Validación y Verificación del Software

Análisis de los casos de prueba

java-etherpad-lite

RAQUEL DÍAZ OTERO
25/11/2018

Contenido

Análisis individual de los casos de prueba

Pruebas de unidad

Los casos de prueba que se analizan a continuación están clasificados dentro de la sección de pruebas de unidad ya que prueban unidades funcionales.

Caso de prueba 1: domain_with_trailing_slash_when_construction_an_api_path

En este caso de prueba se verifica que el método apiPath, sobre una conexión EPLiteConnection cuyo dominio termina en "/", devuelve correctamente la URL asignada a un método de la api.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el funcionamiento del método bajo condiciones normales de uso, y de **caja negra**, ya que no es necesario conocer la implementación concreta del método para diseñar este caso de prueba puesto que solo se verifica que la salida de la ejecución es la correcta para una entrada determinada.

Caso de prueba 2: domain_without_trailing_slash_when_construction_an_api_path

En este caso de prueba se verifica que el método apiPath, sobre una conexión EPLiteConnection cuyo dominio no termina en "/", devuelve correctamente la URL asignada aún método de la api.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el funcionamiento del método bajo condiciones normales de uso, y de **caja negra**, ya que no es necesario conocer la implementación concreta del método para diseñar este caso de prueba puesto que solo se verifica que la salida de la ejecución es la correcta para una entrada determinada.

Caso de prueba 3: query_string_from_map

En este caso de prueba se verifica el correcto funcionamiento del método queryString para una URL sin codificación y un conjunto de argumentos que se pasan como parámetros de entrada. El test comprueba que la query devuelta por el método es la esperada.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el funcionamiento del método bajo condiciones normales de uso, y de **caja negra**, ya que no es necesario conocer la implementación concreta del método para diseñar este caso de prueba puesto que solo se verifica que la salida de la ejecución es la correcta para una entrada determinada.

Caso de prueba 4: url_encoded_query_string_from_map

En este caso de prueba se verifica el correcto funcionamiento del método queryString para una URL con codificación y un conjunto de argumentos que se pasan como parámetros de entrada. El test comprueba que la query devuelta por el método es la esperada.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el funcionamiento del método bajo condiciones normales, y de **caja negra**, ya que no es necesario conocer la implementación concreta del método para diseñar este caso de prueba puesto que solo se comprueba que la salida de la ejecución es la correcta para una entrada determinada.

Caso de prueba 5: api_url_need_to_be_absolute

En este caso de prueba se verifica el correcto funcionamiento del método apiUrl pasándole como parámetros de entrada un path relativo y una query nula. El test comprueba que el método lanza la excepción correspondiente al error provocado.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **negativa**, ya que prueba el funcionamiento del método en condiciones no normales de uso, es decir, comprueba que se produzca un error, y de **caja blanca**, ya que es necesario conocer la implementación concreta del método para diseñar este caso de prueba puesto que se precisa información sobre las condiciones bajo las cuales se produce el error (en la documentación no se especifica que la URL deba ser absoluta y el nombre de la excepción producida no es lo suficientemente descriptivo como para deducir bajo qué condiciones se produce).

Caso se prueba 6: handle_valid_response_from_server

En este caso de prueba se verifica el correcto funcionamiento del método handleResponse al que se le pasa una posible respuesta del servidor formada por un conjunto de pads. El test comprueba que la salida del método contiene el primero de los pads introducidos en la respuesta que se le ha pasado como parámetro de entrada.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el funcionamiento del método en condiciones normales de uso, y de **caja negra**, ya que no es necesario conocer la implementación concreta del método para diseñar este caso de prueba puesto que solo se verifica que la salida de la ejecución es la correcta para una entrada determinada.

