La prova de programes

Objectiu: eliminació del errors (s'estima que s'ocupa el 50% del recursos en aquesta tasca).

Convé començar per diferenciar les tècniques de verificació formal (*proving*) de les de prova (*testing*). Amb les primeres es demostra formalment que el programa satisfà l'especificació. Amb les segones es tracta de fer-ne proves executant el programa amb diferents conjunts de dades, el més completes que es pugui, per comprovar la correctesa del programa.

Crítiques d'aquests mètodes: Les tècniques de verificació formal són complexes i, per tant, cal que les faci personal força qualificat, en conseqüència són d'aplicació cara i complicada.

A més de ser cara, la verificació formal només demostra que el programa satisfà l'especificació, i, per tant, només garantirà l'absència d'errors quan l'especificació ho faci. Dit d'una altra manera, cal que puguem garantir que l'especificació és prou completa i correcte. Malgrat això, en algun país d'Europa hi ha legislació que obliga a fer verificació formal dels programes més crítics (com els que gestionen els reactors de les centrals nuclear).

La prova del programes és el mètode més utilitzat a la indústria per a la depuració d'errors malgrat que pugui semblar, i potser sigui, teòricament menys fiable. Les raons són el menor cost i que poden adaptar-se amb més facilitat a les necessitats de cada cas concret. Vull dir que fàcilment es poden simplificar o fer més completes en funció de la importància que es doni a la fiabilitat del programa.

La pega d'aquestes tècniques de prova és que en realitat només garanteixen que el programa funciona correctament amb les dades i en les condicions en què s'han fet les proves, no pas l'absència total d'errors, tot i que s'intenta que les dades i condicions de funcionament siguin tan completes i pròximes a les reals d'explotació com sigui possible.

Altres objectius. Amb la prova de programes, a més de l'eliminació d'errors, també és vol comprovar que la funcionalitat del programa és completa (que no en manca) i que se satisfan els RNF (requisits no funcionals, que poden incloure condicions sobre el seu funcionament com l'eficiència, etc.)

Depuració d'errors. En aquesta fase del desenvolupament de programes no es tracta només de trobar els possibles errors, sinó també de trobar-ne la causa i d'eliminar-los, és a dir, de depurar-los.

Depurar també vol dir que quan es detecta un error s'ha de buscar en quina fase del cicle de vida es va originar, corregir el que convingui d'aquella fase i veure quines implicacions té a les fases següents, que si convé també caldrà corregir.

Del conjunt format per les dades que s'usen per fer les proves amb el programa (incloent les maniobres amb la perifèria d'entrada, com el ratolí) i pels resultats esperats (incloent els efectes que s'hagin de produir) se'n diu *joc de proves*.

Per a cada execució del programa amb un joc de proves s'ha d'avaluar els resultats produïts i comparar-los amb aquells que s'esperava. Caldrà tractar les diferències entre un resultat (el produït) i l'altre (l'esperat) com a un error.

Cal tenir present que cada vegada que es modifica el programa s'ha de començar el procés de proves de bell nou, des del començament, com si s'hagués fet un nou programa.

Atenció! L'execució de programes en temps real té una problemàtica específica, ja que en aquest cas es compliquen més les coses davant la dificultat de simular la realitat, especialment pel que fa a la coincidència d'esdeveniments.

Classificacions, es fan segons diferents criteris. El primer que podem considerar és segons qui és l'operador o subjecte de la prova:

- El programador
- El dissenyador
- L'analista
- El client o usuari

Observem que els dos primers coneixen prou bé els algorismes, i els dos darrers només coneixeran el comportament que se n'espera, la funcionalitat.

Segons la tècnica emprada:

- *Caixa blanca:* només factibles quan es coneixen els algorismes. Es tracta provocar l'execució, si més no un cop, de totes les sentències, això vol dir passar per totes les branques de les sentències alternatives (*if* i altres) i executar o no les iteracions.
- *Caixa negra:* són proves independents de l'estructura de l'algorisme, basades només en el comportament que se n'espera. Es tracta de establir els conjunts de dades que han de produir un comportament uniforme, això defineix una relació d'equivalència (un joc de proves equival a un altre quan fa que el programa es comporti de la mateixa manera) que permet establir una partició del conjunt en classes d'equivalència. Llavors s'ha de triar un element de cada una de les classes per fer les proves (conjunt que s'anomena *conjunt quocient*).

Entenent que el conjunt de jocs de proves complet el formarien totes les possibles combinacions de dades i maniobres de la perifèria, es diu que un conjunt de jocs de proves és *mínimament complet* en caixa blanca si aconsegueix l'execució de totes les sentències. En caixa negra, si és un conjunt quocient (que conté un representant de cada classe de comportament uniforme). Aquests conjunts de jocs mínimament complets no tenen perquè coincidir, és a dir, que els de caixa blanca poden ser diferents dels de caixa negra.

