PROJECTE PROGRAMACIÓ

TEMA 1.− INTRODUCCIÓ A LA ENGINYERIA DEL SOFTWARE

TEMA 1.− INTRODUCCIÓ A LA ENGINYERIA DEL SOFTWARE

Una perspectiva del hardware i el software

Al llarg dels anys 50 d'existencia del software podem identificar quatre eres diferents:

- Anys 50 i principi dels 60: primers ordinadors d'utilització general ja que fins aquell moment els sistemes estaven dissenyats per a a aplicacions especificauqes. Les aplicacions s'enfocaven al processament per lots (baych processing), es feien a mesura i quasi sense distribució.
- Anys 60−70: comencen els sistemes multi−usuari i de temps real. Primers sistemes de gestió de bases de dades i va néixer el concepte de producte de software. El soft es feia en software houses empreses dedicades estrictament a programar les aplicacions. El manteniment i desenvolupament tenia uns costos molt elevats i impedia la creació de noves aplicacions, i es va començar a parlar de la crisi del software.
- Finals del 70 a finals del 80: neix el microprocesador, l'ordinador personal, les estacions de treball, les xarxe i les arquitectures client−servdor. El nivell de distribució de paquets de software és de l'ordre de cents de mils i milions de còpies, s'ha començat a desenvolupar els primers sistemes distribuïts i les eines CASE com prime auxili a la crisi del software (CASE: dissenyar el sistema de la forma més clara, per trobar els errors més fàcilment).
- Etapa actual: apareixen nous paradigmes: l'orientació a objectes, els GUI (interfíces gràfiques d'usuari), la intel−ligència artifical i els sistemes experts concrets, les xarxes neuronals i els algorismes genètics. Apareix el processament paral−lel i el nivell d'integració dels circuits.

El software i l'engnyeria del software

Etapa inicial

Al principi es concebia l'activitat del desenvoupament del software com un art. Els mètodes formals eren gairebé inexistents i s'utilitzaven menys. El programador aprenia autònomament mitjançant prova i error, i el cost de desenvolupament estava concentrat en el hardware.

Etapa actual

Avui en dia és el software en comptes del hardware el que absorbeix la major part del cost. El que preocupa és per què no es troben els errors abans i com és que tenim tantes dificultats a controlar, gestionar i mesurar el procés de desenvolupament de software.

El software es desenvolupa en el sentit de disseny d'enginyeria i no en el sentit de fabricació clàssic. El soft no es desgasta com u producte físic i en canvi s'ha d'estar modificant constantment perquè s'ha d'adaptar a una realitat canviant. I per últim, el soft s'hauria d'acoblar amb components existents en comptes de dissenyar−se a mida cada vegada.

Per tot això la tendència actual és la de dir que el software s'ha d'emmarcar dins d'una disciplina d'enginyeria. L'enginyeria del software engloba tres elements principals:

- Els *mètodes* ens donen una tècnica per construir software. Permeten afrontar tasques tan diverses com

la planificació de projectes o el disseny d'estructures de dades o algorismes.
- Les *eines* subministren un suport automattzat o semi−automatitzat als mètodes. Així tenim el conjunt d'eines que sota diversos noms podem qualificar de CASE.
- Els *procediments* uneixen les eines i els mètodes, i permeten desenvolupar−los en el temps i amb la qualitat requerits.

L'enginyeria del software es compon d'una sèrie de passos pels quals utlitzrem els mètodes, les eines i els procediments descrits. Aquests passos s'anomenen habitualment paradigmes d'enginyeria del software.

Hi ha cinc paradigmes utilitzats habitualment: el cicle de vida clàssic, el prototipatge, el model d'espiral, les tècnques de 4 generació i el cicle de font de l'orientació a objectes.

El cilce de vida clàssic

Al llibre pàgina 22 tenim les set etapes d'aquest paradigma.