Caso se prueba 7: handle_invalid_parameter_error_from_server

En este caso de prueba se verifica el correcto funcionamiento del método handleResponse al que se le pasa una posible respuesta del servidor formada por un error causado por la

inexistencia del groupID. El test comprueba que el método lanza la excepción correspondiente y que el mensaje de error contiene el texto adecuado.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **negativa**, ya que prueba el funcionamiento del método en condiciones no normales de uso, es decir, comprueba que se produzca un error, y de **caja blanca**, ya que es necesario conocer la implementación concreta del método para diseñar este caso de prueba puesto que se precisa información sobre el error, sobre las condiciones bajo las cuales se produce (el nombre de la excepción producida no es lo suficientemente descriptivo) y sobre el mensaje de error asociado a la excepción.

Caso se prueba 8: handle_internal_error_from_server

En este caso de prueba se verifica el correcto funcionamiento del método handleResponse al que se le pasa una posible respuesta del servidor formada por un error causado por un fallo interno. El test comprueba que el método lanza la excepción correspondiente y que el mensaje de error contiene el texto adecuado.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **negativa**, ya que prueba el funcionamiento del método en condiciones no normales de uso, es decir, comprueba que se produzca un error, y de **caja blanca**, ya que es necesario conocer la implementación concreta del método para diseñar este caso de prueba puesto que se precisa información sobre el error, sobre las condiciones bajo las cuales se produce (el nombre de la excepción producida no es lo suficientemente descriptivo) y sobre el mensaje de error asociado a la excepción.

Caso se prueba 9: handle_no_such_function_error_from_server

En este caso de prueba se verifica el correcto funcionamiento del método handleResponse al que se le pasa una posible respuesta del servidor formada por un error causado por la inexistencia de la función requerida. El test comprueba que el método lanza la excepción correspondiente y que el mensaje de error contiene el texto adecuado.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **negativa**, ya que prueba el funcionamiento del método en condiciones no normales de uso, es decir, comprueba que se produzca un error, y de **caja blanca**, ya que es necesario conocer la implementación concreta del método para diseñar este caso de prueba puesto que se precisa información sobre el error, sobre las condiciones bajo las cuales se produce (el nombre de la excepción producida no es lo suficientemente descriptivo) y sobre el mensaje de error asociado a la excepción.

Caso se prueba 10: handle_invalid_key_error_from_server

En este caso de prueba se verifica el correcto funcionamiento del método handleResponse al que se le pasa una posible respuesta del servidor formada por un error causado por la

inexistencia/no validez de la clave de la API. El test comprueba que el método lanza la excepción correspondiente y que el mensaje de error contiene el texto adecuado.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **negativa**, ya que prueba el funcionamiento del método en condiciones no normales de uso, es decir, comprueba que se produzca un error, y de **caja blanca**, ya que es necesario conocer la implementación concreta del método para diseñar este caso de prueba puesto que se precisa información sobre el error, sobre las condiciones bajo las cuales se produce (el nombre de la excepción producida no es lo suficientemente descriptivo) y sobre el mensaje de error asociado a la excepción.

Caso se prueba 11: unparseable_response_from_the_server

En este caso de prueba se verifica el correcto funcionamiento del método handleResponse al que se le pasa una posible respuesta del servidor formada por un texto HTML que no se puede parsear. El test comprueba que el método lanza la excepción correspondiente y que el mensaje de error contiene el texto adecuado.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **negativa**, ya que prueba el funcionamiento del método en condiciones no normales de uso, es decir, comprueba que se produzca un error, y de **caja blanca**, ya que es necesario conocer la implementación concreta del método para diseñar este caso de prueba puesto que se precisa información sobre el error, sobre las condiciones bajo las cuales se produce (el nombre de la excepción producida no es lo suficientemente descriptivo) y sobre el mensaje de error asociado a la excepción.

Caso se prueba 12: unexpected_response_from_the_server

En este caso de prueba se verifica el correcto funcionamiento del método handleResponse al que se le pasa una posible respuesta del servidor formada por un texto vacío. El test comprueba que el método lanza la excepción correspondiente y que el mensaje de error contiene el texto adecuado.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **negativa**, ya que prueba el funcionamiento del método en condiciones no normales de uso, es decir, comprueba que se produzca un error, y de **caja blanca**, ya que es necesario conocer la implementación concreta del método para diseñar este caso de prueba puesto que se precisa información sobre el error, sobre las condiciones bajo las cuales se produce (el nombre de la excepción producida no es lo suficientemente descriptivo) y sobre el mensaje de error asociado a la excepción.

