PROGRAMA DE CURSO
METODOLOGÍAS DE DISEÑO Y PROGRAMACIÓN

A. Antecedentes generales del curso:

Departamento	Ciencias de la computación					
Nombre del curso	Metodologías de Diseño y Programación		Código	CC3002	Créditos	6
Nombre del curso en inglés	*Design and Programming Methodologies*					
Horas semanales	Docencia	3	Auxiliares	1,5	Trabajo personal	5,5
Carácter del curso	Obligatorio	X		Electivo		
Requisitos	CC3001: Algoritmos y Estructuras de datos					

B. Propósito del curso:

El curso, que se se ubica en el VI semestre, en el ciclo de Licenciatura, tiene como propósito que las y los estudiantes desarrollen software de pequeña y mediana complejidad, considerando técnicas de diseño y programación orientada a objetos. Para ello, aplican patrones de diseño, metodologías de testeo y depuración para producir un software flexible, abierto y robusto.

Asimismo, analizan dilemas éticos asociados al desarrollo e implementación de software, considerando, desde un punto de vista ético, el respeto por la propiedad intelectual, el uso de software libre y licencias de software, y el uso de sistemas de escritura universales más inclusivos como, por ejemplo, Unicode.

El curso tributa a las siguientes competencias específicas (CE) y genéricas (CG):

CE1: Analizar problemas computacionales, construir modelos, expresándolos en representaciones y lenguajes formales adecuados.

CE5: Concebir, diseñar y construir soluciones de software, siguiendo un proceso sistemático y cuantificable, acorde a los fundamentos, eligiendo el paradigma y las técnicas más adecuadas.

CE8: Diagnosticar y resolver problemas en el funcionamiento de software cercano a la plataforma para mejorar su desempeño.

CG3: Compromiso ético

Actuar de manera responsable y honesta, dando cuenta en forma crítica de sus propias acciones y sus consecuencias, en el marco del respeto hacia la dignidad de las personas y el cuidado del medio social, cultural y natural.

C. Resultados de aprendizaje:

Competencias específicas	Resultados de aprendizaje
CE1, CE5	RA1: Desarrolla productos de software de pequeña y mediana complejidad, orientados a objetos, considerando patrones de diseño, de programación y metodologías de testeo, con el objetivo de producir software flexible, abierto y robusto.
CE8	RA2: Utiliza metodologías de diseño, de programación, de testeo y depuración, a fin de desarrollar una aplicación extensible para futuros desarrolladores, mediante el uso de lenguaje de programación orientado a objetos, así como depuradores y framework, tales como JUnit, entre otros.
Competencias genéricas	**Resultados de aprendizaje**
CG3	RA3: Analiza dilemas éticos asociados al desarrollo e implementación de software, considerando el respeto por la propiedad intelectual, el uso de software libre, de licencias y sistemas de escritura universales inclusivos, como el Unicode.

D. Unidades temáticas:

Número	RA al que tributa	Nombre de la unidad	Duración en semanas
1	RA1	Objetos, clases, tipos y polimorfismo	2 semanas
Contenidos		**Indicador de logro**	
1.1. Sintaxis de lenguaje de programación para definir Objetos y Clases. 1.2. Objetos: estado, comportamiento, identidad. 1.3. Mensajes y funciones. 1.4. Propiedad de clases.		La/el estudiante: 1. Programa usando sintaxis de lenguaje de programación para clases y objetos. 2. Identifica los conceptos de estado, comportamiento e identidad, a partir de ejemplos. 3. Traduce a código un problema con su correspondiente abstracción, considerando cuáles son los objetos y su comportamiento. 4. Analiza la sintaxis de un código programado en un lenguaje orientado a objetos.	
Bibliografía de la unidad		[1][4]	

Número	RA al que tributa	Nombre de la unidad	Duración en semanas
2	RA1, RA2	Herramientas para el desarrollo de software	2 semanas
Contenidos		**Indicador de logro**	
2.1. Herramientas IDEs (Integrated Development Environment). 2.2. Versionamiento. 2.3. Refactoring.		La/el estudiante: 1. Analiza las ventajas de usar la herramienta de IDEs, a partir de ejemplos que se le planteen. 2. Utiliza IDEs para escribir un código. 3. Usa el versionamiento para procesar, guardar, mantener cambios en el código fuente de software. 4. Identifica los refactoring más usados para mejorar la legibilidad y la mantención de un código fuente. 5. Mejora la legibilidad y mantención de un código fuente, usando refactoring.	
Bibliografía de la unidad		[9] [10]	

