



## Sociedad Peruana de Computación (SPC)

Programa Profesional de  
Ciencia de la Computación  
Sílabo 2022-I

### 1. CURSO

CS113. Ciencia de la Computación II (Obligatorio)

### 2. INFORMACIÓN GENERAL

2.1 Créditos	:	4
2.2 Horas de teoría	:	2 (Semanal)
2.3 Horas de práctica	:	4 (Semanal)
2.4 Duración del periodo	:	16 semanas
2.5 Condición	:	Obligatorio
2.6 Modalidad	:	Presencial
2.7 Prerrequisitos	:	CS112. Ciencia de la Computación I. (2 <sup>do</sup> Sem)

### 3. PROFESORES

Atención previa coordinación con el profesor

### 4. INTRODUCCIÓN AL CURSO

Este es el tercer curso en la secuencia de los cursos introductorios a la informática. En este curso se pretende cubrir los conceptos señalados por la Computing Curricula IEEE(c)-ACM 2001, bajo el enfoque funcional-first. El paradigma orientado a objetos nos permite combatir la complejidad haciendo modelos a partir de abstracciones de los elementos del problema y utilizando técnicas como encapsulamiento, modularidad, polimorfismo y herencia. El dominio de estos temas permitirá que los participantes puedan dar soluciones computacionales a problemas de diseño sencillos del mundo real.

### 5. OBJETIVOS

- Introducir al alumno a los fundamentos del paradigma de orientación a objetos, permitiendo asimilar los conceptos necesarios para desarrollar un sistema de información

### 6. COMPETENCIAS

- a) Aplicar conocimientos de computación y de matemáticas apropiadas para la disciplina. (**Usar**)
- b) Analizar problemas e identificar y definir los requerimientos computacionales apropiados para su solución. (**Usar**)
- d) Trabajar efectivamente en equipos para cumplir con un objetivo común. (**Usar**)

### 7. COMPETENCIAS ESPECÍFICAS

- a12) Aplicar pensamiento computacional para resolver problemas cotidianos.
- a13) Utilizar de forma eficiente estructuras de control condicionales, repetitivas, funciones, recursividad, ordenamiento y búsqueda.
- b4) Identificar y aplicar de forma eficiente diversas estrategias algorítmicas y estructuras de datos para la solución de un problema dadas ciertas restricciones de espacio y tiempo.
- d1) Desarrollo colaborativo de software utilizando repositorios de código y gestión de versiones (ej. Git, Bitbucket, SVN).

### 8. TEMAS

Unidad 1: Conceptos Fundamentales de Programación (5)	
Competencias esperadas: a,b	
Temas	Objetivos de Aprendizaje
<ul style="list-style-type: none"> <li>• Sintaxis y semántica básica de un lenguaje de alto nivel.</li> <li>• Variables y tipos de datos primitivos (ej., numeros, caracteres, booleanos)</li> <li>• Expresiones y asignaciones.</li> <li>• Operaciones básicas I/O incluyendo archivos I/O.</li> <li>• Estructuras de control condicional e iterativas.</li> <li>• Paso de funciones y parámetros.</li> <li>• Concepto de recursividad.</li> </ul>	<ul style="list-style-type: none"> <li>• Analiza y explica el comportamiento de programas simples que involucran estructuras fundamentales de programación variables, expresiones, asignaciones, E/S, estructuras de control, funciones, paso de parámetros, y recursividad [Usar]</li> <li>• Identifica y describe el uso de tipos de datos primitivos [Usar]</li> <li>• Escribe programas que usan tipos de datos primitivos [Usar]</li> <li>• Modifica y expande programas cortos que usen estructuras de control condicionales e iterativas así como funciones [Usar]</li> <li>• Diseña, implementa, prueba, y depura un programa que usa cada una de las siguientes estructuras de datos fundamentales: cálculos básicos, E/S simple, condicional estándar y estructuras iterativas, definición de funciones, y paso de parámetros [Usar]</li> <li>• Escribe un programa que usa E/S de archivos para brindar persistencia a través de ejecuciones múltiples [Usar]</li> <li>• Escoje estructuras de condición y repetición adecuadas para una tarea de programación dada [Usar]</li> <li>• Describe el concepto de recursividad y da ejemplos de su uso [Usar]</li> <li>• Identifica el caso base y el caso general de un problema basado en recursividad [Usar]</li> </ul>
<b>Lecturas :</b> [stroustrup2013], [Van02], [LE13]	