Caso se prueba 13: valid_response_with_null_data

En este caso de prueba se verifica el correcto funcionamiento del método handleResponse al que se le pasa una posible respuesta del servidor formada por un texto con datos nulos. El test comprueba que el método parsea correctamente el contenido de la respuesta recibida.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el funcionamiento del método en condiciones normales de uso, y de **caja negra**, ya que no es necesario conocer la implementación concreta del método para diseñar este caso de prueba puesto que solo se verifica que la salida de la ejecución es la correcta para una entrada determinada.

Pruebas de integración

Los casos de prueba que se analizan a continuación están clasificados dentro de la sección de pruebas de integración ya que prueban la integración de varios componentes del sistema.

Caso de prueba 1: validate_token

En este caso de prueba se verifica el correcto funcionamiento del método checkToken. El test comprueba que el método se ejecuta sin que se produzca ningún error, pero no comprueba el contenido de ningún valor.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el funcionamiento del método en condiciones normales de uso, y de **caja negra**, ya que no es necesario conocer la implementación concreta del método para diseñar este caso de prueba puesto que solo se verifica que no se produce ningún error durante su ejecución.

Caso de prueba 2: create_and_delete_group

En este caso de prueba se verifica el correcto funcionamiento del método createGroup y del método deleteGroup. El test comprueba que el método createGroup se ejecuta correctamente contrastando la respuesta devuelta por el servidor con la respuesta esperada (verificando así que se ha creado el grupo), y que el método deleteGroup se ejecuta sin que se produzca ningún error, pero no verifica que se haya eliminado el grupo.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el funcionamiento de los métodos en condiciones normales de uso, y de **caja negra**, ya que no es necesario conocer la implementación concreta de los métodos para diseñar este caso de prueba puesto que solo se verifica que la salida de la ejecución es la esperada para una entrada determinada y que no se produce ningún error durante a ejecución.

Caso de prueba 3: create_group_if_not_exists_for_and_list_all_groups

En este caso de prueba se verifica el correcto funcionamiento del método createGroupIfNotExistsFor y del método listAllGroups. El test comprueba que los métodos se ejecutan correctamente contrastando el contenido de las respuestas devueltas por el servidor con el contenido esperado (verificando así que se crean y se listan los grupos).

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el funcionamiento de los métodos en condiciones normales de uso, y de **caja negra**, ya que no es necesario conocer la implementación concreta de los métodos para diseñar este caso de prueba puesto que solo se verifica que la salida de la ejecución es la esperada para una entrada determinada.

Caso de prueba 4: create_group_pads_and_list_them

En este caso de prueba se verifica el correcto funcionamiento de los métodos createGroupPad, setPublicStatus, createGroupPad, setPassword, isPasswordProtected, createGroupPad, getText y listPads. El test comprueba que los métodos se ejecutan correctamente contrastando el contenido de las respuestas devueltas por el servidor con el contenido esperado.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el funcionamiento de los métodos en condiciones normales de uso, y de **caja negra**, ya que no es necesario conocer la implementación concreta de los métodos para diseñar este caso de prueba puesto que solo se verifica que la salida de la ejecución es la esperada para una entrada determinada.

Caso de prueba 5: create_author

En este caso de prueba se verifica el correcto funcionamiento de los métodos createAuthor y getAuthorName. El test comprueba que los métodos se ejecutan correctamente contrastando el contenido de las respuestas devueltas por el servidor con el contenido esperado (verificando así que se crea el autor y que su nombre es el adecuado).

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el funcionamiento de los métodos en condiciones normales de uso, y de **caja negra**, ya que no es necesario conocer la implementación concreta de los métodos para diseñar este caso de prueba puesto que solo se verifica que la salida de la ejecución es la esperada para una entrada determinada.

