simultáneamente.
- 1. Utilice datos históricos y trabaje en equipo para desarrollar una estimación de cuántos días llevará completar cada una de las historias de usuario conocidas al inicio del proyecto.
- 2. Organice de manera flexible las historias de los usuarios en los conjuntos que conformarán cada sprint1 (Sección 3.4) planea completar un prototipo.
- 3. Sume el número de días para completar cada sprint para proporcionar una estimación de la duración del proyecto total.

- 4. Revisar la estimación a medida que se agregan requisitos al proyecto o se crean prototipos. entregado y aceptado por las partes interesadas.
- Tenga en cuenta que duplicar el número de desarrolladores casi nunca reduce el desarrollo. iempo de optimización a la mitad.
- Rosa y Wallshein [Ros17] descubrieron que conocer los requisitos iniciales de software al inicio de un proyecto proporciona una estimación adecuada, pero no siempre precisa, de los tiempos de finalización del proyecto. Para obtener estimaciones más precisas, también es importante conocer el tipo de proyecto y la experiencia del equipo. Describiremos técnicas de estimación más detalladas (por ejemplo, puntos de función o puntos de casos de uso) en la cuarta parte de este libro.

4.4 Construcción del primer prototipo

- En la Sección 2.5.2 describimos la creación de prototipos como un medio para ayudar a las partes interesadas a pasar de declaraciones de objetivos generales e historias de usuarios al nivel de detalle que los desarrolladores necesitarán para implementar esta funcionalidad. Los desarrolladores pueden utilizar el primer prototipo para demostrar que su diseño arquitectónico inicial es un enfoque viable para ofrecer la funcionalidad requerida y al mismo tiempo satisfacer las limitaciones de rendimiento del cliente. Crear un prototipo operativo sugiere que la ingeniería de requisitos, el diseño de software y la construcción se desarrollen en paralelo. Este proceso se muestra en la Figura 4.1. Esta sección describe los pasos que se utilizarán para crear los primeros prototipos.
- Los detalles de las mejores prácticas para el diseño y construcción de software aparecen más adelante en este libro.

- Su primera tarea es identificar las características y funciones que son más importantes para las partes interesadas. Estos ayudarán a definir los objetivos del primer prototipo. Si las partes interesadas y los desarrolladores han creado una lista priorizada de historias de usuarios, debería ser fácil confirmar cuáles son las más importantes.
- A continuación, decida cuánto tiempo se le permitirá crear el primer prototipo. Algunos equipos pueden elegir un tiempo fijo, como un sprint de 4 semanas, para entregar cada prototipo. En este caso, los desarrolladores analizarán su estimación de tiempo y recursos y determinarán cuál de las historias de usuario de alta prioridad se puede terminar en 4 semanas. Luego, el equipo confirmaría con las partes interesadas que las historias de usuarios seleccionadas son las mejores para incluir en el primer prototipo. Un enfoque alternativo sería hacer que las partes interesadas

- y los desarrolladores eligen conjuntamente una pequeña cantidad de historias de usuarios de alta prioridad para incluirlas en el primer prototipo y utilizan sus estimaciones de tiempo y recursos para desarrollar el cronograma para completar el primer prototipo.
- Los ingenieros que trabajan en National Instruments publicaron un documento técnico que describe su proceso para la creación de un primer prototipo funcional [Nat15]. Estos pasos se pueden aplicar a una variedad de proyectos de software:

- 1. Transición del prototipo en papel al diseño de software
- 2. Prototipo de una interfaz de usuario
- 3. Crea un prototipo virtual
- 4. Agregue entradas y salidas a su prototipo
- 5. Diseñe sus algoritmos
- 6. Prueba tu prototipo
- 7. Prototipo pensando en la implementación

- En referencia a estos siete pasos, crear un prototipo en papel para un sistema es muy económico y puede realizarse en una etapa temprana del proceso de desarrollo. Los clientes y las partes interesadas no suelen ser desarrolladores experimentados. Los usuarios no técnicos a menudo pueden reconocer muy rápidamente lo que les gusta o no de una interfaz de usuario una vez que la ven esbozada. Las comunicaciones entre personas suelen estar llenas de malentendidos. Las personas se olvidan de decirse lo que realmente necesitan
- saber o asumen que todos tienen el mismo entendimiento. Crear un prototipo en papel y revisarlo con el cliente antes de realizar cualquier programación puede ayudar a evitar perder tiempo construyendo el prototipo incorrecto. Hablaremos de varios diagramas que se pueden utilizar para modelar un sistema en el Capítulo 8.

- Crear un prototipo de interfaz de usuario como parte del primer prototipo funcional es una buena idea.
- Muchos sistemas se implementan en la Web o como aplicaciones móviles y dependen en gran medida de interfaces de usuario táctiles. Los juegos de computadora y las aplicaciones de realidad virtual requieren una gran comunicación con los usuarios finales para funcionar correctamente. Si los clientes encuentran un producto de software fácil de aprender y utilizar, es más probable que lo utilicen.

- Muchos malentendidos entre desarrolladores y partes interesadas se pueden aliviar comenzando con un prototipo en papel de la interfaz de usuario. A veces, las partes interesadas necesitan ver los conceptos básicos de la interfaz de usuario en acción para poder explicar lo que realmente les gusta y lo que no les gusta de ella.
- Es menos costoso desechar un diseño inicial de interfaz de usuario que terminar el prototipo e intentar ponerle una nueva interfaz de usuario. En el Capítulo 12 se analiza el diseño de interfaces de usuario que proporcionen buenas experiencias de usuario.