Unidad 2: Programación orientada a objetos (7)	
Competencias esperadas: a,b	
Temas	Objetivos de Aprendizaje
<ul style="list-style-type: none"> <li>• Diseño orientado a objetos: <ul style="list-style-type: none"> <li>– Descomposición en objetos que almacenan estados y poseen comportamiento</li> <li>– Diseño basado en jerarquía de clases para modelamiento</li> </ul> </li> <li>• Definición de las categorías, campos, métodos y constructores.</li> <li>• Las subclases, herencia y método de alteración temporal.</li> <li>• Asignación dinámica: definición de método de llamada.</li> <li>• Subtipificación: <ul style="list-style-type: none"> <li>– Polimorfismo artículo Subtipo; upcasts implícitos en lenguajes con tipos.</li> <li>– Noción de reemplazo de comportamiento: los subtipos de actuar como supertipos.</li> <li>– Relación entre subtipos y la herencia.</li> </ul> </li> <li>• Lenguajes orientados a objetos para la encapsulación: <ul style="list-style-type: none"> <li>– privacidad y la visibilidad de miembros de la clase</li> <li>– Interfaces revelan único método de firmas</li> <li>– clases base abstractas</li> </ul> </li> <li>• Uso de colección de clases, iteradores, y otros componentes de la librería estándar.</li> </ul>	<ul style="list-style-type: none"> <li>• Diseñar e implementar una clase [Usar]</li> <li>• Usar subclase para diseñar una jerarquía simple de clases que permita al código ser reusable por diferentes subclases [Usar]</li> <li>• Razonar correctamente sobre el flujo de control en un programa mediante el envío dinámico [Usar]</li> <li>• Comparar y contrastar (1) el enfoque proceduracional/funcional- definiendo una función por cada operación con el uso de la función proporcionando un caso por cada variación de dato - y (2) el enfoque orientado a objetos - definiendo una clase por cada variación de dato con la definición de la clase proporcionando un método por cada operación. Entender ambos enfoques como una definición de variaciones y operaciones de una matriz [Usar]</li> <li>• Explicar la relación entre la herencia orientada a objetos (código compartido y <i>overriding</i>) y subtipificación (la idea de un subtipo es ser utilizable en un contexto en el que espera al supertipo) [Usar]</li> <li>• Usar mecanismos de encapsulación orientada a objetos, tal como interfaces y miembros privados [Usar]</li> <li>• Definir y usar iteradores y otras operaciones sobre agregaciones, incluyendo operaciones que tienen funciones como argumentos, en múltiples lenguajes de programación, seleccionar la forma más natural por cada lenguaje [Usar]</li> </ul>
<b>Lecturas :</b> [Str13]	

