UNIVERSIDAD NACIONAL DE RÍO CUARTO

FACULTAD DE CIENCIAS EXACTAS, FÍSICO-QUÍMICAS Y NATURALES

DEPARTAMENTO DE COMPUTACIÓN

CARRERA/S: Licenciatura en Ciencias de la Computación

ASIGNATURA: Validación y Verificación de Software (3308)

DOCENTE RESPONSABLE: Dra. Valeria Bengolea

EQUIPO DOCENTE: Dra. Valeria Bengolea

AÑO ACADÉMICO: 2019

REGIMEN DE LA ASIGNATURA: Cuatrimestral

RÉGIMEN DE CORRELATIVIDADES:

Aprobada	*Regular*
Programación Avanzada (1948)	Análisis Comparativo de Lenguajes (1956)

CARGA HORARIA TOTAL: 8 horas semanales

TEÓRICAS: 4 hs **PRÁCTICAS:** 4hs

CARÁCTER DE LA ASIGNATURA: Optativa

A. CONTEXTUALIZACIÓN DE LA ASIGNATURA

Validación y Verificación de Software será asignatura optativa de la Licenciatura en Ciencias de la Computación. Según el plan de estudios de la Licenciatura en Ciencias de la Computación, pueden cursarse materias optativas en los años cuarto y quinto de la misma. Las correlatividades fijadas permiten que esta asignatura pueda ser cursada en cualquiera de estos años.

B. OBJETIVOS PROPUESTOS

El objetivo esencial de la materia es lograr que los alumnos se familiaricen con las técnicas más importantes de validación y verificación de software disponibles en la actualidad, que comprendan los fundamentos teóricos que hacen posibles la aplicación de estas técnicas en la práctica, y sepan aplicarlas.

C. CONTENIDOS BÁSICOS DEL PROGRAMA A DESARROLLAR

El rol de la validación y la verificación en la calidad del software. Confiabilidad de software. El problema de construir programas correctos. ¿Qué es un programa *correcto*?

El testing como actividad de validación y verificación. Conceptos fundamentales de testing, sus objetivos y principios. El *testing* como especificación. Pruebas unitarias: conceptos de prueba y granularidad. Automatización de testing usando frameworks xUnit. Estructuración y composición de tests de unidad. Construcción de suites de testing. Revisión de criterios y técnicas de diseño de casos de prueba. Testing de caja negra y caja blanca. Criterios de cobertura. Herramientas de medición de cobertura. Independencia en unidades de test. El concepto de doble de prueba y su necesidad. Tipos de dobles: dummies, stubs, mocks, etc. Herramientas de apoyo. Integración continua. Estrategias de integración y herramientas.

Necesidad de las especificaciones de programas. Revisión de enfoques tradicionales a la validación y verificación de programas. Testing y sus limitaciones. Programas correctos por construcción. Problemas en la automatización de las técnicas tradicionales de validación y verificación de programas. Interacción entre testing y verificación. Generación automática de tests.

El problema de construir y razonar sobre programas concurrentes. Algunas técnicas clásicas para la verificación de programas concurrentes. Las álgebras de procesos. El álgebra de procesos FSP. Modelado de procesos concurrentes en FSP. FSP en el modelado de objetos compartidos y exclusión mutua. Diferentes modelos de interacción entre procesos. Monitores, sincronización. Propiedades de safety y liveness en FSP, y su análisis mediante LTSA.

Comprobación de modelos (*model checking*). La lógica modal. Definiciones fundamentales. Operadores modales y sus distintas interpretaciones. Sintaxis y semántica de la lógica modal proposicional. Estructuras de Kripke. Relación entre estructuras de Kripke y la semántica de programas. Lógicas temporales. La lógica temporal como una lógica modal. Descripción de propiedades relacionadas con el tiempo mediante la utilización de operadores modales. Lógica temporal lineal y lógica temporal ramificada. Model checking para la generación automática de tests.

Sintaxis y semántica formales de la lógica temporal lineal proposicional. Autómatas de Büchi y su relación con modelos de la lógica temporal lineal proposicional. Traducción de

interpretaciones y fórmulas de la lógica temporal lineal proposicional a autómatas de Büchi. Reducción de la comprobación de modelos en lógica temporal lineal proposicional a un problema en Teoría de Autómatas. Automatización en la verificación de propiedades temporales mediante el *model checking*. Ejemplos de su uso en la práctica. La concurrencia y las dificultades en la comprensión de programas concurrentes. Utilización de la herramienta SPIN. El lenguaje ProMeLa. Especificación de programas concurrentes en ProMeLa. Caracterización de propiedades de programas concurrentes mediante el uso de lógica temporal lineal. Ejemplos. Limitaciones del *model checking* en relación a la lógica temporal lineal proposicional.

