	CLASE 1 - Repaso, comunicación y SRS	V/F
1.	Un proceso de software es un conjunto de actividades y resultados asociados que producen un producto de software.	
2.	La elicitación de requisitos consiste en documentar únicamente los requerimientos funcionales del sistema.	
3.	La observación del ambiente de trabajo requiere interrumpir constantemente a las personas para obtener información.	
4.	Los cuestionarios con preguntas abiertas son más fáciles de analizar que los cuestionarios con preguntas cerradas.	
5.	El estándar IEEE Std. 1362-1998 define el formato y contenido para la especificación de requisitos de software (SRS).	
6.	Un buen SRS debe ser no ambiguo, lo que significa que cada requisito debe tener una única interpretación.	
7.	La regla de presentación 10/20/30 de Guy Kawasaki sugiere usar tipografías con cuerpo menor a 30 para las diapositivas.	
8.	En un elevator pitch, se recomienda comenzar con una afirmación o pregunta sorprendente para captar la atención.	
9.	Los requerimientos no funcionales describen el comportamiento y las tareas específicas que debe realizar el sistema.	
10.	El SRS debe evolucionar conjuntamente con el software, registrando los cambios y la aceptación de los mismos.	
	CLASE 2 - GCS, planificación organizativa	
11.	La Gestión de la Configuración del Software (GCS) es un conjunto de actividades orientadas a gestionar los cambios en el software durante su ciclo de vida.	
12.	Una línea base en GCS es una especificación o producto que no ha sido revisado formalmente y puede modificarse sin procedimientos de control de cambio.	
13.	Los elementos de configuración (ECS) incluyen solo los programas ejecutables y el código fuente del software.	
14.	El control de versiones en GCS permite gestionar diferentes variantes de un programa mediante la creación de ramas (branches) en un repositorio.	
15.	La auditoría de configuración verifica si los cambios especificados en una orden de cambio se han realizado correctamente y si se han seguido los estándares de ingeniería de software.	
16.	Un proyecto de software se caracteriza por ser un esfuerzo permanente que produce resultados repetitivos.	
17.	El personal es considerado el recurso más crítico para el éxito de un proyecto de software, representando el capital intelectual de la organización.	
18.	El paradigma cerrado de organización de equipos fomenta la innovación al permitir una comunicación horizontal entre los miembros del equipo.	
19.	Un equipo ágil de software se asemeja al paradigma aleatorio, destacando por su autoorganización y colaboración grupal.	
20.	Un equipo de software consolidado tiende a sufrir fricción entre sus miembros y una atmósfera de trabajo frenética.	
	CLASE 3 - Riesgos	
21.	Un riesgo en un proyecto de software es un evento no deseado que tiene consecuencias negativas.	
22.	La gestión de riesgos en el desarrollo de software busca únicamente evitar los eventos no deseados, pero no minimizar sus consecuencias.	
23.	La deuda técnica se genera exclusivamente por la falta de documentación en un proyecto de software.	
24.	En el desarrollo ágil, la gestión de riesgos es innecesaria porque los equipos se autoorganizan.	
25.	Los riesgos de un proyecto de software pueden categorizarse en riesgos del proyecto, del producto y del negocio.	

