

Ingeniería de Software II

RUP – Proceso Unificado de Rational

Universidad Autónoma de Entre Ríos
Facultad de Ciencia y Tecnología

Lic. Paolo Orundés Cardinali
orundescardinali.paolo@uader.edu.ar

Lic. Damián Cian
cian.damian@uader.edu.ar

¿Qué es RUP?

El **Proceso Racional Unificado** o RUP (por sus siglas en inglés de Rational Unified Process) es un proceso de desarrollo de software desarrollado por la empresa Rational Software, actualmente propiedad de IBM. Junto con el Lenguaje Unificado de Modelado (UML), constituye la metodología estándar más utilizada para el análisis, diseño, implementación y documentación de sistemas orientados a objetos.

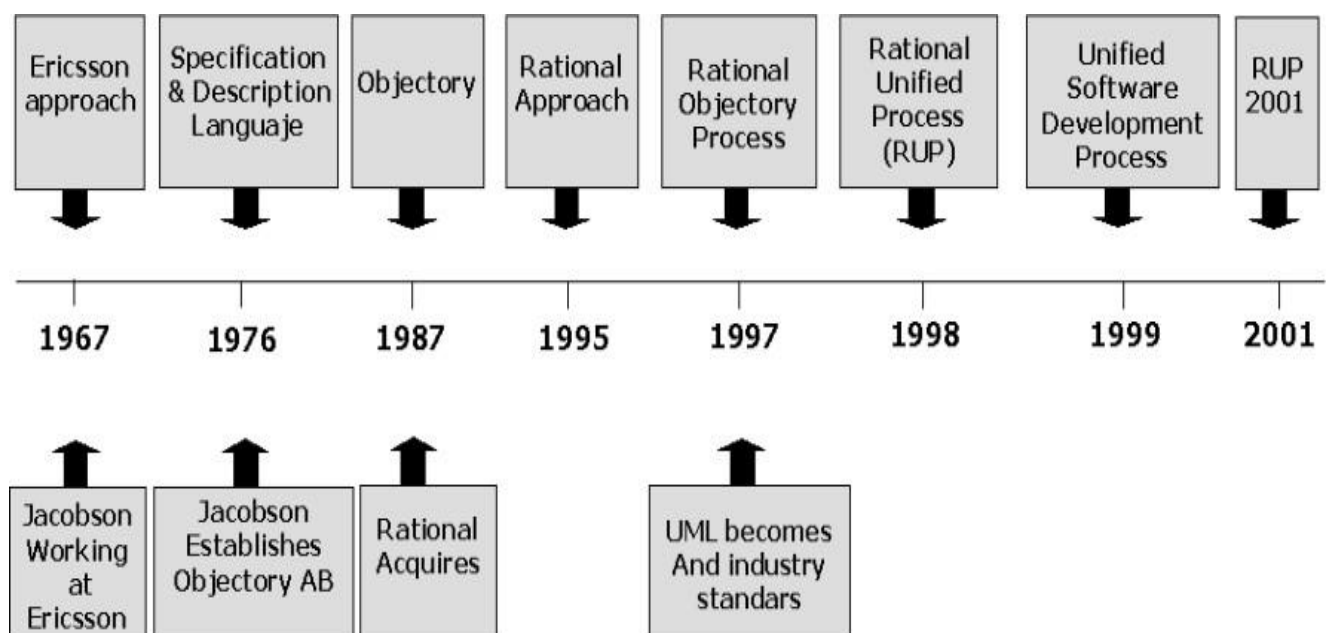
El RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

Aunque RUP se utiliza para proyectos complejos y con equipos extensos, permite realizar actividades y artefactos de acuerdo con la **elección del equipo** y se puede adaptar para **agilizar el proceso**.



Un poco de historia

En la siguiente figura se ilustra la historia de RUP. El antecedente más importante se ubica en 1967 con la Metodología Ericsson (Ericsson Approach) elaborada por Ivar Jacobson, una aproximación de desarrollo basada en componentes, que introdujo el concepto de Caso de Uso. Entre los años de 1987 a 1995 Jacobson fundó la compañía Objectory AB y lanza el proceso de desarrollo Objectory (abreviación de Object Factory).



Posteriormente en 1995 Rational Software Corporation adquiere Objectory AB y entre 1995 y 1997 se desarrolla Rational Objectory Process (ROP) a partir de Objectory 3.8 y del Enfoque Rational (Rational Approach) adoptando UML como lenguaje de modelado. Desde ese entonces y a la cabeza de Grady Booch, Ivar Jacobson y James Rumbaugh, Rational Software desarrolló e incorporó diversos elementos para expandir ROP, destacándose especialmente el flujo de trabajo conocido como modelado del negocio. En junio del 1998 se lanza Rational Unified Process.

