UNIVERSIDAD NACIONAL DEL ALTIPLANO - PUNO

FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

INFORME:

PRUEBA DE APLICACIONES CONVENCIONALES Y PRUEBA DE APLICACIONES ORIENTADAS A OBJETOS

PRESENTADA POR:

CARI CALLATA JOSÉ LUIS

CHAMBILLA CHOQUECOTA DRAKE LEONEL

ASIGNATURA:

EVALUACIÓN Y PRUEBAS DE SOFTWARE

DOCENTE:

CHOQUE CASTRO MARCOS DENYS

PUNO – PERÚ

2022

ÍNDICE

1. RESUMEN

La meta de las pruebas es encontrar errores, y una buena prueba es aquella que tiene una alta probabilidad de encontrar uno. Por tanto, un sistema basado en computadora o un producto debe diseñarse e implementarse teniendo en mente la "comprobabilidad". Al mismo tiempo, las pruebas en sí mismas deben mostrar un conjunto de características que logren la meta de encontrar la mayor cantidad de errores con el mínimo esfuerzo. Mientras que la meta de las pruebas orientadas a objetos es encontrar el número máximo de errores con una cantidad mínima de esfuerzo, es idéntico al de la prueba de software convencional, pero la estrategia y las tácticas de la prueba OO difieren significativamente. La visión de las pruebas se ensancha para incluir la revisión de los modelos de requerimientos y de diseño. Además, el foco de la prueba se mueve alejándose del componente procedimental y acercándose a la clase.

2. INTRODUCCIÓN

Las pruebas presentan una interesante anomalía para los ingenieros de software, quienes por naturaleza son personas constructivas. Las pruebas requieren que el desarrollador deseche nociones preconcebidas sobre lo “correcto” del software recién desarrollado y luego trabajen duro para diseñar casos de prueba a fin de “romper” el software. El software debe probarse para descubrir y corregir los tantos errores. La meta es diseñar una serie de casos de prueba que tengan una alta probabilidad de encontrar errores, para ello se utilizan técnicas que proporcionan lineamientos sistemáticos para diseñar pruebas que revisen la lógica interna y las interfaces de todo componente de software y revisen los dominios de entrada y salida del programa para descubrir errores en el funcionamiento, comportamiento y rendimiento del programa. Los objetos existen en la naturaleza, en entidades hechas por el hombre, en los negocios y en los productos que usamos. Pueden ser clasificados, descritos, organizados, combinados, manipulados y creados. Por eso no es sorprendente que se proponga una visión orientada a objetos para la creación de software de computadora, una abstracción que modela el mundo de forma tal que nos ayuda a entenderlo y gobernarlo mejor. El análisis está orientado a objetos y su diseño se enfoca en los objetos. Los objetos tienen ciertos comportamientos y atributos que determinan la manera en que interactúan y funcionan. No se intenta proporcionar un orden para las acciones al momento del diseño debido a que los objetos funcionan basados en la manera en que funcionan otros objetos. La programación orientada a objetos ayuda a los desarrolladores a crear objetos que reflejan escenarios del mundo real. El enfoque orientado a objetos permite que los objetos estén auto contenidos. Los objetos existen por sí mismos, con una funcionalidad para invocar comportamientos de otros objetos. Al utilizar un enfoque orientado a objetos, los desarrolladores pueden crear aplicaciones que reflejan objetos del mundo real, como rectángulos, elipses y triángulos, además de dinero, números de partes y elementos en un inventario.

3. PRUEBA DE APLICACIONES CONVENCIONALES Y PRUEBA DE APLICACIONES ORIENTADAS A OBJETOS

PRUEBA DE APLICACIONES CONVENCIONALES

Una vez generado el código fuente, el software debe probarse para descubrir (y corregir) tantos errores como sea posible antes de entregarlo al cliente. La meta es diseñar una serie de casos de prueba que tengan una alta probabilidad de encontrar errores; ¿pero cómo? Ahí es donde entran en escena las técnicas de prueba de software. Dichas técnicas proporcionan lineamientos sistemáticos para diseñar pruebas que revisen la lógica interna y las interfaces de todo componente de software y revisen los dominios de entrada y salida del programa para descubrir errores en el funcionamiento, comportamiento y rendimiento del programa.

