TESTING KNOWLEDGE REPORT

1 DE JUNIO DE 2022

E2.07

https://github.com/anac29/Acme-Toolkits

Jaime Borrego Conde: jaiborcon@alum.us.es

Antonio Campos Gil: antcamgil@alum.us.es

Ana Conde Marrón: anaconmar@alum.us.es

Gonzalo Martínez Fernández: gonmarfer2@alum.us.es

Jaime Moscoso Bernal: jaimosber@alum.us.es

Enrique Muñoz Pérez: enrmunper@alum.us.es

Tabla de contenido

Resumen ejecutivo

El objetivo de este documento es detallar los conocimientos adquiridos durante el transcurso de la asignatura acerca del testing de software. Estas ideas se han ido aplicando en el proyecto Acme Toolkits conforme avanzaba su desarrollo.

Tabla de revisión

Número de revisión	Fecha	Descripción
0	29/05/2022	Primera versión

Introducción

El testing de software es el proceso que verifica y valida que las funcionalidades del sistema, por un lado, no tengan defectos, y por otro que actúen como el usuario lo espera.

Este proceso se lleva a cabo paralelamente al desarrollo de software, con el objetivo de identificar errores y, posteriormente, corregir el sistema. Como resultado, permite asegurar la calidad del software desarrollado, y es por ello que es fundamental llevarlo a cabo desde una etapa temprana en el desarrollo del sistema.

Esta disciplina se ha aprendido y aplicado en el proyecto *Acme-Toolkits* a lo largo de la asignatura “Diseño y Pruebas II”, y en este documento se introducirán y señalarán los aspectos más importantes que se han aprendido en la misma.

Conceptos de la disciplina del testing

Los conceptos aprendidos en la asignatura pueden clasificarse en conceptos básicos, conceptos de programación o de testing.

Respecto a los conceptos básicos, se ha estudiado cómo interpretar los requisitos, cómo plantear las funcionalidades a desarrollar en el sistema, cómo localizar fallos y darles solución.

Los conceptos de programación se han aplicado en el desarrollo de *Acme-Toolkits* para el proceso de depuración. Se han considerado errores y posibles pantallas de pánico, tanto por un usuario que no deba acceder a una página, como por el intento de acceso a vistas de entidades que no existen... También, se ha aprendido a diferenciar entre peticiones legales e ilegales, y cómo evitar estas últimas.

Por último, en cuanto a los conceptos de testing, se ha desarrollado en el proyecto un conjunto de pruebas (clases Java), en el que se incluyen tanto pruebas positivas como negativas para la mayoría de las funcionalidades. Cada una de estas clases se centran en una funcionalidad en concreto e intentan cubrir la mayor parte posible de código del proyecto.

Objetivo de las pruebas funcionales

En complemento a los conceptos anteriores, se ha estudiado por qué se deben realizar las pruebas funcionales, que consisten en encontrar fallos en el sistema, obteniendo resultados que no son los esperados. Esto conllevaría a rastrear los errores hasta su origen para posteriormente poder solucionarlos.

Calidad dentro del testing

En el proceso de testing es muy importante conseguir la máxima cobertura. Sin embargo, existen dos tipos de cobertura, la cobertura de rutas y la cobertura de datos.

La cobertura de rutas es una referencia que resulta de calcular el porcentaje de rutas que exploran los casos de prueba. Cuanto mayor sea este porcentaje, más confianza se puede tener en el sistema. Las rutas de ejecución se dividen en secuencias, condicionales y bucles. Para conseguir una mayor cobertura, es imprescindible probar el máximo de caminos.

Por otro lado, la cobertura de datos es el porcentaje resultante de la cantidad de variabilidad en los datos de pruebas. Cuanto más diferentes sean los datos y mayor diversidad aporten a los test, mayor será el porcentaje, y la fiabilidad sobre el sistema será mejor.

Casos de prueba y conjunto de pruebas

Dentro del concepto de testing se pueden distinguir dos términos, dependiendo de la cantidad de características a probar. El primero es el *Caso de prueba* o *Benchmark*, referido a una prueba centrada en la evaluación individual de una característica. El segundo término, *Conjunto de pruebas o Test suite*, hace referencia a una colección de casos de pruebas para un proyecto.

Tipos de casos de prueba

Existen distintos casos de prueba a la hora de crear test. Dentro de esta asignatura se ha aprendido acerca de los tres tipos que existen. Para todos los casos de prueba es necesario añadir datos de prueba, que sirven para comparar, dado el resultado final de cada test, si el resultado era esperado o no.

