Universidad San Ignacio de Loyola (USIL)
Carrera Profesional de
Ingeniería de Software
Sílabo 2021-II

1. CURSO

SFW52095. Programación Orientada a Objetos II (Obligatorio)

2. INFORMACIÓN GENERAL

2.1 Créditos	:	4
2.2 Horas de teoría	:	2 (Semanal)
2.3 Horas de práctica	:	2 (Semanal)
2.4 Horas autónomas	:	128 (horas)
2.5 Duración del periodo	:	16 semanas
2.6 Condición	:	Obligatorio
2.7 Modalidad	:	Presencial
2.8 Prerrequisitos	:	SFW52097. Programación Orientada a Objetos I. (2^{do} 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 functional-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

2) Aplica tópicos de investigación, metodologías, técnicas y mejores prácticas de la Ingeniería de Software para la construcción de soluciones en base al diseño, desarrollo, pruebas, implementación, documentación y mejora continua del Software.. (**Familiarizarse**)

4) Diseña soluciones de Software de acuerdo a los estándares y políticas de seguridad de la información en uno o varios dominios de aplicación siendo socialmente responsables y demostrando ética profesional. (**Familiarizarse**)

7. TEMAS

Unidad 1: Conceptos Fundamentales de Programación (5 horas)	
Competencias esperadas:	
Temas	**Objetivos de Aprendizaje**
• Sintaxis y semántica básica de un lenguaje de alto nivel. • Variables y tipos de datos primitivos (ej., numeros, caracteres, booleanos) • Expresiones y asignaciones. • Operaciones básicas I/O incluyendo archivos I/O. • Estructuras de control condicional e iterativas. • Paso de funciones y parámetros. • Concepto de recursividad.	• 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] • Identifica y describe el uso de tipos de datos primitivos [Usar] • Escribe programas que usan tipos de datos primitivos [Usar] • Modifica y expande programas cortos que usen estructuras de control condicionales e iterativas así como funciones [Usar] • 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] • Escribe un programa que usa E/S de archivos para brindar persistencia a través de ejecuciones múltiples [Usar] • Escoje estructuras de condición y repetición adecuadas para una tarea de programación dada [Usar] • Describe el concepto de recursividad y da ejemplos de su uso [Usar] • Identifica el caso base y el caso general de un problema basado en recursividad [Usar]
Aprendizaje autónomo • Desarrollo de ejercicios prácticos	
Lecturas : [**stroustrup2013**], [Van02], [LE13]	

Unidad 2: Programación orientada a objetos (7 horas)

Competencias esperadas:

Temas	**Objetivos de Aprendizaje**
• Diseño orientado a objetos: – Descomposicion en objetos que almacenan estados y poseen comportamiento – Diseño basado en jerarquia de clases para modelamiento • Definición de las categorías, campos, métodos y constructores. • Las subclases, herencia y método de alteración temporal. • Asignación dinámica: definición de método de llamada. • Subtipificación: – Polimorfismo artículo Subtipo; upcasts implícitos en lenguajes con tipos. – Noción de reemplazo de comportamiento: los subtipos de actuar como supertipos. – Relación entre subtipos y la herencia. • Lenguajes orientados a objetos para la encapsulación: – privacidad y la visibilidad de miembros de la clase – Interfaces revelan único método de firmas – clases base abstractas • Uso de coleccion de clases, iteradores, y otros componentes de la libreria estandar.	• Diseñar e implementar una clase [Usar] • Usar subclase para diseñar una jerarquía simple de clases que permita al código ser reusable por diferentes subclases [Usar] • Razonar correctamente sobre el flujo de control en un programa mediante el envío dinámico [Usar] • Comparar y contrastar (1) el enfoque procedurar/funcional- definiendo una función por cada operación con el cuerdo 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] • Explicar la relación entre la herencia orientada a objetos (codigo compartido y *overriding*) y subtipificación (la idea de un subtipo es ser utilizable en un contexto en el que espera al supertipo) [Usar] • Usar mecanismos de encapsulación orientada a objetos, tal como interfaces y miembros privados [Usar] • Definir y usar iteradores y otras operaciones sobre agregaciones, incluyendo operaciones que tienen funciones como argumentos, en múltiples lenguajes de programación, selecionar la forma mas natural por cada lenguaje [Usar]
Aprendizaje autónomo • Desarrollo de ejercicios prácticos	
Lecturas : [Str13]	