Durante las primeras etapas del proceso, un ingeniero de software realiza todas las pruebas. Sin embargo, conforme avanza el proceso, pueden involucrarse especialistas en pruebas.

Las revisiones y otras acciones SQA pueden y deben descubrir errores, pero no son suficientes. Cada vez que el programa se ejecuta, ¡el cliente lo prueba! Por tanto, tiene que ejecutarse el programa antes de que llegue al cliente, con la intención específica de encontrar y remover todos los errores. Para encontrar el mayor número posible de éstos, las pruebas deben realizarse de manera sistemática y deben diseñarse casos de prueba usando técnicas sistematizadas

Para aplicaciones convencionales, el software se prueba desde dos perspectivas diferentes la primera es la lógica de programa interno en donde se revisa usando técnicas de diseño de casos de prueba de “caja blanca” y la segunda son los requerimientos de software se revisan usando técnicas de diseño de casos de prueba de “caja negra”. El uso de casos auxilia en el diseño de pruebas para descubrir errores de validación del software. En todo caso, la intención es encontrar el máximo número de errores con la mínima cantidad de esfuerzo y tiempo.

Se diseña y documenta un conjunto de casos de prueba elaborados para revisar la lógica interna, las interfaces, las colaboraciones de componentes y los requerimientos externos; se definen los resultados esperados y se registran los resultados reales.

Cuando se realizan pruebas, cambia el punto de vista. ¡Intente con ahínco "romper" el software! Diseñe casos de prueba en forma sistemática y revise minuciosamente los casos de prueba creados. Además, puede evaluar la cobertura de la prueba y rastrear las actividades de detección de errores.

PRUEBAS DE CAJA BLANCA

La prueba de caja blanca, en ocasiones llamada prueba de caja de vidrio, es una filosofía de diseño de casos de prueba que usa la estructura de control descrita como parte del diseño a nivel de componentes para derivar casos de prueba. Al usar los métodos de prueba de caja blanca, puede derivar casos de prueba que garanticen que todas las rutas independientes dentro de un módulo se revisaron al menos una vez, revisen todas las decisiones lógicas en sus lados verdadero y falso, ejecuten todos los bucles en sus fronteras y dentro de sus fronteras operativas y revisen estructuras de datos internas para garantizar su validez.

PRUEBAS DE RUTA BÁSICA

La prueba de ruta o trayectoria básica es una técnica de prueba de caja blanca propuesta por primera vez por Tom McCabe. El método de ruta básica permite al diseñador de casos de prueba derivar una medida de complejidad lógica de un diseño de procedimiento y usar esta medida como guía para definir un conjunto básico de rutas de ejecución. Los casos de prueba derivados para revisar el conjunto básico tienen garantía para ejecutar todo enunciado en el programa, al menos una vez durante la prueba.

NOTACIÓN GRÁFICA

Es una representación de flujo de control llamado gráfico de flujo o gráfico de programa, Cuando en un diseño de procedimiento se encuentran condiciones compuestas, la generación de un gráfico de flujo se vuelve ligeramente más complicada. Una condición compuesta

ocurre cuando uno o más operadores booleanos (OR, AND, NAND, NOR lógicos) se presentan en un enunciado condicional. En la figura 18.3, el segmento en lenguaje de diseño de programa (PDL, por sus siglas en inglés) se traduce en el gráfico de flujo mostrado. Observe que se crea un nodo separado para cada una de las condiciones a y b en el enunciado IF a OR b. Cada nodo que contiene una condición se llama nodo predicado y se caracteriza por dos o más aristas que emanan de él

RUTAS DE PROGRAMAS INDEPENDIENTES

Una ruta independiente es cualquiera que introduce al menos un nuevo conjunto de enunciados de procesamiento o una nueva condición en el programa. Cuando se establece como un gráfico de flujo, una ruta independiente debe moverse a lo largo de al menos una arista que no se haya recorrido antes de definir la ruta.

DERIVACIÓN DE CASO DE PRUEBA

El método de prueba de ruta básica puede aplicarse a un diseño de procedimientos o a un código fuente. En esta sección se presenta la prueba de ruta básica como una serie de pasos. El procedimiento average (promedio), que en la figura 18.4 se muestra en PDL, se usará como ejemplo para ilustrar cada paso del método de diseño de caso de prueba. Observe que average, aunque es un algoritmo extremadamente simple, contiene condiciones y bucles compuestos. Es posible aplicar los siguientes pasos para derivar el conjunto básico:

- Al usar el diseño o el código como cimiento, dibuje el gráfico de flujo correspondiente
- Determine la complejidad ciclomática del gráfico de flujo resultante
- Determine un conjunto básico de rutas linealmente independientes.
- Prepare casos de prueba que fuercen la ejecución de cada ruta en el conjunto básico.