26.	La falta de experiencia en tecnologías nuevas tiene un impacto insignificante en el desarrollo de un sistema.	
27.	Los problemas de integración entre sistemas son considerados riesgos del proyecto.	
28.	Los cambios en la legislación fiscal son un ejemplo de riesgo del producto.	
29.	Un riesgo con probabilidad superior al 70% y un impacto serio o catastrófico se considera crítico según la línea de corte del ejercicio.	
30.	Una estrategia de mitigación para la falta de recursos humanos puede incluir la distribución de tareas críticas entre varios miembros del equipo.	
	CLASE 4 - Conceptos de diseño de software y planificación temporal	
31.	La planificación temporal distribuye el esfuerzo estimado a lo largo de la duración prevista del proyecto, asignándolo a tareas específicas.	
32.	En un diagrama de Gantt, las dependencias entre tareas no son representadas visualmente.	
33.	El método PERT es determinístico y se utiliza cuando las duraciones de las tareas son conocidas con certeza.	
34.	Una tarea con margen total igual a cero pertenece al camino crítico de un proyecto.	
35.	El diseño arquitectónico define la relación entre los elementos estructurales del software y los estilos arquitectónicos.	
36.	Un buen diseño de software debe tener alta cohesión y alto acoplamiento entre módulos.	
37.	La ocultación de información implica que los datos de un módulo son accesibles para todos los demás módulos del sistema.	
38.	El refinamiento en el diseño de software permite revelar detalles de bajo nivel de manera sucesiva.	
39.	El diseño de interfaces se centra únicamente en la comunicación entre sistemas diferentes, no en la interacción con usuarios.	
40.	La creación de un modelo de diseño es incompatible con las metodologías ágiles.	
	CLASE 5 - Diseño de la interfaz de usuario	
41.	El diseño de la interfaz de usuario (UI) se centra exclusivamente en la estética del software, sin considerar la interacción con el usuario.	
42.	El objetivo principal de la UI es facilitar una interacción atractiva y centrada en las necesidades del usuario.	
43.	El diseño de experiencia de usuario (UX) incluye métodos para satisfacer las necesidades del cliente y mejorar su experiencia.	
44.	Una interfaz difícil de usar puede provocar errores en los usuarios o incluso su rechazo al sistema.	
45.	Las tecnologías como hiperenlaces, sonido y realidad virtual deben adaptarse a las preferencias del usuario en el diseño de UI.	
46.	El proceso de diseño de interfaces de usuario es lineal y no requiere iteraciones.	
47.	La investigación de usuarios en UX puede incluir fuentes de datos como encuestas, información de ventas y datos de soporte al usuario.	
48.	El diseño de interacción en UX se enfoca en la apariencia visual de la aplicación, ignorando la satisfacción del usuario.	
49.	En la presentación de la información, se recomienda usar más de 7 colores en la interfaz total del sistema para mayor atractivo visual.	
50.	La interfaz de usuario es solo la punta del iceberg, ya que el diseño más complejo (como la arquitectura de información) está por debajo.	
	CLASE 6 - Diseño arquitectónico	
51.	El diseño arquitectónico se enfoca únicamente en la interfaz de usuario y su interacción con el sistema.	
52.	La arquitectura de software impacta directamente en requerimientos no funcionales como rendimiento, seguridad y disponibilidad.	
53.	En el patrón de repositorio, los subsistemas comparten una base de datos central, lo que facilita la integración de nuevas herramientas.	

54.	En el patrón cliente-servidor, los servidores deben conocer la identidad de los clientes para proporcionar servicios.	
55.	Una ventaja de la arquitectura en capas es que permite el desarrollo incremental y la portabilidad entre plataformas.	
56.	La descomposición modular orientada a flujo de funciones organiza el sistema en objetos débilmente acoplados con interfaces definidas.	
57.	El modelo de control centralizado de tipo "llamada y retorno" es adecuado para sistemas concurrentes.	
58.	En los sistemas dirigidos por eventos con modelo de broadcast, un evento se transmite a todos los subsistemas, y solo los programados lo manejan.	
59.	Las arquitecturas peer-to-peer (P2P) siempre requieren un servidor central para coordinar la comunicación entre nodos.	
60.	La documentación interna en la codificación incluye información detallada sobre algoritmos y estructuras de control para programadores que leen el código fuente.	
	CLASE 7 - Gestión de proyecto (métricas y estimaciones)	
61.	Una métrica de software es simplemente una medida directa sin cálculos ni fórmulas asociadas.	
62.	Las métricas del proyecto tienen propósitos tácticos, como ajustar el calendario para evitar demoras y rastrear riesgos.	
63.	Las métricas del proceso se utilizan para evaluar el desempeño individual de los miembros del equipo.	
64.	La métrica de Fan-in mide la cantidad de funciones que son llamadas por una función específica.	
65.	La métrica de Puntos de Función (PF) es una medida subjetiva independiente del lenguaje de programación.	
66.	En el método GQM, las métricas se definen primero, y luego se establecen las preguntas y objetivos.	
67.	La métrica de Complejidad Ciclomática está relacionada con la comprensión del control de un programa.	
68.	El modelo COCOMO II de composición de aplicación se basa en la estimación de puntos de aplicación.	
69.	Las estimaciones de software predicen únicamente el costo monetario del desarrollo, no el esfuerzo en horas-persona.	
70.	La técnica de Planning Poker es adecuada para metodologías ágiles y fomenta la colaboración en la estimación de historias de usuario.	
	CLASE 8 - Caja negra y caja blanca	
71.	La etapa de prueba del software es la primera instancia en la que se localizan defectos.	
72.	Un defecto por omisión ocurre cuando algún aspecto clave del código falta, como una variable no inicializada.	
73.	Las pruebas de caja negra se centran en los detalles procedimentales del código, como los caminos lógicos.	
74.	El principio de Pareto indica que el 80% de los errores en un software son generados por el 20% del código.	
75.	Las pruebas de caja blanca garantizan que todas las sentencias del programa se ejecuten al menos una vez.	
76.	La prueba de partición equivalente define clases de equivalencia solo para condiciones de entrada válidas.	
77.	El análisis de valores límite (AVL) selecciona casos de prueba en los extremos de las clases de equivalencia.	
78.	La complejidad ciclomática mide la cantidad de errores presentes en un programa.	
79.	Las pruebas de caja negra buscan errores relacionados con la interfaz de usuario y el rendimiento del sistema.	
80.	Un equipo independiente de pruebas es recomendable para evitar conflictos entre la responsabilidad por los defectos y la necesidad de descubrirlos.	