Características clave del RUP

Orientado a casos de uso

Se centra en capturar los requisitos funcionales del sistema a través de **casos de uso**, que describen cómo interactúan los usuarios con el sistema.

Centrado en la arquitectura

Promueve el diseño temprano de una arquitectura robusta como base del sistema.

Iterativo e incremental

El desarrollo se hace en **iteraciones** (ciclos cortos de desarrollo) que permiten obtener versiones parciales del sistema, probarlas, y mejorarlas.



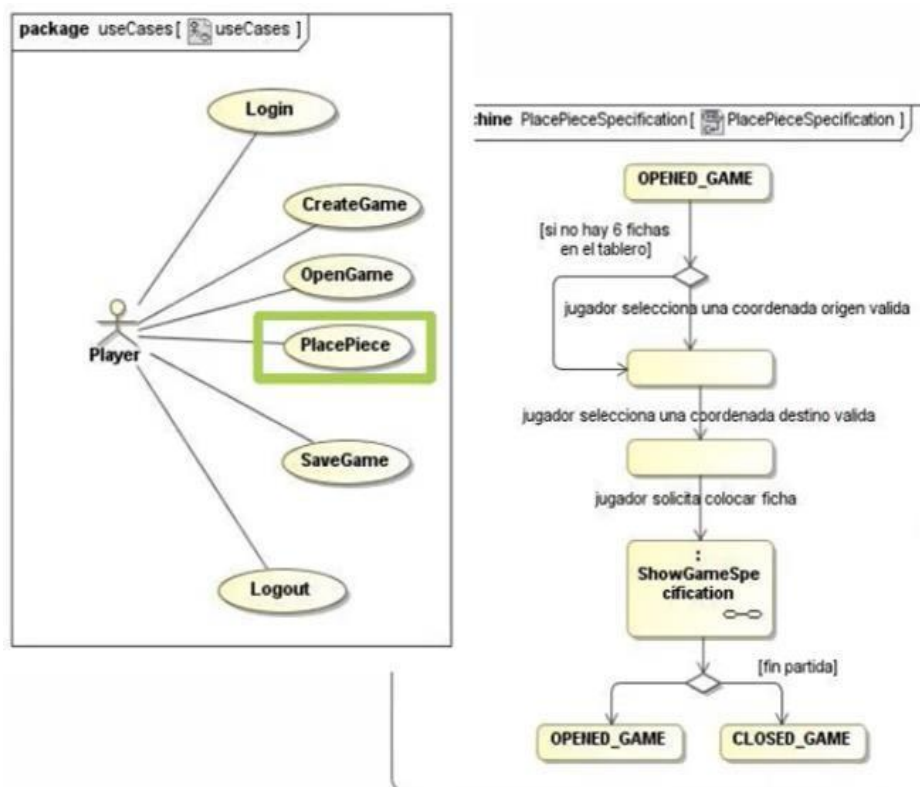
Orientado a Casos de Uso

- Un **caso de uso** es un fragmento de funcionalidad del sistema que proporciona un resultado de valor a un usuario. **Los casos de uso modelan los requerimientos funcionales del sistema.**
- Todos los casos de uso juntos constituyen el **modelo de casos de uso**.
- Los casos de uso también **guían el proceso de desarrollo** (diseño, implementación, y prueba).

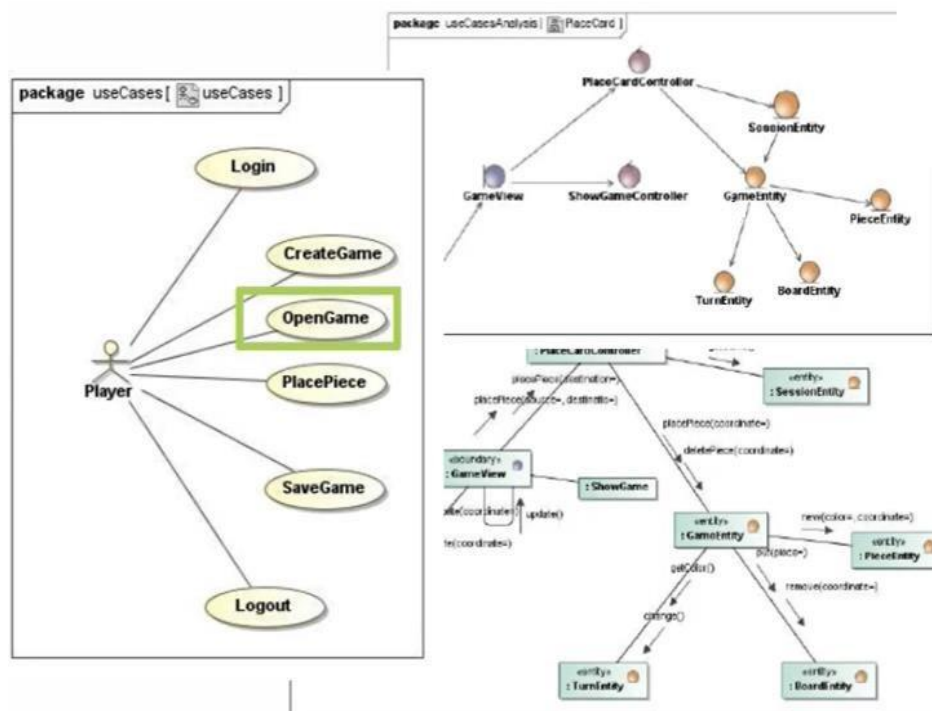
Basándose en los casos de uso los desarrolladores crean una serie de modelos de diseño e implementación que llevan a cabo los casos de uso. De este modo los casos de uso no solo inician el proceso de desarrollo, sino que le proporcionan un hilo conductor, avanza a través de una serie de flujos de trabajo que parten de los casos de uso

