



Universidad Nacional de Ingeniería (UNI)

Escuela Profesional de
Ciencia de la Computación
Sílabo 2027-I

1. CURSO

CS395. Calidad de Software (Electivo)

2. INFORMACIÓN GENERAL

2.1 Curso	: CS395. Calidad de Software
2.2 Semestre	: 8 ^{vo} Semestre.
2.3 Créditos	: 3
2.4 Horas	: 2 HT; 2 HP;
2.5 Duración del periodo	: 16 semanas
2.6 Condición	: Electivo
2.7 Modalidad de aprendizaje	: Presencial
2.8 Prerrequisitos	: CS292. Ingeniería de Software II. (7 ^{mo} Sem)

3. PROFESORES

Titular

- Ernesto Cuadros-Vargas <ecuadrosv@uni.edu.pe>
 - Doctor en Ciencia de la Computación, ICMC-USP, Brasil, 2004.
 - Master en Ciencia de la Computación, ICMC-USP, Brasil, 1998.

Atención previa coordinación con el profesor

4. INTRODUCCIÓN AL CURSO

Entregar software meramente "funcional" no es suficiente para la industria: las organizaciones modernas exigen sistemas confiables, seguros, mantenibles y construidos mediante procesos de ingeniería disciplinados y auditables. Este curso enseña la calidad del software como una disciplina de ingeniería práctica y de primer nivel, ejercida mediante las mismas herramientas y flujos de trabajo usados por equipos profesionales en GitHub: protección de ramas, revisión de código obligatoria, pruebas automatizadas en todos los niveles, integración continua/entrega continua (CI/CD), análisis estático y observabilidad en producción.

Los estudiantes avanzan desde modelos fundamentales de calidad hacia flujos de trabajo colaborativos avanzados, estrategias de pruebas automatizadas (unitarias, de integración, de extremo a extremo y guiadas por comportamiento), pipelines de CI/CD, análisis estático y refactorización, y aspectos de calidad en producción como seguridad de la cadena de suministro, rendimiento e ingeniería de confiabilidad. El curso cierra con énfasis en la cultura DevOps: las prácticas y valores compartidos del equipo (Definition of Done, postmortems sin culpa, aprendizaje continuo) que sostienen la calidad a escala.

5. OBJETIVOS

- Evaluar la calidad del software usando un modelo de características de calidad reconocido internacionalmente y razonar sobre el costo de la mala calidad y la deuda técnica.
- Aplicar un flujo de trabajo profesional de Git/GitHub basado en revisión, incluyendo protección de ramas, CODEOWNERS y conventional commits.
- Diseñar e implementar suites de pruebas unitarias, de integración, de extremo a extremo y guiadas por comportamiento, incluyendo desarrollo guiado por pruebas.
- Construir y operar pipelines de CI/CD que impongan puertas de calidad automatizadas, escaneo de dependencias y versionado semántico.
- Aplicar análisis estático, pruebas de mutación y técnicas de refactorización para identificar y eliminar code smells y deuda técnica.
- Analizar aspectos de calidad en producción, incluyendo pruebas de rendimiento y prácticas de confiabilidad/observabilidad.

- Adoptar prácticas de cultura DevOps – Definition of Done, postmortems sin culpa y aprendizaje continuo del equipo – para sostener la calidad.

6. RESULTADOS DEL ESTUDIANTE

AG-C09) Diseño y Desarrollo de Soluciones: Diseña, implementa y evalúa soluciones para problemas complejos de computación. (Assessment)

AG-C11) Uso de Herramientas: Aplica herramientas modernas de computación en la resolución de problemas. (Assessment)

AG-C12) Usa enfoques sistémicos para seleccionar, desarrollar, aplica, integrar y administrar tecnologías seguras de computación para lograr los objetivos del usuario. (Usage)