La lógica como lenguaje formal y su expresividad. Lógica proposicional. Sus ventajas y sus limitaciones en poder expresivo. Reducción del problema de la deducción a SAT solving proposicional. Automatización de la deducción. La lógica de primer orden. Características y poder expresivo. Restricciones para la aplicación de SAT solving para deducción en lógica de primer orden. Representación de diseños de sistemas mediante lógica de primer orden. Algunas extensiones importantes a la lógica de primer orden. La lógica relacional. Descripción de diseños y propiedades estructurales mediante la lógica relacional. El poder de SAT solving de primer orden como una técnica de validación de sistemas. Limitaciones. La herramienta Alloy. Definición de diseños mediante signaturas y funciones de transformación de estados. Validación de diseños en Alloy. Utilización en la práctica. Ejemplos. Alloy y sus limitaciones para razonar sobre aspectos dinámicos de programas. La extensión DynAlloy. Aplicaciones de DynAlloy. Estrategias para lidiar con problemas de escalabilidad en análisis basado en satisfactibilidad booleana.

D. FUNDAMENTACIÓN DE LOS CONTENIDOS

La confiabilidad es uno de los atributos de calidad de software más importantes, y las actividades de validación y verificación de software son esenciales para su garantía. Todo profesional de Ciencias de la Computación debe conocer las técnicas actuales de validación y verificación, las herramientas de soporte para las mismas, y los fundamentos subyacentes, especialmente dado que estas actividades se consideran de las más costosas en la producción de software.

Debido a que la validación y verificación de software son áreas de las Ciencias de la Computación fuertemente basadas en lógica y matemática discreta, es necesario estudiar con cierto detalle los resultados teóricos que hacen posible la aplicación de las técnicas de validación y verificación presentadas en esta asignatura. Dado que estás áreas son sumamente prácticas, se hace énfasis en la asignatura en el uso de herramientas de software para la asistencia en la verificación.

E. ACTIVIDADES A DESARROLLAR

Se pondrá especial énfasis en la aplicación de las principales técnicas de validación y verificación de software en problemas concretos. Las clases serán por lo tanto teórico-prácticas. Se fomentará el uso de herramientas de software para la asistencia en la verificación (las técnicas elegidas cuentan, todas ellas, con herramientas de soporte).

Los dos trabajos prácticos obligatorios, cuya aprobación es requisito para la regularidad, tienen por objetivo lograr que los alumnos puedan aplicar la teoría aprendida en la resolución, mediante la asistencia de herramientas software para la asistencia en la validación y verificación de software. Además, permiten integrar los contenidos de esta asignatura a

muchos otros aprendidos en otras materias, principalmente de programación y lenguajes (que abarquen programación concurrente y las dificultadas asociadas), y afianzar las capacidades adquiridas en lógica y matemática. Se buscará que los problemas a resolver en los trabajos prácticos hagan evidentes las bondades y limitaciones de las distintas técnicas de validación y verificación presentadas. Se intentará utilizar ejemplos y problemas interesantes, en los cuales las fallas sean difíciles de reconocer, intentando estimular al alumnado.

Además, se proveerá una serie de ejercicios adicionales (cuya resolución es opcional) que acompañarán cada una de las clases prácticas.

Se fomentará la lectura de material adicional y la auto organización de los alumnos en sus actividades. Además, se dejará en manos de los alumnos la instalación y manejo de las herramientas de software utilizadas en la asignatura, como una manera de estimular la práctica en cuestiones más técnicas (no necesariamente ligadas a los tópicos que la asignatura abarca), y la experiencia en la utilización de herramientas nuevas.

CLASES TEÓRICAS: 4 horas semanales

CLASES PRÁCTICAS: 4 horas semanales

CLASES DE TRABAJOS PRÁCTICOS DE LABORATORIO: Todas las clases prácticas serán de laboratorio.

F. NÓMINA DE TRABAJOS PRÁCTICOS

La materia contará con dos trabajos prácticos obligatorios, que serán evaluados por el docente de la materia. Cada uno de los cuales contará con una instancia de evaluación (defensa) oral. La aprobación de dichos trabajos es condición para la regularidad. El plazo para la resolución de cada uno de los trabajos prácticos es de dos semanas.

Además, se proveerá una serie de ejercicios adicionales (cuya resolución es opcional) que acompañarán cada una de las clases prácticas.

