La mayoría de los proyectos de software no cumplen con las expectativas, el presupuesto o los plazos del usuario final. Esto causa una dificultad innecesaria para los desarrolladores, probadores, gerentes y, en última instancia, para los clientes que dependen del producto de software.

# Comentarios.

Protagonistas:

Brooke ---- Product manager

Paul ---- director de marketing

Conner --- Development Manager

Jade -------- Lead developer

Tom -------- software developer

Denise -------- QA manager.

**Episode 1: The Product Strategy Meeting**

Brooke, Connor y Paul tienen una reunion para definer la estrategia de desarrollo de software, done Paul esta muy confiado.

**Episode 2: The Development Team Meets.**

Brooke y Connor conocen a Jane y Tom quienes cominenzan a desarrollar de una vez la aplicacion, con la idea de hacer todo rapido y desechar lo que sobre.

Episode 3: The Prototype Presentation

Brooke, Paul y Connor hace una simple presentacion y todavia no tienen claro a donde va a parar el Proyecto tienen un producto bien visivamente ma sin ninguna funcionalidad.

Episode 4: One Month Later

Los protagonist han trabajado mucho pero estan encontando muchos problemas con el desarrollo de la aplicacion con lo que preventivamente no habian tomado en cuenta, Brooke piensa que seria major pedir ayuda externa para poder respetar la fecha de entrega.

Episode 5: Enter Alex the Contractor

La agencia manda a Alex un desarrollador que pregunta por la documentacion del proyecto y descubre que esta desactualizada y muy ambigua y se da cuenta en el desastre en el que se metio y quiere revisar los termino de su contrato.

Episode 6: Six Weeks Later

Envia la el trabajo a calidad quienes lo rechazan y quieren aportar modificaciones.

Episode 7: Nearing the Deadline

La fecha de entrega se acerca pero el equipo se da cuenta de no haber tomado deciciones correcta para la tecnoogia a utilizar y para efectuar los cambios definidos por calidad necesitan mas tiempo.

Episode 8: "What's done is done."

Empiezan a cometer errore por el cansancio y el estress y sovrescriven codigo sin poder dar roolback.

Epilogue

El producto se entrega en su ultima version pero crea descontento entre los usuario por la falta de opciones.

Jane y Tom cominenzan a desarrollar la aplicacion de forma muy rapida, sin saber que direccion va a tomar el Proyecto y sin tener bien actualizada la documentacion del Proyecto terminando con no tener suficiente tiempo para terminar el Proyecto y publicando una version incomplete.

Principles of Rational Unified Process

Principios:

Adapt the Process

Patrones (comportamientos que permiten este principio y pueden ayudar a los proyectos de desarrollo de software):  
Tamaño correcto del proceso a las necesidades del proyecto  
Adapte la ceremonia de proceso a la fase del ciclo de vida y permita que la formalidad evolucione de leve a fuerte a medida que se resuelven las incertidumbres  
Mejore el proceso continuamente  
Balancee los planes y las estimaciones con el nivel de incertidumbre  
Anti-Patrones (comportamientos contrarios a este principio y pueden dañar los proyectos de desarrollo de software):  
Siempre vea más procesos y una planificación inicial más detallada como mejor  
Forzar estimaciones tempranas y atenerse a esas estimaciones  
Desarrolle planes precisos y administre el proyecto haciendo un seguimiento contra el plan estático  
Utilice siempre el mismo grado de proceso durante todo el ciclo de vida.

Balance Business Needs and User Needs

Capturar procesos comerciales  
Priorizar los proyectos y las capacidades del software que respaldan las necesidades comerciales.

Involucre al cliente para asegurarse de que comprende sus necesidades usando:

Desarrollo impulsado por el uso del caso

Diseño centrado en el usuario.

Collaborate Across Teams

Motivar a las personas para que rindan al máximo y auto gestionados  
Fomentar la colaboración multifuncional (por ejemplo, analistas, desarrolladores, probadores)  
Proporcionar entornos colaborativos efectivos  
Administre artefactos y tareas en evolución para mejorar la colaboración, el progreso y la visión de calidad con entornos integrados.  
Integrar negocios, software y equipos de operación

Anti-Patrones:   
Nutrir a los desarrolladores heroicos dispuestos a trabajar horas extremadamente largas, incluidos los fines de semana  
Tener personas altamente especializadas equipadas con potentes herramientas para hacer su trabajo, con una colaboración limitada entre los diferentes miembros del equipo, y una integración limitada entre las diferentes herramientas  
Suponiendo que si todos hacen su trabajo, el resultado final será bueno.

