Clase 2

Proceso de desarrollo de software

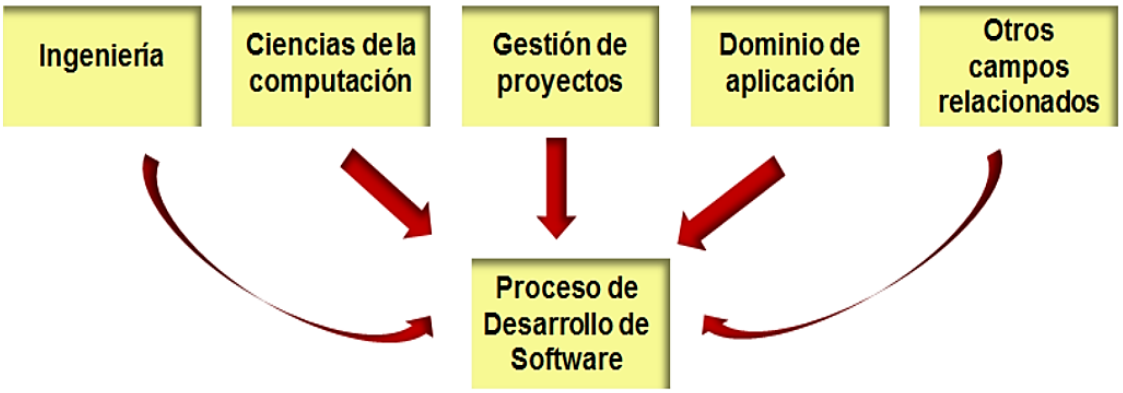
2.1 Introducción

En esta clase veremos los pasos o consideraciones a tener presentes al momento de planificar la creación de software. Generalmente al referirse a desarrollo de software se piensa inmediatamente en la programación o codificación de dicho software, sin embargo, este paso es uno más dentro de lo que se llama **proceso de desarrollo de software**.

2.2 Proceso de desarrollo de software

Un proceso de desarrollo de software es el conjunto estructurado de las actividades requeridas para construir un sistema. El proceso de desarrollo de software se utiliza para mejorar la comprensión del problema a resolver, la comunicación entre los participantes del proyecto y el mantenimiento de un sistema complejo.

En la siguiente figura se ilustran los aspectos involucrados durante el proceso de desarrollo de software: la aplicación de principios, técnicas y prácticas de la ingeniería, las ciencias de la computación, la gestión de proyectos, el dominio de aplicación y otros campos relacionados.



Los métodos del proceso de desarrollo indican como construir "técnicamente" el software, y abarcan, entre otras, las siguientes actividades fundamentales:

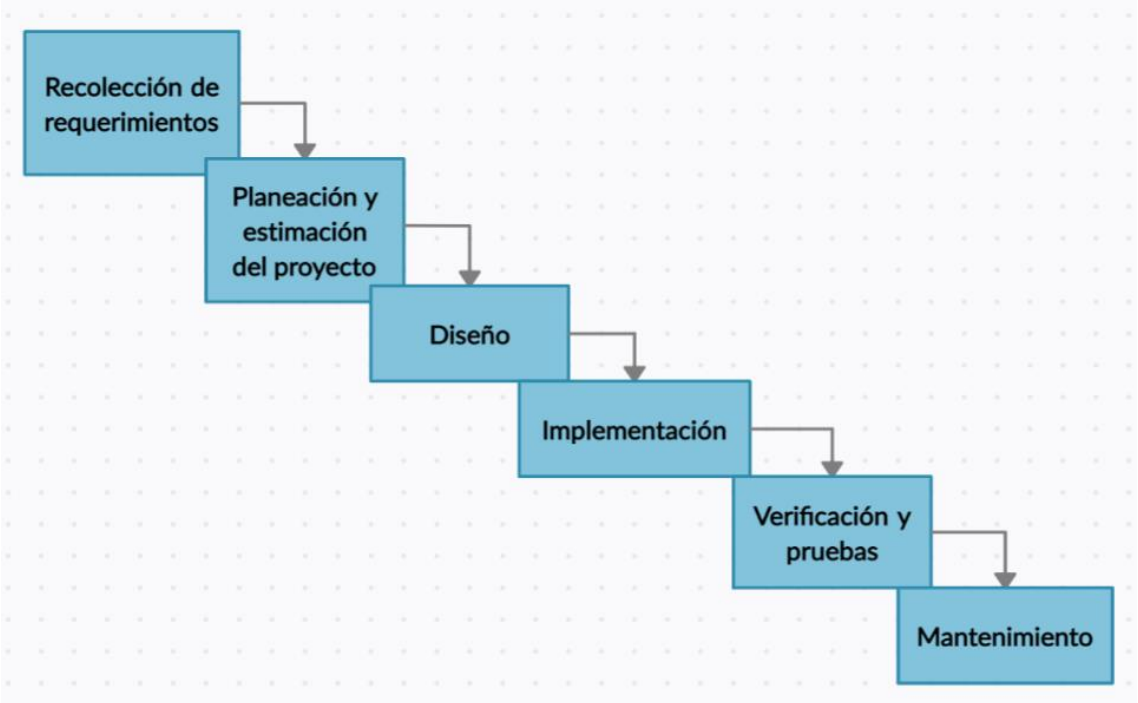
1. La especificación del software: donde los clientes junto con los desarrolladores del software definen los requerimientos del software a producir, sus funciones, características, propiedades y restricciones sobre su operación.