Aquest model presuposa que fins que no s'acaba una etapa no es pot iniciar la següent i queno hi ha marxa enrera. Aquests dos suposists no són realistes, almenys en la situació actul dels requeriments que s'estan fent a l'enginyeria del sftware. Es per això que es diu que el model clàssic ha entrat en crisi i s'han proposat models alternatius. Malgrat tot, aquest model és vàlid per a un curs de projecte de programació.

La primera etapa és la de les especificacions (anàlisi prèvia o anàlisi o enginyeria de sistemes). En ella es defineixen els grans trets del sistema que s'ha de desenvolupar en el marc d'un sistema més gran que és el que hem d'estudiar. Em de tenir en compte els recursos de què disposen, quines interfícies hi haurà amb altres elements com ara el hardware, base de dades i perosnes.

La segona etapa és la de l'analisis. S'amplien els requeriments anteriors pensant especialment en el software i l'analista ha de conèixer profundament el domini de l'aplicació com també la funcionalitat nencessària, el rendiment que s'espera obtenir i les interfíces.

A continuació es realitza el disseny. S'han de realitzar diversos passos en aquesta estapa: estructura de dades, arquitectura del software, detall de procediments i carcateització de les interfícies. El disseny tradueix els requeriments descrits en l'analisi a una representacióque servirà com a descripció del que s'haurà de fer en la propera etapa de codificació.

Un cop arribat a aquest punt es pot iniciar la programació o codificació. Haurem de traduir el disseny a codi procesable per l'ordinador i provarem cada un dels programes que anem desenvolupant separadament.

Les proves són la penúltima etapa. Es comprova que el conjunt de la programació executi correctament el que s'espera d'ella i que ho faci amb el rendiment desitjat.

La darrera etapa no s'acaba mai. És el manteniment. Durarà el temps que duri la vida de l'aplicació.

Prototipatge

Sovint l'usuari no és capaç de definir exactament les seves necessitats. Altres vegades l'analista no pot calcular exactament el rendiment de cert algorisme, del sistema operatiu o la manera millor de dissenyar una interfície per a l'usuari.

Hi ha tres alternatives:

· La primera consisteix en la construcció d'una maqueta d'una primera solució al problema. Una maqueta és la

construcció de les interfícies i els menus amb què l'usuari haurà d'interactuar.

· La segona es crear un petit sistema, normalment un subconjunt operatiu del sistema final, moltes vegades amb un llenguatge d'alt nivell.

· La tercera alternativa és usar un programa ja existent que fa part del que vol l'usuari i un cop revisat introduir en una segona etapa la funcionalitat adicional que vol l'usuari.

El model d'espiral

Al llibre de l'assignatura, pagina 24 tens un esquema

El model d'espiral de Boehm ha estat desenvolupat per agafar les millors aportacions del model clàssic i del de prototipatge i alhora introduir un nou element, l'anàlisi de risc, que no existia abans.

S'enten per planificació la determinació d'objectius, les alternatives i les restriccions. L'analisi de risc és l'anàlisi d'alternatives i la identificació i resolució dels riscos. Enginyeris és el desenvolupament del producte . Per últim, l'avalació del client és la valoració dels resultats de l'enginyeria.

Es van duent a terme quatre tasques cíclicament, i de mica en mica ens anem apropant al producte final. A cada cicle obtenim un nou producte per mitj`del prototipatge o segons el cilce de vida clàssic. El més important és a cada pas anar avaluant els riscos de seguir amb elmodel obtingut i eplantejar quines són les millors alternatives per seguir amb el projecte.

En definitiva, el que es pretén és que si es detecten els factors de risc i s'avaluen les alternatives abans de cada etapa es pot garantir l'èxit del projecte en temps, recursos i satisfacció del client.

Tècniques de 4ª generació

Al llibre de l'assignatura, pagina 26 tens un esquema

Aquest model es basa en la utilització extensa de llenguatge de quarta generació. Aquests llenguatges són de molt alt nivell, en gran part no procedimentals i amb un acoblament molt fort amb les bases de dades relacionals.