**Communication and Collaboration.**

Motive a las personas del equipo para que rindan al máximo.

Las personas que se sienten realmente responsables por el resultado final están motivadas para hacer un buen trabajo

Fomentar la colaboración multifuncional.

Romper las paredes que a menudo existen entre

Analistas, desarrolladores y probadores

Equipos que dirigen el negocio, desarrollan aplicaciones e implementan las aplicaciones

Demonstrate Value Iteratively

Habilite la retroalimentación entregando valor de usuario incremental en cada iteración  
Adapta tus planes usando un proceso iterativo  
Acepta y gestiona el cambio  
Abordar temprano los principales riesgos técnicos, comerciales y programáticos.

**Elevate Level of Abstraction**

**Patterns**

Reutilizar activos existentes

Use herramientas e idiomas de alto nivel para reducir la cantidad de documentación producida

Centrarse en la arquitectura primero

Arquitecto de resiliencia, calidad, comprensibilidad y control de la complejidad.

**RUP:** Un conjunto subyacente de principios para el desarrollo de software exitoso.

Estos principios son la base sobre la cual se ha desarrollado el RUP.

Un marco de contenido de métodos reutilizables y bloques de creación de procesos.

Una familia de plug-ins de métodos define un marco de trabajo de métodos desde el que puede crear sus propias configuraciones de métodos y procesos personalizados.

El método subyacente y el lenguaje de definición de proceso.

Un meta-modelo de arquitectura de método unificado que proporciona un lenguaje para describir el contenido y los procesos del método.

RUP plugin.

* Rational Unified Process
* Base Concepts
* RUP Formal Resources
* RUP Informal Resources
* Business Modeling
* Service-Oriented Architecture
* RUP for J2EE
* Rational Software Architect
* Legacy Evolution

¿Por qué debería usar RUP?

RUP proporciona un profesional de desarrollo de software con un entorno de proceso configurable basado en estándares. Ese entorno de proceso:

Permite que un método personalizado se publique y se ponga a disposición de todo el equipo del proyecto

Permite que ese método se configure para adaptarse a las necesidades únicas de cada proyecto

Proporciona a cada usuario un filtrado personalizado

Proporciona un conjunto recopilado de prácticas de ingeniería de software que se mejoran continuamente de forma periódica para reflejar los cambios en las prácticas de la industria.

Los primeros ancestros de RUP son el Enfoque Racional y el Proceso de Objeción, desde principios de los años noventa. Los dos procesos se fusionaron en el proceso de objeción racional en 1996. El proceso de objeción proporcionó la estructura básica del proceso y el concepto del caso de uso. Desde el lado racional, el proceso hereda el desarrollo iterativo y su enfoque arquitectónico.

Summary:

RUP es: Un conjunto subyacente de filosofías y principios para el desarrollo exitoso de software.  
Un marco de contenido de métodos reutilizables y bloques de creación de procesos.  
El método subyacente y el lenguaje de definición de proceso.  
Los principios clave para el desarrollo impulsado por las empresas:  
Adapta el proceso  
Equilibrar las prioridades de las partes interesadas en competencia  
Colabora entre equipos  
Demostrar valor iterativamente  
Elevar el nivel de abstracción  
Enfocarse continuamente en la calidad

¿Cuáles son los factores que pueden causar la adición de iteraciones?  
Inicio: trabajando con nuevas funcionalidades, entorno de negocios desconocido, alcance altamente volátil, decisiones de make-buy  
Elaboración: trabajar con un nuevo entorno de sistema (nuevas características arquitectónicas), elementos arquitectónicos no probados, necesidad de prototipos del sistema  
Construcción: muchos códigos para escribir y verificar  
Transición: necesidad de alfas y versiones beta, conversiones de la base de clientes, entrega incremental a los clientes

Ahora ha visto las cuatro fases utilizadas en RUP para representar el proceso iterativo a lo largo del tiempo: Inicio, Elaboración, Construcción y Transición. Ha visto que cada fase contiene hitos de fase que son puntos en los que se toman decisiones sobre si el proyecto está listo para continuar según lo planificado para la siguiente fase o si se necesitan modificaciones.

También ha visto que cada fase contiene iteraciones, y que el número de iteraciones en cada fase es una decisión tomada al considerar diferentes factores sobre ese proyecto en particular.