- Agregar entradas y salidas a su prototipo de interfaz de usuario proporciona una manera fácil de comenzar a probar el prototipo en evolución. Las pruebas de las interfaces de los componentes de software deben realizarse antes de probar el código que constituye los algoritmos del componente. Para probar los propios
- algoritmos, los desarrolladores suelen utilizar un "marco de prueba" para garantizar que los algoritmos implementados funcionen según lo previsto. Crear un marco de prueba separado y desecharlo suele ser un mal uso de los recursos del proyecto. Si se diseña correctamente, la interfaz de usuario puede servir como marco de prueba para los algoritmos de los componentes, eliminando así el esfuerzo necesario para construir marcos de prueba separados.

- Diseñar sus algoritmos se refiere al proceso de transformar sus ideas y bocetos en código de lenguaje de programación. Es necesario considerar tanto lo funcional requisitos establecidos en la historia del usuario y las limitaciones de rendimiento (tanto explícitas como implícitas) al diseñar los algoritmos necesarios. Este es el punto donde es probable que se identifique y agregue funcionalidad de soporte adicional al alcance del proyecto, si aún no existe en una biblioteca de códigos

- probar su prototipo demuestra la funcionalidad requerida e identifica defectos aún no descubiertos antes de mostrárselo al cliente. A veces es aconsejable involucrar al cliente en el proceso de prueba antes de que se termine el prototipo para evitar desarrollar una funcionalidad
- incorrecta. El mejor momento para crear casos de prueba es durante la recopilación de requisitos o cuando se han seleccionado los casos de uso para su implementación. Las estrategias y tácticas de prueba se analizan en los capítulos 19 al 21.

- Crear prototipos teniendo en cuenta la implementación es muy importante porque le ayuda a evitar tomar atajos que conduzcan a la creación de software que será difícil de mantener en el futuro. Esto no quiere decir que cada línea de código llegue al producto de software final. Como muchas tareas creativas, desarrollar un prototipo es iterativo. Se esperan borradores y revisiones.
- A medida que se desarrolla el prototipo, debe considerar cuidadosamente las elecciones arquitectónicas del software que realiza. Es relativamente económico cambiar algunas líneas de código si detecta errores antes de la implementación. Es muy costoso cambiar la arquitectura de una aplicación de software una vez que se ha lanzado a los usuarios finales de todo el mundo.

Hogar seguro

Diseñador de habitaciones

Considerando el primer prototipo

La escena: la oficina de Doug Miller.

Los jugadores: Doug Miller, director de ingeniería de software; Jamie Lazar, miembro del equipo de software; Vinod Raman, miembro del equipo de software.

La conversación: (Un golpe en la puerta.
Jamie y Vinod entran a la oficina de Doug.)

Jamie: Doug, ¿tienes un minuto?

Doug: Claro, Jamie, ¿qué pasa?

Jamie: Hemos estado pensando en el alcance de esta herramienta de diseño de habitaciones SafeHome.

Doug: ¿Y?

Vinod: Hay mucho trabajo por hacer en la parte final del proyecto antes de que la gente pueda empezar a colocar sensores de alarma y probar diseños de muebles.

Jamie: No queremos trabajar en la parte final durante meses y luego cancelar el proyecto cuando la gente de marketing decide que odia el producto.

Doug: ¿Has intentado elaborar un prototipo en papel y revisarlo con el grupo de marketing?

Vinod: Bueno, no. Pensamos que era importante conseguir rápidamente un prototipo de computadora que funcionara y no queríamos tomarnos el tiempo para hacerlo.

Doug: Mi experiencia es que la gente necesita ver algo antes de saber si les gusta o no.

Jamie: Tal vez deberíamos dar un paso atrás y crear un prototipo en papel de la interfaz de usuario y hacer que trabajen con él y ver si les gusta el concepto.

Vinod: Supongo que no sería demasiado difícil programar una interfaz de usuario ejecutable usando el motor de juego que estábamos considerando usar para la versión de realidad virtual de la aplicación.

Doug: Suena como un plan. Pruebe ese enfoque y vea si tiene la confianza que necesita para comenzar a desarrollar su prototipo.

4.5 Evaluación del tipo de prototipo

- Los desarrolladores realizan las pruebas a medida que se construye el prototipo y se convierten en una parte importante de la evaluación del prototipo. Las pruebas demuestran que los componentes del prototipo están operativos, pero es poco probable que los casos de prueba hayan encontrado todos los defectos.
- En el modelo en espiral, los resultados de la evaluación permiten a las partes interesadas y a los desarrolladores evaluar si es deseable continuar con el desarrollo y crear el siguiente prototipo.
- Parte de esta decisión se basa en la satisfacción del usuario y de las partes interesadas, y otra parte se deriva de una evaluación de los riesgos de sobrecostos y de no entregar un producto que funcione cuando el proyecto esté terminado.
- Dam y Siang [Dam17] sugieren varios consejos de mejores prácticas para recopilar comentarios sobre su prototipo.

- 1. Ofrezca una base cuando solicite comentarios sobre el prototipo.
- 2. Pruebe su prototipo con las personas adecuadas.
- 3. Haga las preguntas correctas.
- 4. Sea neutral al presentar alternativas a los usuarios.
- 5. Adáptese durante las pruebas.
- 6. Permitir que el usuario aporte ideas.

