VALIDACIÓ DE PROGRAMES

Islam El Omrani Akhazzan
CFGS Desenvolupament Web 1
15/05/22

ÍNDEX

INTRODUCCIÓ:
Amb aquests exercicis aprendrem lo que es el Test Drive Development, una forma de programar en la que en comptes de programar de a a b programem per tests, fent fallar d'objectiu en objectiu fins a aconseguir el resultat final que és un programa sense errades ni forats negres.

Pregunta 1. Els meus errors

- Un dels errors que més recordo és el que vaig tenir amb un exercici de la botiga de vins. Hi havia un test del Junit, l'elimina vi per referència que no em funcionava de cap manera I vaig estar dies I dies donant-li voltes.

- La detecció va ser que em fallava tota l'estona el mateix test, per tant trobar l'error no va ser gaire difícil.

- Vaig arribar a tal grau de desesperació que vaig haver d'anar als fitxers del prgtest per saber amb exactitud com funcionava el test. Vaig copiar-lo i vaig emular-lo al meu sistema amb un debugger i PER FI vaig veure que el mètode cridava a l'afegir vi, i aquest afegia per nom, no per referència. Això era el que fallava perquè el vi que volíem eliminar era un vi que no havíem pogut afegir perquè tenia el mateix nom que un altre vi.

La satisfacció final d'arreglar l'error i veure com passa tots els tests va pagar tot el sofriment que vaig tenir mentre feia l'exercici. Mai ho oblidaré.

Pregunta 2. El cicle de TDD

1. Afegir un test que passi les especificacions requerides, l'aspecte clau aquí és que el desenvolupador es fixa primer en els requisits abans de fer el codi.

2. Fem córrer tots els tests, el nou test ha de fallar perquè el codi encara no l'hem modificat.

3. Escriure el codi més simple que faci passar el nou test.

4. El codi hauria de passar tots els tests ara.

5. Refactoritzar el codi, és a dir millorar la llegibilitat o fer-lo més curt sense canviar el seu comportament exterior.

Com a velocitat ho veig més lent perquè fer test després codi a mi al menys em costaria molt. Però crec que a la llarga és una manera molt millor de programar, el codi que generes no té cap errada i en projectes més grans crec que em pot ser molt útil. Mantenir-lo també és més fàcil perquè afegeixes nous tests i nou codi que solucioni els tests així que mai pots espatllar el que et funcionava en el passat.

Pregunta 3. Qui fa què

Sincerament crec que qualsevol testejador pot fer cadascun dels 4 tipus de proves perquè tot i les seves diferències al final son proves i no hi veig molta diferència fer-les, així que deixo les explicacions de cada tipus de proves, m'han quedat bastant clars els 4 tipus.

També crec que les proves s'han de fer abans de codificar així que o no he entés la pregunta o no tenia gaire sentit per mí, cosa que dubto.

Les proves de sistema (també end-to-end) posen tots els elements de l'aplicació en conjunt, i sense tenir en compte els detalls de codificació, ens enfoquem en les funcionalitats que demanà el client. Aquestes proves es solen realitzar en un entorn el més semblant possible al de l'usuari final. Sovint són proves de caixa negra o grisa.

Les proves d'integració proven com diferents parts del sistema treballen quan les posem juntes. Poden incloure des de vàries classes a tot un subsistema. A diferència de les de sistema, no cal que totes les parts estiguin disponibles. Habitualment són proves de caixa grisa.

Les proves funcionals proven una funcionalitat concreta (cas d'ús) del sistema. Per exemple, una prova de que podem afegir correctament una nau industrial. Aquestes proves sovint són de caixa grisa o blanca.

Les proves unitàries s'enfoquen en una petita peça de codi aïllada de la resta de l'aplicació. Miren de minimitzar dependències amb altres parts i de provar tots els possibles camins (ex. totes les branques de cada condicional). Aquest tipus de proves és generalment automatitzable i, solen considerar-se de caixa blanca.

CONCLUSIONS:
Com a conclusió penso que és una forma molt interessant de programar i sense dubte la posaré en marxa algún dia, sembla un codi tan segur i quan ho domines es pot fer tant ràpid...