Universidad de San Carlos

Facultad de Ingeniería

Laboratorio, Análisis y Diseño de Sistemas 1

Carlos Ché 201800624

Primer Semestre 2022

# Clase 1

## Ingeniería del Software

Creación de productos para un cliente especifico teniendo en cuenta factores como costos, tiempos, planificación y calidad.

El ciclo de vida del software tiene 4 fases:

* Concepción. Determina la repercusión del proyecto y diseño del modelo del negocio.
* Elaboración. Planificación del proyecto, especificando características y arquitectura.
* Construcción. Elaboración del producto.
* Transición. Entrega del producto terminado

## Metodologías Robustas o Tradicionales

### Metodología Cascada

Es una metodología fácil de entender, con documentación excelente y total, fechas estipuladas, el cliente aprueba la documentación. Pero no se puede regresar a un paso anterior y no se puede avanzar a otra fase hasta terminar la actual.

* Análisis
* Diseño
* Implementación
* Verificación
* Mantenimiento

### Metodología de Prototipos

Reduce los riesgos, aumenta la probabilidad de éxito y tiene retroalimentación rápida.

* Requerimientos
* Modelado
* Construcción
* Desarrollo
* Refinamiento
* Prototipo terminado

### Metodología Espiral

Respuesta a los problemas del modelo en cascada, acepta cambios en requerimientos, identifica riesgos.

Diagram

Description automatically generated

### RUP (Rational Unified Process)

Es un conjunto de metodologías no pasos, es el estándar más utilizado. Evalúa cada fase, es sencilla. No es sencilla, tiene una posición incomoda para el cliente ya que se necesita para detallar las actividades, posee altos costos al implementar un cambio. Pero se definen claramente los roles y etapas y tiene una documentación exhaustiva. Fases:

* Requerimiento
* Análisis e implementación
* Prueba
* Evaluación

## Metodologías ágiles

El proyecto se divide en varias fases o iteraciones y el stakeholder siempre se involucra.

### Principios

* Equipo. Individuos e interacciones encima de procesos y herramientas
* Software Funcional. Tener software funcional en cada iteración para entregar en vez de una documentación extensa que solo nos quita tiempo.
* Colaboración con el cliente. Interacción constante con el cliente, da retroalimentación y no hay solo un contrato definido.
* Responder al cambio. Se permiten cambios a medio proyecto y son fáciles de adaptar.

# Clase 2

## Requerimiento

Condición que se debe satisfacer en un sistema para que este cumpla, contrato o estándar.

### Requerimiento No Funcionales

No son funciones del sistema, son propiedades, es el como hace algo. Existen por políticas del negocio, presupuestos, factores externos y el cómo se involucran con otros sistemas. A continuación, se muestran unos tipos:

* Requisitos de Productos
  + Rendimiento
  + Memoria
  + Seguridad
  + Eficiencia
* Requisitos Organizacionales
  + Patrones de diseño
  + Estándares de empresa
* Necesidades Externas
  + Comunicación con otros sistemas
  + Legislativos
  + Regulatorios
  + Éticos

Ejemplos como:

* Sistema debe ser capaz de procesar N transacciones por segundo.
* La respuesta debe ser menos a 5 segundos.
* Debe desarrollar con el patrón de diseño observer.
* Se usará la metodología TDD.

### Requerimientos Funcionales

Actividades y servicios que debe tener el sistema, y como procesara la entrada de datos y la salida. También se indica que no debe hacer el sistema.

Ejemplos como:

* Login
* Registro
* Enviar Correo de confirmación
* Asignar ID a X dato
* El usuario podrá ver el catalogo en la página principal

## Casos de uso

Es una técnica para la captura de requerimientos, proporciona información de cómo debe interactuar el sistema con los usuarios o con otro sistema. Cada caso se centra en describir una característica del sistema.

Se deben complementar con las reglas de negocio pues no permiten identificar los requerimientos no funcionales.

## Historias de Usuario

Cambia la forma de obtener los requisitos, a describir estos con los involucrados. Se debe considerar el peso de cada historia a la hora de escoger las iteraciones, y tiene relación con otras historias del backlog.