Model de font per a l'orientació a objectes

En l'orientació a objectes els cicles de vida que es proposen són radicalment diferents als anteriors, per dues raons. La primera raó és l'existència de classes que es reutilitzen en diferents aplicacions. La segona raó és que el cicle de vida d'una aplicació concreta està influït pels conceptes de simulació de processos i de desenvolupament iteratiu.

Com l'orientació a objectes se centra en la construcció de sistemes a partir de components senzills (classes) es necessari introduir el concepte de cicle de vida d'una classe o d'un cluster de classes (conjunt de classes molt inter−relacionades que resolen una problemàtica específica.

En aquest cicle de vida cada classe o cluster passa per tres etapes: especificació, disseny/implementació i validació/generalització. Tampoc no estan sincronitzats els cicles de vida de cada cluster ja que tenen vida independent i seran reutilitzas, possiblement, en projectes diferents i probablement sotmesos a un desenvolupament futur a través de generalitzacions i especialitzacions.

Tamany de projecte

Categoria	Nombre tècnics	Durada	Grandaria	Exemple
Trivial	1	1 a 4 mesos	500 línies	Aplicació personal
Petit	1	1 a 6 mesos	1 a 2K	Projectes d'estudiants
Mitjà	2 a 5	1 a 2 anys	5 a 50K	Projecte informàtics, aplicacions actuals.
Gran	5 a 20	2 a 3 anys	50 a 100K	Paquets de base de dades, sistemes gràfics, etc
Molt gran	100 a 1000	4 a 5 anys	1M	Sistemes de temps real, de telecomunicacions, etc.
Enorme	2000 a 5000	5 a 10 anys	1M a 10M	Sistemes de trànsit aeri o de defensa.

Desenvolupar software: disseny i construcció de programes

L'objectiu del dissenyador és produir un model o la representació d'una unitat que serà construïda posteriorment. El procés pel qual es desenvolupa el model combina intuïció i criteris basats en l'experiència construint entitas similars, un conjunt de principis i/o heurístiques que el guien en el camí pel qual evoluciona el model, un conjunt de criteris per avaluar−ne la qualitat i un procés iteratiu que porta finalment a la representació del disseny.

El disseny és el nucli de l'enginyeria del software i s'aplica independentment del paradigma que s'utilitzi. El disseny és la primera de les tres activitats −disseny, codificació, proves− que són necessàries per construir i verificar software.

Quan parlem de disseny en aquest context té un significat bastant ampli i en ell es realitzen tres funcions. Anàlisi del problema, abstracció de les entitats que s'han de modelar i disseny de la seva concrecio informática.

El procés de disseny des d'un punt de vista de direcció de projectes consta de dos passos: disseny preliminar, on es creen el disseny de les dades i l'arquitectura, i el disseny detallat, on es refina l'arquitectura fins arribbar a un disseny detallat de les ddes, unes representacions algorísmiques del software i, en tota aplicació moderna, elmolt important disseny d'interfície.

Hi ha una sèrie de principis desenvolupats en aquests anys que són comuns a tots els mètodes. Aquests són els següents:

- Abstracció a cada nivell de les entitats rellevants.
- Refinament successiu per etapes en un nivell més detallat
- Modularitat en descompondre un problema complex
- Arquitectura del software: jerarquia de control o estructura del programa, estructura de dades.
- Ocultació d'informació per esmorteir l'impacte dels canvis.

Evolució del programa. El manteniment

El manteniment del software consumeix el 70% de tots els esforços i les despeses d'una organització de software.

Hi ha diferents tipus de manteniment:

- *Correctiu*: corregir errors no trobats durant el desenvolupament inicial del software.
- *Adaptatiu*: aquell que sorgeix com a necessitat d'adaptar el software per canvis de l'entorn.
- *Perfectiu*: afegir funcionalitats, modificacions a funcions actuals , en general, millores suggerides pels usuaris.
- *Preventiu*: que es caracteritza com a enginyeria inversa. Això es fa per facilitar o millorar el manteniment futur o la fiabilitat.