7. TEMAS

Unidad 1: Calidad y Evaluación del Diseño (4 horas)	
Resultados esperados: AG-C09,AG-C12	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
■ Diseño de datos Modelado de Datos ● Estructuras de datos ● Sistemas de almacenamiento ■ Trazabilidad de requisitos ● Comprender qué requisitos son satisfechos por un diseño ■ Modelado de diseño, por ejemplo con diagramas de clases, diagramas de entidad-relación o diagramas de secuencia ■ Medición y análisis de la calidad del diseño ■ Principios de diseño y codificación seguros Diseño e Ingeniería de Controles de Seguridad, Análisis de Amenazas e Ingeniería de Seguridad, Computación Confiable e Ingeniería de Privacidad ● Principio de privilegio mínimo ● Principio de valores predeterminados seguros ante fallos (fail-safe defaults) ● Principio de aceptabilidad psicológica ■ Evaluar compensaciones de diseño (tradeoffs) (por ejemplo, eficiencia vs confiabilidad, seguridad vs usabilidad) ■ Modelo de características de calidad ISO/IEC 25010 y costo de la mala calidad ● Características de calidad: adecuación funcional, eficiencia de desempeño, compatibilidad, usabilidad, confiabilidad, seguridad, mantenibilidad, portabilidad ● Costo de la mala calidad y la deuda técnica como preocupación de negocio/económica, no solo técnica	■ Diseñar un conjunto de estructuras de datos para implementar una superficie de API proporcionada [Diseñar] ■ Identificar qué requisitos son satisfechos por un diseño de software proporcionado [Analizar] ■ Traducir un diseño de software en lenguaje natural a diagramas de clases [Traducir] ■ Adaptar un diseño de sistema defectuoso para que siga mejor los principios de privilegio mínimo y valores predeterminados seguros ante fallos [Crear] ■ Contrastar dos diseños de software a través de diferentes cualidades, como eficiencia o usabilidad [Contrastar] ■ Evaluar un sistema de software frente a las características de calidad ISO/IEC 25010 y estimar el costo de negocio de la mala calidad/deuda técnica [Evaluar]
Lecturas : [ISO25010], [Sommerville2015]	

Unidad 2: Control de Versiones y CI/CD (12 horas)	
Resultados esperados: AG-C11,AG-C12	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
■ Gestión de configuración de software y control de versiones: Prácticas de Desarrollo de Software ● Configuración en control de versiones, builds/configuración reproducibles. ● Estrategias de ramificación en control de versiones. Ramas de desarrollo vs ramas de lanzamiento. Desarrollo basado en tronco (<i>trunk-based development</i>). ● Estrategias de fusión/rebase, cuando sean relevantes. ■ Gestión de lanzamientos (<i>release management</i>). ■ Herramientas de prueba, incluyendo herramientas de análisis estático y dinámico. Prácticas de Desarrollo de Software, Flujo de Información y No Interferencia, Inyección y Validación de Entrada, Seguridad de Memoria y Tipos, Análisis de Malware y Seguridad Avanzada ■ Automatización de procesos de software: ● Sistemas de construcción (<i>build systems</i>): el valor de builds rápidos, herméticos y reproducibles, comparar/contrastar enfoques para construir un proyecto. ● Integración Continua (CI): el uso de automatización y pruebas automatizadas para hacer una validación preliminar de que la revisión actual del tronco se construye y pasa las pruebas (básicas). ● Despliegue Continuo (CD): el uso de automatización para liberar de forma automática cada cambio que supera las pruebas hacia el entorno de producción, garantizando entregas frecuentes y confiables. ● Gestión de dependencias: actualización de dependencias externas/aguas arriba, gestión de paquetes, SemVer. ■ Flujo de revisión colaborativa como puerta de calidad ● Revisión obligatoria de <i>pull requests</i> antes de fusionar ● Propiedad del código mediante reglas de tipo CODEOWNERS ● Convenciones de mensajes de <i>commit</i> (<i>conventional commits</i>) ● Reglas de protección de ramas que exigen verificaciones antes de fusionar ■ Escaneo de vulnerabilidades de dependencias y de la cadena de suministro integrado en los pipelines de CI	■ Describir la diferencia entre la gestión de configuración de software centralizada y distribuida [Describir] ■ Describir cómo el control de versiones puede usarse para ayudar en la gestión de lanzamientos de software [Describir] ■ Identificar elementos de configuración y usar una herramienta de control de código fuente en un proyecto pequeño basado en equipo [Analizar] ■ Describir cómo las herramientas de prueba estáticas y dinámicas disponibles pueden integrarse en el entorno de desarrollo de software [Describir] ■ Comprender el uso de sistemas de CI/CD como una fuente de verdad para el estado del código compartido del equipo (éxito en la construcción y pruebas) [Explicar] ■ Aplicar un flujo de revisión de <i>pull requests</i> con reglas de propiedad de código y protección de ramas para imponer puertas de calidad en un proyecto de equipo [Aplicar] ■ Usar un escáner de vulnerabilidades de dependencias/cadena de suministro como parte de un pipeline de CI para detectar dependencias vulnerables [Usar]
4	
Lecturas : [Kim2021]	