## Control de versiones

Forma de organizar los elementos de algo que es modificado a lo largo del tiempo. Antes se utilizaba un sistema de nombre donde se ponía la versión pero este es demasiado difícil usar en grupos grande y en proyectos donde se realizan grandes cambios en todo momento por lo que se utilizan sistemas de control de versiones para ayudarnos. Tipos:

* Sistema Local. Es copiar los archivos a otro directorio con diferencia en color, fecha o numeral para identificar los cambios. Cada uno lleva su control y es difícil juntar los cambios de todo el equipo.
* Sistema centralizado. Permite obtener las diversas versiones y hacer commits nuevos, posee un único repositorio en un servidor. Pero tiene inconvenientes de que siempre debe estar disponible y que es demasiado vulnerable al tener todo en un solo servidor.
* Sistema distribuido. Todos los colaboradores tienen una copia del sistema central, si el central falla todos tienen un backup, las acciones realizadas no afectan al servidor central y no hay tanta dependencia de red.

# Clase 3

## SCRUM

Framework de trabajo donde se pueden manejar problemas complejos y adaptativos mientras se incrementa el valor del producto

### Product Owner

Maneja el product backlog, es solo una persona no un comité. Ordenas los ítems y escoge su valor.

### Scrum Master

Se asegura que el equipo de desarrollo entienda el alcance del producto, facilita los eventos de scrum, guía al equipo de desarrollo a crear productos de alto valor y remueve los impedimentos del equipo que retrasen el proyecto.

### Equipo de desarrollo

No hay sub-equipos, nadie posee un título especifico, y los miembros tienen una habilidad y enfoque específicos por cumplir, menores a 3 tiene poca productividad y mayores a 9 cuesta mucho organizarse.

### Product Backlog

Lista dinámica y ordena de las funcionalidades que necesita el producto.

### Sprint planning meeting

Se planifica que se espera lograr al final del sprint y se da por comenzado el sprint.

### Sprint diario

15 minutos máximos, ¿qué hice ayer?, ¿qué haré hoy?, ¿algún impedimento?

### Sprint review

¿Qué se logró?, ¿Se alcanzó el objetivo?, se presenta un incremento y se da retroalimentación a las tareas completadas.

### Sprint retrospective

Se discute como mejorar el proceso de desarrollo. El ¿cómo se hizo algo?

## Git

Software de control de versiones para sistemas distribuidos, permite ramas, software libre y multiplataforma. Tenemos las opciones de un hosting local o hostings en la nube como gitlab o github.

### Buenas practicas

* Cada desarrollador posee un único usuario
* No push a master
* No colocar información sensible como llaves
* Realizar un buen gitignore
* Mensajes en commits en tercera persona y descriptivos

### Tags

Los tags son referencias a puntos específicos en la historia de Git. Comúnmente se utiliza el versionamiento semántico.

Diagram

Description automatically generated

### Gitflow

Es una recomendación de cómo usar git, permite integrar código rápidamente, rastrea errores y posee una identificación rápida de las versiones. Posee las siguientes ramas:

* Master. Es la versión en producción, se debe generar el tag de la versión y de esta salen los hotfix a develop.
* Nace del master, de esta nacen los feature nuevos a desarrollar y los realeses a publicar.
* Feature. Su formato es feature/nombre. Nace de develop y se utiliza para crear nuevas funcionalidades de las historias de usuario.
* Release. Su formato es reléase/versión, en esta rama se realizan las pruebas, acepta commits de bugfix y deben hacerse los merge a master.
* Hotfix. Formato hotfix/1.0.1. Se crean desde el master, tienen prioridad y se hacen hacia la versión en producción.

# Clase 4

## RUP

Proceso de ingeniería de software, otorga una forma disciplinada de asignar tareas en el equipo. Es iterativo, incremental, proceso dirigido por casos de uso, proceso centrado en la arquitectura su enfoque es la calidad del producto.

RUP se puede mostrar como una gráfica de dos dimensiones en el eje horizontal encontramos el tiempo y aspectos del ciclo y en el vertical las actividades a realizar.