- Proporcionar un andamiaje es un mecanismo que permite al usuario ofrecer comentarios que no generen confrontación. Los usuarios suelen ser reacios a decirles a los desarrolladores que odian el producto que están utilizando. Para evitar esto, a menudo es más fácil pedirle al usuario que brinde comentarios utilizando un marco como "Me gusta, deseo, ¿Y si?" como una forma de brindar comentarios abiertos y honestos. Las declaraciones que me gustan alientan a los usuarios a brindar comentarios positivos sobre el prototipo. Deseo que las declaraciones impulsen a los usuarios a compartir ideas sobre cómo se puede mejorar el prototipo. Estas declaraciones pueden proporcionar comentarios negativos y críticas constructivas. ¿Qué pasaría si las declaraciones animaran a los usuarios a sugerir ideas para que su equipo las explore al crear prototipos en futuras iteraciones?

- Conseguir que las personas adecuadas evalúen el prototipo es esencial para reducir el riesgo de desarrollar el producto equivocado. No es aconsejable que los miembros del equipo de desarrollo realicen todas las pruebas porque probablemente no sean representativos de la población de usuarios prevista. Es importante contar con la combinación adecuada de usuarios (p. ej., novatos, típicos y avanzados) para brindarle comentarios sobre el prototipo.

- Hacer las preguntas correctas implica que todos los interesados estén de acuerdo en objetivos prototipo. Como desarrollador, es importante mantener la mente abierta y hacer todo lo posible para convencer a los usuarios de que sus comentarios son valiosos. La retroalimentación impulsa el proceso de creación de prototipos mientras planifica futuras actividades de desarrollo de productos. Además de los comentarios generales, intente hacer preguntas específicas sobre cualquier característica nueva incluida en el prototipo.

- Ser neutral al presentar alternativas permite al equipo de software evitar que los usuarios sientan que les están “vendiendo” una forma de hacer las cosas. Si desea comentarios honestos, informe a los usuarios que aún no ha decidido que sólo hay una manera correcta de hacer las cosas. La programación sin ego es una filosofía de desarrollo que se centra en producir el mejor producto que el equipo puede crear para los usuarios previstos. Aunque no es deseable crear prototipos desechables, la programación sin ego sugiere que las cosas que no funcionan deben arreglarse o repararse.

- descartado. Así que trate de no apegarse demasiado a sus ideas al crear los primeros prototipos.
- Adaptarse durante las pruebas significa que necesita una mentalidad flexible mientras los usuarios trabajan con el prototipo. Esto podría significar modificar su plan de pruebas o realizar cambios rápidos en el prototipo y luego reiniciar las pruebas. El objetivo es obtener la mejor retroalimentación posible de los usuarios, incluida la observación directa de ellos mientras interactúan con el prototipo. Lo importante es que obtenga la retroalimentación que necesita para ayudar a decidir si construir el próximo prototipo o no.
- Permitir que los usuarios aporten ideas significa lo que dice. Asegúrese de tener una forma de registrar sus sugerencias y preguntas (de forma electrónica o de otro tipo). Discutiremos formas adicionales de realizar pruebas de usuario en el Capítulo 12.

Hogar seguro

Diseñador de habitaciones

Evaluación del primer prototipo

La escena: la oficina de Doug Miller.

Los jugadores: Doug Miller, director de ingeniería de software; Jamie Lazar, miembro del equipo de software; Vinod Raman, miembro del equipo de software.

La conversación: (Un golpe en la puerta.
Jamie y Vinod entran a la oficina de Doug.)

Jamie: Doug, ¿tienes un minuto?

Doug: Claro, Jamie, ¿qué pasa?

Jamie: Completamos la evaluación de la herramienta de diseño de salas SafeHome trabajando con nuestras partes interesadas en marketing.

Doug: ¿Cómo fueron las cosas?

Vinod: Nos centramos principalmente en la interfaz de usuario que permitirá a los usuarios colocar sensores de alarma en la habitación.

Jamie: Me alegro de que les permitiéramos revisar un prototipo en papel antes de crear el prototipo para PC.

Doug: ¿ Por qué es eso?

Vinod: Hicimos algunos cambios y a la gente de marketing le gustó más el nuevo diseño, así que ese fue el diseño que utilizamos cuando empezamos a programarlo.

Doug: Bien. ¿Cuál es el siguiente paso?

Jamie: Completamos el análisis de riesgos y, dado que no detectamos ninguna historia de usuario nueva, creemos que es razonable trabajar en la creación del próximo prototipo incremental, ya que todavía estamos a tiempo y dentro del presupuesto.