2. La codificación del software[1]: donde el software se diseña y programa.
3. Validación del software: donde el software se valida y se testea para asegurar su normal funcionamiento y que cumple con lo que el cliente requiere.
4. Evolución del software: donde el software se modifica para adaptarlo a los cambios requeridos por el cliente y el mercado.

Diferentes tipos de sistemas necesitan diferentes procesos de desarrollo. Por ejemplo, el software de tiempo real en un avión tiene que ser completamente especificado antes de que empiece el desarrollo, no se puede ir diseñando "sobre la marcha"; mientras que en un sistema de comercio electrónico, la especificación y el programa normalmente son desarrollados juntos.

2.3 Ciclo de vida del software

Al proceso de desarrollo de software también se le conoce como ciclo de vida. Un ciclo de vida es el proceso que se sigue para construir, entregar y hacer funcionar el software, **desde la concepción de la idea del sistema hasta su entrega y el desuso**.



[1] En varios libros de referencia, como Ingeniería del software de Ian Sommerville, esta actividad figura como *desarrollo de software*, pero en este apunte se cambia para evitar confusión con el concepto general de proceso de desarrollo de software.

La figura anterior describe el ciclo de vida de un producto de software. Para efectos didácticos, las actividades o etapas de la figura se muestran como parte de una secuencia lineal en cascada, sin embargo, según el modelo de ciclo de vida adoptado, el proceso puede ser cíclico e iterativo, siendo común regresar a etapas anteriores.

- El ciclo de vida primero nace con la concepción del sistema, su planeación y la especificación de los requerimientos.
- Luego se lleva a cabo su implementación que consiste en su diseño, codificación (la programación) y pruebas.
- Posteriormente el producto se entrega y sigue viviendo durante su operación y mantenimiento.
- El ciclo de vida del sistema de software termina cuando éste se deja de utilizar.

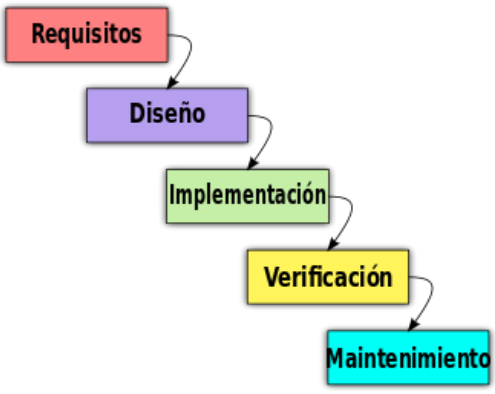
2.4 Modelo de desarrollo de software

Un modelo de desarrollo de software es una representación del Proceso de Desarrollo de software, y determina el orden en el que se llevan a cabo las actividades del proceso de desarrollo de software.

El modelo indica el orden de las etapas involucradas en el desarrollo del software y nos proporciona un criterio para comenzar, para continuar a la siguiente etapa y para finalizar.

