LOGALI

TEORÍA

ABAP Unit Test – Clases de Test

SAP ABAP Programación Orientada a Objetos

Contenido

1. **Pruebas unitarias ABAP** 3
2. **Crear unidades de test** 4
3. **Test sistemático** 4
4. **Clase de test** 5
5. **Propiedades de las clases de test** 6
6. **Clase estándar CL_AUNIT_ASSERT** 7

1. Pruebas unitarias ABAP

Las pruebas unitarias se organizan en clases y luego en los métodos de prueba que forman parte de la TU - TestUnit. El sistema verifica las pequeñas unidades dentro de la unidad de test y de ahí el nombre de Unidad ABAP. El objetivo de un método es comprobar si una unidad devuelve el resultado deseado. Para este propósito, hay métodos de la clase de servicio (Service Class) CL_AUNIT_ASSERT que ejecutan las comparaciones entre los valores teóricos y reales calculados por una unidad. Los resultados de todas las pruebas unitarias de una TU se muestran en la pantalla de resultados Unidad ABAP.

La unidad ABAP le permite probar el código a nivel de unidad, como las pruebas de cada pieza por separado, por lo tanto se conocerá si podría funcionar o no con los resultados deseados cuando aquellas piezas se arman juntas en el puzle.

Por la filosofía de pruebas unitarias tenemos que crear pruebas de unidades funcionales aisladas. Las unidades de prueba individuales deberían, si es posible, ser de tamaño más pequeño e independientes unas de otras. Debe elegir como unidades de la unidad más pequeña que suministra un resultado que puede ser comprobado. Lo ideal sería que las unidades no tengan acceso unas a otras. Sin embargo, este requisito es realmente una situación ideal, y no siempre es práctico. En cualquier caso, las pruebas que se ejecutan unas tras otras no deben trabajar con datos u objetos que ya se han cambiado por otra prueba.

Ventajas del test de pequeñas unidades de código:

- En el caso de un error, el problema se puede localizar fácilmente si la unidad es muy pequeña.
- Idealmente, un error no puede influir en las otras pruebas en un ensayo debido a los requisitos de aislamiento.
- El test de unidades simples requiere menos esfuerzo que los escenarios de test complejos.

2. Crear unidades de test

Las pruebas unitarias se ejecutan regularmente durante el desarrollo. Esto tiene la consecuencia de que las fases de test pueden ser fácilmente acortadas o incluso descartadas como en el caso clásico donde las pruebas se producen sólo después del desarrollo.

2.1. Especificaciones precisas de test

En las pruebas unitarias el desarrollador crea definiciones precisas en cuanto a lo que un determinado procedimiento debe hacer y qué valores debe devolver. Si no puede crear definiciones precisas en cualquier unidad de prueba para un procedimiento particular, esto podría ser una indicación de que se debe replantear el pliego de condiciones de este procedimiento.

Por el contrario, es una buena idea definir en primer lugar las pruebas unitarias para un procedimiento y después aplicar el procedimiento. En cualquier caso, la aplicación de pruebas de unidad le ayudará a especificar sus procedimientos con mayor precisión y para averiguar si la aplicación de hecho coincide con la especificación.

2.2. Adaptar la modularización para los test

Si intenta dividir su unidad de prueba en unidades que se pueden probar de forma independiente unas de las otras, va lograr un mayor grado de modularización. Mientras que está diseñando las pruebas unitarias, se dará cuenta si la codificación es o no suficientemente modularizada y se adaptará en consecuencia. De esta manera, su codificación productiva será mejor.

3. Test sistemático

Las pruebas unitarias son sistemáticas. Lo ideal es cubrir todas las partes de una unidad de test y ejecutar los test siempre con los mismos datos de prueba. Esto tiene muchas ventajas en comparación con las pruebas

irregulares que los desarrolladores ejecutan donde los casos de test se pierden. Así se puede navegar sin ninguna dificultad a las líneas de código problemáticas después del test.