Unidad 3: Algoritmos y Diseño (5)	
Competencias esperadas: a,b,d	
Temas	Objetivos de Aprendizaje
<ul style="list-style-type: none"> <li>• Conceptos y propiedades de los algoritmos <ul style="list-style-type: none"> <li>– Comparación informal de la eficiencia de los algoritmos (ej., conteo de operaciones)</li> </ul> </li> <li>• Rol de los algoritmos en el proceso de solución de problemas</li> <li>• Estrategias de solución de problemas <ul style="list-style-type: none"> <li>– Funciones matemáticas iterativas y recursivas</li> <li>– Recorrido iterativo y recursivo en estructura de datos</li> <li>– Estrategias Divide y Conquistar</li> </ul> </li> <li>• Conceptos y principios fundamentales de diseño <ul style="list-style-type: none"> <li>– Abstracción</li> <li>– Descomposición de Program</li> <li>– Encapsulamiento y camuflaje de información</li> <li>– Separación de comportamiento y aplicación</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Discute la importancia de los algoritmos en el proceso de solución de un problema [Usar]</li> <li>• Discute como un problema puede ser resuelto por múltiples algoritmos, cada uno con propiedades diferentes [Usar]</li> <li>• Crea algoritmos para resolver problemas simples [Usar]</li> <li>• Usa un lenguaje de programación para implementar, probar, y depurar algoritmos para resolver problemas simples [Usar]</li> <li>• Implementa, prueba, y depura funciones recursivas simples y sus procedimientos [Usar]</li> <li>• Determina si una solución iterativa o recursiva es la más apropiada para un problema [Usar]</li> <li>• Implementa un algoritmo de divide y vencerás para resolver un problema [Usar]</li> <li>• Aplica técnicas de descomposición para dividir un programa en partes más pequeñas [Usar]</li> <li>• Identifica los componentes de datos y el comportamiento de múltiples tipos de datos abstractos [Usar]</li> <li>• Implementa un tipo de dato abstracto coherente, con la menor pérdida de acoplamiento entre componentes y comportamientos [Usar]</li> <li>• Identifica las fortalezas y las debilidades relativas entre múltiples diseños e implementaciones de un problema [Usar]</li> </ul>
Lecturas : [stroustrup2013], [Weert16], [LE13]	

Unidad 4: Análisis Básico (3)	
Competencias esperadas: a,b	
Temas	Objetivos de Aprendizaje
<ul style="list-style-type: none"> <li>• Diferencias entre el mejor, el esperado y el peor caso de un algoritmo.</li> <li>• Análisis asintótico de complejidad de cotas superior y esperada.</li> <li>• Definición formal de la Notación Big O.</li> <li>• Clases de complejidad como constante, logarítmica, lineal, cuadrática y exponencial.</li> <li>• Medidas empíricas de desempeño.</li> <li>• Compensación entre espacio y tiempo en los algoritmos.</li> <li>• Uso de la notación Big O.</li> <li>• Notación Little o, Big omega y Big theta.</li> <li>• Relaciones recurrentes.</li> <li>• Análisis de algoritmos iterativos y recursivos.</li> <li>• Teorema Maestro y Árboles Recursivos.</li> </ul>	<ul style="list-style-type: none"> <li>• Explique a que se refiere con “mejor”, “esperado” y “peor” caso de comportamiento de un algoritmo [Usar]</li> <li>• En el contexto de a algoritmos específicos, identifique las características de data y/o otras condiciones o suposiciones que lleven a diferentes comportamientos [Usar]</li> <li>• Determine informalmente el tiempo y el espacio de complejidad de diferentes algoritmos [Usar]</li> <li>• Indique la definición formal de Big O [Usar]</li> <li>• Lista y contraste de clases estándares de complejidad [Usar]</li> <li>• Realizar estudios empíricos para validar una hipótesis sobre runtime stemming desde un análisis matemático Ejecute algoritmos con entrada de varios tamaños y compare el desempeño [Usar]</li> <li>• Da ejemplos que ilustran las compensaciones entre espacio y tiempo que se dan en los algoritmos [Usar]</li> <li>• Use la notación formal de la Big O para dar límites superiores asintóticos en la complejidad de tiempo y espacio de los algoritmos [Usar]</li> <li>• Usar la notación formal Big O para dar límites de casos esperados en el tiempo de complejidad de los algoritmos [Usar]</li> <li>• Explicar el uso de la notación theta grande, omega grande y o pequeña para describir la cantidad de trabajo hecho por un algoritmo [Usar]</li> <li>• Usar relaciones recurrentes para determinar el tiempo de complejidad de algoritmos recursivamente definidos [Usar]</li> <li>• Resuelve relaciones de recurrencia básicas, por ejemplo. usando alguna forma del Teorema Maestro [Usar]</li> </ul>
Lecturas : [Str13]	