A continuación, se muestran y explican brevemente algunos de los modelos de desarrollo comúnmente utilizados:

2.4.1 Modelo de cascada



Denominado así por la posición de las fases (que parecen caer en cascada hacia las siguientes fases), es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior. Al final de

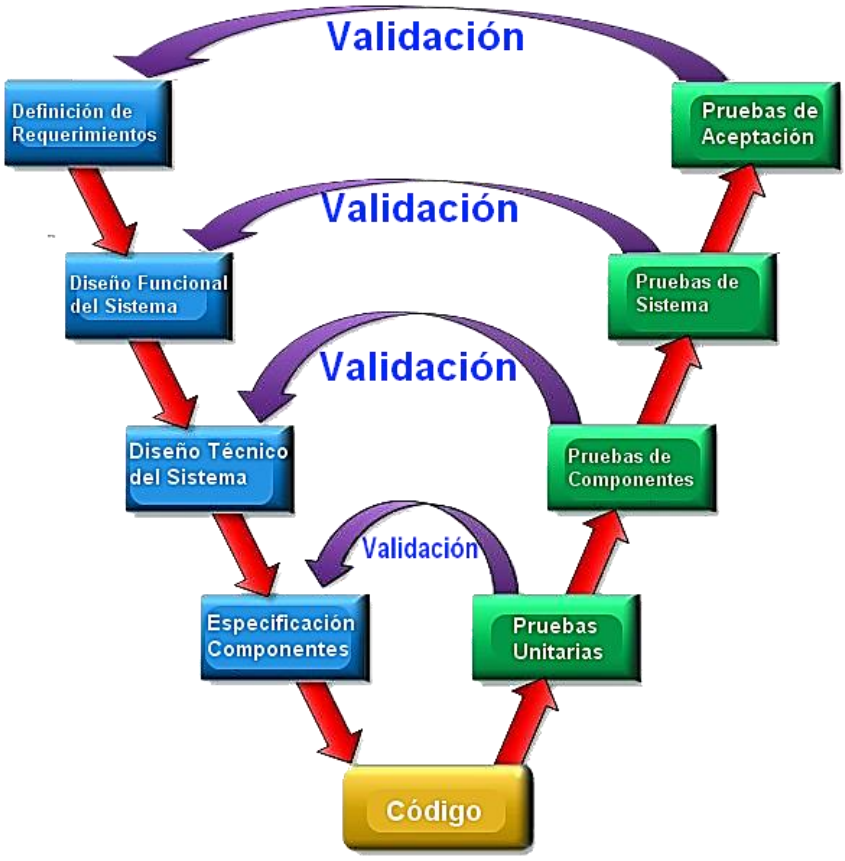
cada etapa, el modelo está diseñado para llevar a cabo una revisión final, que se encarga de determinar si el proyecto está listo para avanzar a la siguiente fase. Este modelo fue el primero en originarse y es la base de todos los demás modelos de ciclo de vida.

2.4.2 Modelo en V

Podría considerarse como una versión mejorada del modelo de cascada y por lo tanto mantiene gran parte de su concepto de secuencialidad, sin embargo, contempla, en ciertos casos, volver a etapas previas.

La "V" del nombre del modelo hace referencia a la forma como el modelo compara las fases de desarrollo con las fases de control de la calidad correspondientes.

- El brazo izquierdo de la letra V contiene las tareas de diseño y desarrollo del sistema.
- El brazo derecho contiene las medidas de control de calidad de cada fase.
- En la unión entre los dos brazos, se sitúa la implementación del producto. En los proyectos de software, esto se refiere a la programación del software.



La unión entre las fases de la parte izquierda y las correspondientes pruebas de la derecha representa una doble información:

- Sirve para indicar en qué fase de desarrollo se deben definir las pruebas correspondientes (determinar que pruebas se realizarán para probar si dicha etapa del desarrollo fue exitosa)
- Por otro lado, sirve para saber a qué fase de desarrollo hay que volver si se encuentran fallos en las pruebas correspondientes.

Por lo tanto, el modelo en V hace más explícita parte de las iteraciones y repeticiones de trabajo que están ocultas en el modelo en cascada.

El modelo en V se centra en las actividades y la corrección, se recomienda solo para el desarrollo de productos pequeños con equipos de trabajo de hasta cinco integrantes.

Su principal desventaja es que las pruebas comienzan a efectuarse después de la implementación, esto puede conducir a un retroceso de todo un proceso que costó tiempo y dinero.