El proceso de entrega del ciclo de vida de RUP clásico es una descripción integral de un método de desarrollo iterativo. Contiene una estructura jerárquica de fases, iteraciones, actividades y subactividades. Se puede ver por tareas (vista de proceso de estructura de desglose de trabajo), roles (vista de proceso de asignación de equipo) o productos de trabajo (vista de proceso de uso de producto de trabajo).

En el próximo módulo, veremos brevemente cómo se puede personalizar RUP.

# Principles of Analysis I with UML 2.0

**A use-case specification** es el artefacto más valioso para descubrir clases. Verá cómo se descubren las clases utilizando este artefacto más adelante en este módulo.  
  
**A glossary** define términos importantes para el proyecto. Muchos de estos términos pueden usarse como clases.  
  
**A stakeholder request** **document** incluye las solicitudes que un interesado podría tener sobre el sistema que se desarrollará. A menudo contiene referencias a fuentes externas a las que el sistema debe cumplir.  
  
**A vision document** incluye requisitos de alto nivel para el proyecto. Este documento proporciona una base contractual para requisitos más detallados. A menudo, este documento es una buena fuente para las clases.

Boundary Class?

Una clase de límite se utiliza para modelar la interacción entre el entorno del sistema y su funcionamiento interno. Tal interacción transforma y traduce los cambios de eventos y notas en la presentación del sistema (como la interfaz).

Las clases de límites se pueden usar para capturar los requisitos en una interfaz de usuario. Debido a que muchas interfaces de usuario creadas hoy en día están orientadas a objetos, y la tendencia apunta hacia una orientación aún más objetiva, vale la pena hacer un modelo de objetos de la interfaz de usuario.

Aunque muchas decisiones de diseño de interfaz de usuario se realizan mejor durante el prototipado y el rápido desarrollo de la interfaz de usuario, el razonamiento sobre la estructura y los requisitos de usabilidad en la interfaz de usuario es natural en términos de un modelo de objetos.

Use una clase de límite para representar una ventana principal.  
  
Utilice una clase de límite para representar cada objeto (tipo) lógico y relevante manipulado por el actor dentro de la ventana principal. Recuerde que dichos objetos a menudo tienen sus propias ventanas secundarias, como ventanas de propiedades.  
  
Use las responsabilidades en las clases de límites para describir las operaciones requeridas de sus ventanas y objetos correspondientes.  
Use atributos en las clases de límites para describir las propiedades de sus ventanas y objetos correspondientes.  
Utilice relaciones entre clases de límites para representar relaciones, como jerarquías de agregación y rutas de navegación, de sus ventanas y objetos correspondientes.  
  
Use requisitos especiales en las clases de límites para capturar los requisitos de usabilidad y otros tipos de requisitos, como los requisitos de diseño e implementación, de sus ventanas y objetos correspondientes.

Haga bocetos o use capturas de pantalla de un prototipo de interfaz de usuario para ilustrar el comportamiento y la apariencia de los objetos de contorno.

Modele solo las abstracciones clave del sistema. No modele cada botón, lista y widget en la GUI. El objetivo del análisis es formar una buena imagen de cómo se compone el sistema, no para diseñar hasta el último detalle. En otras palabras, identifique las clases de límites solo para los fenómenos en el sistema o para las cosas mencionadas en el flujo de eventos del caso de uso.

# A control class controls the behavior of a use case.

Una clase de control se puede definir como un tipo de clase utilizado para modelar el comportamiento de control específico para uno o unos pocos casos de uso. Los objetos de control (instancias de clases de control) a menudo controlan otros objetos. Por lo tanto, se comportan como coordinadores. Las clases de control encapsulan el comportamiento específico del caso de uso.

A diferencia de una clase de límite, una clase de control es independiente del entorno. La misma clase de control se puede usar para coordinar muchas clases de límites diferentes. Cuando el sistema realiza el caso de uso, se crea un objeto de control. Los objetos de control generalmente mueren cuando se ha realizado su caso de uso correspondiente.

Las clases de control deberían hacer solo la secuencia.

Una clase de control debería decirles a otras clases que "hagan algo" y nunca deberían "hacer nada", excepto para dirigir.

Las clases de control se utilizan para desacoplar las clases de frontera y entidad.

# Una clase de entidad modela los conceptos clave del sistema.

Usualmente modela información que es duradera (persistente)

Es independiente del entorno

Puede ser utilizado en casos de uso múltiple

Una clase de entidad se puede definir como un tipo de clase utilizado para modelar información y comportamientos asociados que deben almacenarse.