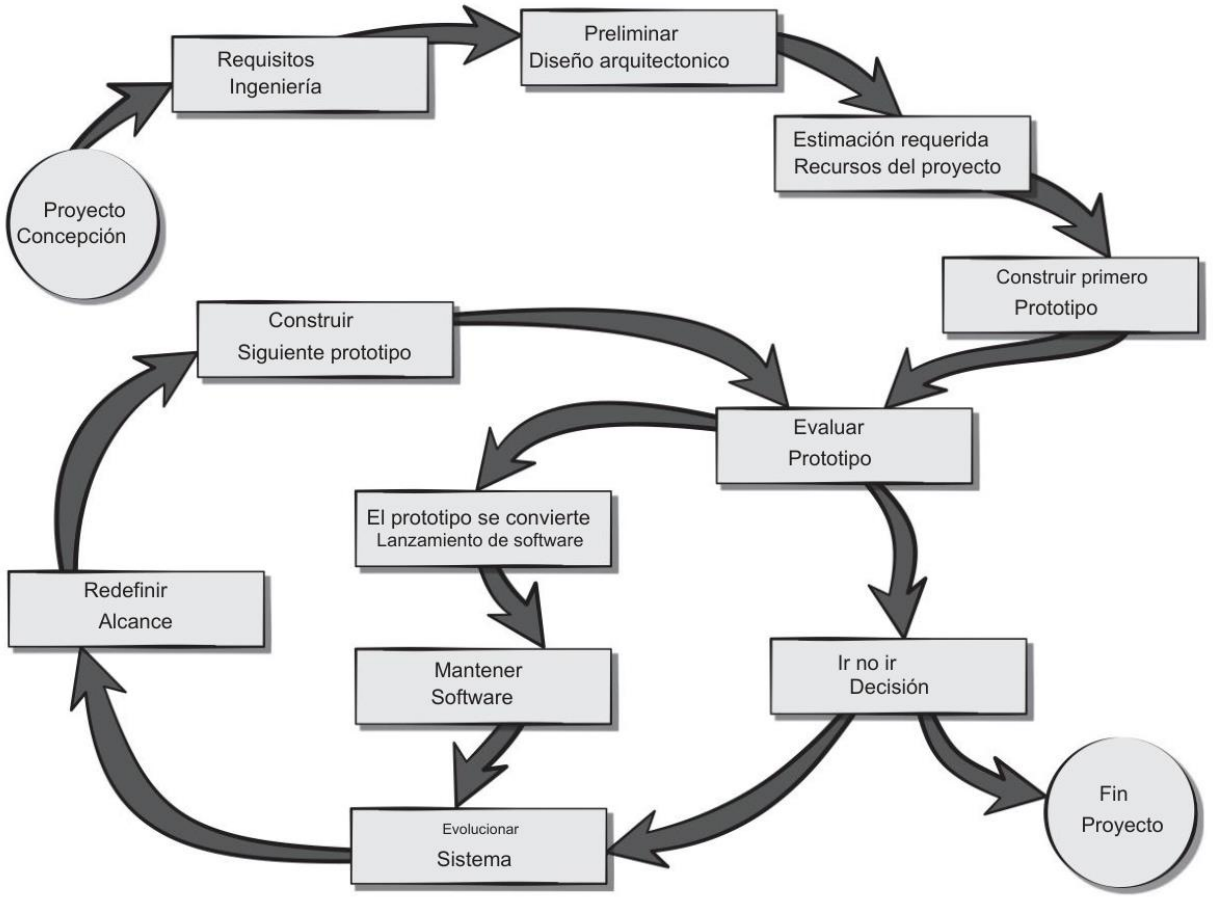
Vinod: Entonces, si está de acuerdo, reuniremos a los desarrolladores y a las partes interesadas y comenzaremos a planificar el próximo incremento de software.

Doug: Estoy de acuerdo. Mantenme informado e intenta mantener el tiempo de desarrollo del próximo prototipo en 6 semanas o menos.

4.6 Decisión de ir, no ir

- Una vez evaluado el prototipo, las partes interesadas del proyecto deciden si continúan con el desarrollo del producto de software. Si se refiere a la Figura 4.4, una decisión ligeramente diferente basada en la evaluación del prototipo podría ser entregar el prototipo al

Modelo de proceso de software recomendado



- usuarios finales y comenzar el proceso de mantenimiento. Las secciones 4.8 y 4.9 analizan esta decisión con más detalle. Discutimos el uso de la evaluación de riesgos en el modelo espiral como parte del proceso de evaluación del prototipo en la Sección 2.5.3. El primer circuito alrededor de la espiral podría usarse para solidificar los requisitos del proyecto. Pero realmente debería ser más.
- En el método que proponemos aquí, cada viaje alrededor de la espiral desarrolla un incremento significativo del producto de software final. Puede trabajar con la historia de usuario del proyecto o el trabajo pendiente de funciones para identificar un subconjunto importante del producto final para incluirlo en el primer prototipo y repetir esto para cada prototipo posterior.
- Un paso por la región de planificación sigue al proceso de evaluación. Se proponen estimaciones de costos revisadas y cambios en el cronograma con base en lo que se descubrió al evaluar el prototipo de software actual. Esto puede implicar agregar nuevas historias de usuarios o características al trabajo pendiente del proyecto a medida que se evalúa el prototipo. El riesgo de exceder el presupuesto y no cumplir con la fecha de entrega del proyecto se evalúa comparando las nuevas estimaciones de costos y tiempos con las anteriores. El riesgo de no satisfacer las expectativas del usuario también se considera y discute con las partes interesadas y, a veces, con la alta dirección antes de decidir crear otro prototipo.

