CLASIFICACIÓN DE LAS PRUEBAS

Pruebas en EPLiteConnectionTest

Los tests de este archivo trabajan sólo con la clase EPLiteConnection, pero hay algunos métodos de esa clase que por debajo tienen llamadas a otra clase, y si no se mockean, las pruebas de dichos métodos serían de integración. Dichos métodos son getObject y postObject, llamados a su vez por algún método más, pero da la casualidad de que justo esos métodos no se prueban aquí, y dado que los métodos que sí se prueban no tienen llamadas a otras clases, y los tests tampoco, todas las pruebas de EPLiteConnectionTest son de unidad.

- domain_with_trailing_slash_when_construction_an_api_path:
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra, ya que no necesita conocer el código por dentro, con conocer el formato del string devuelto por el método le sería suficiente.
  - Positiva, ejecuta el software en condiciones normales.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De unidad.
- domain_without_trailing_slash_when_construction_an_api_path
  - Dinámica, ya que requiere que se ejecute el código.
  - Podría ser de caja negra, ya que es la misma prueba que la anterior, con un pequeño cambio. El primer parámetro pasado al crear la conexión lo manda sin una “/” al final, ya que en el constructor de la conexión se comprueba si la URL lleva o no “/” al final (para poder eliminarla si es el caso). Si esta característica estuviese en la documentación, este test sería de caja negra, pero creo que en este caso el diseñador de la prueba conocía el código y por ello probó a pasar y no pasar la “/”, por lo que es de caja blanca.
  - Positiva, ejecuta el software en condiciones normales.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De unidad.
- query_string_from_map
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra, no necesita conocer el código por dentro, con conocer el formato del string devuelto por el método le sería suficiente.
  - Positiva, ejecuta el software en condiciones normales.
  - Funcional, se preocupa por qué es lo que hace el código
  - De unidad.

- url_encoded_query_string_from_map
  - Dinámica, ya que requiere que se ejecute el código.
  - Podría ser de caja negra ya que no necesita conocer el código por dentro, con conocer el formato del string devuelto por el método le sería suficiente.
  - Positiva, ejecuta el software en condiciones normales, aunque le pasa valores frontera (caracteres especiales) para ver si acepta diferentes codificaciones.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De unidad.
- api_url_need_to_be_absolute
  - Dinámica, ya que requiere que se ejecute el código.
  - Podría ser de caja negra ya que no necesita conocer el código por dentro, simplemente necesita saber que el método sólo acepta URLs absolutas, y es algo que aparece explícitamente en el javadoc del método llamado.
  - Negativa, espera que al ejecutar se lance una excepción.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De unidad.
- handle_valid_response_from_server
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra, ya que no necesita conocer el código por dentro, con conocer cómo es el formato de llamadas y respuestas del servidor es suficiente.
  - Positiva, ejecuta el software en condiciones normales.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De unidad.
- handle_invalid_parameter_error_from_server
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra, ya que no necesita conocer el código por dentro, solo debe conocer el formato y la codificación de los mensajes recibidos por el servidor.
  - Negativa, espera que salte una excepción al pasarle unos parámetros concretos.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De unidad.

- handle_internal_error_from_server
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra, ya que no necesita conocer el código por dentro, solo debe conocer el formato formato y la codificación de los mensajes recibidos por el servidor.
  - Negativa, espera que salte una excepción al pasarle unos parámetros concretos.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De unidad.
- handle_no_such_function_error_from_server
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra, ya que no necesita conocer el código por dentro, solo debe conocer el formato formato y la codificación de los mensajes recibidos por el servidor.
  - Negativa, espera que salte una excepción al pasarle unos parámetros concretos.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De unidad.
- handle_invalid_key_error_from_server
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra, ya que no necesita conocer el código por dentro, solo debe conocer el formato formato y la codificación de los mensajes recibidos por el servidor.
  - Negativa, espera que salte una excepción al pasarle unos parámetros concretos.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De unidad.
- unparsable_response_from_the_server
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra, ya que no necesita conocer el código por dentro, solo debe conocer que el servidor requiere formato JSON, y le pasa algo que no es JSON.
  - Negativa, espera que salte una excepción al pasarle unos parámetros concretos.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De unidad.

- unexpected_response_from_the_server
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra, ya que no necesita conocer el código por dentro, solo debe conocer que el servidor requiere una serie de parámetros, y le pasa un mensaje vacío.
  - Negativa, espera que salte una excepción al pasarle unos parámetros concretos.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De unidad.
- valid_response_with_null_data
  - Dinámica, ya que requiere que se ejecute el código.
  - Podría ser de caja negra ya que no necesita conocer el código por dentro, solo debe conocer que el formato requerido por el servidor.
  - Positiva, ejecuta el software en condiciones normales, aunque le pase un valor frontera (null en vez de datos).
  - Funcional, se preocupa por qué es lo que hace el código.
  - De unidad.

Pruebas en EPLiteClientIntegrationTest

Los tests en este archivo sólo trabajan con la clase EPLiteClient, no hacen llamadas a ninguna otra clase, pero los métodos a los que llaman tienen llamadas por debajo a otra clase, EPLiteConnection, que al no estar mockeada, hace que las pruebas pasen a ser de integración.