Unidad 5: Sistemas de tipos básicos (5)	
Competencias esperadas: a,b	
Temas	Objetivos de Aprendizaje
<ul style="list-style-type: none"> <li>• Tipos como conjunto de valores junto con un conjunto de operaciones. <ul style="list-style-type: none"> <li>– Tipos primitivos (p.e. números, booleanos)</li> <li>– Composición de tipos contruídos de otros tipos (p.e., registros, uniones, arreglos, listas, funciones, referencias)</li> </ul> </li> <li>• Asociación de tipos de variables, argumentos, resultados y campos.</li> <li>• Tipo de seguridad y los errores causados por el uso de valores de manera incompatible dadas sus tipos previstos.</li> <li>• Metas y limitaciones de tipos estáticos <ul style="list-style-type: none"> <li>– Eliminación de algunas clases de errores sin ejecutar el programa</li> <li>– Indecisión significa que un análisis estatico puede aproximar el comportamiento de un programa</li> </ul> </li> <li>• Tipos genéricos (polimorfismo paramétrico) <ul style="list-style-type: none"> <li>– Definición</li> <li>– Uso de librerías genéricas tales como colecciones.</li> <li>– Comparación con polimorfismo ad-hoc y polimorfismo de subtipos</li> </ul> </li> <li>• Beneficios complementarios de tipos estáticos y dinámicos: <ul style="list-style-type: none"> <li>– Errores tempranos vs. errores tardíos/evitados.</li> <li>– Refuerzo invariante durante el desarrollo y mantenimiento del código vs. decisiones pospuestas de tipos durante la la creación de prototipos y permitir convenientemente la codificación flexible de patrones tales como colecciones heterogéneas.</li> <li>– Evitar el mal uso del código vs. permitir más reuso de código.</li> <li>– Detectar programas incompletos vs. permitir que programas incompletos se ejecuten</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Tanto para tipo primitivo y un tipo compuesto, describir de manera informal los valores que tiene dicho tipo [Usar]</li> <li>• Para un lenguaje con sistema de tipos estático, describir las operaciones que están prohibidas de forma estática, como pasar el tipo incorrecto de valor a una función o método [Usar]</li> <li>• Describir ejemplos de errores de programa detectadas por un sistema de tipos [Usar]</li> <li>• Para múltiples lenguajes de programación, identificar propiedades de un programa con verificación estática y propiedades de un programa con verificación dinámica [Usar]</li> <li>• Dar un ejemplo de un programa que no verifique tipos en un lenguaje particular y sin embargo no tenga error cuando es ejecutado [Usar]</li> <li>• Usar tipos y mensajes de error de tipos para escribir y depurar programas [Usar]</li> <li>• Explicar como las reglas de tipificación definen el conjunto de operaciones que legales para un tipo [Usar]</li> <li>• Escribir las reglas de tipo que rigen el uso de un particular tipo compuesto [Usar]</li> <li>• Explicar por qué indecidibilidad requiere sistemas de tipo para conservadoramente aproximar el comportamiento de un programa [Usar]</li> <li>• Definir y usar piezas de programas (tales como, funciones, clases, métodos) que usan tipos genéricos, incluyendo para colecciones [Usar]</li> <li>• Discutir las diferencias entre, genéricos (<i>generics</i>), subtipo y sobrecarga [Usar]</li> <li>• Explicar múltiples beneficios y limitaciones de tipificación estática en escritura, mantenimiento y depuración de un software [Usar]</li> </ul>
Lecturas : [Str13]	