	CLASE 9 - Estrategias de prueba	
81.	Una estrategia de pruebas del software solo describe los pasos a seguir sin considerar el esfuerzo, tiempo o recursos requeridos.	
82.	La verificación asegura que el software satisface las expectativas del cliente, mientras que la validación comprueba que cumple con las especificaciones.	
83.	Las pruebas de unidad verifican el correcto funcionamiento de componentes individuales, como módulos o clases.	
84.	En las pruebas de integración descendente, se inicia probando los módulos de los niveles más bajos de la jerarquía de control.	
85.	Las pruebas de regresión se realizan para verificar que los cambios en el software no han introducido efectos colaterales no deseados.	
86.	En el enfoque ascendente de pruebas de integración, se eliminan los controladores, pero se requieren resguardos para los módulos subordinados.	
87.	Las pruebas de sistema evalúan el software como un todo para verificar que todos los elementos estén integrados correctamente.	
88.	Las pruebas de aceptación alfa se realizan en el lugar del cliente sin la presencia del desarrollador.	
89.	La depuración es un proceso que siempre ocurre como consecuencia de una prueba efectiva que descubre un error.	
90.	En las pruebas de unidad para software orientado a objetos, una clase encapsulada es típicamente el foco de la prueba.	
	CLASE 10 - Mantenimiento	
91.	El mantenimiento de software comienza únicamente después de que el sistema ha sido entregado al cliente y nunca durante el desarrollo.	
92.	Según las leyes de Lehman, un programa que se usa en un entorno real debe cambiar continuamente o se volverá menos útil en ese entorno.	
93.	El mantenimiento correctivo se realiza para mejorar la eficiencia o el rendimiento del sistema en respuesta a comentarios de los usuarios.	
94.	El mantenimiento preventivo incluye actividades como instalar programas antimalware para evitar fallos futuros en el software.	
95.	La re-estructuración del software implica analizar el código fuente para generar documentación como diagramas de flujo o tablas de interfaces.	
96.	La ingeniería inversa parte del código fuente para recuperar el diseño o la especificación de un sistema, especialmente cuando no hay documentación.	
97.	El ciclo de mantenimiento incluye fases como análisis, diseño, implementación, prueba y actualización de la documentación de apoyo.	
98.	El mantenimiento de software representa entre un 10% y 30% del costo total del desarrollo de un sistema.	
99.	Las herramientas de mantenimiento, como los comparadores de archivos, ayudan a identificar diferencias entre versiones del código fuente.	
100.	La re-ingeniería del software solo implica regenerar el código sin modificar la especificación o el diseño del sistema.	
	CLASE 11 - Auditoría Informática	
101.	La auditoría informática es una actividad exclusivamente correctiva que se realiza solo después de detectar problemas en los sistemas.	
102.	Uno de los objetivos de la auditoría informática es garantizar la integridad de los datos y la confidencialidad de la información.	
103.	La auditoría interna es realizada por personas externas a la empresa y no puede ser disuelta por decisión de la organización.	
104.	La metodología Octave incluye pasos como identificar activos, analizar amenazas y determinar vulnerabilidades.	
105.	El principio de independencia obliga al auditor a mantener autonomía en su trabajo, incluso si tiene una relación laboral con la empresa auditada.	

106.	Herramientas como Nessus y Wireshark se utilizan en auditorías informáticas para analizar vulnerabilidades y tráfico de red.	
107.	La metodología Magerit comienza con la evaluación de controles antes de la planificación de la auditoría.	
108.	Los reportes de hallazgos en auditorías informáticas clasifican los problemas por nivel de criticidad y proponen recomendaciones específicas.	
109.	Un consultor informático tiene un enfoque más técnico y específico que un auditor informático, centrándose en identificar riesgos y vulnerabilidades.	
110.	El principio de discreción exige al auditor evitar divulgar datos obtenidos durante la auditoría, incluso si parecen inofensivos.	