Ejemplo

Comenzamos con un caso de uso y una especificación para analizar los requisitos



Luego Una vez elaborados los requerimientos de los casos de uso, se analizará y se realizarán diagramas de clases, colaboración, etc para repartir responsabilidades, de manera de saber que gestionar.



Y así vamos modelando los diferentes casos de uso para continuar con el diseño, con las pruebas, etc.

Centrado en la Arquitectura

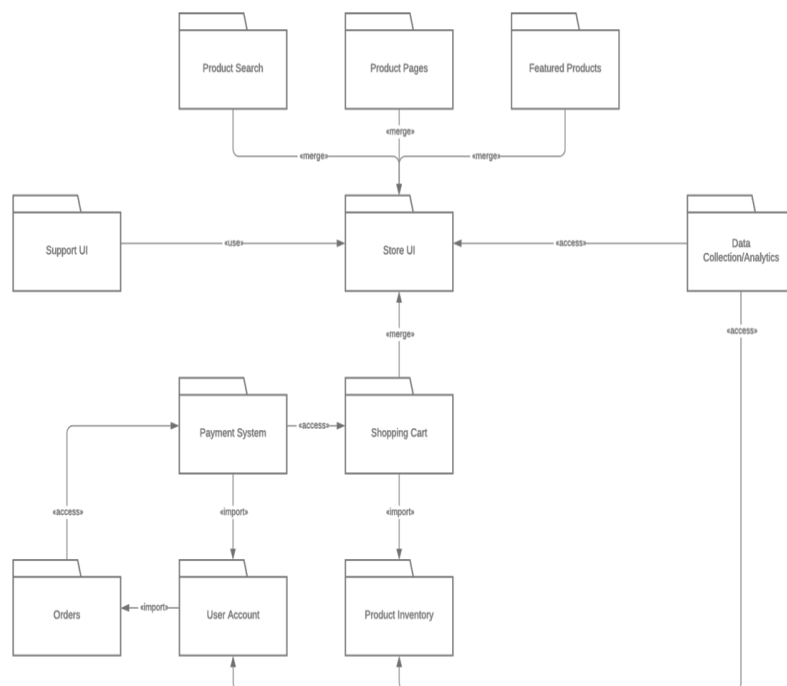
La arquitectura de un sistema software se describe mediante diferentes vistas del sistema en construcción. El concepto de arquitectura software incluye los aspectos estáticos y dinámicos más significativos del sistema. La arquitectura es una vista del diseño completo con las características más importantes resaltadas, dejando los detalles de lado.

Arquitectura: Conjunto de decisiones significativas acerca de la organización de un sistema software, la selección de los elementos estructurales a partir de los cuales se compone el sistema, las interfaces entre ellos, su comportamiento, sus colaboraciones, y su composición.

Los casos de uso y la arquitectura están profundamente relacionados. Los casos de uso deben encajar en la arquitectura, y a su vez la arquitectura debe permitir el desarrollo de todos los casos de uso requeridos, actualmente y a futuro. El arquitecto desarrolla la forma o arquitectura a partir de la comprensión de un conjunto reducido de casos de uso fundamentales o críticos (usualmente no más del 10 % del total). En forma resumida, podemos decir que el arquitecto:

- Crea un esquema en borrador de la arquitectura comenzando por la parte no específica de los casos de uso (por ejemplo, la plataforma) pero con una comprensión general de los casos de uso fundamentales.
- A continuación, trabaja con un conjunto de casos de uso claves o fundamentales. Cada caso de uso es especificado en detalle y realizado en términos de subsistemas, clases, y componentes.
- A medida que los casos de uso se especifican y maduran, se descubre más de la arquitectura, y esto a su vez lleva a la maduración de más casos de uso.