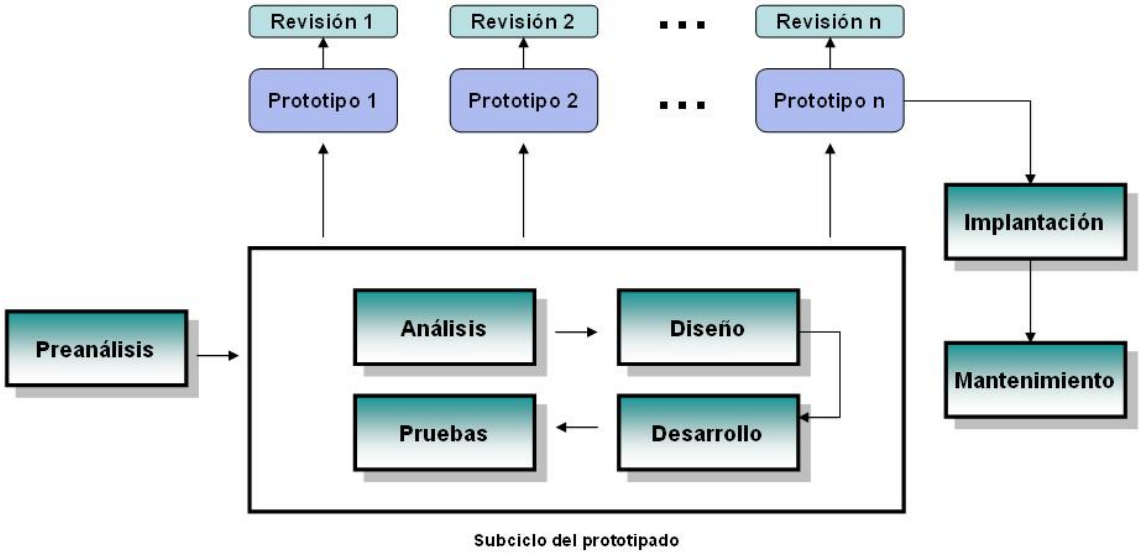
2.4.3 Modelo de prototipos

Este modelo, que pertenece a los del tipo evolutivo, se fundamenta en el desarrollo de un producto inicial que se presenta al usuario para obtener su aprobación y se perfecciona, a través de diferentes versiones o prototipos, hasta obtener el producto adecuado.

Para que la elección de este modelo adquiera sentido, cada prototipo debe ser construido en poco tiempo y no se deben utilizar muchos recursos.

El diseño rápido se centra en una representación de aquellos aspectos del software que serán visibles para el cliente o el usuario final. Este diseño conduce a la construcción de un prototipo, el cual es evaluado por el cliente para una retroalimentación; gracias a ésta se refinan los requisitos del software que se desarrollará y se procede al desarrollo de un nuevo prototipo.

Esto permite que al mismo tiempo el desarrollador entienda mejor lo que se debe hacer y el cliente vea resultados a corto plazo.



El modelo por prototipo se recomienda para el desarrollo de productos pequeños o de tamaño medio y es especialmente útil cuando se desconocen los requerimientos del producto o son poco estables.

2.4.4 Modelo de espiral

Este modelo de ciclo de vida del software fue definido por primera vez por Barry Boehm en 1988 y es muy utilizado en la ingeniería de software. Pertenece, al igual que el modelo de prototipo, a los de tipo evolutivo.

El modelo en espiral describe el ciclo de vida de un software por medio de espirales, que se repiten hasta que se puede entregar el producto terminado.

El producto se trabaja continuamente y las mejoras a menudo tienen lugar en pasos muy pequeños.

Una característica clave del desarrollo en espiral es la minimización de los riesgos en el desarrollo de software, lo que podría resultar en un aumento de los costes totales, más esfuerzo y una demora en el lanzamiento del producto.

Estos riesgos son contrarrestados dividiendo un proyecto de software en "miniproyectos" o prototipos. Cada prototipo se centra en uno o más riesgos importantes hasta que todos éstos estén controlados.