Número	RA al que tributa	Nombre de la unidad	Duración en semanas
3	RA1, RA2	Testeo y desarrollo iterativo de una aplicación simple	3 semanas
Contenidos		**Indicador de logro**	
3.1. Métodos de testeo, basado en JUnit, entre otros. 3.2. Diseño por contratos. 3.3. Técnicas básicas de diseño con UML (diagrama de clase y de secuencia). 3.4. Técnica de depuración. 3.5. Ejemplos simples mediante el uso de desarrollo iterativo.		La/el estudiante: 1. Identifica distintos métodos de testeo para software, considerando ventajas y limitaciones de cada uno de los métodos. 2. Escribe un test para cada clase, considerando su interfaz pública y privada. 3. Prueba la interfaz de una clase, usando frameworks tales como JUnit, entre otros. 4. Valida las funcionalidades de la aplicación, considerando las operaciones y acciones asignadas a los objetos. 5. Aplica la metodología de desarrollo iterativo para una aplicación simple, usando UML. 6. Escribe una aplicación simple, flexible, abierta y robusta, aplicando técnicas de testeo y desarrollo iterativo.	
Bibliografía de la unidad		[2],[4]	

Número	RA al que tributa	Nombre de la unidad	Duración en semanas
4	RA2	Patrones de diseño de software	4 semanas
Contenidos		**Indicador de logro**	
4.1. Importancia de las objetos, clases, tipos y polimorfismos, métodos y metodologías para el diseño de software. 4.2. Patrones de diseño: adaptador, proxy, template, observador, puente, singleton y fábrica abstracta, visitor, entre otros. 4.3. Ejemplos de desarrollo de software basado en patrones.		La/el estudiante: 1. Identifica y analiza situaciones en las que se requiere usar un patrón de diseño. 2. Analiza ejemplos de desarrollo de software basado en patrones. 3. Determina potenciales mejoras que se podrían realizar a un algoritmo para facilitar la extensibilidad del software. 4. Aplica y programa patrones de diseño adecuados para el desarrollo de software flexible y fácil de extender. 5. Diseña software flexibles y fáciles de extender.	
Bibliografía de la unidad		[4],[5]	

Número	RA al que tributa	Nombre de la unidad	Duración en semanas
5	RA1, RA3	Aspectos éticos asociados al desarrollo de software	1,5 semanas
Contenidos		**Indicador de logro**	
5.1. Consideraciones éticas para el desarrollo de software: 5.2. Uso ético de licencias de softwares; software libre y propiedad intelectual. 5.3. Importancia del uso de diferentes sistemas de escritura. (ejemplo Unicode) para desarrollar softwares amigables para el usuario.		La/el estudiante: 1. Analiza problemas éticos relacionados con la propiedad intelectual de un software o artefacto, considerando la importancia de respetar los términos del contrato o licencia. 2. Reflexiona sobre la distribución y uso del software o artefacto, identificando distintos tipos de licencias y derechos del autor. 3. Analiza los derechos asociados al uso de un software libre, considerando su importancia para el diseño de software. 4. Determina la importancia de procesar datos, considerando diferentes sistemas de escritura (Unicode).	
Bibliografía de la unidad		[6], [7], [8]	

Número	RA al que tributa	Nombre de la unidad	Duración en semanas
6	RA1, RA2	Tópicos avanzados de programación	2,5 semanas
Contenidos		Indicador de logro	
6.1. Interfaz gráfica. 6.2. Excepciones. 6.3. Thread. 6.4. Generic. 6.5. Rendimiento de aplicación. 6.6. Metodología de depuración de rendimiento.		La/el estudiante: 1. Utiliza un framework para definir la interfaz gráfica del usuario, por ejemplo, JavaFX y Swing. 2. Usa excepciones para modelar errores. 3. Identifica y evalúa fallas de rendimiento asociadas a la CPU y memoria.	
Bibliografía de la unidad		[1], [3]	

E. Estrategias de enseñanza - aprendizaje:

La metodología de enseñanza y aprendizaje del curso es activo–participativa; incluye **clases expositivas** donde se presentan los principales conceptos a trabajar en la sesión, considerando la participación activa de los estudiantes al analizar y resolver tareas y ejercicios en base a ejemplos, problemas o desafíos.