En primer lugar, los casos de prueba positivos simulan el funcionamiento común de la aplicación, cómo funcionaría el sistema al ser usado por una persona con inteligencia natural y que realizase acciones correctas y sin errores.

Por otra parte, los casos de prueba negativos son similares a los positivos, pero con el objetivo de encontrar errores en el sistema, se introducen datos erróneos para poder probar las funcionalidades en los casos contrarios a los positivos. Esto podría relacionarse con un usuario malintencionado o que se equivocase al realizar acciones.

Por último, las pruebas de hacking permiten para validar la autorización y seguridad del sistema. Dentro de este tipo de test se prueban los inicios de sesión de la aplicación, y las funcionalidades accesibles para cada rol del sistema.

Pruebas realizadas sobre Acme-Toolkits

Esta sección recopila el proceso general que han seguido los tester del grupo para probar las funcionalidades del sistema. En general, todas han sido pruebas end-to-end.

Listados — Pruebas positivas

Las pruebas para los listados comienzan iniciando sesión en la aplicación con el rol adecuado. Posteriormente, se accede a la vista correspondiente, y se comprueba que haya un listado. Entonces, por cada entrada de la tabla que representa el listado, se comprueba columna a columna que los datos que contienen se corresponden con los datos que contiene esa misma entrada en el fichero `.csv` de pruebas.

Finalmente, si es necesario, se cierra sesión.

```
@ParameterizedTest
@CsvFileSource(resources = "/any/chirp/list.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(10)
public void positiveTest(final int recordIndex, final String creationMoment,final String title, final String author,
    final String body, final String email ) {


    super.clickOnMenu("Anonymous","List Chirp");
    super.checkListingExists();
    super.sortListing(0, "asc");

    super.checkColumnHasValue(recordIndex, 0, creationMoment);
    super.checkColumnHasValue(recordIndex, 1, title);
    super.checkColumnHasValue(recordIndex, 2, author);
    super.checkColumnHasValue(recordIndex, 3, body);
    super.checkColumnHasValue(recordIndex, 4, email);

}
```

Vistas Show — Pruebas positivas

Las vistas *show* muestran los detalles de una entidad. El comienzo de una prueba de esta vista implica iniciar sesión con el rol adecuado y acceder al listado que contiene las entidades a probar, tal y como se hacía en los test de listado.

Una vez en esa vista, se simula el acceso a la entidad a través de un clic, y se comprueba si la vista de *show* (formulario) existe. Si la acción es satisfactoria se procede a comparar los valores de cada campo con los valores de la entrada `.csv` correspondiente.

Por último, se cierra la sesión en la aplicación si fuese necesario.

```
@ParameterizedTest
@CsvFileSource(resources = "/authenticated/announcement/list.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(10)
public void positiveTest(final int recordIndex, final String creationMoment, final String title,
    final String body, final String flag, final String link)
{
    super.signIn("patron1", "patron1");

    super.clickOnMenu("Authenticated","List Announcements");
    super.checkListingExists();
    super.sortListing(1, "asc");

    super.checkColumnHasValue(recordIndex, 0, creationMoment);
    super.checkColumnHasValue(recordIndex, 1, title);
    super.checkColumnHasValue(recordIndex, 2, flag);

    super.clickOnListingRecord(recordIndex);
    super.checkFormExists();
    super.checkInputBoxHasValue("creationMoment", creationMoment);
    super.checkInputBoxHasValue("title", title);
    super.checkInputBoxHasValue("body", body);
    super.checkInputBoxHasValue("flag", flag);
    super.checkInputBoxHasValue("link", link);

}
```

Creación de entidades

El primer paso para probar la creación de entidades es acceder a la aplicación con el usuario con el rol apropiado. Una vez hecho esto, es menester navegar hacia la vista de creación de la entidad deseada, que puede estar localizada en diversos lugares.

Casos positivos

A partir del punto anterior, un caso positivo rellena el formulario de creación con los valores necesarios para hacer que la entidad se cree correctamente. Tras ello, se accede a la vista de listado y show de la nueva entidad para corroborar que contiene los datos adecuados.

```
@ParameterizedTest
@CsvFileSource(resources = "/any/chirp/create-positive.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(10)
public void positiveTest(final int recordIndex, final String title, final String author, final String body, final String email) {
    super.clickOnMenu("Anonymous", "List Chirp");
    super.checkListingExists();

    super.clickOnButton("Create Chirp");
    super.fillInputBoxIn("title", title);
    super.fillInputBoxIn("author", author);
    super.fillInputBoxIn("body", body);
    super.fillInputBoxIn("email", email);
    super.fillInputBoxIn("confirmation", "true");
    super.clickOnSubmit("Create");

    super.clickOnMenu("Anonymous", "List Chirp");
    super.checkListingExists();
    super.sortListing(1, "asc");
    super.checkColumnHasValue(recordIndex, 1, title);
    super.checkColumnHasValue(recordIndex, 2, author);
    super.checkColumnHasValue(recordIndex, 3, body);
    super.checkColumnHasValue(recordIndex, 4, email);

}
```