El objetivo de los trabajos prácticos obligatorios es poder evaluar la aplicación de las técnicas aprendidas en situaciones concretas de tamaño mediano, y la correcta comprensión de los fundamentos teóricos subyacentes a las técnicas estudiadas. Permiten también detectar y corregir problemas tanto en las actividades de cálculo y deducción en matemática y lógica como en la construcción de programas, y así afianzar los conocimientos adquiridos en materias anteriores, de las áreas de matemática discreta, lógica, programación e ingeniería de software.

G. HORARIOS DE CLASES:

El dictado de la asignatura estará compuesto por cuatro horas semanales de clases teóricas y 4 horas semanales de clases prácticas. Las clases se desarrollarán los Jueves de 16 a 20, y los Viernes de 16 a 20.

HORARIO DE CLASES DE CONSULTAS: Las clases de consulta se darán semanalmente bajo demanda de los alumnos, en horario a convenir con los mismos.

H. MODALIDAD DE EVALUACIÓN:

- **Evaluaciones Parciales:** La materia contará con dos trabajos prácticos obligatorios cada uno con una instancia de evaluación oral (defensa). Cada trabajo cuenta con una una instancia de recuperación. La aprobación de dichos trabajos es condición para la regularidad. El plazo para la resolución de cada uno de los trabajos prácticos será de dos semanas.

- **Evaluación Final:** El examen final para alumnos regulares se llevará a cabo mediante evaluación oral. Abarcará la totalidad de los contenidos de la asignatura, verificando que los alumnos hayan adquirido los conocimientos teóricos y puedan aplicarlos en casos concretos en la práctica.

CONDICIONES DE REGULARIDAD: Para regularizar la materia se deberá: aprobar los dos trabajos prácticos de programación obligatorios, con nota no menor a 5.

CONDICIONES DE PROMOCIÓN: Para ser promovido en la materia se deberá aprobar los trabajos prácticos con promedio no menor a 7. La nota de promoción se calculará como un promedio ponderado de las notas de los trabajos prácticos.

PROGRAMA ANALÍTICO

A. CONTENIDOS

El rol de la validación y la verificación en la calidad del software. Confiabilidad de software. El problema de construir programas correctos. ¿Qué es un programa *correcto*?

El testing como actividad de validación y verificación. Conceptos fundamentales de testing, sus objetivos y principios. El *testing* como especificación. Pruebas unitarias: conceptos de prueba y granularidad. Automatización de testing usando frameworks xUnit. Estructuración y composición de tests de unidad. Construcción de suites de testing. Revisión de criterios y técnicas de diseño de casos de prueba. Testing de caja negra y caja blanca. Criterios de cobertura. Herramientas de medición de cobertura. Independencia en unidades de test. El concepto de doble de prueba y su necesidad. Tipos de dobles: dummies, stubs, mocks, etc. Herramientas de apoyo. Integración continua. Estrategias de integración y herramientas.

Necesidad de las especificaciones de programas. Revisión de enfoques tradicionales a la validación y verificación de programas. Testing y sus limitaciones. Programas correctos por construcción. Problemas en la automatización de las técnicas tradicionales de validación y verificación de programas. Interacción entre testing y verificación. Generación automática de tests.

El problema de construir y razonar sobre programas concurrentes. Algunas técnicas clásicas para la verificación de programas concurrentes. Las álgebras de procesos. El álgebra de procesos FSP. Modelado de procesos concurrentes en FSP. FSP en el modelado de objetos compartidos y exclusión mutua. Diferentes modelos de interacción entre procesos. Monitores, sincronización. Propiedades de safety y liveness en FSP, y su análisis mediante LTSA.

Comprobación de modelos (*model checking*). La lógica modal. Definiciones fundamentales. Operadores modales y sus distintas interpretaciones. Sintaxis y semántica de la lógica modal proposicional. Estructuras de Kripke. Relación entre estructuras de Kripke y la semántica de programas. Lógicas temporales. La lógica temporal como una lógica modal. Descripción de propiedades relacionadas con el tiempo mediante la utilización de operadores modales. Lógica temporal lineal y lógica temporal ramificada. Model checking para la generación automática de tests.

Sintaxis y semántica formales de la lógica temporal lineal proposicional. Autómatas de Büchi y su relación con modelos de la lógica temporal lineal proposicional. Traducción de interpretaciones y fórmulas de la lógica temporal lineal proposicional a autómatas de Büchi. Reducción de la comprobación de modelos en lógica temporal lineal proposicional a un problema en Teoría de Autómatas. Automatización en la verificación de propiedades temporales mediante el *model checking*. Ejemplos de su uso en la práctica. La concurrencia y las dificultades en la comprensión de programas concurrentes. Utilización de la herramienta SPIN. El lenguaje ProMeLa. Especificación de programas concurrentes en ProMeLa. Caracterización de propiedades de programas concurrentes mediante el uso de lógica temporal lineal. Ejemplos. Limitaciones del *model checking* en relación a la lógica temporal lineal proposicional.