Ejemplo de arquitectura a través de un diagrama de paquetes



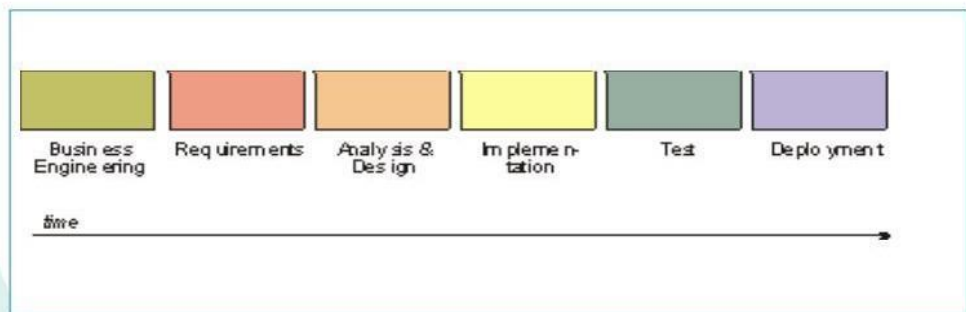
Iterativo e Incremental

Es práctico dividir el esfuerzo de desarrollo de un proyecto de software en partes más pequeñas o mini proyectos. Cada mini proyecto es **una iteración que resulta en un incremento**. Las iteraciones hacen referencia a pasos en el flujo de trabajo, y los incrementos a crecimientos en el producto.

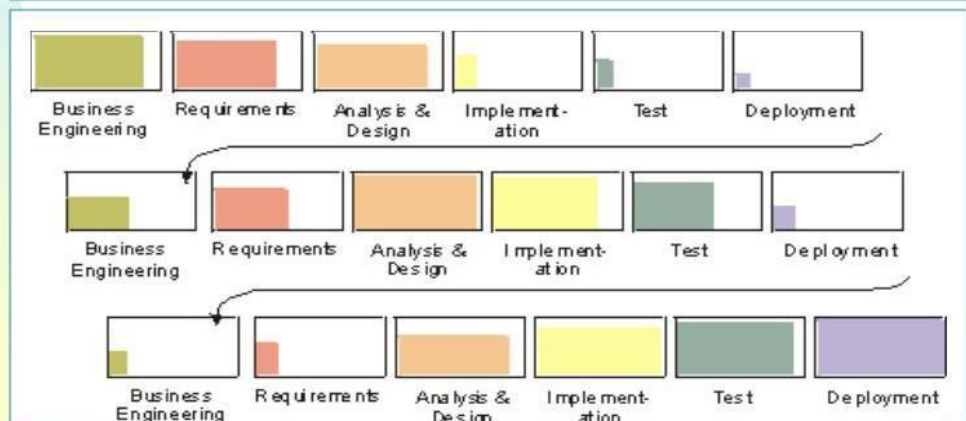
Las iteraciones deben estar controladas. Esto significa que deben seleccionarse y ejecutarse de una forma planificada. Los desarrolladores basan la selección de lo que implementarán en cada iteración en dos cosas: el conjunto de casos de uso que amplían la funcionalidad, y en los riesgos más importantes que deben mitigarse. En cada iteración los desarrolladores identifican y especifican los casos de uso relevantes, crean un diseño utilizando la arquitectura seleccionada como guía, para implementar dichos casos de uso. Si la iteración cumple sus objetivos, se continúa con la próxima. Si no deben revisarse las decisiones previas y probar un nuevo enfoque.

Proceso Iterativo e Incremental

Enfoque
Secuencial

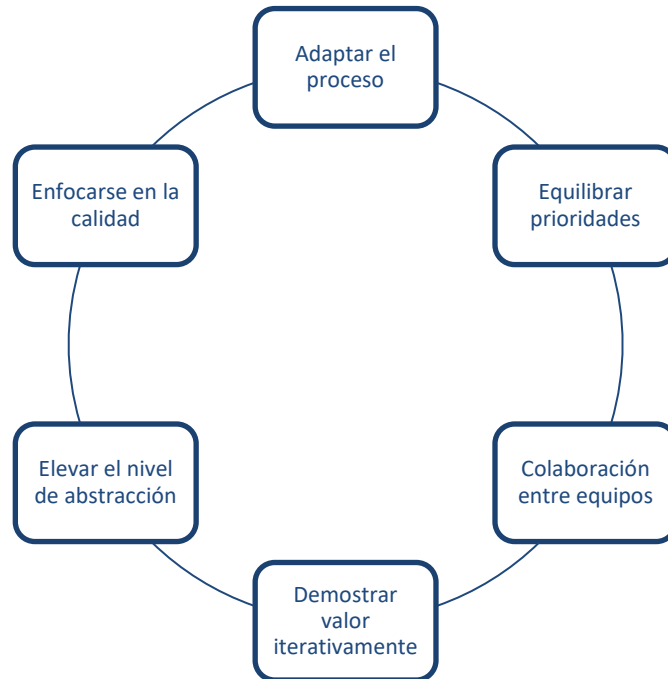


Enfoque
Iterativo e
Incremental



Principios de desarrollo

La Filosofía del RUP está basada en 6 principios claves



Adaptar el proceso

El proceso deberá adaptarse a las necesidades del cliente ya que es muy importante interactuar con él. Las características propias del proyecto, el tamaño del mismo, así como su tipo o las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto.