Casos negativos

Retornando al punto de la introducción, un caso de prueba negativo para la creación de entidades intentará rellenar el formulario con datos erróneos o no válidos que provoquen una advertencia. Simplemente, se comprueba que al intentar enviar el formulario se produzcan errores en la entidad.

```
@ParameterizedTest
@CsvFileSource(resources = "/any/chirp/create-negative.csv", encoding = "utf-8", numLinesToSkip = 1)
@Order(20)
public void negativeTest(final int recordIndex, final String title, final String author,
        final String body, final String email, final String confirmation) {
    super.clickOnMenu("Anonymous", "List Chirp");
    super.checkListingExists();

    super.clickOnButton("Create Chirp");
    super.fillInputBoxIn("title", title);
    super.fillInputBoxIn("author", author);
    super.fillInputBoxIn("body", body);
    super.fillInputBoxIn("email", email);
    super.fillInputBoxIn("confirmation", confirmation);
    super.clickOnSubmit("Create");

    super.checkErrorsExist();
}
```

Actualización de entidades

La funcionalidad de este apartado tiene exactamente el mismo procedimiento que la creación de entidades, exceptuando que, en vez de enviar el formulario como creación, se hace como actualización (mediante otro botón).

Eliminación de entidades

Eliminar una entidad se prueba de una manera particular pues, se comprobará seleccionando una entidad existente en la base de datos y al borrarla, se comprobará que esa entidad ya no existe en ella. Anteriormente se ha debido iniciar sesión en la aplicación con el rol adecuado, y se ha debido navegar a la vista desde la que se pueda eliminar la entidad deseada.

Casos positivos

Por tanto, un caso positivo es acceder a la vista donde se encuentran las entidades y borrar la primera de ellas. Anteriormente solo se han añadido dos entradas por tanto al eliminar la primera solo se tendrá que comprobar que al listar las entidades ahora la primera entrada corresponde a la segunda entidad pues la primera de ellas ha sido satisfactoriamente eliminada.

```
@ParameterizedTest
@CsvFileSource(resources = "/inventor/item/delete-positive.csv", encoding = "utf-8", numLinesToSkip = 
@Order(10)
public void positiveTest(final int recordIndex,
    final String name, final String code, final String technology, final String description,
    final String retailPrice, final String itemType, final String link, final String nameAd,
    final String codeAd,
    final String technologyAd, final String descriptionAd) {

    super.signIn("inventor2", "inventor2");

    super.clickOnMenu("Inventor","List my Items");
    super.checkListingExists();
    super.sortListing(1, "asc");

    super.clickOnButton("Create");
    super.fillInputBoxIn("name", name);
    super.fillInputBoxIn("code", code);
    super.fillInputBoxIn("technology", technology);
    super.fillInputBoxIn("description", description);
    super.fillInputBoxIn("retailPrice", retailPrice);
    super.fillInputBoxIn("itemType", itemType);
    super.fillInputBoxIn("link", link);
    super.clickOnSubmit("Create");

    super.clickOnMenu("Inventor","List my Items");
    super.checkListingExists();
    super.sortListing(1, "asc");

    super.checkColumnHasValue(recordIndex, 0, name);
    super.checkColumnHasValue(recordIndex, 1, code);
    super.checkColumnHasValue(recordIndex, 2, itemType);
    super.checkColumnHasValue(recordIndex, 3, technology);

    super.clickOnListingRecord(recordIndex);
    super.checkFormExists();
    super.checkInputBoxHasValue("name", name);
    super.checkInputBoxHasValue("code", code);
    super.checkInputBoxHasValue("technology", technology);
    super.checkInputBoxHasValue("description", description);
    super.checkInputBoxHasValue("retailPrice", retailPrice);

    super.clickOnSubmit("Delete");

    super.clickOnMenu("Inventor","List my Items");
    super.checkListingExists();
    super.sortListing(1, "asc");

    super.checkColumnHasValue(recordIndex, 0, nameAd);
    super.checkColumnHasValue(recordIndex, 1, codeAd);
    super.checkColumnHasValue(recordIndex, 3, technologyAd);
    super.checkColumnHasValue(recordIndex, 4, descriptionAd);

    super.clickOnListingRecord(recordIndex);
    super.checkFormExists();
    super.checkInputBoxHasValue("name", nameAd);
    super.checkInputBoxHasValue("code", codeAd);
    super.checkInputBoxHasValue("technology", technologyAd);
    super.checkInputBoxHasValue("description", descriptionAd);

    super.clickOnSubmit("Delete");
    super.checkNotErrorsExist();
}
```