Unidad 3: Planificación y Tipos de Pruebas (8 horas)	
Resultados esperados: AG-C09	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
■ Tipos de pruebas ● Unitarias ● Integración ● Validación ● Sistema ■ Diferencias estilísticas entre el código de prueba y el código de producción: DAMP vs DRY - se justifica más duplicación en el código de prueba. ■ Planificación y generación de pruebas ● Generación de casos de prueba, a partir de modelos formales, especificaciones, etc. ● Cobertura de pruebas ○ Matrices de prueba ○ Cobertura de código - ¿cuánto del código está probado? ○ Cobertura de entorno - ¿cuántas arquitecturas de hardware, sistemas operativos, navegadores, etc. están probados? ● Datos y entradas de prueba	■ Comparar y contrastar los diferentes tipos y niveles de pruebas (regresión, unitarias, integración, sistemas y aceptación) [Comparar] ■ Describir técnicas para crear un plan de pruebas y generar casos de prueba [Describir] ■ Crear un plan de pruebas para un segmento de código de tamaño mediano que incluya una matriz de pruebas y generación de datos y entradas de prueba [Crear] ■ Implementar un plan de pruebas para un segmento de código de tamaño mediano [Implementar]
Lecturas : [Sommerville2015]	

Unidad 4: Prácticas Avanzadas de Pruebas (4 horas)	
Resultados esperados: AG-C11	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
■ Desarrollo de pruebas ● Desarrollo guiado por pruebas ● Pruebas orientadas a objetos, uso de mocks e inyección de dependencias ● Técnicas de prueba de caja opaca (anteriormente, caja negra) y caja transparente (anteriormente, caja blanca) ● Herramientas de prueba, incluyendo cobertura de código, análisis estático y fuzzing ■ Verificación y validación en el ciclo de desarrollo ● Revisiones de código ● Automatización de pruebas, incluyendo automatización de herramientas ● Pruebas pre-commit y post-commit ● Compensaciones entre cobertura de pruebas y rendimiento/latencia de las pruebas ● Seguimiento y priorización de defectos: reproducibilidad de los defectos reportados ■ Desafíos de verificación y validación específicos del dominio ● Pruebas de rendimiento y evaluación comparativa (benchmarking) ● Asincronía, paralelismo y concurrencia ● Crítico para la seguridad (<i>safety-critical</i>) ● Numérico	■ Identificar los principios fundamentales de los métodos de desarrollo guiado por pruebas y explicar el papel de las pruebas automatizadas en estos métodos [Analizar] ■ Discutir problemas relacionados con las pruebas de software orientado a objetos [Debatir] ■ Describir el uso de mocks y la inyección de dependencias y su aplicación [Describir] ■ Realizar, como parte de una actividad en equipo, una revisión de código de un segmento de código de tamaño mediano [Evaluar] ■ Describir el papel que las herramientas pueden desempeñar en la validación del software [Describir] ■ Automatizar las pruebas en un proyecto de software pequeño [Diseñar] ■ Explicar los roles, pros y contras de las pruebas pre-commit y post-commit [Explicar] ■ Discutir las compensaciones entre la cobertura de pruebas y el rendimiento/latencia de las pruebas y cómo esto puede afectar la verificación [Debatir] ■ Usar una herramienta de seguimiento de defectos para gestionar defectos de software en un proyecto de software pequeño [Usar] ■ Discutir las limitaciones de las pruebas en ciertos dominios [Debatir]
Lecturas : [Beck2002]	