- El objetivo del proceso de evaluación de riesgos es conseguir el compromiso de todas las partes interesadas y de la dirección de la empresa para proporcionar los recursos necesarios para crear el próximo prototipo. Si el compromiso no existe porque el riesgo de fracaso del proyecto es demasiado grande, entonces el proyecto puede darse por terminado. En el marco de Scrum (Sección 3.4), la decisión de ir o no podría tomarse durante la retrospectiva de Scrum.
- reunión celebrada entre la demostración del prototipo y la reunión de planificación del nuevo sprint. En todos los casos, el equipo de desarrollo expone los argumentos a los propietarios del producto y les permite decidir
- si continúan con el desarrollo del producto o no. En el Capítulo 26 se presenta una discusión más detallada de los métodos de evaluación de riesgos del software.

4.7 Evolución del tipo de prototipo

- Una vez que el equipo de desarrollo y otras partes interesadas han desarrollado y revisado un prototipo, es hora de considerar el desarrollo del siguiente prototipo. El primer paso es recopilar todos los comentarios y datos de la evaluación del prototipo actual. Luego, los desarrolladores y las partes interesadas inician negociaciones para planificar la creación de otro prototipo. Una vez que se han acordado las características deseadas para el nuevo prototipo, se consideran las limitaciones conocidas de tiempo y presupuesto, así como la viabilidad técnica de implementar el prototipo. Si los riesgos de desarrollo se consideran aceptables, el trabajo continúa.
- El modelo de proceso de creación de prototipos evolutivos se utiliza para adaptarse a los cambios que inevitablemente ocurren a medida que se desarrolla el software. Cada prototipo debe diseñarse para permitir cambios futuros y evitar desecharlo y crear el siguiente prototipo desde cero. Esto sugiere favorecer tanto las características importantes como las bien comprendidas al establecer los objetivos de cada prototipo. Como siempre, en este proceso se debe dar gran importancia a las necesidades del cliente.

4.7.1 Alcance del nuevo prototipo

- El proceso de determinar el alcance de un nuevo prototipo es como el proceso de determinar el alcance del prototipo inicial. Los desarrolladores podrían: (1) seleccionar características para desarrollar dentro del tiempo asignado a un sprint, o (2) asignar tiempo suficiente para implementar las características necesarias para satisfacer los objetivos establecidos por los desarrolladores con el aporte de las partes interesadas. Cualquiera de los enfoques requiere que los desarrolladores mantengan una lista priorizada de funciones o historias de usuarios. Las prioridades utilizadas para ordenar la lista deben estar determinadas por los objetivos establecidos para el prototipo por las partes interesadas y los desarrolladores.
- En XP (Sección 3.5.1), las partes interesadas y los desarrolladores trabajan juntos para agrupar las historias de usuarios más importantes en un prototipo que se convertirá en la próxima versión del software y determinará su fecha de finalización. En Kanban (Sección 3.5.2), los desarrolladores y las partes interesadas utilizan un tablero que les permite centrarse en el estado de finalización de cada historia de usuario. Esta es una referencia visual que se puede utilizar para ayudar a los desarrolladores a utilizar cualquier modelo de proceso de prototipo incremental para planificar y monitorear el progreso del desarrollo de software. Las partes interesadas pueden ver fácilmente la acumulación de funciones y ayudar a los desarrolladores a ordenarlas para identificar las historias más útiles necesarias en el próximo prototipo.

- Probablemente sea más fácil estimar el tiempo necesario para completar las historias de usuario seleccionadas que encontrar las historias de usuario que deben encajar en un bloque de tiempo fijo. Pero se debe seguir el consejo de mantener el tiempo de desarrollo del prototipo entre 4 y 6 semanas para garantizar una participación y retroalimentación adecuadas de las partes interesadas.

4.7.2 Construcción de nuevos prototipos

- Una historia de usuario debe contener tanto una descripción de cómo el cliente planea interactuar con el sistema para lograr un objetivo específico como una descripción de cuál es la definición de aceptación del cliente. La tarea del equipo de desarrollo es crear componentes de software adicionales para implementar las historias de usuario seleccionadas para su inclusión en el nuevo prototipo junto con los casos de prueba necesarios. Los desarrolladores deben continuar la comunicación con todas las partes interesadas mientras crean el nuevo prototipo.
- Lo que hace que este nuevo prototipo sea más complicado de construir es que los componentes de software creados para implementar nuevas características en el prototipo en evolución necesitan trabajar con los componentes utilizados para implementar las características incluidas en el prototipo anterior. Se vuelve aún más complicado si los desarrolladores necesitan eliminar componentes o modificar los que se incluyeron en el prototipo anterior porque los requisitos han cambiado. Las estrategias para gestionar estos tipos de cambios de software se analizan en el Capítulo 22.

- Es importante que los desarrolladores tomen decisiones de diseño que hagan que el prototipo de software sea más fácilmente extensible en el futuro. Los desarrolladores deben documentar las decisiones de diseño de una manera que facilite la comprensión del software al crear el próximo prototipo. El objetivo es ser ágil tanto en el desarrollo como en la documentación. Los desarrolladores deben resistir la tentación de sobrediseñar el software para acomodar características que pueden o no incluirse en el producto final.
- También deben limitar su documentación a aquello a lo que deben hacer referencia durante el desarrollo o cuando sea necesario realizar cambios en el futuro.