MATRICES DE GRAFO

El procedimiento para derivar el gráfico de flujo e incluso determinar un conjunto de rutas básicas es sensible a la mecanización. Una estructura de datos llamada matriz de un grafo, puede ser bastante útil para desarrollar una herramienta de software que auxilie en la prueba

de ruta básica. Una matriz de grafo es una matriz cuadrada cuyo tamaño de las filas y columnas son iguales, cada fila y columna corresponde a un nodo identificado y las entradas de la matriz corresponden a conexiones entre nodos.

PRUEBAS DE LA ESTRUCTURA DE CONTROL

Es un método de diseño de casos de prueba que revisa las condiciones lógicas contenidas en un módulo de programa. Una condición simple es una variable booleana o una expresión relacional, posiblemente precedida de un operador NOT (¬). Una expresión relacional toma la forma

PRUEBA DE FLUJO DE DATOS

El método de prueba de flujo de datos selecciona rutas de prueba de un programa de acuerdo con las ubicaciones de las definiciones y con el uso de variables en el programa. Para ilustrar el enfoque de prueba de flujo de datos, suponga que a cada enunciado en un programa se le asigna un número de enunciado único y que cada función no modifica sus parámetros o variables globales.

PRUEBAS BASADAS EN MODELOS

La prueba basada en modelo (PBM) es una técnica de prueba de caja negra que usa la información contenida en el modelo de requerimientos como la base para la generación de casos de prueba. En muchos casos, la técnica de prueba basada en modelo usa diagramas de estado UML, un elemento del modelo de comportamiento, como la base para el diseño de los casos de prueba.7 La técnica PBM requiere cinco pasos:

1. Analizar un modelo de comportamiento existente para el software o crear uno
2. Recorrer el modelo de comportamiento y especificar las entradas que forzarán al software a realizar la transición de estado a estado.
3. Revisar el modelo de comportamiento y observar las salidas esperadas, conforme el software realiza la transición de estado a estado.
4. Ejecutar los casos de prueba

5. Comparar los resultados reales y esperados y adoptar una acción correctiva según se requiera.

PRUEBAS PARA ENTORNOS, ARQUITECTURAS Y APLICACIONES ESPECIALIZADOS

En ocasiones, los lineamientos y enfoques únicos para pruebas se garantizan cuando se consideran entornos, arquitecturas y aplicaciones especializadas.

PRUEBA DE INTERFACES DE GRAFICA DE USUARIO

Las interfaces gráficas para usuario (GUI, por sus siglas en inglés) presentan interesantes retos de prueba. Puesto que los componentes reutilizables ahora son parte común en los entornos de desarrollo GUI, la creación de la interfaz para el usuario se ha vuelto menos consumidora de tiempo y más precisa. Pero, al mismo tiempo, la complejidad de las GUI ha crecido, lo que conduce a más dificultad en el diseño y ejecución de los casos de prueba

PRUEBA DE APLICACIONES ORIENTADAS A OBJETOS

Según (Cajigas, 2014), el objetivo general de las pruebas orientadas a objeto es encontrar el número máximo de errores con el mínimo esfuerzo; el cual es idéntico al objetivo de las pruebas de software convencional. Pero la estrategia y táctica difiere significativamente en las pruebas OO.

Según (Orozco, 2014), para probar adecuadamente los sistemas OO:

1. Ampliar la definición de prueba para incluir las técnicas de descubrimiento de error aplicadas al análisis orientado a objetos y a modelos de diseño.
2. Cambiar significativamente la estrategia para prueba de unidad e integración.
3. Explicar las características únicas del software OO mediante el diseño de casos de prueba.

AMPLIACIÓN DE DEFINICIÓN DE PRUEBAS

Según (Orozco, 2014), los modelos de análisis (AOO) y diseño (DOO) orientados a objetos proporcionan información sustancial acerca de la estructura y comportamiento del sistema. Todos los modelos orientados a objetos deben probarse en relación con su exactitud, completitud y consistencia dentro del contexto de la sintaxis, la semántica y la pragmática del modelo.