Unidad 6: Algoritmos y Estructuras de Datos fundamentales (3)	
Competencias esperadas: a,b,d	
Temas	Objetivos de Aprendizaje
<ul style="list-style-type: none"> <li>• Algoritmos numéricos simples, tales como el cálculo de la media de una lista de números, encontrar el mínimo y máximo.</li> <li>• Algoritmos de búsqueda secuencial y binaria.</li> <li>• Algoritmos de ordenamiento de peor caso cuadrático (selección, inserción)</li> <li>• Algoritmos de ordenamiento con peor caso o caso promedio en <math>O(N \lg N)</math> (Quicksort, Heapsort, Mergesort)</li> <li>• Tablas Hash, incluyendo estrategias para evitar y resolver colisiones.</li> <li>• Árboles de búsqueda binaria: <ul style="list-style-type: none"> <li>– Operaciones comunes en árboles de búsqueda binaria como seleccionar el mínimo, máximo, insertar, eliminar, recorrido en árboles.</li> </ul> </li> <li>• Grafos y algoritmos en grafos: <ul style="list-style-type: none"> <li>– Representación de grafos (ej., lista de adyacencia, matriz de adyacencia)</li> <li>– Recorrido en profundidad y amplitud</li> </ul> </li> <li>• Montículos (Heaps)</li> <li>• Grafos y algoritmos en grafos: <ul style="list-style-type: none"> <li>– Problema de corte máximo y mínimo</li> <li>– Búsqueda local</li> </ul> </li> <li>• Búsqueda de patrones y algoritmos de cadenas/texto (ej. búsqueda de subcadena, búsqueda de expresiones regulares, algoritmos de subsecuencia común más larga)</li> </ul>	<ul style="list-style-type: none"> <li>• Implementar algoritmos numéricos básicos [Usar]</li> <li>• Implementar algoritmos de búsqueda simple y explicar las diferencias en sus tiempos de complejidad [Usar]</li> <li>• Ser capaz de implementar algoritmos de ordenamiento comunes cuadráticos y <math>O(N \log N)</math> [Usar]</li> <li>• Describir la implementación de tablas hash, incluyendo resolución y el evitamiento de colisiones [Usar]</li> <li>• Discutir el tiempo de ejecución y eficiencia de memoria de los principales algoritmos de ordenamiento, búsqueda y hashing [Usar]</li> <li>• Discutir factores otros que no sean eficiencia computacional que influyan en la elección de algoritmos, tales como tiempo de programación, mantenibilidad, y el uso de patrones específicos de la aplicación en los datos de entrada [Usar]</li> <li>• Explicar como el balanceamiento del arbol afecta la eficiencia de varias operaciones de un arbol de búsqueda binaria [Usar]</li> <li>• Resolver problemas usando algoritmos básicos de grafos, incluyendo búsqueda por profundidad y búsqueda por amplitud [Usar]</li> <li>• Demostrar habilidad para evaluar algoritmos, para seleccionar de un rango de posibles opciones, para proveer una justificación por esa selección, y para implementar el algoritmo en un contexto en específico [Usar]</li> <li>• Describir la propiedad del heap y el uso de heaps como una implementación de colas de prioridad [Usar]</li> <li>• Resolver problemas usando algoritmos de grafos, incluyendo camino más corto de una sola fuente y camino más corto de todos los pares, y como mínimo un algoritmo de arbol de expansion minima [Usar]</li> <li>• Trazar y/o implementar un algoritmo de comparación de string [Usar]</li> </ul>
Lecturas : [stroustrup2013], [PA18]	