Chart, diagram

Description automatically generated

RUP posee 4 fases, y al final de cada fase se realiza una evaluación para ver si se alcanzaron los objetivos y se puede pasar a la siguiente fase, las fases son las siguientes:

* Incepción. Define ámbito y objetivos del proyecto, funcionalidades y capacidades del producto.
* Elaboración. Se define la arquitectura y se planifica el proyecto considerando los recursos.
* Construcción. El producto se desarrolla en iteraciones, cada iteración involucra tareas de análisis, diseño e implementación. Se documenta todo y se generan las pruebas.
* Transición. Se libera el producto y se entrega al usuario final, se terminan los manuales de usuario y las tareas también se realizan en iteraciones.

## QA

Quality Assurance, es un proceso que asegura que todos los procesos de ingeniería de software, métodos, código, sean monitoreados y cumplan con los estándares definidos.

Nos asegura la calidad de lo que el cliente espera, y que se cumplan los criterios acordados, y que el producto final no tenga errores, todo el equipo de desarrollo debe contar con al menos un responsable para esto.

El analista de QA, encuentra los errores pero no los corrige. Documenta los errores, verifica estándares y busca optimizaciones. Para tener un equipo de SCRUM completo se debe contar con alguien encargado de QA para verificar la calidad del producto a entregar.

# Clase 5

## Pruebas Unitarias

Prueba unitaria o unit test, es un método que prueba una unidad estructural de código, son simples y fáciles de codificar no deben llevar más de 5 minutos. Son pruebas de solo porciones de código, independiente, automatizable, repetible y predecible y las pruebas deben ser profesionales.

### Ventajas

* Fomentan el cambio. Promueven la refactorización del código.
* Simplifican la integración.
* Documentan el código.
* Separación de la interfaz y la implementación.
* Menos errores y mayor facilidad de localizarlos.
* Pueden mejorar el diseño.
* Puede ser la forma más simple de verificar el funcionamiento.
* Ahorra tiempo en el futuro.
* Mejora el código.

### Consideraciones

* No descubren errores de integración, rendimiento u otros problemas similares.
* En algunos casos es difícil identificar las salidas esperadas en las pruebas.
* Se necesitan condiciones iniciales reales para poder hacer una prueba útil.
* Muchas veces se hace uso de objetos simulados para las pruebas, para así no crear los objetos reales solo para hacer las pruebas.

### Objetos simulados

Son objetos que parecen reales, pero no tiene dependencia de otros. Son fáciles de programar con las necesidades que tengamos y permite reproducir cualquier interacción obtenida con el objeto.

* STUB. Objeto falso que se le programan objetos de retorno, solo da respuestas y es el más liviano. Siempre regresa los mismos valores sin importar el input, se puede utilizar para simular objetos de base de datos, reduce la complejidad.
* MOCK. Objeto falso que se le programa a objetos de retornos, guarda llamadas a métodos, es usado para guardar y verificar interacciones, considerado el más poderoso y flexible de los objetos, otorga control total del comportamiento de los objetos.
* SPY/PROXY. Objeto falso que se le programa a objetos de retorno, cualquier comportamiento que no se haya mockeado son reales y funcionan como un objeto real, técnicamente es sustituir el real y reemplazar solo un método útil para testear objetos con muchos métodosClase 6

# Clase 6

## Extreme Programming

Metodología basada en el no nosense. Se enfoca en crear código, testear el código, escuchar para saber que codificar, y diseñar para poder codificar y testear.

Utiliza practicas de XP, se modifica en pasos pequeños y mientras una función no se solicite no se codifica.

### Feedback Fino

Se trabaja en ciclos muy cortos para poder comprobar el código una y otra vez.

* El test-driven development indica que se hacen las pruebas primero y el código después.
* Pair Programming. Un desarrollador escribe código y el otro lo prueba.
* On-site customer. Un representante del cliente forma parte del equipo para resolver dudas y aportar ideas.

### Proceso Continuo

La integración continua de código en los equipos permite a los programadores escribir varias fracciones de código diariamente y luego poder comprobar que funcione.