La definición de las pruebas deben ampliarse para incluir técnicas de detección de errores aplicados a los modelos AOO y DOO. La estrategia para las pruebas de unidad e integración deben cambiar significativamente. El diseño de casos de prueba deben tener en cuenta las características propias del software orientado a objetos (Cajigas, 2014).

ANÁLISIS ORIENTADO A OBJETOS (AOO)

Según (virtual itca edu, n.d.), es un método de análisis que examina los requisitos desde la perspectiva de las clases y objetos que se encuentran en el vocabulario del dominio del problema".

La orientación a objetos puede describirse como el conjunto de disciplinas que desarrollan y modelizan software que facilitan la construcción de sistemas complejos a partir de componentes.

En el contexto del desarrollo de sistemas de software con orientación a objetos, se entiende por Análisis Orientado a Objetos al proceso de construcción de modelos del dominio del problema, identificando y especificando un conjunto de objetos semánticos que interactúan y se comportan de acuerdo a los requerimientos del sistema. Los objetos semánticos son aquellos que poseen un significado específico en el dominio del problema.

Documentos básicos del análisis orientado a objetos:

- Documentos de análisis. Contiene la documentación que aporta el cliente que encarga la aplicación. También contiene las actas de las reuniones de trabajo del grupo de análisis.
- Especificación de requisitos o requerimientos. La captura de requisitos es el proceso de averiguar, normalmente en circunstancias difíciles, lo que se debe construir. La captura de requisitos es complicada pues a veces los usuarios no explican bien lo que quieren, o por ser difícil tener una visión global del problema a resolver.
- Diagramas de casos de uso. Es uno de los 5 tipos de diagramas UML que se utilizan para el modelado de los aspectos dinámicos de un sistema. Estos representan una vista externa del sistema. Un modelo de casos de uso se construye mediante un proceso iterativo durante las reuniones entre los desarrolladores del sistema y los clientes, conduciendo a una especificación de requisitos sobre la que todos coinciden.
- Escenarios y subescenarios. Cada caso de uso da lugar a múltiples escenarios. Se codifican siguiendo la codificación de los casos de uso. Se estudia cada escenario utilizando guiones como los que usan en el cine. Cada equipo que pasa por un escenario identifica los objetos y sus responsabilidades, así como los mecanismos que relacionan los objetos.
- Prototipos y su evaluación. El prototipado consiste en la elaboración de un modelo o maqueta que se construye para evaluar mejor los requisitos que se desea que cumpla. Estos modelos o prototipos suelen consistir en versiones reducidas, demos o conjuntos de pantallas (que no son totalmente operativos) de la aplicación pedida.

DISEÑO ORIENTADO A OBJETOS (DOO)

Según (virtual itca edu, n.d.), el diseño Orientado a Objetos (DOO) difiere considerablemente del diseño estructurado ya que en DOO no se realiza un problema en términos de tareas

(subrutinas) ni en términos de datos, sino que se analiza el problema como un sistema de objetos que interactúan entre sí.

Un problema desarrollado con técnicas orientadas a objetos requiere, en primer lugar, saber cuáles son los objetos del programa. Como tales objetos son instancias de clases, la primera etapa en el desarrollo orientado a objetos requiere de la identificación de dichas clases (atributos y comportamiento), así como las relaciones entre éstas y su posterior implementación en un lenguaje de programación.

Aunque no siempre están bien delimitadas las etapas de análisis y diseño en la OO, se pueden sintetizar de alguna forma las ideas claves de las distintas tecnologías existentes dentro del desarrollo orientado a objetos al que denominaremos diseño.

En el diseño orientado a objetos, los objetos necesitan interactuar y comunicarse, para realizar dicha comunicación, los objetos utilizan su propia interfaz pública, dicha interfaz se compone principalmente de métodos y propiedades.

El diseño orientado a objetos transforma el modelo del análisis en un modelo de diseño que sirve como anteproyecto para la construcción de software.