Unidad 7: Programación reactiva y dirigida por eventos (2)	
Competencias esperadas: a,b	
Temas	Objetivos de Aprendizaje
<ul style="list-style-type: none"> <li>• Eventos y controladores de eventos.</li> <li>• Usos canónicos como interfaces gráficas de usuario, dispositivos móviles, robots, servidores.</li> <li>• Uso de frameworks reactivos. <ul style="list-style-type: none"> <li>– Definición de controladores/oyentes (handles/listeners) de eventos.</li> <li>– Bucle principal de eventos no controlado por el escritor controlador de eventos (event-handler-writer)</li> </ul> </li> <li>• Eventos y eventos del programa generados externamente generada.</li> <li>• La separación de modelo, vista y controlador.</li> </ul>	<ul style="list-style-type: none"> <li>• Escribir manejadores de eventos para su uso en sistemas reactivos tales como GUIs [Usar]</li> <li>• Explicar porque el estilo de programación manejada por eventos es natural en dominios donde el programa reacciona a eventos externos [Usar]</li> <li>• Describir un sistema interactivo en términos de un modelo, una vista y un controlador [Usar]</li> </ul>
Lecturas : [stroustrup2013], [Wil11]	

Unidad 8: Árboles y Grafos (7)	
Competencias esperadas: a,b,d	
Temas	Objetivos de Aprendizaje
<ul style="list-style-type: none"> <li>• Árboles. <ul style="list-style-type: none"> <li>– Propiedades</li> <li>– Estrategias de recorrido</li> </ul> </li> <li>• Grafos no dirigidos</li> <li>• Grafos dirigidos</li> <li>• Grafos ponderados</li> <li>• Árboles de expansión/bosques.</li> <li>• Isomorfismo en grafos.</li> </ul>	<ul style="list-style-type: none"> <li>• Ilustrar mediante ejemplos la terminología básica de teoría de grafos, y de alguna de las propiedades y casos especiales de cada tipo de grafos/árboles [Usar]</li> <li>• Demostrar diversos métodos de recorrer árboles y grafos, incluyendo recorridos pre, post e inorden de árboles [Usar]</li> <li>• Modelar una variedad de problemas del mundo real en ciencia de la computación usando formas adecuadas de grafos y árboles, como son la representación de una topología de red o la organización jerárquica de un sistema de archivos [Usar]</li> <li>• Demostrar como los conceptos de grafos y árboles aparecen en estructuras de datos, algoritmos, técnicas de prueba (inducción estructurada), y conteos [Usar]</li> <li>• Explicar como construir un árbol de expansión de un grafo [Usar]</li> <li>• Determinar si dos grafos son isomorfos [Usar]</li> </ul>
Lecturas : [Nak13]	