### Otros datos

El bienestar del equipo es lo primordial, ya que a mayor motivación mejor trabajo. El Extreme Programming propone una semana laboral de 40 hora sin horas extras.

## Test Driven Development

Se realizan pruebas antes de agregar funcionalidad al sistema, para así definir éxito desde el comienzo y descomponer el diseño en pequeñas partes.

### Reglas

* No se puede escribir código en producción a menos que sea para arreglar un error.
* No escribir más de la prueba unitaria si ya falla.
* No realizar más código para que la única prueba fallida pase.

### Pasos

* Escribir test para caso
* Correr test
* Test falla
* Se escribe código
* Se evalúa si se puede refactorizar el código y proceder
* Si el refactor afecta las demás pruebas se realiza mal
* Corre test
* Pasa test
* Repetir

## Behavior Driven Development

Es una rama de TDD que usa descripciones de software fácil de entender para el usuario.

* Cada requerimiento debe ser una historia de usuario
* Cada caso debe ser un escenario de un usuario en el sistema
* Se debe tomar un enfoque de como haría un test el usuario en vez de una prueba unitaria
* Especificar el escenario y condiciones GIVEN
* Condiciones que se deben cumplir WHEN
* Definir el resultado esperado THEN
* No se definen pruebas si no comportamientos
* Mejora la comunicación entre desarrolladores, testes y usuarios y gerencia
* Curva de aprendizaje corta
* Cubre a un público más amplio

# Clase 7

## Prueba Funcionales

Se centran en comprobar que los sistemas cumplan con los requerimientos de los clientes. Tiene las siguientes fases:

* Análisis de requerimientos
* Plan de Pruebas
* Casos de Pruebas
* Ejecución
* Reporte de Cierre

Además, contamos con los siguientes tipos de pruebas funcionales:

* Prueba Unitarias. Se ejecutan de forma independiente para garantizar la funcionalidad de una porción de código.
* Prueba de humo. Se verifican que las funcionalidades más importantes funcionen.
* Prueba de integración. Generalmente son automatizadas, prueba componentes individuales con el objetivo que los módulos individuales funcionen y permite que todo actué como parte de un solo sistema en vez de partes aisladas.
* Prueba de regresión. Aseguran que los cambios realizados al código no arruinen lo que ya funcionaba.
* Prueba de cordura. Detecta errores accidentales, analiza profundamente la compilación.
* Pruebas de aceptación del usuario. Los usuarios reales prueban todo para que funcione correctamente.

## Pruebas No Funcionales

Son pruebas para validar el funcionamiento del sistema por medio de sus requerimientos no funcionales.

Graphical user interface, text

Description automatically generated with medium confidence

# Clase 8

## Kanban

Metodología ágil que se creó en Toyota, basado en el sistema de producción just intime, el cual se enfoca en la producción bajo demanda para así reducir los desperdicios, la metodología se enfoca en llevar a cabo tareas pendientes.

Posee 4 principios los cuales son los siguientes:

* Calidad Garantizada
* Reducción del desperdicio
* Mejora continua
* Flexibilidad

### Tablero

Todo el trabajo gira en torno al tablero, dentro de este se pueden visualizar a todos los miembros del equipo y el flujo y los bloqueos en los que se encuentren.

### Tarjetas

Son la representación gráfica de las tareas y deben moverse según su estado, en realizadas, en proceso y por hacer.

### WIP

Work in progress, son las tareas que se están en proceso.

### Flujo

Son los pasos y procesos dentro el tablero de KANBAN en donde todas las tareas deben seguir los pasos del flujo, la tarea KABAN pasa estar terminada hasta después de seguir el flujo.

### Practicas

* Visualizar el flujo. Visualizar todas las tareas y el flujo en el tablero.
* Limitar WIP. Establecer metas realistas y definir el equilibrio de trabajo a realizar
* Políticas administrativas. Criterio con el cual se dice que una tarea termino alguna fase del flujo.
* Administrar el flujo. Ver como distribuir las tareas dependiendo el avance.
* Mejorar colaborativamente. Ver que se puede mejorar en el grupo.