Adaptar el ciclo de vida

- Al comienzo del proyecto se debe ser menos riguroso con los procesos
- Mejorar los procesos continuamente
- Evaluar después de cada iteración
- Lecciones aprendidas

Factores que condicionan el adaptar el proceso

- Tamaño del proyecto
- Las distribuciones de los equipos
- La complejidad de la tecnología
- El número de interesados
- El cumplimiento de los requisitos
- La posición en el ciclo vital del proyecto

Equilibrar prioridades

Los requisitos de los diversos participantes pueden ser diferentes, contradictorios o disputarse recursos limitados. Debe poder encontrarse un equilibrio que satisfaga los deseos de todos. Gracias a este equilibrio se podrán corregir desacuerdos que surjan en el futuro. Al igual esta metodología está acorde con el Lenguaje Unificado de Modelado (UML).

Conflicto

- Más alcance
- Menos tiempo
- Menos dinero

Gestionar requisitos

- Priorizar las necesidades
- Implicar los interesados
- Desarrollar las necesidades de los interesados
- Identificar los activos disponibles

Colaboración entre equipos

El desarrollo de software no lo hace una única persona sino múltiples equipos. Debe haber una comunicación fluida para coordinar requisitos, desarrollo, evaluaciones, planes, resultados, etc.

Respondemos preguntas como

- ¿Cómo motivar?
- ¿Cómo colaborar con equipos?

Demostrar valor iterativamente

Los proyectos se entregan, aunque sea de un modo interno, en etapas iteradas. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto, así como también los riesgos involucrados.

Este principio explica porque el desarrollo del software se beneficia enormemente del proceso iterativo.



Elevar el nivel de abstracción

Este principio dominante motiva el uso de conceptos reutilizables tales como patrones de diseño del software, lenguajes 4GL o esquemas (frameworks) por nombrar algunos. Estos se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con UML.

Este principio describe como reducir la complejidad elevando el nivel de abstracción.

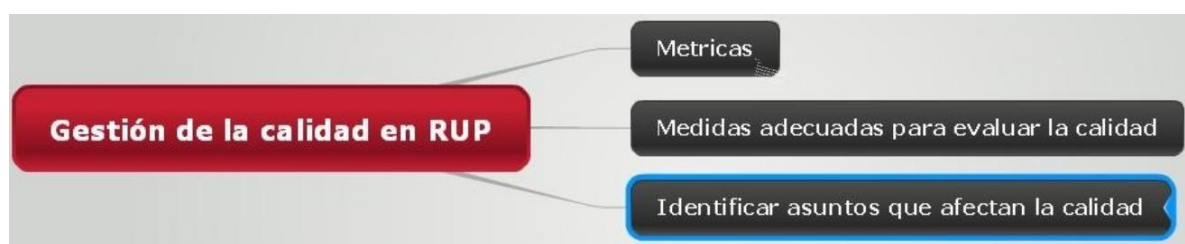
- Centrarse en la arquitectura
- Modelado de UML
- Bloques de alto nivel y componentes más importantes, sus responsabilidades y sus interfaces
- Reutilizar activos existentes

Enfocarse en la calidad

El control de calidad no debe realizarse al final de cada iteración, sino en todos los aspectos de la producción. El aseguramiento de la calidad forma parte del proceso de desarrollo y no de un grupo independiente, también es una estrategia de desarrollo de software.