Unidad 9: Diseño de Software (6)	
Competencias esperadas: a,b	
Temas	Objetivos de Aprendizaje
<ul style="list-style-type: none"> <li>● Principios de diseño del sistema: niveles de abstracción (diseño arquitectónico y el diseño detallado), separación de intereses, ocultamiento de información, de acoplamiento y de cohesión, de reutilización de estructuras estándar.</li> <li>● Diseño de paradigmas tales como diseño estructurado (descomposición funcional de arriba hacia abajo), el análisis orientado a objetos y diseño, orientado a eventos de diseño, diseño de nivel de componente, centrado datos estructurada, orientada a aspectos, orientado a la función, orientado al servicio.</li> <li>● Modelos estructurales y de comportamiento de los diseños de software.</li> <li>● Diseño de patrones.</li> <li>● Relaciones entre los requisitos y diseños: La transformación de modelos, el diseño de los contratos, invariantes.</li> <li>● Conceptos de arquitectura de software y arquitecturas estándar (por ejemplo, cliente-servidor, n-capas, transforman centrados, tubos y filtros).</li> <li>● El uso de componentes de diseño: selección de componentes, diseño, adaptación y componentes de ensamblaje, componentes y patrones, componentes y objetos (por ejemplo, construir una GUI usando un standard widget set)</li> <li>● Diseños de refactorización utilizando patrones de diseño</li> <li>● Calidad del diseño interno, y modelos para: eficiencia y desempeño, redundancia y tolerancia a fallos, trazabilidad de los requerimientos.</li> <li>● Medición y análisis de la calidad de un diseño.</li> <li>● Compensaciones entre diferentes aspectos de la calidad.</li> <li>● Aplicaciones en frameworks.</li> <li>● Middleware: El paradigma de la orientación a objetos con middleware, requerimientos para correr y clasificar objetos, monitores de procesamiento de transacciones y el sistema de flujo de trabajo.</li> <li>● Principales diseños de seguridad y codificación (cross-reference IAS/Principles of secure design). <ul style="list-style-type: none"> <li>– Principio de privilegios mínimos</li> <li>– Principio de falla segura por defecto</li> <li>– Principio de aceptabilidad psicológica</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>● Formular los principios de diseño, incluyendo la separación de problemas, ocultación de información, acoplamiento y cohesión, y la encapsulación [Usar]</li> <li>● Usar un paradigma de diseño para diseñar un sistema de software básico y explicar cómo los principios de diseño del sistema se han aplicado en este diseño [Usar]</li> <li>● Construir modelos del diseño de un sistema de software simple los cuales son apropiado para el paradigma utilizado para diseñarlo [Usar]</li> <li>● En el contexto de un paradigma de diseño simple, describir uno o más patrones de diseño que podrían ser aplicables al diseño de un sistema de software simple [Usar]</li> <li>● Para un sistema simple adecuado para una situación dada, discutir y seleccionar un paradigma de diseño apropiado [Usar]</li> <li>● Crear modelos apropiados para la estructura y el comportamiento de los productos de software desde la especificaciones de requisitos [Usar]</li> <li>● Explicar las relaciones entre los requisitos para un producto de software y su diseño, utilizando los modelos apropiados [Usar]</li> <li>● Para el diseño de un sistema de software simple dentro del contexto de un único paradigma de diseño, describir la arquitectura de software de ese sistema [Usar]</li> <li>● Dado un diseño de alto nivel, identificar la arquitectura de software mediante la diferenciación entre las arquitecturas comunes de software, tales como 3 capas (<i>3-tier</i>), <i>pipe-and-filter</i>, y cliente-servidor [Usar]</li> <li>● Investigar el impacto de la selección arquitecturas de software en el diseño de un sistema simple [Usar]</li> <li>● Aplicar ejemplos simples de patrones en un diseño de software [Usar]</li> <li>● Describir una manera de refactorar y discutir cuando esto debe ser aplicado [Usar]</li> <li>● Seleccionar componentes adecuados para el uso en un diseño de un producto de software [Usar]</li> <li>● Explicar cómo los componentes deben ser adaptados para ser usados en el diseño de un producto de software [Usar]</li> <li>● Diseñar un contrato para un típico componente de software pequeño para el uso de un dado sistema [Usar]</li> <li>● Discutir y seleccionar la arquitectura de software adecuada para un sistema de software simple para un dado escenario [Usar]</li> </ul>