Unidad 5: Herramientas y Análisis de Pruebas (8 horas)	
Resultados esperados: AG-C11	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
■ Herramientas y automatización de verificación y validación ● Análisis estático ● Cobertura de código ● Fuzzing ● Análisis dinámico y contención de fallos (sanitizadores, etc.) ● Registro y seguimiento de fallos ■ Planificación y generación de pruebas ● Estimación de fallos y terminación de pruebas, incluyendo siembra de defectos ● Uso de números aleatorios y pseudoaleatorios en las pruebas ■ Pruebas de sistemas asíncronos, paralelos y concurrentes ■ Verificación y validación de artefactos que no son código (documentación, materiales de capacitación) ■ Desarrollo guiado por comportamiento (BDD) ● Expresar criterios de aceptación como escenarios Gherkin Given/When/Then ● Ejecutar escenarios BDD como pruebas automatizadas de extremo a extremo ■ Pruebas de mutación y análisis estático de calidad de código ● Pruebas de mutación para evaluar la efectividad de una suite de pruebas ● Detección estática de <i>code smells</i> y complejidad ciclomática excesiva	■ Describir y comparar diferentes herramientas para verificación y validación [Describir] ■ Automatizar el uso de diferentes herramientas en un proyecto de software pequeño [Diseñar] ■ Explicar cómo y cuándo se deben usar números aleatorios en las pruebas [Explicar] ■ Describir enfoques para la estimación de fallos [Describir] ■ Estimar el número de fallos en una aplicación de software pequeña basándose en la densidad de fallos y la siembra de fallos [Estimar] ■ Describir técnicas y problemas con las pruebas de software asíncrono, concurrente y paralelo [Describir] ■ Crear un plan de pruebas para un segmento de código de tamaño mediano que contenga código asíncrono, concurrente y/o paralelo, incluyendo una matriz de pruebas y generación de datos y entradas de prueba [Crear] ■ Describir técnicas para la verificación y validación de artefactos que no son código [Describir] ■ Escribir escenarios Gherkin Given/When/Then para los criterios de aceptación de una funcionalidad y ejecutarlos como pruebas automatizadas de extremo a extremo [Escribir] ■ Analizar el puntaje de mutación de una suite de pruebas y refactorizar las pruebas para eliminar los mutantes sobrevivientes [Analizar]
Lecturas : [Wynne2012]	

Unidad 6: Compatibilidad y Versionado de API (4 horas)	
Resultados esperados: AG-C11,AG-C12	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
■ Ley de Hyrum / Ley de las Interfaces Implícitas ■ Compatibilidad con versiones anteriores ● La compatibilidad no es una propiedad de una sola entidad, es una propiedad de una relación. ● La compatibilidad con versiones anteriores debe evaluarse en términos de proveedor + consumidor(es) o con un modelo bien especificado de qué formas de compatibilidad aspira/promete un proveedor. ■ Control de versiones (versioning) ● Versionado Semántico (SemVer) ● Desarrollo basado en tronco (<i>trunk-based development</i>)	■ Identificar tanto el comportamiento explícito como el implícito de una interfaz e identificar riesgos potenciales de la Ley de Hyrum [Analizar] ■ Identificar cambios que pueden considerarse ampliamente "compatibles con versiones anteriores", potencialmente con declaraciones explícitas sobre qué uso está o no soportado [Analizar] ■ Evaluar si un cambio propuesto es lo suficientemente seguro dada la metodología de control de versiones en uso para un proyecto dado [Evaluar]
Lecturas : [Sommerville2015]	