Unidad 3: Algoritmos y Diseño (5 horas)

Competencias esperadas:

Temas	**Objetivos de Aprendizaje**

• Conceptos y propiedades de los algoritmos
  – Comparación informal de la eficiencia de los algoritmos (ej., conteo de operaciones)
• Rol de los algoritmos en el proceso de solución de problemas
• Estrategias de solución de problemas
  – Funciones matemáticas iterativas y recursivas
  – Recorrido iterativo y recursivo en estructura de datos
  – Estrategias Divide y Conquistar
• Conceptos y principios fundamentales de diseño
  – Abstracción
  – Descomposición de Program
  – Encapsulamiento y camuflaje de información
  – Separación de comportamiento y aplicación
	• Discute la importancia de los algoritmos en el proceso de solución de un problema [Usar]
	• Discute como un problema puede ser resuelto por múltiples algoritmos, cada uno con propiedades diferentes [Usar]
	• Crea algoritmos para resolver problemas simples [Usar]
	• Usa un lenguaje de programación para implementar, probar, y depurar algoritmos para resolver problemas simples [Usar]
	• Implementa, prueba, y depura funciones recursivas simples y sus procedimientos [Usar]
	• Determina si una solución iterativa o recursiva es la más apropiada para un problema [Usar]
	• Implementa un algoritmo de divide y vencerás para resolver un problema [Usar]
	• Aplica técnicas de descomposición para dividir un programa en partes más pequeñas [Usar]
	• Identifica los componentes de datos y el comportamiento de mútiples tipos de datos abstractos [Usar]
	• Implementa un tipo de dato abstracto coherente, con la menor pérdida de acoplamiento entre componentes y comportamientos [Usar]
	• Identifica las fortalezas y las debilidades relativas entre múltiples diseños e implementaciones de un problema [Usar]

Aprendizaje autónomo • Desarrollo de ejercicios prácticos	
Lecturas : [**stroustrup2013**], [**Weert16**], [LE13]	

Unidad 4: Análisis Básico (3 horas)

Competencias esperadas:

Temas	Objetivos de Aprendizaje
• Diferencias entre el mejor, el esperado y el peor caso de un algoritmo. • Análisis asintótico de complejidad de cotas superior y esperada. • Definición formal de la Notación Big O. • Clases de complejidad como constante, logarítmica, lineal, cuadrática y exponencial. • Medidas empíricas de desempeño. • Compensación entre espacio y tiempo en los algoritmos. • Uso de la notación Big O. • Notación Little o, Big omega y Big theta. • Relaciones recurrentes. • Análisis de algoritmos iterativos y recursivos. • Teorema Maestro y Árboles Recursivos.	• Explique a que se refiere con "mejor", "esperado" y "peor" caso de comportamiento de un algoritmo [Usar] • En el contexto de a algoritmos específicos, identifique las características de data y/o otras condiciones o suposiciones que lleven a diferentes comportamientos [Usar] • Determine informalmente el tiempo y el espacio de complejidad de diferentes algoritmos [Usar] • Indique la definición formal de Big O [Usar] • Lista y contraste de clases estándares de complejidad [Usar] • Realizar estúdios empíricos para validar una hipótesis sobre runtime stemming desde un análisis matemático Ejecute algoritmos con entrada de varios tamaños y compare el desempeño [Usar] • Da ejemplos que ilustran las compensaciones entre espacio y tiempo que se dan en los algoritmos [Usar] • Use la notación formal de la Big O para dar límites superiores asintóticos en la complejidad de tiempo y espacio de los algoritmos [Usar] • Usar la notación formal Big O para dar límites de casos esperados en el tiempo de complejidad de los algoritmos [Usar] • Explicar el uso de la notación theta grande, omega grande y o pequeña para describir la cantidad de trabajo hecho por un algoritmo [Usar] • Usar relaciones recurrentes para determinar el tiempo de complejidad de algoritmos recursivamente definidos [Usar] • Resuelve relaciones de recurrencia básicas, por ejemplo. usando alguna forma del Teorema Maestro [Usar]

Aprendizaje autónomo

- Desarrollo de ejercicios prácticos

Lecturas : [Str13]

Unidad 5: Sistemas de tipos básicos (5 horas)