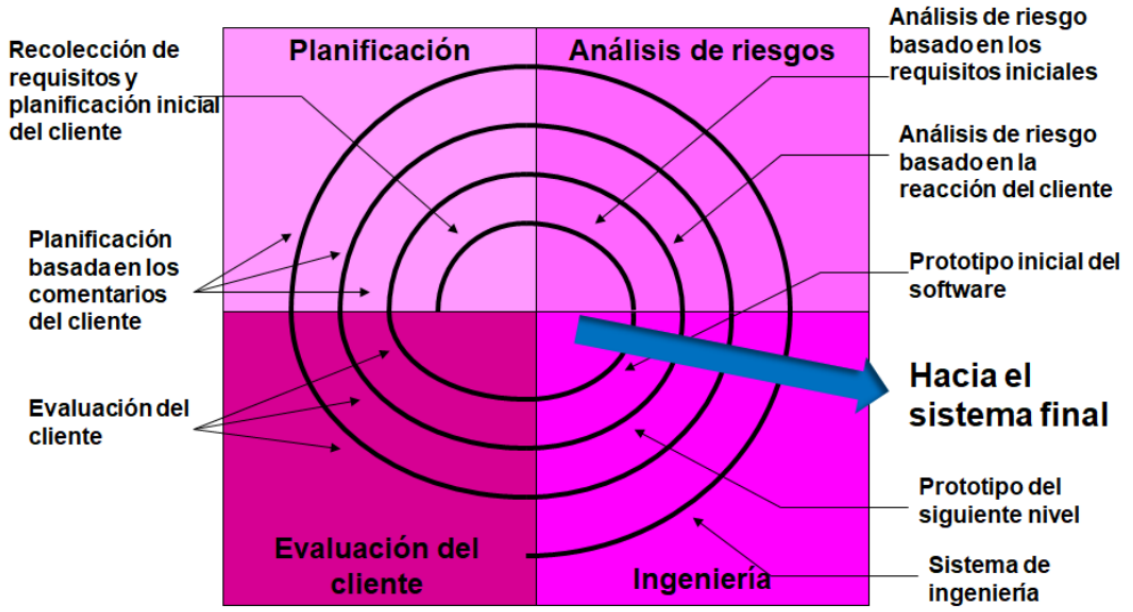
El concepto "riesgo" puede referirse a requerimientos poco comprensibles, arquitecturas poco comprensibles, problemas de ejecución importantes, problemas con la tecnología en la que se basa el software, entre otros.

La idea es, en cada ciclo, ir cumpliendo con lo requerimientos y objetivos del software a través de las alternativas que supongan menos riesgos para el proyecto.

El modelo en espiral combina una parte iterativa, que es la construcción de prototipos, con la parte secuencial del modelo en cascada para ir obteniendo resultados parciales en los que se tiene cierto control.

Durante las primeras iteraciones, la versión obtenida suele ser un prototipo, que se presenta para evaluación al cliente.

En la siguiente figura se ilustra el desarrollo en espiral enfocado en el proceso de desarrollo de software: análisis, diseño, implementación y pruebas.



Básicamente en el análisis de riesgo se realiza lo siguiente:

- Se identifican y evalúan los riesgos potenciales.
- También se evalúan las alternativas existentes.
- Los riesgos son registrados, evaluados y luego reducidos utilizando prototipos, simulaciones y software de análisis.
- En este ciclo, existen varios prototipos como plantillas de diseño o componentes funcionales.

Durante las últimas iteraciones, se producen versiones cada vez más completas del sistema. En la última iteración se produciría la versión final que pasaría a la fase de mantenimiento (no hay una cantidad definida de iteraciones, depende de que no queden riesgos por controlar y resolver).

Este sistema puede ser utilizado en proyectos grandes y complejos como puede ser, por ejemplo, la creación de un sistema operativo.

Cómo habrán notado los modelos de desarrollo de software tienen cierta complejidad, aquí mencionamos algunos modelos tradicionales y los describimos brevemente. Al momento de emprender el desarrollo de software hay que hacer un análisis de los modelos disponibles y cuál sería acorde a nuestro proyecto.

La aplicación de modelos de desarrollo de software nos posibilita que nuestro proyecto cumpla lo que Ian Sommerville caracteriza como "los atributos de un buen software[2]".