Componentes del Diseño Orientado a Objetos:

- La identificación de objetos, sus atributos y servicios
- La organización de objetos dentro de una jerarquía
- La construcción de descripciones dinámicas de objetos que muestran cómo se usan los servicios
- La especificación de interfaces de objetos

Elementos básicos del Diseño Orientado a Objetos:

- Diagramas de interacción
- Diagramas de clases y consulta de patrones de diseño
- Diagramas de objetos
- Diagramas de actividades
- Diagramas de estados
- Diagramas de componentes
- Diagramas de despliegue

MODELOS DE PRUEBA AOO Y DOO

Según (Orozco, 2014), los modelos de análisis y diseño no pueden probarse de la manera convencional porque no pueden ejecutarse, Sin embargo pueden usarse revisiones técnicas para examinar su:

- Exactitud
- Consistencia

EXACTITUD DE MODELOS AOO Y DOO

La exactitud sintáctica se juzga mediante el uso adecuado de la simbología, cada modelo se revisa para garantizar que se mantienen las convenciones de modelado adecuadas. La exactitud semántica puede valorarse con base en la conformidad del modelo con el dominio de problemas del mundo real (Orozco, 2014).

CONSISTENCIA DE LOS MODELOS AOO y DOO

Según (Orozco, 2014), la consistencia de los modelos orientados a objetos puede juzgarse al considerar las relaciones entre entidades en el modelo. Un modelo de análisis o diseño inconsciente tiene representaciones en una parte del modelo que no se reflejan de manera correcta en otras porciones.

- Modelo clase-responsabilidad-colaboración (CRC): Cada tarjeta CRC menciona el nombre de la clase, sus responsabilidades (operaciones) y sus colaboradores.
- Diagrama de objeto-relación: Proporciona una representación gráfica de las conexiones entre clases.

ESTRATEGIAS DE PRUEBAS ORIENTADAS A OBJETOS

Según (Orozco, 2014), dicho en el lenguaje de las pruebas de software comienza con la:

- Prueba de unidad
- Prueba de integración
- Pruebas de validación y sistema

PRUEBA DE UNIDAD

A diferencia de la prueba de unidad del software convencional, que tiende a enfocarse en el detalle algorítmico de un módulo y en los datos que fluyen a través de la interfaz de módulo la prueba de clase para el software OO se activa mediante las operaciones encapsuladas por la clase y por el comportamiento de estado de la misma.

PRUEBA DE INTEGRACIÓN

Existen dos diferentes estrategias para la prueba de integración de los sistemas OO.

- Prueba basada en Hebra: Integra el conjunto de clases requeridas para responder a una entrada o evento del sistema.
- Prueba basada en Uso: Comienza la construcción del sistema al probar aquellas clases (llamadas independientes) que usan muy pocas clases del servidor (si es que emplean alguna).

PRUEBAS DE VALIDACIÓN Y SISTEMA

Se enfoca en las pruebas visibles para el usuario y en las salidas del sistema reconocibles por el mismo. para auxiliar en la derivación de pruebas de validación, el examinador debe recurrir a casos de uso que sean parte del modelo de requerimientos de interacción de usuario.

MÉTODOS DE PRUEBA ORIENTADA A OBJETOS

(Orozco, 2014).

1. Cada caso de prueba debe identificarse de manera única y explícita asociado con la clase que se va a probar.
2. Debe establecerse el propósito de la prueba.
3. Debe desarrollarse una lista de pasos de prueba para cada una de ellas, que deben contener:
    a. Una lista de estados
    b. Una lista de mensajes y operadores
    c. Una lista de excepciones
    d. Una lista de condiciones externas

e. Información complementaria que ayudará a comprender o a implementar la prueba

IMPLICACIONES DEL DISEÑO DE CASOS DE PRUEBA DE LOS CONCEPTOS OO

La encapsulación es un concepto de diseño esencial para OO, puede crear un obstáculo menor cuando se prueba, las pruebas requieren reportar el estado concreto y abstracto de un objeto, no obstante la encapsulación puede hacer que esta información sea un poco difícil de obtener.

La herencia también puede presentar retos adicionales durante el diseño de casos de prueba.

APLICABILIDAD DE MÉTODOS CONVENCIONALES DE DISEÑO DE CASOS DE PRUEBA

Los métodos de prueba de la caja blanca pueden aplicarse a las operaciones definidas para una clase Las técnicas de ruta básica, prueba de bucle o flujo de datos pueden ayudar pueden ayudar a garantizar que se probaron los enunciados en una operación.