Competencias esperadas:

Temas	Objetivos de Aprendizaje

• Tipos como conjunto de valores junto con un conjunto de operaciones.
  – Tipos primitivos (p.e. números, booleanos)
  – Composición de tipos construídos de otros tipos (p.e., registros, uniones, arreglos, listas, funciones, referencias)
• Asociación de tipos de variables, argumentos, resultados y campos.
• Tipo de seguridad y los errores causados por el uso de valores de manera incompatible dadas sus tipos previstos.
• Metas y limitaciones de tipos estáticos
  – Eliminación de algunas clases de errores sin ejecutar el programa
  – Indecisión significa que un análisis estatico puede aproximar el comportamiento de un programa
• Tipos genéricos (polimorfismo paramétrico)
  – Definición
  – Uso de librerías genéricas tales como colecciones.
  – Comparación con polimorfismo ad-hoc y polimorfismo de subtipos
• Beneficios complementarios de tipos estáticos y dinámicos:
  – Errores tempranos vs. errores tardíos/evitados.
  – 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.
  – Evitar el mal uso del código vs. permitir más reuso de código.
  – Detectar programas incompletos vs. permitir que programas incompletos se ejecuten
	• Tanto para tipo primitivo y un tipo compuesto, describir de manera informal los valores que tiene dicho tipo [Usar]
	• 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]
	• Describir ejemplos de errores de programa detectadas por un sistema de tipos [Usar]
	• Para múltiples lenguajes de programación, identificar propiedades de un programa con verificación estática y propiedades de un programa con verificación dinámica [Usar]
	• Dar un ejemplo de un programa que no verifique tipos en un lenguaje particular y sin embargo no tenga error cuando es ejecutado [Usar]
	• Usar tipos y mensajes de error de tipos para escribir y depurar programas [Usar]
	• Explicar como las reglas de tipificación definen el conjunto de operaciones que legales para un tipo [Usar]
	• Escribir las reglas de tipo que rigen el uso de un particular tipo compuesto [Usar]
	• Explicar por qué indecidibilidad requiere sistemas de tipo para conservadoramente aproximar el comportamiento de un programa [Usar]
	• Definir y usar piezas de programas (tales como, funciones, clases, métodos) que usan tipos genéricos, incluyendo para colecciones [Usar]
	• Discutir las diferencias entre, genéricos (*generics*), subtipo y sobrecarga [Usar]
	• Explicar múltiples beneficios y limitaciones de tipificación estática en escritura, mantenimiento y depuración de un software [Usar]

Aprendizaje autónomo

- Desarrollo de ejercicios prácticos

Lecturas : [Str13]

Unidad 6: Algoritmos y Estructuras de Datos fundamentales (3 horas)

Competencias esperadas:

Temas	**Objetivos de Aprendizaje**

• Algoritmos numéricos simples, tales como el cálculo de la media de una lista de números, encontrar el mínimo y máximo.
• Algoritmos de búsqueda secuencial y binaria.
• Algoritmos de ordenamiento de peor caso cuadrático (selección, inserción)
• Algoritmos de ordenamiento con peor caso o caso promedio en O(N lg N) (Quicksort, Heapsort, Mergesort)
• Tablas Hash, incluyendo estratégias para evitar y resolver colisiones.
• Árboles de búsqueda binaria:
  – Operaciones comunes en árboles de búsqueda binaria como seleccionar el mínimo, máximo, insertar, eliminar, recorrido en árboles.
• Grafos y algoritmos en grafos:
  – Representación de grafos (ej., lista de adyacencia, matriz de adyacencia)
  – Recorrido en profundidad y amplitud
• Montículos (Heaps)
• Grafos y algoritmos en grafos:
  – Problema de corte máximo y mínimo
  – Busqueda local
• Búsqueda de patrones y algoritmos de cadenas/texto (ej. búsqueda de subcadena, búsqueda de expresiones regulares, algoritmos de subsecuencia común más larga)
	• Implementar algoritmos numéricos básicos [Usar]
	• Implementar algoritmos de busqueda simple y explicar las diferencias en sus tiempos de complejidad [Usar]
	• Ser capaz de implementar algoritmos de ordenamiento comunes cuádraticos y O(N log N) [Usar]
	• Describir la implementación de tablas hash, incluyendo resolución y el evitamiento de colisiones [Usar]
	• Discutir el tiempo de ejecución y eficiencia de memoria de los principales algoritmos de ordenamiento, busqueda y hashing [Usar]
	• 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]
	• Explicar como el balanceamiento del arbol afecta la eficiencia de varias operaciones de un arbol de búsqueda binaria [Usar]
	• Resolver problemas usando algoritmos básicos de grafos, incluyendo busqueda por profundidad y busqueda por amplitud [Usar]
	• 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]
	• Describir la propiedad del heap y el uso de heaps como una implementación de colas de prioridad [Usar]
	• 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]
	• Trazar y/o implementar un algoritmo de comparación de string [Usar]