Ingeniería de Software II - 25-06-2024 - TEMA 1

Nombre y apellido: .....

Nro. Alumno: .....

Responda V (Verdadero) o F (Falso). Con tinta, no se puede utilizar lápiz.

1.	La planificación temporal es una actividad que distribuye el esfuerzo estimado a lo largo de la duración prevista del proyecto.	
2.	Las métricas post-mortem (como LDC) pueden conformar una línea base para futuras estimaciones.	
3.	La GCS es una actividad que se aplica al finalizar todo el proceso del software.	
4.	Las 4 P de la gestión de un proyecto de software son: Personal, Producto, Proceso, Permanencia.	
5.	El paradigma cerrado de gestión de equipos de desarrollo se caracteriza porque el trabajo se realiza de manera colaborativa y la toma de decisiones consensuadas constituyen las características de los equipos.	
6.	La planificación es el conjunto de actividades que asegura que el software implemente correctamente una función específica.	
7.	Una estrategia de resolución requiere tener estrategias de tratamiento del riesgo.	
8.	Los riesgos de desarrollo de software se categorizan en: producto, proyecto y negocio.	
9.	La etapa de diseño de software es una etapa en la que no se involucra la calidad porque es una representación gráfica.	
10.	El concepto de "abstracción" en el diseño de software permite concentrarse en un problema a un nivel de generalización sin tener en cuenta los detalles de bajo nivel.	
11.	En la representación de la información, además de utilizar un color para indicar el significado es necesario agregarle información textual.	
12.	En el diseño UX el "diseño visual" es la interacción entre un usuario y producto, donde el objetivo es que sea agradable para el usuario.	
13.	Los requerimientos no funcionales NO se ven afectados por la arquitectura de software.	
14.	Una de las desventajas del patrón de repositorio, en la arquitectura de software, es que los subsistemas deben estar acordes a los modelos de datos del repositorio.	
15.	La “evaluación administrativa del área de informática” es uno de los campos de acción de la auditoría informática.	
16.	La principal desventaja del enfoque descendente en las pruebas de integración es la necesidad de los resguardos.	
17.	La etapa de prueba no debería ser la primera en que se localizan defectos.	
18.	La prueba de caja blanca se refiere a las pruebas que se llevan a cabo sobre la interfaz del software.	
19.	La auditoría es una actividad meramente mecánica.	
20.	Las tareas de mantenimiento nunca provocan reiniciar las fases de análisis, diseño e implementación.	

Ingeniería de Software II - 25-06-2024 - TEMA 1

Nombre y apellido: .....

Nro. Alumno: .....

Responder en cada recuadro con tinta, no se puede utilizar lápiz.

1. Exprese 4 características de un buen SRS.

……

……

……

……

……

……

……

……

2. Explique 4 diferencias entre experiencia de usuario (UX) y desarrollo de interface (UI).

……

……

……

……

……

……

……

……

3. ¿Cuáles son las características del mantenimiento de software?

……

……

……

……

……

……

……

……

Ingeniería de Software II - 27-06-2024 - TEMA 2

Nombre y apellido: .....

Nro. Alumno: .....

Responda V (Verdadero) o F (Falso). Con tinta, no se puede utilizar lápiz.