4.7.3 Prueba de nuevos prototipos

- Probar el nuevo prototipo debería ser relativamente sencillo si el equipo de desarrollo creara casos de prueba como parte del proceso de diseño antes de completar la programación. Cada historia de usuario debería haber tenido criterios de aceptación adjuntos cuando se creó. Estas declaraciones de aceptación deben guiar la creación de los casos de prueba destinados a ayudar a verificar que el prototipo satisface las necesidades del cliente. También será necesario probar el prototipo para detectar defectos y problemas de rendimiento.
- Una preocupación adicional de las pruebas para los prototipos evolutivos es garantizar que la adición de nuevas funciones no rompa accidentalmente las funciones que funcionaban correctamente en el prototipo anterior. La prueba de regresión es el proceso de verificar que el software que se desarrolló y probó previamente aún funciona de la misma manera después de haber sido modificado. Es importante utilizar el tiempo de prueba de forma inteligente y utilizar los casos de prueba diseñados para detectar defectos en los componentes con mayor probabilidad de verse afectados por las nuevas funciones. Las pruebas de regresión se analizan con más detalle en el Capítulo 20.

4.8 Lanzamiento del tipo de prototipo

- Cuando se aplica un proceso de creación de prototipos evolutivos, puede resultar difícil para los desarrolladores saber cuándo un producto está terminado y listo para su lanzamiento a los clientes. Los desarrolladores de software no quieren lanzar un producto de software con errores a los usuarios finales y hacer que decidan que el software es de mala calidad. Un prototipo que se considere candidato a lanzamiento debe estar sujeto a pruebas de aceptación del usuario además de las pruebas funcionales y no funcionales (de rendimiento) que se habrían realizado durante la construcción del prototipo.
- Las pruebas de aceptación del usuario se basan en los criterios de aceptación acordados que se registraron a medida que se creaba y agregaba cada historia de usuario al trabajo pendiente del producto. Esto permite

- representantes de los usuarios para verificar que el software se comporta como se esperaba y recopilar sugerencias para futuras mejoras. David Nielsen [Nie10] tiene varias sugerencias para realizar pruebas de prototipos en entornos industriales.
- Al probar una versión candidata, se deben repetir las pruebas funcionales y no funcionales utilizando los casos de prueba que se desarrollaron durante las fases de construcción de los prototipos incrementales. Se deben crear pruebas no funcionales adicionales para verificar que el desempeño del prototipo sea consistente con los puntos de referencia acordados para el producto final. Los puntos de referencia de rendimiento típicos pueden tener que ver con el tiempo de respuesta del sistema, la capacidad de datos o la usabilidad. Uno de los requisitos no funcionales más importantes que se deben verificar es garantizar que la versión candidata se ejecute en todos los entornos de tiempo de ejecución planificados y en todos los dispositivos de destino. El proceso debe centrarse en pruebas limitadas a los criterios de aceptación establecidos antes de que se creara el prototipo. Las pruebas no pueden demostrar que un producto de software esté libre de errores, solo que los casos de prueba se ejecutaron correctamente.

- Los comentarios de los usuarios durante las pruebas de aceptación deben organizarse mediante funciones visibles para el usuario, tal como se muestran en la interfaz de usuario. Los desarrolladores deben examinar el dispositivo en cuestión y realizar cambios en la pantalla de la interfaz de usuario si la implementación de estos cambios no retrasará el lanzamiento del prototipo. Si se realizan cambios, es necesario verificarlos en una segunda ronda de pruebas antes de continuar. No debe planificar más de dos iteraciones de pruebas de aceptación del usuario.
- Es importante, incluso para proyectos que utilizan modelos de procesos ágiles, utilizar un sistema de seguimiento de problemas o de informe de errores (por ejemplo, Bugzilla2 o Jira3) para capturar los resultados de las pruebas.
- Esto permite a los desarrolladores registrar fallas de prueba y facilita la identificación de los casos de prueba que deberán ejecutarse nuevamente para verificar que una reparación corrija adecuadamente el problema descubierto.
- En cada caso, los desarrolladores deben evaluar si los cambios se pueden realizar en el software sin causar sobrecostos o retrasos en la entrega del producto. Las implicaciones de no solucionar un problema deben documentarse y compartirse tanto con el cliente como con los altos directivos, quienes pueden decidir cancelar el proyecto en lugar de comprometer los recursos necesarios para entregar el proyecto final.