Los métodos de prueba de caja negra son tan apropiados para los sistemas OO como para los sistemas desarrollados, usando métodos de ingeniería del software convencional.

PRUEBA BASADA EN FALLO

El objeto de la prueba basada en fallo dentro de un sistema OO es diseñar pruebas que tengan una alta probabilidad de descubrir fallos plausibles.

Puesto que el producto o sistema debe adecuarse a los requerimientos del cliente, la planificación preliminar requerida para realizar alguna prueba basado en fallo comienza con el modelo de análisis.

CASOS DE PRUEBA Y JERARQUÍA DE CLASE

Cada uno tendrá un conjunto de requerimientos de prueba derivadas de la especificación y la implementación. Dichos requerimientos de prueba sondean fallos plausibles: de integración, de condición, de frontera, etc. Pero es probable que las operaciones sean similares.

DISEÑO DE PRUEBAS BASADAS EN ESCENARIO

La prueba basada en escenario se concentra en lo que hace el usuario, no en lo que hace el producto. Esto significa capturar las tareas (por medio de casos de uso) que el usuario tiene que realizar y luego aplicar estas y sus variantes como pruebas.

Los escenarios descubren errores de interacción. Pero para lograr esto, los casos de prueba deben ser más complejos y más realistas que las pruebas basadas en fallos. La prueba basada en escenario tiende a ejercitar múltiples subsistemas en una sola prueba.

PRUEBAS DE LAS ESTRUCTURAS SUPERFICIAL Y PROFUNDA

La estructura superficial se hace referencia a la estructura observable externamente de un programa OO, es decir la estructura que es inmediatamente obvia para un usuario final, las pruebas se basan todavía en tareas de usuario. Capturar estas tareas involucran compresión, observación y hablar con usuarios representativos.

Cuando se habla de estructura profunda, se hace referencia a los detalles técnicos internos de un programa OO, es decir la estructura que se comprende al examinar el diseño y/o el código.

La prueba de estructura profunda se diseña para ejercitar dependencias, comportamientos y mecanismos de comunicación que se establezcan como parte del modelo de diseño de software OO.

MÉTODOS DE PRUEBA APLICABLE A NIVEL DE CLASE

Según (Orozco, 2014), la prueba se enfoca en una sola clase y en los métodos que encapsula esta. La prueba aleatoria y la participación son métodos que pueden usarse para ejercitar una clase durante la prueba OO.

PRUEBA ALEATORIA PARA CLASES OO

Estas y otras pruebas de orden aleatorio se realizan para ejercitar diferentes historias de vida de las instancias de clase.

PRUEBA DE PARTICIÓN EN EL NIVEL DE CLASE

La prueba de participación reduce el número de casos de prueba requeridos para ejercitar la clase, en una forma muy similar a la partición de equivalencia para el software tradicional.

- La partición con base en estado categoriza las operaciones de clase a partir de su capacidad para cambiar el estado de la clase.
- La partición con base en atributos categoriza las operaciones de clase con base en los atributos que usan.
- La partición basada en categoría jerarquiza las operaciones de clase con base en la función genérica que cada una realiza.

4. CONCLUSIONES

Las pruebas orientadas a objetos a diferencia de las pruebas convencionales, tienen mejores resultados en encontrar el número máximo de errores con una cantidad mínima de esfuerzo, gracias a la estrategia y tácticas de la prueba OO, ya que al realizar la revisión de los modelos de requerimientos y de diseño se tiene una perspectiva más y encontrar errores se vuelve más sencillo realizando las pruebas a nivel de código y el análisis y diseño OO.

5. BIBLIOGRAFÍA

Cajigas, D. (2014). *Pruebas Orientadas a Objeto*. SlidePlayer. Retrieved 12 27, 2022, from https://slideplayer.es/slide/1086375/

Orozco, A. (2014). *Prueba de aplicaciones orientadas a objetos*. SlidesShare. Retrieved 12 27, 2022, from https://es.slideshare.net/marcopolosilvasegovia/prueba-de-aplicaciones?from_action=save

virtual itca edu. (n.d.). *Aplicación de la ingeniería del software orientado a objetos para realizar el diseño y pruebas de un sistema*. Selección de técnicas de ingeniería de software. Retrieved 12 27, 2022, from https://virtual.itca.edu.sv/Mediadores/stis/index.html