1.	Las restricciones en el presupuesto o calendario NO son el origen del cambio en la Gestión de Configuración del Sistema (GCS).	
2.	Un proyecto es un esfuerzo temporal que se lleva a cabo para crear un producto, servicio o resultado único.	
3.	El equipo de software ágil se parece a la organización del paradigma abierto.	
4.	En la planeación de riesgos la estrategia denominada "plan de contingencia" establece que, siguiendo esta estrategia, la probabilidad que el riesgo se presente se reduce.	
5.	En la planeación temporal de un proyecto un "hito" es algo que se espera que esté hecho para alguna fecha, un logro que sea objetivo, fácil de evaluar y notable.	
6.	Del gráfico de PERT-CPM se puede obtener una ventana temporal para el desarrollo de cada actividad.	
7.	En el diseño de software se busca que los módulos tengan mucho acoplamiento y baja cohesión.	
8.	El refactoring de software es una técnica de reorganización que simplifica el diseño de un componente sin cambiar su función o comportamiento.	
9.	Los colores y el diseño visual son características centrales de la experiencia de usuario (UX).	
10.	El principio de consistencia de Jacob Nielsen en el desarrollo de interfaces indica que no debe haber ambigüedades en la terminología y lo visual.	
11.	El modelo de control centralizado para el diseño de una arquitectura de software se caracteriza por responder a eventos externos al subsistema.	
12.	Una de las desventajas de la arquitectura de sistemas distribuidos es la seguridad.	
13.	En la arquitectura distribuida interorganizacional cada nodo de procesamiento maneja su propia carga de trabajo, pero la carga informática general se comparte dinámicamente entre todos los nodos.	
14.	En la arquitectura Cliente-Servidor, los clientes y servidores son procesos diferentes. Además, los servidores pueden atender varios clientes donde un reflected puede brindar varios servicios y los clientes no se conocen entre sí.	
15.	La métrica LCD (líneas de código) es una métrica post mortem porque se calcula con datos de otros proyectos de software ya finalizados.	
16.	Si se realizan pruebas de software se garantiza la ausencia de defectos.	
17.	En las estrategias de prueba la verificación responde a la pregunta de ¿Estamos construyendo el producto correcto? de manera de asegurar que el software satisface las expectativas del cliente.	
18.	El mantenimiento de software es una actividad que incluye: corregir errores, mejorar las capacidades, eliminar funciones obsoletas y optimizar otras.	
19.	En las leyes de Lehman la "complejidad creciente" hace referencia a que la evolución de los programas es un proceso autorregulativo donde los atributos (tamaño, número de errores documentados, etc.) son aproximadamente invariantes en el tiempo.	
20.	En el rejuvenecimiento del software la "re-documentación" representa un análisis estático del código para producir la documentación del sistema.	

Ingeniería de Software II - 27-06-2024 - TEMA 2

Nombre y apellido: .....

Nro. Alumno: .....

Responder en cada recuadro con tinta, no se puede utilizar lápiz.

1. Indique 5 consejos para una buena comunicación del proyecto en formato presentación.

……

……

……

……

……

……

……

……

2. Describa 4 pautas de criterios técnicos para un buen diseño de software.

……

……

……

……

……

……

……

……

3. Indique y describa 3 principios aplicados al auditor informático.

……

……

……

……

……

……

……

……

Ingeniería de Software II - 28-06-2024 - TEMA 1

Nombre y apellido: .....

Nro. Alumno: .....

Responda V (Verdadero) o F (Falso). Con tinta, no se puede utilizar lápiz.

1.	La GCS es una actividad de autoprotección que se aplica durante el proceso del software.	
2.	El control de cambios permite al usuario especificar configuraciones alternativas del sistema mediante la selección de versiones adecuadas.	
3.	La autoridad de control de cambios (ACC) se encarga de realizar las revisiones técnicas formales.	
4.	La línea base es un concepto de GCS que nos ayuda a controlar los cambios.	
5.	Dependiendo del modelo de proceso seleccionado, el modelo está compuesto de conjuntos de tareas o actividades o no.	
6.	La mala administración del personal no es uno de los principales factores del fracaso de los proyectos.	
7.	El camino crítico puede obtenerse utilizando el cálculo del margen total.	
8.	Una tarea se puede describir como secuencia de acciones a realizar sin determinar el tiempo a utilizar.	
9.	Una tarea crítica es aquella cuyo retraso genera un retraso en todo el proyecto.	
10.	La planificación establece una secuencia operativa de tareas.	
11.	El riesgo de que personal experimentado abandone la organización antes de que finalice el proyecto es un riesgo del negocio.	
12.	Los riesgos de gran impacto con una probabilidad de moderada a alta y los riesgos de poco impacto, pero con gran probabilidad no deberían tomarse en cuenta.	
13.	Para la toma de decisión acerca del tratamiento de los riesgos, se debe tener en cuenta el costo de la aplicación de las estrategias.	
14.	Los riesgos conocidos se extrapolan de la experiencia en proyectos.	
15.	Una métrica proporciona una visión profunda que permite al gestor de proyectos ajustar el proceso para que las cosas salgan mejor.	
16.	Una de las reglas básicas del diseño de interfaz es no darle el control al usuario.	
17.	Para mejorar la seguridad del sistema no es conveniente agrupar las operaciones críticas en un conjunto reducido de subsistemas.	
18.	Una de las desventajas del patrón de repositorio es que los subsistemas deben estar acordes a los modelos de datos del repositorio.	
19.	La verificación del software responde a la pregunta: ¿Estamos construyendo el producto correctamente?	
20.	La auditoría es una actividad reactiva donde el auditor actúa.	