2.5 Los atributos de un buen software

Así como los servicios que proveen, los productos de software tienen un cierto número de atributos asociados que reflejan la calidad de ese software. Estos atributos no están directamente asociados con lo que el software hace. Más bien, reflejan su comportamiento durante su ejecución y en la estructura y organización del código del programa y en la documentación asociada.

El conjunto específico de atributos que se espera de un sistema de software depende obviamente de su aplicación. Por ejemplo, un sistema bancario debe ser seguro, un juego debe tener capacidad de respuesta, un interruptor de un sistema de alarma debe ser fiable, etc. Esto se generaliza en el conjunto de atributos que se muestra en la siguiente tabla, el cual tiene las características esenciales de un sistema de software bien diseñado.

Atributos esenciales de un buen software	
Mantenibilidad	El software debe escribirse de tal forma que pueda evolucionar para cumplir las necesidades de cambio de los clientes. Este es un atributo crítico debido a que la necesidad del cambio en el software es, en general, inevitable.
Confiabilidad	La confiabilidad del software tiene un gran número de características, incluyendo la fiabilidad, la protección y la seguridad. El software confiable no debe causar daños en caso de una falla del sistema.
Eficiencia	El software no debe hacer que se malgasten los recursos del sistema, como la memoria y los ciclos de procesamiento de la CPU. Por lo tanto, la eficiencia incluye tiempos de respuesta y de procesamiento, utilización de la memoria, etcétera.

[2] Sommerville Ian. Ingeniería del Software. Pearson 7a. Edición. 2005.

Usabilidad	El software debe ser fácil de utilizar, sin esfuerzo adicional, por el usuario para quien está diseñado. Esto significa que debe tener una interfaz de usuario apropiada y una documentación adecuada.

2.6 Los requerimientos

Durante esta clase se han mencionado varias veces los requerimientos, que son el primer paso en el ciclo de desarrollo de software, previo al diseño, programación y pruebas del proyecto. Vamos a desarrollar brevemente este concepto.

Los requerimientos especifican qué es **lo que un sistema de software debe hacer (sus funciones) y sus propiedades esenciales y deseables**.

Un requerimiento expresa el propósito del sistema sin considerar cómo se va a implantar. En otras palabras, los requerimientos identifican qué hace el sistema, mientras que el diseño establece el cómo lo hace el sistema.

2.6.1 ¿Para qué sirven los requerimientos?

Los requerimientos pueden servir a tres propósitos:

- Primero, permiten que los desarrolladores expliquen cómo han entendido lo que el cliente espera del sistema.
- Segundo, indican a los desarrolladores qué funcionalidad y qué características debe tener el sistema resultante.
- Tercero, indican qué demostraciones se deben llevar a cabo para convencer al cliente de que el sistema que se le entrega es de hecho lo que había ordenado.

2.6.2 Captura de los Requerimientos

La captura o extracción de los requerimientos tiene como objetivo principal la comprensión de lo que los clientes y los usuarios esperan que haga el sistema. Durante la captura se identifican los aspectos clave que el sistema requiere y se descartan los aspectos irrelevantes. Existen diversas técnicas para la extracción de los requerimientos:

- Una de ellas es la entrevista con el cliente y los usuarios.
- También está la observación de las tareas que debe realizar el cliente con el software (por ejemplo si actualmente lo realiza con una planilla Excel), en la que se revelan problemas, detalles, estrategias y estructuras de trabajo que son difíciles de captar claramente con las entrevistas.

- Finalmente, con la construcción de prototipos intermedios se va modificando el proyecto hasta que cumpla con las expectativas del cliente.

2.6.3 Verificación de requerimientos

Una característica esencial de un requerimiento de software es que debe ser posible verificar que el producto final lo satisface. Una tarea importante es planificar cómo verificar cada requerimiento. La verificación de requerimientos pretende asegurar que el sistema de software satisfará las necesidades del cliente y se lleva a cabo mediante las pruebas de aceptación.

Las pruebas de aceptación sirven para convencer al cliente de que el sistema cumple con lo que él pidió. Debe diseñarse por lo menos una prueba para cada requerimiento. Si un requerimiento no se puede probar significa que el requerimiento está mal hecho y es necesario replantearlo.

En conclusión, la captura, el análisis y la especificación de los requerimientos del sistema es una de las fases más importantes para que el proyecto tenga éxito.

Y nos evitará situaciones cómo la siguiente… :)