Aprendizaje autónomo • Desarrollo de ejercicios prácticos	
Lecturas : [**stroustrup2013**], [PA18]	

Unidad 7: Programación reactiva y dirigida por eventos (2 horas)	
Competencias esperadas:	
Temas	**Objetivos de Aprendizaje**
• Eventos y controladores de eventos. • Usos canónicos como interfaces gráficas de usuario, dispositivos móviles, robots, servidores. • Uso de frameworks reactivos. – Definición de controladores/oyentes (handles/listeners) de eventos. – Bucle principal de enventos no controlado po el escritor controlador de eventos (event-handler-writer) • Eventos y eventos del programa generados externamente generada. • La separación de modelo, vista y controlador.	• Escribir manejadores de eventos para su uso en sistemas reactivos tales como GUIs [Usar] • Explicar porque el estilo de programación manejada por eventos es natural en dominios donde el programa reacciona a eventos externos [Usar] • Describir un sistema interactivo en términos de un modelo, una vista y un controlador [Usar]
Aprendizaje autónomo • Desarrollo de ejercicios prácticos	
Lecturas : [**stroustrup2013**], [Wil11]	

Unidad 8: Árboles y Grafos (7 horas)	
Competencias esperadas:	
Temas	**Objetivos de Aprendizaje**
• Árboles. – Propiedades – Estrategias de recorrido • Grafos no dirigidos • Grafos dirigidos • Grafos ponderados • Arboles de expansion/bosques. • Isomorfismo en grafos.	• 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] • Demostrar diversos métodos de recorrer árboles y grafos, incluyendo recorridos pre, post e inorden de árboles [Usar] • 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] • Demuestrar como los conceptos de grafos y árboles aparecen en estructuras de datos, algoritmos, técnicas de prueba (inducción estructurada), y conteos [Usar] • Explicar como construir un árbol de expansión de un grafo [Usar] • Determinar si dos grafos son isomorfos [Usar]
Aprendizaje autónomo • Desarrollo de ejercicios prácticos	
Lecturas : [Nak13]	

Unidad 9: Diseño de Software (6 horas)	
Competencias esperadas:	
Temas	**Objetivos de Aprendizaje**
• 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. • 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. • Modelos estructurales y de comportamiento de los diseños de software. • Diseño de patrones. • Relaciones entre los requisitos y diseños: La transformación de modelos, el diseño de los contratos, invariantes. • Conceptos de arquitectura de software y arquitecturas estándar (por ejemplo, cliente-servidor, n-capas, transforman centrados, tubos y filtros). • El uso de componentes de diseño: seleccion de componentes,diseño,adaptacion y componentes de ensamblaje, componentes y patrones, componentes y objetos(por ejemplo,construir una GUI usando un standar widget set) • Diseños de refactorización utilizando patrones de diseño • Calidad del diseño interno, y modelos para: eficiencia y desempeño, redundancia y tolerancia a fallos, trazavilidad de los requerimientos. • Medición y análisis de la calidad de un diseño. • Compensasiones entre diferentes aspectos de la calidad. • Aaplicaciones en frameworks. • Middleware: El paradigma de la orientacion a objetos con middleware, requerimientos para correr y clasificar objetos, monitores de procesamiento de transacciones y el sistema de flujo de trabajo. • Principales diseños de seguridad y codificación(cross-reference IAS/Principles of securre design). – Principio de privilegios mínimos – Principio de falla segura por defecto – Principio de aceptabilidad psicológica	• Formular los principios de diseño, incluyendo la separación de problemas, ocultación de información, acoplamiento y cohesión, y la encapsulación [Usar] • 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] • Construir modelos del diseño de un sistema de software simple los cuales son apropiado para el paradigma utilizado para diseñarlo [Usar] • 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] • Para un sistema simple adecuado para una situación dada, discutir y seleccionar un paradigma de diseño apropiado [Usar] • Crear modelos apropiados para la estructura y el comportamiento de los productos de software desde la especificaciones de requisitos [Usar] • Explicar las relaciones entre los requisitos para un producto de software y su diseño, utilizando los modelos apropiados [Usar] • 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] • Dado un diseño de alto nivel, identificar la arquitectura de software mediante la diferenciación entre las arquitecturas comunes de software, tales como 3 capas (*3-tier*), *pipe-and-filter*, y cliente-servidor [Usar] • Investigar el impacto de la selección arquitecturas de software en el diseño de un sistema simple [Usar] • Aplicar ejemplos simples de patrones en un diseño de software [Usar] • Describir una manera de refactorar y discutir cuando esto debe ser aplicado [Usar] • Seleccionar componentes adecuados para el uso en un diseño de un producto de software [Usar] • Explicar cómo los componentes deben ser adaptados para ser usados en el diseño de un producto de software [Usar] • Diseñar un contrato para un típico componente de software pequeño para el uso de un dado sistema [Usar] • Discutir y seleccionar la arquitectura de software adecuada para un sistema de software simple para un dado escenario [Usar]