Caso de prueba 6: create_author_with_author_mapper

En este caso de prueba se verifica el correcto funcionamiento de los métodos createAuthorIfNotExistsFor (2 parámetros) y createAuthorIfNotExistsFor (1 parámetro). El test comprueba que los métodos se ejecutan correctamente contrastando el contenido de las respuestas devueltas por el servidor con el contenido esperado.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el funcionamiento de los métodos en condiciones normales de uso, y de **caja negra**, ya que no es necesario conocer la implementación concreta de los métodos para diseñar este caso de prueba puesto que solo se verifica que la salida de la ejecución es la esperada para una entrada determinada.

Caso de prueba 7: create_and_delete_session

En este caso de prueba se verifica el correcto funcionamiento de los métodos createSession, getSessionInfo, listSessionsOfGroup, listSessionsOfAuthor y deleteSession. El test comprueba que los métodos createSession, getSessionInfo, listSessionsOfGroup y listSessionsOfAuthor se ejecutan correctamente contrastando el contenido de las respuestas devueltas por el servidor con el contenido esperado; y del método deleteSession verificando que su ejecución no produce ningún error, pero no se comprueba que se elimine la sesión.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el funcionamiento de los métodos en condiciones normales de uso, y de **caja negra**, ya que no es necesario conocer la implementación concreta de los métodos para diseñar este caso de prueba puesto que solo se verifica que la salida de la ejecución es la esperada para una entrada determinada y que no se produce ningún error durante la ejecución.

Caso de prueba 8: create_pad_set_and_get_content

En este caso de prueba se verifica el correcto funcionamiento de los métodos createPad, setText, setHTML, getHTML, getText, getRevisionsCount, getRevisionChangeset (1 parámetro), getRevisionChangeset (2 parámetros), createDiffHTML, appendText, getAttributePool, saveRevision (1 parámetro), saveRevision (2 parámetros), getSavedRevisionsCount, listSavedRevisions, padUsersCount, padUsers, getReadOnlyID, getPadID, listAuthorsOfPad, getLastEdited, sendClientsMessage y deletePad. El test comprueba que los métodos se ejecutan correctamente contrastando el contenido de las respuestas devueltas por el servidor con el contenido esperado y verificando que su ejecución no produce ningún error. En el caso del método deletePad, no se comprueba que se elimine el pad.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el funcionamiento de los métodos en condiciones normales de uso, y de **caja negra**, ya que no es necesario conocer la implementación concreta de los métodos para diseñar este caso de prueba puesto que solo se verifica que la salida de la ejecución es la esperada para una entrada determinada y que no se produce ningún error durante la ejecución.

Caso de prueba 9: create_pad_move_and_copy

En este caso de prueba se verifica el correcto funcionamiento de los métodos copyPad (2 parámetros), movePad (2 parámetros), copyPad (3 parámetros) y movePad (3 parámetros). El test comprueba que los métodos se ejecutan correctamente contrastando el contenido de las respuestas devueltas por el servidor con el contenido esperado, antes y después de las modificaciones realizadas.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el

funcionamiento de los métodos en condiciones normales de uso, y de **caja negra**, ya que no es necesario conocer la implementación concreta de los métodos para diseñar este caso de prueba puesto que solo se verifica que la salida de la ejecución es la esperada para una entrada determinada y que no se produce ningún error durante la ejecución.

Caso de prueba 10: create_pads_and_list_them

En este caso de prueba se verifica el correcto funcionamiento del método listAllPads. El test comprueba que el método se ejecuta correctamente verificando que la respuesta devuelta por el servidor contiene todos lo pads creados previamente en el test.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el funcionamiento del método en condiciones normales de uso, y de **caja negra**, ya que no es necesario conocer la implementación concreta del método para diseñar este caso de prueba puesto que solo se verifica que la salida de la ejecución es la esperada para una entrada determinada.