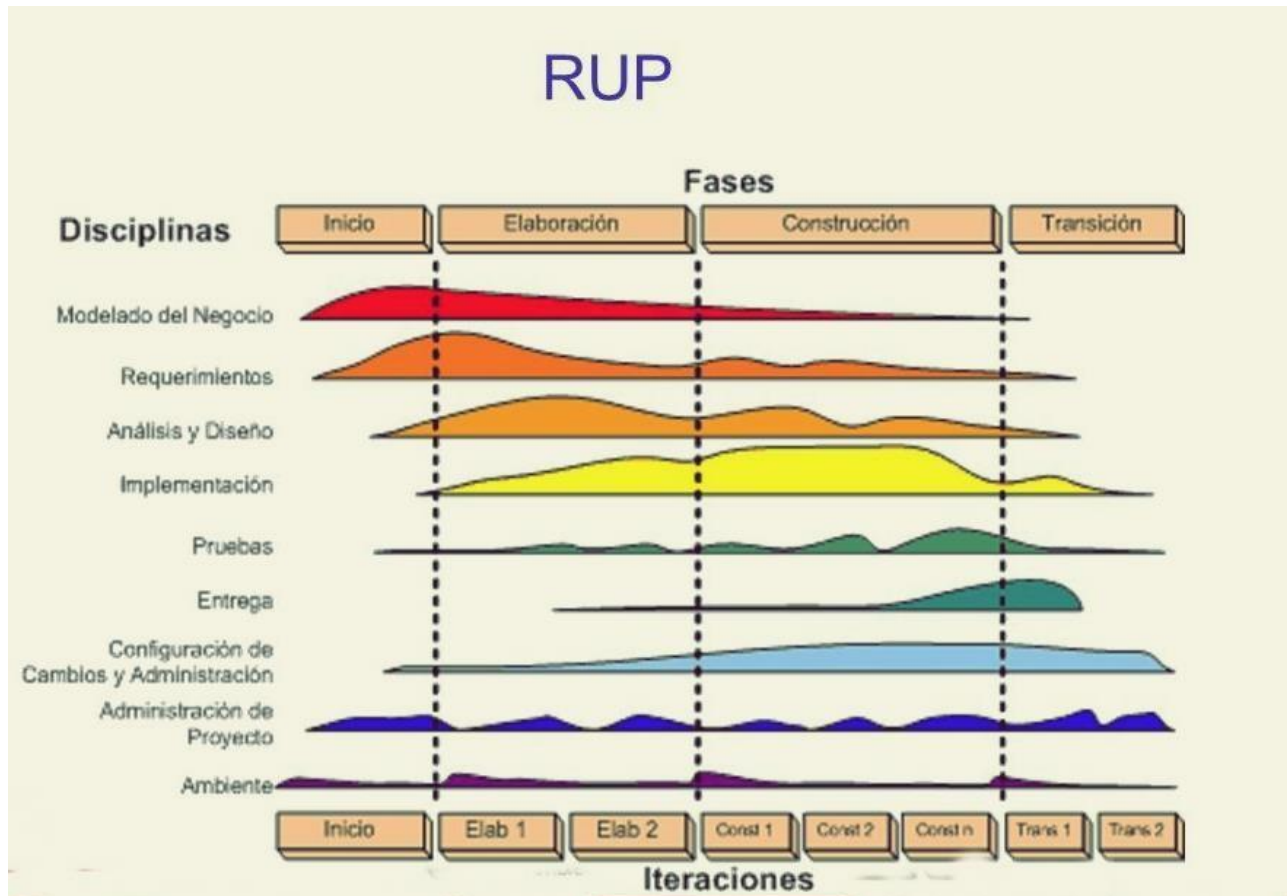
Este principio introduce la idea de calidad y describe cómo tratarla en todo el proceso.

- Como el proceso iterativo apoya la calidad
- Apropiar el concepto de calidad en todo el grupo de trabajo
- Implementar el diseño de pruebas al principio de cada iteración
- Implementar métricas



Descripción de las actividades o fases

Divididas en cuatro fases: Es en este momento que se le da planificación al proyecto, relevamiento de recursos, implementación, pruebas, entre otros.



Eje horizontal: Representa los aspectos dinámicos del proceso en RUP (concepción o inicio, elaboración, construcción y transición) expresados en fases, iteraciones e hitos.

Eje vertical: Representa los aspectos estáticos del proceso en RUP en términos de componentes, flujos de trabajo, actividades, artefactos y roles.

Disciplinas

En procesos paralelos a las fases:

- Modelado del negocio
- Requisitos
- Análisis y diseño
- Implementación
- Pruebas
- Gestión de la configuración
- Gestión del proyecto
- Entorno (infraestructura y herramientas)

Fase inicial (Inception)

Objetivo: Definir el alcance del proyecto y su viabilidad.

Es en este momento que se elabora la planificación del proyecto con los stakeholders, son ellos quienes han descrito los requisitos para el sistema a desarrollar. La etapa se realiza en un corto período de tiempo. Guía al equipo para analizar la viabilidad del proyecto y cómo empezar a definir los primeros pasos.

Esta etapa se enfoca que “que” vamos a hacer

- Se identifican los principales casos de usos
- Se identifican los riesgos
- Se define el alcance del proyecto

Modelado del negocio

- Entender la estructura y dinámica de la organización
- Entender los problemas actuales de la organización y sus posibles mejoras
- Asegurar que clientes, usuarios finales, programadores tengan un entendimiento común de la organización objetivo

Requisitos

- Establecer y mantener un acuerdo entre clientes y otros stakeholders sobre lo que el sistema podría hacer.
- Proveer a los desarrolladores un mejor entendimiento de los requisitos del sistema
- Definir el ámbito del sistema
- Proveer una base para estimar costos y tiempo del desarrollo del sistema
- Definir una interfaz de usuario para el sistema enfocadas a las necesidades y metas del sistema

Fase Elaboración (Elaboration)

Objetivo: Reducir los riesgos y definir la arquitectura del sistema.