- validate_token
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra ya que no necesita conocer el código por dentro, llama a un método.
  - Positiva, ejecuta el código en condiciones normales.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De integración.
- create_and_delete_group
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra ya que no necesita conocer el código por dentro, conoce el formato de las respuestas del servidor y con eso comprueba que esté lo necesario.
  - Positiva, ejecuta el código en condiciones normales.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De integración.
- create_group_if_not_exists_for_and_list_all_groups
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra ya que no necesita conocer el código por dentro, conoce el formato de las respuestas del servidor y con eso comprueba que esté lo necesario.
  - Positiva, ejecuta el código en condiciones normales, aunque comprueba que no se puedan crear dos grupos iguales.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De integración.
- create_group_pads_and_list_them
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra ya que no necesita conocer el código por dentro, conoce el formato de las llamadas y respuestas del servidor y con eso comprueba que esté lo necesario.
  - Positiva, ejecuta el código en condiciones normales.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De integración.

- create_author
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra ya que no necesita conocer el código por dentro, conoce el formato de las llamadas y respuestas del servidor y con eso comprueba que esté lo necesario.
  - Positiva, ejecuta el código en condiciones normales.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De integración.
- create_author_with_author_mapper
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra ya que no necesita conocer el código por dentro, conoce el formato de las llamadas y respuestas del servidor y con eso comprueba que esté lo necesario.
  - Positiva, ejecuta el código en condiciones normales, pero probando valores frontera, como crear varios autores con una función que crea si no existe ninguno.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De integración.
- create_and_delete_session
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra, no necesita conocer el código por dentro, conoce la documentación del cliente donde vienen explicados todos los formatos recibidos y devueltos por dicho cliente y los utiliza para ejecutar las pruebas.
  - Positiva, ejecuta el código en condiciones normales.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De integración.
- create_pad_set_and_get_content
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra ya que no necesita conocer el código por dentro, conoce el formato de las llamadas y respuestas del servidor y con eso comprueba que esté lo necesario.
  - Positiva, ejecuta el código en condiciones normales, aunque con valores frontera para ver que acepta caracteres con distintas codificaciones.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De integración.

- create_pad_move_and_copy
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra, no necesita conocer el código por dentro, conoce la documentación del cliente donde vienen explicados todos los formatos recibidos y devueltos por dicho cliente y los utiliza para ejecutar las pruebas.
  - Positiva, ejecuta el código en condiciones normales.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De integración.
- create_pads_and_list_them
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra, no necesita conocer el código por dentro, conoce la documentación del cliente donde vienen explicados todos los formatos recibidos y devueltos por dicho cliente y los utiliza para ejecutar las pruebas.
  - Positiva, ejecuta el código en condiciones normales.
  - Funcional, se preocupa por qué es lo que hace el código.
  - De integración.
- create_pad_and_chat_about_it
  - Dinámica, ya que requiere que se ejecute el código.
  - Caja negra, no necesita conocer el código por dentro, conoce la documentación del cliente donde vienen explicados todos los formatos recibidos y devueltos por dicho cliente y los utiliza para ejecutar las pruebas.
  - Positiva, ejecuta el código en condiciones normales, y además probando valores frontera (caracteres en otra codificación).
  - Funcional, se preocupa por qué es lo que hace el código.
  - De integración.

Conclusión

A la vista de esto queda claro que aunque el número de pruebas y la cobertura parezcan adecuadas, pueden quedar zonas sin probar, al faltar muchos tipos de test. Todos los tests son dinámicos, no hay ninguno que no necesite que se ejecute el código o que lo simbolice; casi todas son de caja negra; y son todas funcionales.

Al no haber realmente pruebas de caja blanca ni estructurales, sólo se está probando el funcionamiento del código, pero no su calidad interna, lo que puede dejar sin comprobar flujos mal diseñados o caminos mal estructurados en el código. Tampoco se comprueba el rendimiento del código al no haber pruebas no funcionales, y en una aplicación con llamadas a servidor constantes, sería muy interesante comprobar que es capaz de manejar múltiples solicitudes en un período corto de tiempo, por ejemplo.

Las pruebas de unidad de la clase EPLiteConnection se dejan por probar varios métodos, como por ejemplo el método trustServerAndCertificate, que aunque sea privado, las pruebas de unidad deberían de contemplarlo, sobre todo siendo un método que tiene una complejidad bastante alta. En las pruebas de integración, por otra parte, a pesar de cubrir la mayor parte del código, se echa en falta algún caso de prueba que lance errores.

En cuanto a la cobertura en sí misma, gracias a Coveralls se puede ver qué lineas están cubiertas y cuáles no, y esto nos muestra que la clase EPLiteClient tiene un 94% de cobertura y en general prueba todo, excepto dos pequeños métodos. Como es una clase con poca complejidad de código (pocos ifs, ramificaciones…) los tests cubren todos los posibles caminos. No es así en EPLiteConnection, una clase cuyos métodos tienen más ifs, diferentes try catch, y Coveralls nos indica que hay un par de catchs a los que no se llega. Esto puede ser un efecto colateral de que los tests hayan sido de caja negra y, al no tener conocimiento del código, se hayan pasado por alto condiciones. Sería interesante lanzar estos catch con un diseño de particiones equivalentes buscando encontrar todas las posibles combinaciones. Lo que sí se prueba, y es muy interesante, es una especie de valores frontera para los métodos que aceptan strings, y es pasarles strings con caracteres de distintas codificaciones, para ver si se manejan bien.

En general, a pesar de tener una cobertura muy alta, estos tests tienen varias lagunas. Se reducen a un par de clases de tests, y esto hace que queden muchas cosa por probar, como el rendimiento, en una aplicación en la que es una parte crucial. Además, para los tipos de tests que sí que están, faltan por probar condiciones y caminos, en parte debido a ser unos tests de caja negra en su gran mayoría.