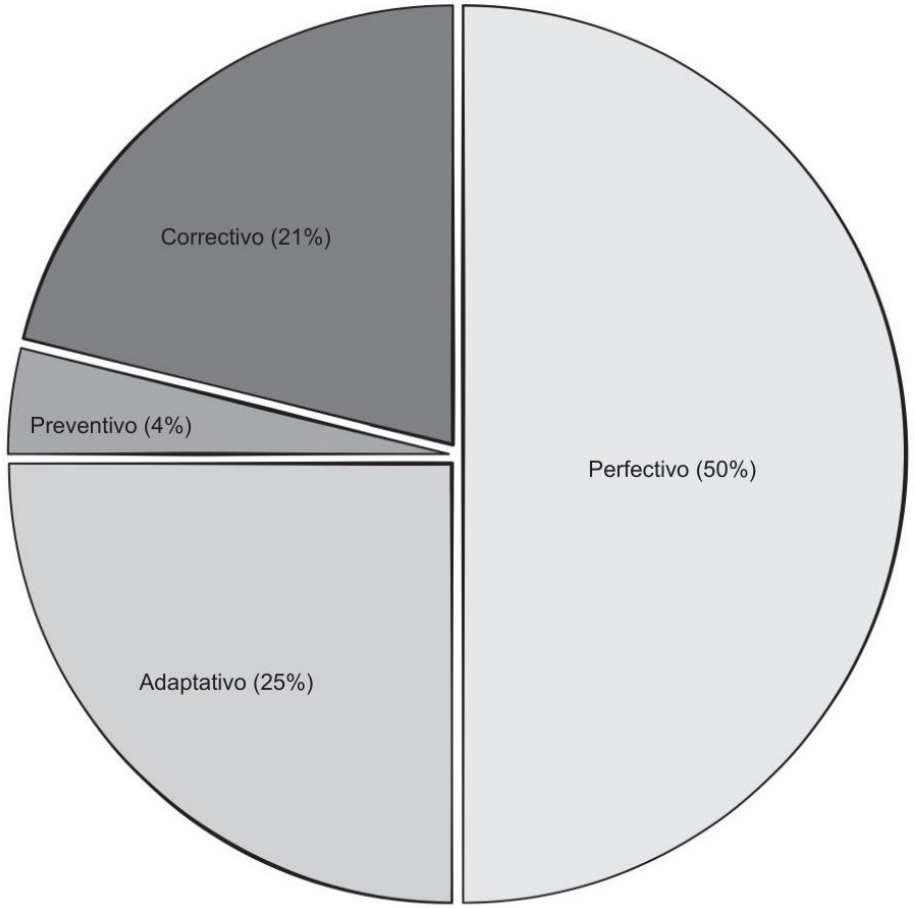
- Los problemas y las lecciones aprendidas al crear la versión candidata deben ser documentados y considerados por los desarrolladores y las partes interesadas como parte de la autopsia del proyecto. Esta información debe considerarse antes de decidir emprender el desarrollo futuro de un producto de software después de su lanzamiento a la comunidad de usuarios.
- Las lecciones aprendidas del producto actual pueden ayudar a los desarrolladores a realizar mejores estimaciones de costos y tiempos para proyectos similares en el futuro.
- Las técnicas para realizar pruebas de aceptación del usuario se analizan en los Capítulos 12 y 20 En el Capítulo 17 se presenta una discusión más detallada sobre el aseguramiento de la calidad del software.

Mantener el software de lanzamiento

- El mantenimiento se define como las actividades necesarias para mantener el software operativo después de que haya sido aceptado y entregado (liberado) en el entorno del usuario final. El mantenimiento continuará durante la vida útil del producto de software. Algunos ingenieros de software creen que la mayor parte del dinero gastado en un producto de software se gastará en actividades de mantenimiento. El mantenimiento correctivo es la modificación reactiva del software para reparar problemas.
- descubierto después de que el software ha sido entregado al usuario final del cliente. El mantenimiento adaptativo es la modificación reactiva del software después de la entrega para mantenerlo utilizable en un entorno cambiante del usuario final. El mantenimiento perfectivo es la modificación proactiva del software después de la entrega para proporcionar nuevas funciones de usuario, una mejor estructura del código del programa o una documentación mejorada. El mantenimiento preventivo es la modificación proactiva del software después de la entrega para detectar y corregir fallas del producto antes de que sean descubiertas por los usuarios en el campo [SWEBOK4]. Se puede programar y planificar el mantenimiento proactivo. El mantenimiento reactivo a menudo se describe como extinción de incendios porque no se puede planificar y debe realizarse de inmediato para los sistemas de software que son críticos para el éxito de las actividades de los usuarios finales. La Figura 4.5 muestra que sólo el 21 por ciento del tiempo de los desarrolladores normalmente se dedica al mantenimiento correctivo.

- Para un modelo de proceso evolutivo ágil como el que se describe en este capítulo, los desarrolladores lanzan soluciones parciales funcionales con la creación de cada prototipo incremental. Gran parte del trabajo de ingeniería realizado es mantenimiento preventivo o perfectivo a medida que se agregan nuevas funciones al sistema de software en evolución. Es tentador pensar que el mantenimiento se maneja simplemente planeando hacer otro viaje alrededor de la espiral.
- Pero los problemas de software no siempre se pueden anticipar, por lo que es posible que sea necesario realizar reparaciones rápidamente y los desarrolladores pueden verse tentados a tomar atajos al intentar reparar el software roto.

Distribución del esfuerzo de mantenimiento.



- software. Es posible que los desarrolladores no quieran perder tiempo en la evaluación o planificación de riesgos. Sin embargo, los desarrolladores no pueden permitirse el lujo de realizar cambios en el software sin considerar la posibilidad de que los cambios necesarios para solucionar un problema causen nuevos problemas en otras partes del programa.
- Es importante comprender el código del programa antes de realizar cambios en él. Si los desarrolladores han documentado el código, será más fácil entender si otras personas necesitan realizar el trabajo de mantenimiento. Si el software está diseñado para ampliarse, el mantenimiento también se puede realizar más fácilmente, además de las reparaciones de emergencia de Defectos
- Es esencial probar cuidadosamente el software modificado para garantizar que los cambios de software tengan el efecto deseado y no dañen el software en otros lugares.
- La tarea de crear productos de software que sean fáciles de soportar y mantener requiere una ingeniería cuidadosa y reflexiva. En el Capítulo 27 se presenta una discusión más detallada sobre la tarea de mantener y soportar el software después de la entrega.