En la fase de elaboración, busca relevar casos, documentación, estudios base, es decir, modelos para orientar el proyecto. Esto es para orientar cuál será la mejor manera de acuerdo con las premisas de los interesados.

Tras todo este conocimiento, se elabora un plan de proyecto, con todas las características y especificidades, de la forma más detallada posible.

Esta fase se enfoca “como” se va hacer

- Se hace un plan de proyectos
- Se contemplan los casos de uso
- Se eliminan los riesgos

Análisis y diseño

- Se especifican los requerimientos del sistema y se describe cómo se va a implementar el sistema
- Transformar los requisitos al diseño del sistema
- Desarrollar una arquitectura para el sistema
- Adaptar el diseño para que sea consistente con el entorno de implementación

Fase Construcción (Construction)

Objetivo: Construir el producto completo o una versión funcional sólida.

Ahí es cuando se termina la construcción del proyecto, por eso tiene ese nombre. El principal objetivo es la elaboración del producto. Dado que el método se basa en el desarrollo de software, estamos hablando de **crear códigos**.

Además, es en esta etapa que se realizan las primeras pruebas para que se prepare la base inicial para la etapa de transición.

- Se concentra en la elaboración del producto totalmente operativo y eficiente
- Se genera el manual de usuarios

Implementación

- Se implementan las clases y objetos en código fuente, archivos, ejecutables, etc. El resultado final es un sistema ejecutable
 - Planificación de subsistemas a implementar, orden de integración formando el Plan de Integración
 - Cada implementador decide en qué orden implementa los elementos de los subsistemas
 - Si encuentra errores de diseño, se notifica
 - Se integra el sistema siguiendo el plan

Pruebas

- Se evalúa la calidad del trabajo y se realizan las correcciones necesarias para integrar al ciclo de vida
 - Encontrar y documentar los errores o defectos de calidad
 - Se puede asesorar sobre la calidad del software percibido
 - Provee la validación del diseño y requisitos por medio de demostraciones concretas
 - Verificar las funciones del producto según lo diseñado
 - Verificar que los requisitos tengan su apropiada implementación

Despliegue

- Esta actividad tiene como objetivo producir con éxito las distribuciones del producto y a las distribuciones a los usuarios
 - Probar el producto en su entorno de ejecución final
 - Empaquetar el software para su distribución
 - Distribuir el software
 - Instalar el software
 - Proveer asistencia a los usuarios
 - Formar a los usuarios
 - Migrar software existente o base de datos

Fase Transición (Transition)

Objetivo: Entregar el sistema al usuario final.

Es la fase que pasa el proyecto desde el punto de prueba hasta la implementación.

Después de todas las pruebas realizadas y con el objeto listo, llega el momento de ponerlo a disposición del usuario final, es decir, la entrega del proyecto.





Además de la entrega, esta fase incluye la realización de capacitaciones y asegurar que el objeto final **resuelva todos los problemas de las partes interesadas**.

Dadas todas las fases que componen un proyecto utilizando la metodología RUP, es importante destacar que en el desarrollo de estas actividades todo el equipo necesita estar orientado a algunas prácticas y realizar los artefactos de forma alineada.

- El objetivo es llegar a obtener el reléase del proyecto
- Se realiza la instalación del producto para el cliente y se realiza el entrenamiento de los usuarios.
- En esta etapa se incluye: manufactura, envío, entrenamiento, soporte, mantenimiento del producto hasta que el cliente queda satisfecho

Artefactos

En cada una de las fases de RUP (pertenecientes a la estructura estática) se realizan una serie de artefactos llamados también entregables que sirven para comprender mejor tanto el análisis como el diseño del sistema (entre otros). Algunos de los artefactos son los siguientes:

Fase	Entregables principales
 Inicio	<ul style="list-style-type: none">- Documento de visión del sistema- Modelo preliminar de casos de uso- Identificación de riesgos iniciales- Plan de negocio- Estimación inicial de esfuerzo y costos- Prototipo básico (opcional)- Plan del proyecto inicial
 Elaboración	<ul style="list-style-type: none">- Modelo de casos de uso detallado- Arquitectura del sistema (modelo arquitectónico)- Modelo de análisis y diseño inicial- Prototipo arquitectónico ejecutable- Modelo de datos preliminar- Plan de proyecto detallado- Plan de pruebas preliminar- Actualización de riesgos
 Construcción	<ul style="list-style-type: none">- Código fuente funcional- Modelo de diseño completo (clases, componentes, etc.)- Casos de prueba y resultados- Manual de usuario (borrador o final)- Guía de instalación y despliegue- Versión ejecutable del sistema (release beta)- Informe de calidad del software
 Transición	<ul style="list-style-type: none">- Versión final del sistema- Documentación técnica completa- Manual del usuario final- Capacitación a usuarios- Plan de despliegue- Plan de mantenimiento- Informe de retroalimentación de usuarios- Registro de errores corregidos y pendientes