Segons la unitat de prova, la prova es pot fer:

- Sobre un *component* (en general els components seran rutines, operacions, funcions, etc. En l'orientació a l'objecte (OO) el component serà sempre una classe, mai es provarà un element o mòdul menor).
- Amb part dels components del programa durant el procés d'*integració*.
- Amb el programa o sistema complet per *validació* de la correctesa del producte per part del desenvolupador.

- Amb el programa o sistema complet per *acceptació* del producte acabat per part del client (o usuari).

La validació es farà, en principi, en l'entorn operatiu de desenvolupament amb els criteris del desenvolupador (que són l'especificació), mentre que l'acceptació es farà sempre a casa del client, en el seu entorn operatiu i amb el seu criteri (basat en allò que volia o necessitava, normalment seran proves *beta*). S'entén que són proves *alfa* aquelles que es fan amb l'assistència o proximitat del programador o del tècnics que han desenvolupat el programa, mentre que és diuen *beta* quan aquests tècnics no són a l'abast.

En tota prova cal definir:

- Unitat de prova (objecte de la prova: un component, la integració d'uns quants, etc.)
- Tècnica a emprar (caixa blanca o negra).
- Subjecte de la prova (usuari, analista...)
- Jocs de proves, que en conjunt han de ser mínimament complets, si més no.

Proves d'integració: consisteix en provar conjuntament components ja provats individualment. Això es pot fer amb un criteri ascendent, descendent o mixt.

Ascendent (*bottom-up*) vol dir començar pels components més baixos en l'estructura del programa, elements que no n'integren cap i que estan dissenyats per integrar-se en d'altres. S'avança pujant de nivell en l'estructura (ascendint) i formant cada cop estructures més complexes fins arribar a tenir el programa complet.

El criteri descendent (*top-down*) opera al revés. Comença pel component integrador de tot el programa al qual li anem incorporant altres components del nivell inferior (descendint) que conformen la seva estructura fins arribar a integrar-hi els més simples.

Els criteris mixts parteixen d'un component, que pot ser qualsevol, i usen dels dos criteris anteriors, ara un ara l'altre, segons convingui. Sempre s'acaba amb el programa o sistema complet.

Quan es treballa de manera ascendent, per a cada component (o conjunt integrat) convé fer un programa que permeti accedir a les seves operacions (o funcionalitats), és a dir, un programa que permeti fer les crides adequades per executar-les. Aquestes peces de programació s'anomenen *drivers*, són programes senzills i a vegades força rudimentaris especialitzats en poder usar les funcionalitats de la classe o conjunt de classes que s'està provant.

Quan es treballa de forma descendent, haurem de substituir els components que han de formar la classe provada per petites peces de programa que emulin d'alguna manera el seu comportament o simplement ens permetin saber que s'ha fet correctament la crida del component encara no integrat. Aquestes peces de programació s'anomenen *stubs*, són programes senzills i a vegades força rudimentaris que permeten saber que hem cridat correctament aquell component. Els *stubs* a vegades permeten introduir les respostes a mà, d'altres donen respostes fixes (valors constants), d'altres avaluen la precondició de la funcionalitat cridada, etc.

En l'OO es fa la integració seguint les herències, agregacions, associacions i dependències. Normalment per capes i dins de les capes es diferencia els controladors

(de domini o de vista) de les classes bàsiques. L'element més petit a provar (o component) és una classe, mai menys.

Tècniques de depuració: Normalment es basen en el seguiment de l'execució del programa, alguns llenguatges, com Eiffel, permeten, a més, fer ús de l'especificació.

Se sol partir del punt on s'ha produït el resultat erroni (no esperat) i tirar arrere en el sentit del flux d'execució del programa. També es pot intentar determinar els punts sospitosos.

Durant el procés de depuració cal anar amb molta cura per evitar la producció d'errors nous i recordar sempre de revisar totes les fases per detectar-ne la causa real.

Eines de depuració: normalment les aporta el compilador o entorn de desenvolupament amb què treballem, destaquen:

- Programes de *debugging,* a vegades són complexos o embolicats d'usar.
- Les eines CASE solen dur eines auxiliars de depuració que fins i tot abasten diferents fases del cicle de vida.
- Aprofitament de l'especificació formal.

Formulari: recordeu que en el segon lliurament heu de documentar cada executable de prova (moltes vegades serà un *driver*) segons el formulari normalitzat a l'assignatura. En aquest formulari es deixen en blanc els camps que no calgui omplir, però no s'eliminen. Recordeu també, que, a més, us demano les opcions de compilació que s'han usat en generar cada executable.

Al directori on hi ha l'executable hi haurà tres fitxers amb el mateix nom: el font (.java), l'executable (.class) i el formulari (.txt, .doc, .html, o htm) que documenta la prova.

Un programa de proves ha d'admetre dades noves sense haver de ser compilat de nou. És a dir, que les dades dels jocs de proves han d'estar fora del codi font del programa. Òbviament aquestes dades poden estar en fitxers fàcilment editables (per exemple en format de text pla), no és imprescindible que el programa de prova sigui interactiu.