Caso de prueba 11: create_pad_and_chat_about_it

En este caso de prueba se verifica el correcto funcionamiento de los métodos appendChatMessage, getChatHead, getChatHistory (1 parámetro) y getChatHistory (3 parámetros). El test comprueba que los métodos se ejecutan correctamente contrastando el contenido de las respuestas devueltas por el servidor con el contenido esperado, y verificando que su ejecución no produce ningún error.

Se trata de una prueba **funcional**, ya que pretende certificar que el software hace lo que tiene que hacer, **dinámica**, ya que requiere la ejecución del software, **positiva**, ya que prueba el funcionamiento de los métodos en condiciones normales de uso, y de **caja negra**, ya que no es necesario conocer la implementación concreta del método para diseñar este caso de prueba puesto que solo se verifica que la salida de la ejecución es la esperada para una entrada determinada y que no se produce ningún error durante la ejecución.

Conclusiones

Todas las pruebas realizadas (tanto las de unidad como las de integración) son pruebas funcionales, es decir, verifican el funcionamiento del software desarrollado, pero no se han diseñado pruebas estructurales que verifiquen la calidad interna del software o pruebas no funcionales que se encarguen de comprobar el rendimiento. De igual forma, no existen pruebas estáticas que inspeccionen el código sin necesidad de ejecutarlo, ya que todas las pruebas desarrolladas son dinámicas, y apenas existen pruebas de caja blanca que tengan en cuenta la implementación concreta de los elementos del sistema, en lugar de comprobar simplemente que los valores de salida son los correctos para unas entradas determinadas como se hace en las pruebas de caja negra. Al no tener en cuenta esos aspectos, el software puede tener carencias y bugs relacionados con ellos que no afecten al funcionamiento de los elementos y componentes testeados y que por tanto se pasen por alto con las pruebas elaboradas.

En relación a las pruebas de unidad, existen tanto pruebas positivas como negativas, es decir, se comprueba el comportamiento de las unidades en condiciones normales de ejecución y también se comprueban casos de error. Sin embargo, las pruebas se han diseñado teniendo en cuenta un escaso conjunto de valores de prueba, sin emplear criterios como las particiones equivalentes, valores-frontera, la selección de datos aleatorios… y en algunos casos las comprobaciones son parciales y no se verifican completamente los valores de retorno de los métodos. Esto disminuye la calidad de las pruebas y puede ocasionar que haya errores que no se detecten.

En relación a las pruebas de integración, todas las pruebas elaboradas son positivas, es decir, solo se verifica el funcionamiento de los componentes en condiciones normales de uso, pero en ningún momento se prueban casos de error. De igual manera que para los test de unidad, los casos de prueba se han diseñado teniendo en cuenta un escaso conjunto de valores de prueba, sin emplear distintos criterios para la elección de los mismos, lo que puede ocasionar que no se detecten algunos errores. Además, en la mayoría de los test faltan comprobaciones puesto que no se verifica que los métodos de borrado de los elementos eliminen dicho objeto correctamente. Hay que tener en cuenta que al tratarse de pruebas que únicamente verifican el funcionamiento del software en condiciones normales, podrían clasificarse dentro de la categoría de pruebas de aceptación ya que este tipo de pruebas se emplean para demostrar el correcto funcionamiento del software mediante el testeo de los casos de uso más relevantes.

En cuanto a la cobertura de las pruebas, el porcentaje del 94% representa la cantidad de líneas de código cubiertas por los casos de prueba. Este nivel de cobertura demuestra que existen líneas del código que no se ejecutan en ningún momento y además, esta métrica no proporciona ninguna información respecto a la calidad de las pruebas realizadas.

En conclusión, la gran parte de las pruebas realizadas verifican el correcto funcionamiento de los elementos del sistema en condiciones normales de uso, pero existe una carencia a la hora de probar condiciones de error y de probar aspectos no relacionados con el funcionamiento, como

la robustez del sistema, la calidad interna del software, el rendimiento de los componentes, el rendimiento general del sistema, la usabilidad...