Roles






Roles	
 Analistas	Analistas de proceso de negocio Diseño del negocio Analista del sistema Especificador de requisitos
 Desarrolladores	Arquitecto de software Diseñador Diseñador de interfaz de usuario Diseñador de cápsulas Diseñador de base de datos Implementador Integrador
 Gestores	Jefe de proyecto Jefe de control de cambios Jefe de configuración Jefe de pruebas Jefe de despliegue Ingeniero de procesos Revisor de gestión del proyecto Gestor de pruebas
 Apoyo	Documentador técnico Administrador de sistema Especialista en herramientas Desarrollador de cursos Artista gráfico
 Especialista en pruebas	Especialista en pruebas (tester) Analista de pruebas Diseñador de pruebas

Tabla comparativa: RUP vs Cascada vs Scrum

Característica	RUP	Cascada	Scrum
Tipo de proceso	Iterativo e incremental	Secuencial	Iterativo e incremental
Enfoque	Casos de uso, arquitectura	Documentación, planificación	Personas y colaboración
Fases	4 fases	5-6 fases	Sprints
Flexibilidad	Media	Baja	Alta
Adecuado para	Proyectos medianos a grandes	Proyectos bien definidos	Proyectos con cambios frecuentes
Roles definidos	Sí, bien estructurados	Sí	Sí, pero más dinámicos
Documentación	Detallada	Muy detallada	Mínima y ligera
Gestión de riesgos	Activa y temprana	Tardía	Continua

Ejemplo práctico aplicado

Imaginemos que vas a desarrollar una app para gestionar turnos en una clínica médica:

- **Inicio:**
 - Definir que la app debe permitir a pacientes reservar turnos y a médicos ver su agenda.
 - Identificar riesgos como “falta de internet” o “resistencia al cambio del personal”.
- **Elaboración:**
 - Hacer un diseño de pantallas y diagramas de casos de uso.
 - Definir qué vas a usar una arquitectura MVC.
 - Validar que los requisitos sean correctos.
- **Construcción:**
 - Desarrollar la funcionalidad de login, reserva, notificaciones.
 - Entregar versiones funcionales cada 2 semanas.
 - Realizar pruebas automáticas y manuales.
- **Transición:**
 - Entrenar al personal.
 - Ajustar la app según sus comentarios.
 - Entregar la versión final y brindar soporte.

Paso a paso de una iteración en RUP

Supongamos que estamos en la fase de **Construcción**, pero estos pasos aplican en todas las fases iterativas (excepto Transición, que es más lineal).

1. Planificación de la iteración

- Se definen:
 - Los **objetivos** de la iteración.
 - Qué **casos de uso o funcionalidades** se van a desarrollar.
 - Tareas para análisis, diseño, implementación y pruebas.
 - Recursos (personas, tiempo, herramientas).
- Se actualiza el **plan del proyecto**.

Ejemplo: “Implementar la funcionalidad de reserva de turnos y verificar que se guarda en la base de datos.”

2. Captura y refinamiento de requisitos

- Se detalla qué espera el usuario de esa funcionalidad.
- Se refinan los **casos de uso** relacionados.
- Se discuten restricciones o validaciones necesarias.

Ejemplo: “El paciente solo puede reservar turnos en horarios disponibles.”

3. Análisis y diseño

- Se diseña cómo se implementará la funcionalidad:
 - Diagramas de clases, secuencia, componentes.
 - Decisiones arquitectónicas.
- Se actualiza el modelo de diseño si es necesario.

Ejemplo: Crear clases `Turno`, `Paciente`, `Agenda`, con sus relaciones.

4. Implementación (codificación)

- Se escribe el código de la funcionalidad planificada.
- Se integran componentes nuevos o modificados.
- Se hacen pruebas unitarias del código.

Ejemplo: Programar el formulario de reserva y guardar los datos correctamente.

5. Pruebas (verificación de calidad)

- Se prueban:
 - Casos de uso implementados.
 - Integración con otras partes del sistema.
 - Validación contra requisitos.

- Se documentan los resultados.

Ejemplo: Probar que no se puedan reservar dos turnos en el mismo horario.

6. Evaluación y cierre de iteración

- Se revisa:
 - Qué se logró.
 - Qué falló o debe mejorarse.
 - Lecciones aprendidas.
- Se preparan entregables si corresponde.
- Se ajusta el plan para la siguiente iteración.

Ejemplo: “La funcionalidad está lista, pero se detectó que falta enviar confirmaciones por email. Se agendará para la próxima iteración.”

Resultado

Al final de una iteración, se obtiene una **versión ejecutable del sistema** (aunque sea parcial), que se puede probar, mostrar al cliente o usar como base para la siguiente iteración.