Pasos del proceso de software recomendados

1. Ingeniería de requisitos
   - Reúna historias de usuarios de todas las partes interesadas.
   - Haga que las partes interesadas describan la aceptación Historias de usuario de criterios.
2. Diseño arquitectónico preliminar
   - Hacer uso de prototipos y modelos en papel.
   - Evaluar alternativas utilizando herramientas no funcionales. requisitos.
   - Decisiones de diseño de arquitectura documental.
3. Estimar los recursos necesarios del proyecto.
   - Utilice datos históricos para estimar el tiempo hasta Completa cada historia de usuario.
   - Organizar las historias de usuario en sprints.
   - Determinar el número de sprints necesarios para completar el producto.
   - Revisar las estimaciones de tiempo como historias de uso. se agregan o eliminan.
4. Construir el primer prototipo.
   - Seleccionar el subconjunto de historias de usuarios más importantes para las partes interesadas.
   - Crear prototipo en papel como parte del proceso de diseño.
   - Diseñar un prototipo de interfaz de usuario con entradas y salidas.
   - Diseñar los algoritmos necesarios para los primeros prototipos.
   - Prototipo pensando en la implementación.
5. Evaluar el prototipo
   - Crear casos de prueba mientras se crea el prototipo. diseñado.
   - Prototipo de prueba utilizando usuarios adecuados.
   - Capturar los comentarios de las partes interesadas para utilizarlos en el proceso de revisión.
6. Decisión de ir o no ir
   - Determinar la calidad del prototipo actual.
   - Revisar las estimaciones de tiempo y costos para completando el desarrollo.
   - Determinar el riesgo de no cumplir con las expectativas de las partes interesadas.
   - Obtener el compromiso de continuar desarrollo.
7. Evolucionar el sistema
   - Definir el alcance del nuevo prototipo.
   - Construir nuevo prototipo.
   - Evaluar nuevo prototipo e incluir pruebas de regresión.
   - Evaluar los riesgos asociados con la continuación evolución.
8. Prototipo de lanzamiento
   - Realizar pruebas de aceptación.
   - Defectos del documento identificados.
   - Compartir los riesgos de calidad con la dirección.
9. Mantener el software
   - Comprenda el código antes de realizar cambios.
   - Pruebe el software después de realizar cambios.
   - Cambios de documentos.
   - Comunicar los defectos y riesgos conocidos a todos los interesados.

4.10 Resumen

- Cada proyecto es único y cada equipo de desarrollo está formado por personas únicas.
- Todo proyecto de software necesita una hoja de ruta, y el proceso de desarrollo de software requiere un conjunto predecible de tareas básicas (comunicación, planificación, modelado, construcción e implementación). Sin embargo, estas tareas no deben realizarse de forma aislada y es posible que deban adaptarse para satisfacer las necesidades de cada nuevo proyecto. En este capítulo, sugerimos el uso de un proceso de creación de prototipos incremental y altamente interactivo. Creemos que esto es mejor que producir planes de productos rígidos y documentos grandes antes de realizar cualquier programación. Los requisitos cambian. Los aportes y comentarios de las partes interesadas deben ocurrir temprano y con frecuencia en el proceso de desarrollo para garantizar la entrega de un producto útil.

- Sugerimos el uso de un modelo de proceso evolutivo que enfatice la participación frecuente de las partes interesadas en la creación y evaluación de prototipos de software incrementales. Limitar los artefactos de ingeniería de requisitos al conjunto mínimo de documentos y modelos útiles permite la producción temprana de prototipos y casos de prueba. La planificación para crear prototipos evolutivos reduce el tiempo perdido repitiendo el trabajo necesario para crear prototipos desechables.
- El uso de prototipos en papel en las primeras etapas del proceso de diseño también puede ayudar a evitar programar productos que no satisfagan las expectativas del cliente. También es importante realizar el diseño arquitectónico justo antes de comenzar el desarrollo real para evitar retrasos en el cronograma y sobrecostos.
- La planificación es importante, pero debe realizarse con rapidez para evitar retrasar el inicio del desarrollo. Los desarrolladores deben tener una idea general de cuánto tiempo llevará completar un proyecto, pero deben reconocer que es probable que no conozcan todos los requisitos del proyecto hasta que se entreguen los productos de software. Los desarrolladores harían bien en evitar una planificación detallada que vaya más allá de la planificación del prototipo actual.
- Los desarrolladores y las partes interesadas deben adoptar un proceso para agregar características que se implementarán en futuros prototipos y evaluar el impacto de estos cambios en el cronograma y el presupuesto del proyecto.

- La evaluación de riesgos y las pruebas de aceptación son una parte importante del proceso de evaluación del prototipo. También es importante tener una filosofía ágil sobre la gestión de requisitos y la adición de nuevas funciones al producto final. El mayor desafío que enfrentan los desarrolladores con los modelos de procesos evolutivos es gestionar la variación del alcance y al mismo tiempo entregar un producto que cumpla con las expectativas del cliente y hacer todo esto entregando el producto a tiempo y dentro del presupuesto. Eso es lo que hace que la ingeniería de software sea tan desafiante y gratificante.

Actividad

- Analice las ventajas y desventajas entre los diversos flujos de proceso y plasmarlo en un cuadro comparativo.

¿Preguntas o dudas?

- ¿Qué me llevo de la clase?

Referencias

- Roger s. Pressman (2020). Process Models. Software Engineering. (pp 53–69). Mc Graw Gill.

GRACIAS

Ingeniería de Sistemas
Universidad Nacional de Cajamarca