A partir de lo anterior, el curso considera, además, las siguientes metodologías:

- **Resolución de problemas:** a partir de tareas y/o ejercicios, a los y las estudiantes se les presenta un problema, el que deben resolver dados los requerimientos iniciales.
- **Análisis de caso:** las y los estudiantes analizan ejemplos de diseño adecuados a un tipo de problema a resolver y aplican los conocimientos adquiridos a otras situaciones que se le presentan.

F. Estrategias de evaluación:

Al inicio de cada semestre, la académica o académico informará a las y los estudiantes sobre los tipos y cantidad de evaluaciones, así como las ponderaciones correspondientes.

Para esta propuesta de programa, el curso considera las siguientes instancias de evaluación**:**

- **Controles:** donde se evalúan conceptos teóricos o se resuelven problemas asociados a los resultados de aprendizaje y a los contenidos de las unidades.
- **Tareas:** incluye trabajo personal por parte del estudiante para analizar y resolver problemas atingentes que se le presenten.
- **Examen:** busca evaluar de forma integradora los aprendizajes adquiridos en el curso.

G. Recursos bibliográficos:

Bibliografía obligatoria:

[1] Evans, B., Flanagan, D. (2018). *Java in a nutshell*. O'Reilly Media, Inc. ISBN: 9781492037255. https://www.oreilly.com/library/view/java-in-a/9781492037248/

[2] Tudose, C., Tahchiev, P., Leme, F., Massol, V., and Gregory, G. (2019). *JUnit in Action*. Manning Publication. ISBN: 978-1617297045. https://www.manning.com/books/junit-in-action-third-edition

[3] Sharan, K. (2015). *Learn JavaFX 8: Building User Experience and Interfaces with Java 8*. Apress. ISBN: 978-1484211434 https://www.amazon.com/Learn-JavaFX-Building-Experience-Interfaces/dp/148421143X.

[4] Timothy C. Lethbridge Robert Laganière. (2005). *Object-Oriented Software Engineering. Practical Software Development using UML and Java.* McGraw- hill Education. Second Edition. ISBN: 978-0077109080 https://fall14cs.files.wordpress.com/2016/03/object-oriented-software-engineering-practical-software-development-using-uml-and-java-2005.pdf.

[5] Freeman, E., Bates, B., Sierra, K., Robson, E. (2004). *Head first in design patterns*. O'Relly First Edition. https://www.amazon.com/Head-First-Design-Patterns-Brain-Friendly/dp/0596007124.

[6] Seoane Pascual, J., González Barahona, J.M., Robles, G. (2007). *Introducción al software libre.* http://softlibre.unizar.es/manuales/softwarelibre/sobre.pdf.

[7] Seoane Pascual, J., González Barahona, J.M., Robles, G. (2007). *Introduction to Free Software.* https://archive.org/details/ost-computer-science-fta-m1-intro_to_fs-v1.

[8] Korpela, J.K. (2006). *Unicode Explained.* O'Reilly Media, Inc. ISBN: 9780596101213 https://www.oreilly.com/library/view/unicode-explained/059610121X/.

[9] Fowler, M. (2018). Refactoring: Improving the Design of Existing Code (2nd Edition). https://www.amazon.com/Refactoring-Improving-Existing-Addison-Wesley-Signature/dp/0134757599.

[10] Meszaros, G. (2007). xUnit Test Patterns: Refactoring Test Code. https://www.amazon.com/xUnit-Test-Patterns-Refactoring-Code/dp/0131495054.

Bibliografía complementaria:

[11] https://www.eclipse.org

[12] https://junit.org/junit5/

H. Datos generales sobre elaboración y vigencia del programa de curso:

Vigencia desde:	Primavera, 2021
Elaborado por:	Alexandre Bergel, Nancy Hitschfeld
Validado por:	Revisión y validación entre académicos: Jocelyn Simmonds. Validación CTD de Computación
Revisado por:	Área de Gestión Curricular