Unidad 10: Ingeniería de Requisitos (1 horas)

Competencias esperadas:

Temas	**Objetivos de Aprendizaje**
• Al describir los requisitos funcionales utilizando, por ejemplo, los casos de uso o historias de los usuarios. • Propiedades de requisitos, incluyendo la consistencia, validez, integridad y viabilidad. • Requisitos de software elicitatión. • Descripción de datos del sistema utilizando, por ejemplo, los diagramas de clases o diagramas entidad-relación. • Requisitos no funcionales y su relación con la calidad del software. • Evaluación y uso de especificaciones de requisitos. • Requisitos de las técnicas de modelado de análisis. • La aceptabilidad de las consideraciones de certeza/incertidumbre sobre el comportamiento del software/sistema. • Prototipos. • Conceptos básicos de la especificación formal de requisitos. • Especificación de requisitos. • Validación de requisitos. • Rastreo de requisitos.	• Enumerar los componentes clave de un caso de uso o una descripción similar de algún comportamiento que es requerido para un sistema [Usar] • Describir cómo el proceso de ingeniería de requisitos apoya la obtención y validación de los requisitos de comportamiento [Usar] • Interpretar un modelo de requisitos dada por un sistema de software simple [Usar] • Describir los retos fundamentales y técnicas comunes que se utilizan para la obtención de requisitos [Usar] • Enumerar los componentes clave de un modelo de datos (por ejemplo, diagramas de clases o diagramas ER) [Usar] • Identificar los requisitos funcionales y no funcionales en una especificación de requisitos dada por un sistema de software [Usar] • 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] • 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] • 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] • Usar un método común, no formal para modelar y especificar los requisitos para un sistema de software de tamaño medio [Usar] • 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] • Crear un prototipo de un sistema de software para reducir el riesgo en los requisitos [Usar] • Diferenciar entre el rastreo (*tracing*) hacia adelante y hacia atrás y explicar su papel en el proceso de validación de requisitos [Usar]

Aprendizaje autónomo

- Desarrollo de ejercicios prácticos

Lecturas : [Str13]

8. PLAN DE TRABAJO

8.1 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.

8.2 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.

8.3 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.

9. SISTEMA DE EVALUACIÓN

Cada uno de los rubros del esquema de evaluación y la nota final del curso son redondeados a números enteros. La nota final del curso es el promedio ponderado de los rubros correspondientes: evaluación permanente, examen parcial y examen final.

Los promedios calculados componentes del rubro 'Evaluación Permanente' mantendrán su cálculo con 2 decimales.

	%	Observaciones	Semana	Rezagable
Evaluación Continua	70%			
Práctica Calificada	70%			
Práctica Calificada$_1$		Se elimina la práctica con la menor nota	4	No
Práctica Calificada$_2$		Se elimina la práctica con la menor nota	8	No
Práctica Calificada$_3$		Se elimina la práctica con la menor nota	12	No
Proyecto	30%		15	
Examen final	30%			

10. 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 Platforml, 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.