Unidad 7: Técnicas de Refactorización (4 horas)	
Resultados esperados: AG-C09,AG-C12	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
■ Refactorización ● Patrones de refactorización estándar (renombrar, integrar, extraer, etc.) ● Uso de herramientas de refactorización en el IDE ● Aplicación de herramientas de análisis estático (para identificar código que necesita refactorización, generar cambios, etc.) ● Valor de la refactorización como remedio para la deuda técnica ■ Refactorización "a Gran Escala": técnicas cuando un cambio de refactorización es demasiado grande para comprometerlo de forma segura (proyectos grandes), o cuando es imposible sincronizar el cambio entre proveedor + todos los consumidores (múltiples repositorios, consumidores con código privado). ● Expresar tanto las API antiguas como las nuevas para que puedan coexistir. ● Minimizar el tamaño de los cambios de comportamiento. ● ¿Por qué se requieren estas técnicas?. Por ejemplo, "consumidores de API que puedo ver" vs "consumidores que no puedo ver").	■ Considerar las entradas de herramientas de análisis estático y/o los principios de Diseño de Software para identificar código que necesita refactorización [Analizar] ■ Refactorizar la implementación de una interfaz para mejorar el diseño, claridad, etc., con impacto mínimo/cero en los usuarios existentes [Crear] ■ Planificar una refactorización compleja de múltiples pasos para cambiar el comportamiento predeterminado de una API de manera segura [Planear]
Lecturas : [Fowler2017]	

Unidad 8: Pruebas de Rendimiento y Evaluación Comparativa (4 horas)	
Resultados esperados:	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
▪ Pruebas de rendimiento y evaluación comparativa • Rendimiento (<i>throughput</i>) y latencia • Degradación bajo carga (pruebas de estrés, manejo FIFO vs LIFO de solicitudes) • Aceleración (<i>speedup</i>) y escalado (<i>scaling</i>) ◦ Ley de Amdahl ◦ Ley de Gustafson ◦ Escalado suave (<i>soft</i>) y débil (<i>weak</i>) • Identificar y medir figuras de mérito (<i>figures of merit</i>) • Cuellos de botella de rendimiento comunes ◦ Limitado por cómputo (<i>compute-bound</i>) ◦ Limitado por ancho de banda de memoria ◦ Limitado por latencia • Métodos estadísticos y mejores prácticas para la evaluación comparativa (benchmarking) ◦ Estimación de la incertidumbre ◦ Intervalos de confianza • Análisis y presentación (gráficos, etc.) • Técnicas de medición de tiempo (<i>timing</i>)	▪ Describir el rendimiento (<i>throughput</i>) y la latencia y proporcionar ejemplos de cada uno [Describir] ▪ Explicar la aceleración (<i>speedup</i>) y las diferentes formas de escalado (<i>scaling</i>) y cómo se calculan [Explicar] ▪ Describir cuellos de botella de rendimiento comunes [Describir] ▪ Describir métodos estadísticos y mejores prácticas para la evaluación comparativa (benchmarking) de software [Describir] ▪ Explicar técnicas y desafíos para medir el tiempo al construir una evaluación comparativa (benchmark) [Explicar] ▪ Identificar las figuras de mérito, construir y ejecutar una evaluación comparativa (benchmark) y analizar estadísticamente y visualizar los resultados para un proyecto de software pequeño [Analizar]
Lecturas : [Bondi2015], [Gregg2020]	

Unidad 9: Ingeniería de Confiabilidad (4 horas)	
Resultados esperados: AG-C11,AG-C12	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
■ Modelos de confiabilidad de software ■ Técnicas y modelos de tolerancia a fallos de software ● Diferencias contextuales en la tolerancia a fallos (por ejemplo, hacer fallar un sistema crítico para la aviación se evita firmemente, hacer fallar un sistema de procesamiento de datos antes de que se escriban datos corruptos en el almacenamiento es muy valioso) ■ Prácticas de ingeniería de confiabilidad de software - incluyendo revisiones, pruebas, verificación de modelos práctica (practical model checking) ■ Identificación de dominios de fallos dependientes e independientes, y su impacto en la confiabilidad del sistema ■ Análisis basado en mediciones de la confiabilidad del software - telemetría, monitorización y alertas, paneles de control, métricas de calificación de lanzamientos, etc.	■ Demostrar la capacidad de aplicar múltiples métodos para desarrollar estimaciones de confiabilidad para un sistema de software [Demostrar] ■ Identificar métodos que conducirán a la realización de una arquitectura de software que logre un nivel especificado de confiabilidad [Analizar] ■ Identificar formas de aplicar redundancia para lograr tolerancia a fallos [Analizar] ■ Identificar dependencias de punto único de fallo (single-point-of-failure, SPF) en un diseño de sistema [Analizar]
Lecturas : [Kim2021], [Forsgren2018]	