Ingeniería de Software II - 28-06-2024 - TEMA 1

Nombre y apellido: .....

Nro. Alumno: .....

Responder en cada recuadro con tinta, no se puede utilizar lápiz.

1. Explique los 4 tipos de diseño.

2. Explique 4 principios de Nielsen.

3. Explique 4 tipos de rejuvenecimiento de software.

Ingeniería de Software II - 28-06-2024 - TEMA 2

Nombre y apellido: .....

Nro. Alumno: .....

Responder en cada recuadro con tinta, no se puede utilizar lápiz.

1. Explique 4 tipos de mantenimiento.

……

……

……

……

……

……

……

……

2. Explique 4 tipos de prueba de software.

……

……

……

……

……

……

……

……

3. Explique 4 organizaciones de equipo de trabajo.

……

……

……

……

……

……

……

……

Ingeniería de Software II - 06-08-2024

Nombre y apellido: .....

Nro. Alumno: .....

Responda V (Verdadero) o F (Falso). Con tinta, no se puede utilizar lápiz. **(Total 4pts)**

1.	La “línea base” en el GCS es una especificación o producto que aún no se ha revisado formalmente y sobre el que se ha llegado a un acuerdo, y que de ahí en adelante sirve como base para un desarrollo posterior y que puede cambiarse solamente a través de procedimientos formales de control de cambio.	
2.	El software de una organización es el activo más grande, representa el capital intelectual.	
3.	El líder de un equipo de desarrollo de software debe alentar, celebrar los logros individuales y generar espíritu comunitario.	
4.	Realizar un desarrollo ágil de software implica dejar de lado la gestión de riesgos.	
5.	Los riesgos de poco impacto, pero con gran probabilidad de ocurrencia deberían tomarse en cuenta para la línea de corte.	
6.	En la planificación temporal un “hito” es una secuencia de acciones a realizar en un plazo determinado.	
7.	En la planificación temporal tener un margen total de 5 días significa que la tarea puede iniciarse con 5 días de retraso sin que ello afecte a la duración total del proyecto.	
8.	El diseño de experiencias de usuario (Ux) es un conjunto de métodos aplicados al proceso de diseño que buscan satisfacer las necesidades del cliente y proporciona una buena experiencia a los usuarios destinatarios.	
9.	Organizar la arquitectura de la información de una interfaz es uno de los principios del diseño de experiencia de usuario.	
10.	En el diseño de software la cohesión es una medida de fuerza o relación funcional existente entre las sentencias o grupos de sentencias de un mismo módulo.	
11.	El diseño arquitectónico de software que utiliza un patrón de cliente servidor tiene como ventaja que soporta el desarrollo incremental.	
12.	En el diseño arquitectónico de software el modelo de control basado en eventos es un subsistema que tiene la responsabilidad de iniciar y detener otro subsistema.	
13.	Las métricas postmortem como LDC permiten conformar una línea base para futuras métricas.	
14.	La técnica de estimación Delphi es desarrollada por cualquier grupo de personas con conocimientos o capacitación especializados en el tipo de desarrollo y se lleva un registro de cada persona que aportó a la estimación para volver a consultar.	
15.	La revisión de requerimientos y el diseño contribuyen a mejorar las pruebas dado que permiten descubrir los problemas (defectos) en las etapas tempranas del desarrollo de software.	
16.	Los costos de prueba obtenidos en la prueba de camino básico garantizan que se ejecuta al menos una vez cada sentencia del programa.	
17.	En la estrategia de prueba de integración considera un módulo crítico aquel que tiene un alto nivel de control.	
18.	Las pruebas de validación comienzan cuando terminan las pruebas de caja negra.	
19.	La ley de Lehman sobre estabilidad organizacional dice que el cambio incremental de cada entrega de desarrollo de software es aproximadamente constante.	
20.	El Mantenimiento adaptativo de software tiene que ver con las tecnologías cambiantes, así como con las políticas y reglas relacionadas con su software.	

Ingeniería de Software II - 06-08-2024

Nombre y apellido: .....

Nro. Alumno: .....

Responder en cada recuadro con tinta, no se puede utilizar lápiz.

1. Nombrar las 4 estrategias de prueba del software (no caja blanca o negra).

……

……

……

……

……

……

……

……

2. Mencione los 4 principios de Nielsen.

……

……

……

……

……

……

……

……

3. Mencione 4 tipos de rejuvenecimiento de software.

……

……

……

……

……

……

……

……