Las pruebas unitarias son típicas pruebas de caja blanca donde sólo el desarrollador las puede ejecutar. El desarrollador sabe su codificación tan bien que él o ella puede crear unidades funcionales pequeñas. En conjunto, los test unitarios le proporcionan la seguridad de que todo seguirá funcionando después de los cambios del programa si funcionaba de antemano.

En comparación con las pruebas de integración la unidad de test es muy pequeña y las pruebas de la unidad se producen muy pronto durante el desarrollo. Las pruebas de integración se pueden centrar en los errores que se producen a través de la integración ya que las unidades individuales ya son probadas antes de la integración. En las pruebas unitarias aparecen errores en una fase temprana del ciclo de vida de un componente. De esta manera, como programador, tiene mucho más tiempo para corregirlos.

4. Clase de test

Para crear una clase como clase de test tiene que añadir las palabras claves FOR TESTING en la definición de la clase. Para definir un método de test tiene que añadir las mismas palabras claves FOR TESTING en la declaración del método.

```
CLASS lcl_test DEFINITION FOR TESTING.

  PUBLIC SECTION.

    METHODS test_factorial FOR TESTING.

ENDCLASS.
```

5. Propiedades de las clases de test

Cada clase de test tiene propiedades que se tienen que definir en la definición de la clase de test. El adicional FOR TESTING es muy importante para definir una clase de TEST. Si omite la adición, la clase va ser considerada una clase ABAP habitual. El adicional FOR TESTING la hace una clase de test. También se necesita al menos un método FOR TESTING la clase de TEST.

Las propiedades de las clases de test de la unidad ABAP son:

- **Risk Level** o el nivel de riesgo
- **Duration** o la duración del test, hablando en tiempos de ejecución

La propiedad **Risk Level** define el nivel de riesgo de un Test. Al ejecutar la prueba, el sistema comprueba el nivel de riesgo frente a nivel de riesgo predefinido en la configuración del cliente. Si la prueba tiene un nivel de riesgo más alto, la Unidad ABAP ignoraría esa prueba y no se ejecutará el test.

Puede especificar uno de estos tres valores posibles:

- **Harmless** – Inofensiva. La ejecución de esta prueba no afecta a los procesos o la base de datos. Esta es la configuración por defecto.
- **Dangerous** – Peligroso (alarmante). Este tipo de prueba realiza cambios en los datos persistentes o en la base de datos.
- **Critical** – Crítica. Este tipo de prueba podría realizar cambios en Customizing, así como los datos persistentes. Este tipo de prueba tiene un nivel alto de responsabilidad y requiere un cuidadoso uso.

Estos son los tres niveles de riesgo: Harmless, Dangerous and Critical de la propiedad Risk Level

La segunda propiedad DURATION tiene sus tres posibles niveles. Similar al nivel de riesgo, toda la prueba tendría que tener una duración de tiempo de ejecución esperado. ABAP Unity no ejecutará las pruebas con una duración superior a la definida en la configuración de nivel de cliente. En Duración, se especifican los siguientes valores:

- **Short** – Corto. Es ejecutado muy rápido. Esta es la configuración por defecto en la configuración del cliente. En general menos de 1 minuto
- **Medium** – Medio. Es ejecutado en el intervalo de 1 minuto a 10 minutos
- **Long** – Larga. Toma un tiempo para procesar la prueba. El tiempo de ejecución sería de más de 10 minutos.

6. Clase estándar CL_AUNIT_ASSERT

Para trabajar con ABAP UNIT tenemos la clase estándar de la CL_AUNIT_ASSERT de la unidad ABAP tiene varios métodos para probar varios escenarios diferentes.

Uno de los métodos más utilizados es: ASSERT_EQUALS

Este método toma valores actuales y esperados como argumentos y le dice a la unidad ABAP si el test pasa o falla.