Unidad 10: Ingeniería de Requisitos (1)	
Competencias esperadas: a,b	
Temas	Objetivos de Aprendizaje
<ul style="list-style-type: none"> <li>• Al describir los requisitos funcionales utilizando, por ejemplo, los casos de uso o historias de los usuarios.</li> <li>• Propiedades de requisitos, incluyendo la consistencia, validez, integridad y viabilidad.</li> <li>• Requisitos de software elicitation.</li> <li>• Descripción de datos del sistema utilizando, por ejemplo, los diagramas de clases o diagramas entidad-relación.</li> <li>• Requisitos no funcionales y su relación con la calidad del software.</li> <li>• Evaluación y uso de especificaciones de requisitos.</li> <li>• Requisitos de las técnicas de modelado de análisis.</li> <li>• La aceptabilidad de las consideraciones de certeza/incertidumbre sobre el comportamiento del software/sistema.</li> <li>• Prototipos.</li> <li>• Conceptos básicos de la especificación formal de requisitos.</li> <li>• Especificación de requisitos.</li> <li>• Validación de requisitos.</li> <li>• Rastreo de requisitos.</li> </ul>	<ul style="list-style-type: none"> <li>• Enumerar los componentes clave de un caso de uso o una descripción similar de algún comportamiento que es requerido para un sistema [Usar]</li> <li>• Describir cómo el proceso de ingeniería de requisitos apoya la obtención y validación de los requisitos de comportamiento [Usar]</li> <li>• Interpretar un modelo de requisitos dada por un sistema de software simple [Usar]</li> <li>• Describir los retos fundamentales y técnicas comunes que se utilizan para la obtención de requisitos [Usar]</li> <li>• Enumerar los componentes clave de un modelo de datos (por ejemplo, diagramas de clases o diagramas ER) [Usar]</li> <li>• Identificar los requisitos funcionales y no funcionales en una especificación de requisitos dada por un sistema de software [Usar]</li> <li>• Realizar una revisión de un conjunto de requisitos de software para determinar la calidad de los requisitos con respecto a las características de los buenos requisitos [Usar]</li> <li>• Aplicar elementos clave y métodos comunes para la obtención y el análisis para producir un conjunto de requisitos de software para un sistema de software de tamaño medio [Usar]</li> <li>• Comparar los métodos ágiles y el dirigido por planes para la especificación y validación de requisitos y describir los beneficios y riesgos asociados con cada uno [Usar]</li> <li>• Usar un método común, no formal para modelar y especificar los requisitos para un sistema de software de tamaño medio [Usar]</li> <li>• Traducir al lenguaje natural una especificación de requisitos de software (por ejemplo, un contrato de componentes de software) escrito en un lenguaje de especificación formal [Usar]</li> <li>• Crear un prototipo de un sistema de software para reducir el riesgo en los requisitos [Usar]</li> <li>• Diferenciar entre el rastreo (<i>tracing</i>) hacia adelante y hacia atrás y explicar su papel en el proceso de validación de requisitos [Usar]</li> </ul>
Lecturas : [Str13]	

## 9. PLAN DE TRABAJO

### 9.1 Metodología

Se fomenta la participación individual y en equipo para exponer sus ideas, motivándolos con puntos adicionales en las diferentes etapas de la evaluación del curso.

### 9.2 Sesiones Teóricas

Las sesiones de teoría se llevan a cabo en clases magistrales donde se realizarán actividades que propicien un aprendizaje activo, con dinámicas que permitan a los estudiantes interiorizar los conceptos.

### 9.3 Sesiones Prácticas

Las sesiones prácticas se llevan en clase donde se desarrollan una serie de ejercicios y/o conceptos prácticos mediante planteamiento de problemas, la resolución de problemas, ejercicios puntuales y/o en contextos aplicativos.

## 10. SISTEMA DE EVALUACIÓN

\*\*\*\*\* EVALUATION MISSING \*\*\*\*\*

## 11. BIBLIOGRAFÍA BÁSICA

- [LE13] Stanley B. Lippman and Barbara E.Moo. *C++ Primer*. 5th. O'Reilly, 2013. ISBN: 9780133053043.
- [Nak13] S. Nakariakov. *The Boost C++ Libraries: Generic Programming*. CreateSpace Independent Publishing Platform, 2013.
- [PA18] Praseed Pai and Peter Abraham. *C++ Reactive Programming*. 1st. Packt, 2018.
- [Str13] B Stroustrup. *The C++ Programming Language, 4th edition*. Addison-Wesley, 2013.
- [Van02] David Vandervoorde. *C++ Templates: The Complete Guide*. 1st. Addison-Wesley, 2002. ISBN: 978-0134448237.
- [Wil11] Anthony Williams. *C++ Concurrency in Action*. 1st. Manning, 2011.