La lógica como lenguaje formal y su expresividad. Lógica proposicional. Sus ventajas y sus limitaciones en poder expresivo. Reducción del problema de la deducción a SAT solving

proposicional. Automatización de la deducción. La lógica de primer orden. Características y poder expresivo. Restricciones para la aplicación de SAT solving para deducción en lógica de primer orden. Representación de diseños de sistemas mediante lógica de primer orden. Algunas extensiones importantes a la lógica de primer orden. La lógica relacional. Descripción de diseños y propiedades estructurales mediante la lógica relacional. El poder de SAT solving de primer orden como una técnica de validación de sistemas. Limitaciones. La herramienta Alloy. Definición de diseños mediante signaturas y funciones de transformación de estados. Validación de diseños en Alloy. Utilización en la práctica. Ejemplos. Alloy y sus limitaciones para razonar sobre aspectos dinámicos de programas. La extensión DynAlloy. Aplicaciones de DynAlloy. Estrategias para lidiar con problemas de escalabilidad en análisis basado en satisfactibilidad booleana.

A. CRONOGRAMA DE CLASES Y PARCIALES

Semana	Día/Fecha	Teóricos	Día/Fecha	Prácticos	Día/Fecha	Laboratorios	Parciales / Recuperatorios
1	14/03/19	Calidad de software. Testing.	15/03/19	Herramientas básicas de testing de unidad.			
2	21/03/19	Revisión de concepto s básicos de testing	22/03/19	Herramientas avanzadas para testing de unidad y medidas de cobertura			
3	28/03/19	Generación de tests. Generación aleatoria- Generación exhaustiva acotada	29/03/19	Herramientas de generación aleatoria de tests y generación exhastiva acotada.			
4	04/04/19	Generación de tests. Constraint solving.	05/05/19	Herramientas de generación basadas en constraint solving. Presentación del primer trabajo práctico obligatorio			Presentación del primer trabajo práctico obligatorio
5	11/04/19	Verificación de programas concurrentes. Modelos.	12/04/19	Herramientas básicas para modelos de programas concurrentes.			

6	18/04/19		19/04/19				19/04/19: Fecha Límite para el envío de la resolución del primer trabajo práctico obligatorio.
7	25/04/19	Propiedades de sistemas concurrentes. Lógica temporal	26/04/19				26/04/19: Defensa oral del primer trabajo práctico
8	02/05/19	(continuación) Propiedades de sistemas concurrentes. Lógica temporal	03/05/19	Model checkers.			
9	09/05/19	Principios de model checking	10/05/19	Model checkers.			
10	16/05/19	Revisión de lógica y álgebra SAT Solvers	17/05/19	SAT Solvers			
11	23/05/19	Modelos formales de software	24/05/19	Alloy como herramienta de verificación. Presentación del segundo trabajo práctico obligatorio			Presentación del segundo trabajo práctico obligatorio
12	30/05/19	Satisfactibilidad booleana y testing	31/05/19	Alloy para generar tests a partir de especificaciones.			
13	06/06/19	Alloy, modelos dinámicos	07/06/19	DynAlloy, verificación acotada de programas			07/06/19: Fecha Límite para el envío de la resolución del segundo trabajo práctico
14	13/06/19		14/06/19	verificación acotada de programas			13/06/19: Defensa oral del segundo trabajo

							práctico obligatorio

B. BIBLIOGRAFÍA

Obligatoria

- Glenford J. Myers, *The Art of Software Testing*, John Wiley & Sons, Inc., 2012.D. Jackson, Software Abstractions, MIT Press, 2006.
- Feedback-directed random test generation, Carlos Pacheco, Shuvendu K. Lahiri, Michael D. Ernst, and Thomas Ball (ICSE 2007).
- Pex – White Box Test Generation for .NET, Nikolai Tillmann and Jonathan de Halleux (TAP 2008)
- Korat: A Tool for Generating Structurally Complex Test Inputs, Aleksandar Milicevic, Sasa Misailovic, Darko Marinov, Sarfraz Khurshid, (ICSE 2007).
- Evolutionary Generation of Whole Test Suites, G. Fraser and A. Arcuri, (*QSIC* 2011)
- Model Checking Programs, Willem Visser, Klaus Havelund, Guillaume Brat, SeungJoon Park and Flavio Lerda (J-ASE, vol. 10, no. 2, April 2003).
- Jeff Magee & Jeff Kramer. Concurrency: State Models & Java Programs (2nd edition). Wiley. 2006.

De Consulta

- Tutoriales y manuales de las herramientas utilizadas.