Los objetos de entidad (instancias de clases de entidad) se usan para mantener y actualizar información sobre algún fenómeno, como un evento, una persona o un objeto de la vida real. Por lo general, son persistentes y tienen atributos y relaciones necesarios durante un período prolongado, a veces durante la vida del sistema.

Las fuentes frecuentes de inspiración para las clases de entidad son:

Glossary.

Business domain model.

Las clases deben descubrirse en los artefactos utilizados por el proyecto.  
Las clases deben reflejar el dominio comercial modelado.  
Cada clase debe tener un nombre que lo distinga de otras clases.  
Una responsabilidad es un contrato u obligación de una clase. Cada clase debe tener varias responsabilidades.

CRC card contains:

Nombre de la clase y descripción

Responsabilidades de la clase

Colaboradores para las responsabilidades

Se puede organizar físicamente para representar colaboraciones

Puede ayudar a identificar jerarquías de generalización entre las clases

Son más efectivos para grupos con técnicas nuevas de orientación a objetos, ya que las tarjetas evitan enfocarse en el diseño de procedimientos, previenen la generalización prematura y fomentan el "Object Think"

**Resumen Principles of Analysis II with UML 2.0**

Diagramas de interacción

El UML proporciona dos tipos de diagramas de interacción que modelan las interacciones de objetos. Son diagramas de secuencia y comunicación.

Un diagrama de interacción muestra una interacción, que consiste en un conjunto de objetos y sus relaciones, incluidos los mensajes que pueden enviarse entre ellos.

Diagramas de secuencia

Un diagrama de secuencia enfatiza el orden de tiempo de los mensajes. Los contenidos de un diagrama de secuencia incluyen:

Objetos

Actores

Mensajes

Ocurrencia de ejecución

Objeto Lifeline

Diagramas de comunicación

Un diagrama de comunicación muestra cómo los objetos interactúan para realizar el comportamiento de un caso de uso particular o una parte de un caso de uso. Los contenidos de un diagrama de comunicación incluyen:

Objetos

Actores

Mensajes

Links

Similitudes y diferencias

Los diagramas de secuencia y comunicación son semánticamente equivalentes. Ambos modelan los aspectos dinámicos de un sistema, y ambos modelan un escenario de caso de uso.

Los diagramas de secuencia y comunicación expresan información similar pero la muestran de diferentes maneras.

Cuando se modela la vista estática de un sistema, los diagramas de clases se usan generalmente en una de las tres formas de modelar. Se usan para modelar:

El vocabulario de un sistema

Colaboraciones simples

Un esquema de base de datos lógica

Los diagramas de clases le permiten modelar el vocabulario de su sistema porque determinó las abstracciones que forman parte o están fuera de los límites de su sistema. Los diagramas de clases especifican estas abstracciones y sus responsabilidades.

Una colaboración es una agrupación de clases y otros elementos que trabajan en conjunto para proporcionar una solución que es más grande que la suma de los elementos en la colaboración. Ninguna clase está sola. Funciona en colaboración con otros elementos para llevar a cabo algún tipo de solución. Los diagramas de clase le permiten modelar estas colaboraciones.

Un esquema de base de datos es similar a los planos para el diseño conceptual de una base de datos. Muchos de los sistemas que diseñará tienen objetos persistentes, lo que significa que deberán almacenarse en una base de datos para su posterior recuperación. Puede modelar esquemas para estas bases de datos usando diagramas de clase. Los diagramas de clase de UML son un superconjunto de diagramas de relación de entidad (E-R). Sin embargo, donde los diagramas típicos de E-R se enfocan solo en los datos, los diagramas de clases van un paso más allá y permiten modelar el comportamiento también. El comportamiento, modelado como operaciones, generalmente se convierte en desencadenantes o procedimientos almacenados en la base de datos.

No es inusual que un sistema en desarrollo contenga cientos o incluso miles de clases diferentes. Manejar números tan grandes genera su propio problema. ¿Cómo se pueden organizar estas clases y no perder la organización del modelo? Puede organizar estas clases con paquetes: "bolsas virtuales" de propósito general para organizar elementos en grupos.

## Preguntas importantes para el examen.

* ¿Los cambios en una clase requieren cambios en otra.
* Si
* ¿Si la clase específica es "eliminada" del sistema, ¿se afectan seriamente otras clases?
* Las clases que se vuelven superfluas como resultado de eliminar la clase específica de alguna manera están conectadas a ella.
* ¿Las clases interactúan con una gran cantidad de mensajes o comparten una comunicación complicada?
* Si
* ¿Las clases interactúan de una clase de límite para presentar los detalles de una clase de entidad?
* Si