Caso negativo

Cuando publicas una entidad esta no puede ser borrada, por ello solo se tiene que publicar primero una entidad y comprobar que se obtiene un error tras aplicar la operación de borrado.

```
@ParameterizedTest
@CsvFileSource(resources = "/inventor/item/delete-negative.csv", encoding = "utf-8", numLinesToSkip =
@Order(10)
public void negativeTest(final int recordIndex,
    final String name, final String code, final String technology, final String description,
    final String retailPrice, final String itemType, final String link) {

    super.signIn("inventor2", "inventor2");

    super.clickOnMenu("Inventor","List my Items");
    super.checkListingExists();
    super.sortListing(1, "asc");

    super.clickOnButton("Create");
    super.fillInputBoxIn("name", name);
    super.fillInputBoxIn("code", code);
    super.fillInputBoxIn("technology", technology);
    super.fillInputBoxIn("description", description);
    super.fillInputBoxIn("retailPrice", retailPrice);
    super.fillInputBoxIn("itemType", itemType);
    super.fillInputBoxIn("link", link);
    super.clickOnSubmit("Create");

    super.clickOnMenu("Inventor","List my Items");
    super.checkListingExists();
    super.sortListing(1, "asc");

    super.checkColumnHasValue(recordIndex, 0, name);
    super.checkColumnHasValue(recordIndex, 1, code);
    super.checkColumnHasValue(recordIndex, 2, itemType);
    super.checkColumnHasValue(recordIndex, 3, technology);

    super.clickOnListingRecord(recordIndex);
    super.checkFormExists();
    super.checkInputBoxHasValue("name", name);
    super.checkInputBoxHasValue("code", code);
    super.checkInputBoxHasValue("technology", technology);
    super.checkInputBoxHasValue("description", description);
    super.checkInputBoxHasValue("retailPrice", retailPrice);

    super.clickOnSubmit("Publish");

    super.clickOnMenu("Inventor","List my Items");
    super.checkListingExists();
    super.sortListing(1, "asc");

    super.checkColumnHasValue(recordIndex, 0, name);
    super.checkColumnHasValue(recordIndex, 1, code);
    super.checkColumnHasValue(recordIndex, 3, technology);
    super.checkColumnHasValue(recordIndex, 4, description);

    super.clickOnListingRecord(recordIndex);
    super.checkFormExists();

    super.checkInputBoxHasValue("name", name);
    super.checkInputBoxHasValue("code", code);
    super.checkInputBoxHasValue("technology", technology);
    super.checkInputBoxHasValue("description", description);

    super.clickOnMenu("Inventor","List my Items");
    super.checkListingExists();
    super.sortListing(1, "asc");
    super.clickOnListingRecord(recordIndex);
    super.checkNotButtonExists("Delete");
}
```

Conclusiones

Durante esta asignatura se ha dado la oportunidad de aprender numerosos conceptos importantes acerca del proceso de testing de software, que debe realizarse de manera conjunta al desarrollo de software con el fin de detectar problemas en la funcionalidad del sistema.

Dicho proceso permite identificar de manera temprana errores que pueda haber en el software, lo que facilita solucionarlos a medida que avanza el desarrollo de Acme-Toolkits. Todo este proceso permite garantizar la calidad, seguridad y el alto rendimiento del software que el equipo de trabajo desarrolle.

En el proyecto presentado, el rol de tester ha sido compartido por todos los miembros del equipo, por lo tanto, ha sido posible que todos los integrantes conserven al menos una parte de los conocimientos adquiridos y hayan podido ponerlos en práctica.

En conclusión, la disciplina que se ha estudiado y desarrollado en la asignatura ha resultado ser muy útil para poder aplicarla en futuros proyectos en el ámbito laboral.

Bibliografía

[1] — S03 - Functional testing (Theory).pptx
[2] — S04 - Functional testing (Laboratory).pptx
[3] — S05 - Performance testing (Theory, Laboratory).pptx
[4] — S05 - Perfomance testing (Erratum).docx