Unidad 10: Construcción a Escala y Proceso (4 horas)	
Resultados esperados: AG-C12	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
■ Pruebas a mayor escala ● Dobles de prueba (test doubles) (<i>stubs, mocks, fakes</i>) ● Inyección de dependencias (<i>dependency injection</i>) ■ Secuenciación del trabajo, incluyendo identificación de dependencias, hitos y retiro de riesgo ● Identificación de dependencias: identificar las dependencias entre diferentes tareas ● Hitos: una colección de tareas que sirve como marcador de progreso cuando se completa. Idealmente, el hito abarca una unidad útil de funcionalidad. ● Retiro de riesgo: identificar qué elementos de un proyecto son riesgosos y priorizar la finalización de tareas que aborden esos riesgos. ■ Problemas potenciales de seguridad en programas Flujo de Información y No Interferencia, Inyección y Validación de Entrada, Seguridad de Memoria y Tipos, Análisis de Malware y Seguridad Avanzada ● Desbordamientos de búfer y otros tipos de desbordamientos ● Condiciones de carrera (race conditions) ● Inicialización incorrecta, incluida la elección de privilegios ● Validación de entrada (input validation) ■ Documentación (autogenerada) ■ Contexto de desarrollo: "campo verde" (<i>green field</i>) vs base de código existente ● Análisis de impacto del cambio (<i>change impact analysis</i>) ● Realización del cambio (<i>change actualization</i>) ■ Gestión de lanzamientos (release management) ■ Prácticas de DevOps ● Definition of Done como umbral de calidad compartido para el trabajo terminado ● Postmortems sin culpa (<i>blameless postmortems</i>) para aprender de incidentes/fallas ● Cultura de calidad en equipo: propiedad compartida, retroalimentación y aprendizaje continuo	■ Reescribir un programa simple para eliminar vulnerabilidades comunes, como desbordamientos de búfer, desbordamientos de enteros y condiciones de carrera [Aplicar] ■ Escribir un componente de software que realice alguna tarea no trivial y sea resistente a errores de entrada y de tiempo de ejecución [Escribir]
Lecturas : [Forsgren2018]	

Unidad 11: Proyecto Final: Demostración de un Pipeline de Calidad de Extremo a Extremo (8 horas)	
Resultados esperados: AG-C09	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
- Integración de todas las prácticas del curso en un único repositorio en vivo: protección de ramas, revisión obligatoria de PRs, suites de pruebas automatizadas (unitarias, de integración, E2E/BDD), pipeline de CI/CD con puertas de calidad, escaneo de dependencias, análisis estático y observabilidad. - Revisión en vivo del repositorio: demostración y defensa del pipeline de calidad del equipo ante compañeros/instructor. - Retrospectiva y postmortem sin culpa del recorrido de calidad del proyecto.	- Integrar control de versiones, pruebas automatizadas, CI/CD, análisis estático y observabilidad en un único pipeline de calidad de extremo a extremo para un repositorio real. [Usar] - Defender el diseño y la efectividad del pipeline de calidad de un equipo mediante una revisión en vivo del repositorio. [Usar] - Realizar un postmortem sin culpa para identificar mejoras de proceso para proyectos futuros. [Evaluar]
Lecturas : [Forsgren2018]	

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

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.

Sesiones de Laboratorio:

Para verificar que los alumnos hayan alcanzado el logro planteado para cada una de las unidades de aprendizaje, realizarán actividades que les permita aplicar los conocimientos adquiridos durante las sesiones de teoría y se les propondrá retos que permitan evaluar el desempeño de los alumnos.

Exposiciones individuales o grupales:

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.

Lecturas:

A lo largo del curso se proporcionan diferentes lecturas, las cuales podrían ser evaluadas. El uso de herramientas Online permite a cada estudiante acceder a la información del curso, e interactuar fuera de aula con el profesor y con los otros estudiantes.

La nota final NF depende de de las siguientes notas:

- EP : Examen Parcial.
- EF : Examen Final.
- P : Evaluacion permanente (Promedio de prácticas y Trabajos calificados).

La nota final NF se calcula así:

$$NF = \frac{EP + EF + P}{3}$$

10. BIBLIOGRAFÍA BÁSICA