Podem dir que hi ha dos tipus de manteniment, l'estruturat i el no estructurat. Si l'única documentació que tenim és el programa font, el manteniment esdevé una tasca dificil d'avaluar. Qüestions com l'estructura del programa, dades globals, interfícies, restriccions de rendiment no són conegudes i freqüentment es malinterpreten. Aquest tipus de manteniment s'anomena no estructurat.

En canvi si existeix una informació completa el pla que es desenvolupa és el següent: avaluar el disseny i els costos d'incorporació de la modificació, preparar un pla de desenvolupament, modificar el disseny, recodificar i provar. Realitzat d'aquesta manera, es diu que el manteniment és estructurat.

El cost de manteniment ha anat creixent del 35−40% als anys 70 al 80% als anys 90.

El manteniment arriba a reduir de 40 a 1 la productivitat informàtica. És a dir el cost de mantenir una línia de codificació és 40 vegades superior al d'escriure una línia nova.

Reusabilitat dels components

La reusabilitat és una característica important del software de qualitat. És a dir, un component s'hauria de dissenyar i implementar de forma que pugui ser reutilitzat en futures aplicacions o programes.

Actualment un component reutilitzable és una encapsulació de codi més dades en un únic paquet anomenat classe o objecte que permet a l'informàtic crear noves aplicacions a partir de parts reusables

Però aquest guany té un preu. En primer lloc, el reciclatge d'un programador entrenat en programació estructurada al nou paradigma d'orientació a objectes dura un mínim d'un any. En segon lloc, el cost de desenvolupament de les primeres aplicacions orientades a objectes es considerablement superior al de les clàssiques per l'esforç de creació molt acurada de la llibreria de classes.

Malgrat tot, l'opinió unànime és que els guanys que s'obtenen a partir d'aquest punt compensen sobradament l'esforç invertit, ja que s'arriba a nivells de reutilització per sobre del 80% en comptes del 5−10% en instal−lacions clàssiques.

TEMA 2.− INTRODUCCIÓ A L'ORIENTACIÓ A OBJECTES

Perspectiva històrica

L'orientació a objectes surgeix com reacció als alts costos de manteniment. Els primers conceptes provenen del llenguatge Simula67, aquests volien construir models de sistemes físics complexos que podien contenir milers de components. Els conceptes que incorporava van tenir una influència significativa en els llenguatges orientats a objectes posterior, com Smalltalk i Eiffel.

Introducció a la filosofia i la terminologia

L'OO consisteix a gafar els components normals de qualsevol sistema informàtic (dades + procediments) i reduir l'èmfasi en els procesos i enforitr l'encapsulació en un mòdul autònom de dades i funcions juntes, amb una especificació clara i precisa de la interfície.

L'anàlisi i el disseny d'alt nivell es realitza no solament en funció d'aquest objecte, sinó també en la forma com els objectes interactuen entre ells a través de missatges que es passen informació, invocant que els objectes executin un procediment o que respongui amb detalls del seu estat.

Els detalls de l'implementació es postpont fins a molt més tard en el procés d desenvolupament. Els detalls d'implementació són privats a l'objecte.

El paradigma de l'OO es basa en la construcció d'un model informàtic d'un sistema real, es a dir, intenta modelar el mon real.

Una segona característica dels entorns de desnvolupament orientats a objectes és la de la reusabilitat.

Tècinicament, l'orientació a objectes es basa en tres conceptes:

- ***Encapsulació*** + ocultació de la informació: ocultació de dades i procediments dins d'un mòdul (objecte)
- ***Abstracció, classificació*** i tipus abstractes de dades: dissenyar en funció del tipus de dades i/o objectes. Un tipus definit per l'usuari pot ser qualsevol cosa (una taula, un comptador). El concepte de classificació és la idea d'agrupar idees de software en classes de coses. És a dir, que no solament treballem amb un conjunt de codi més dades que representa una taula, sinó amb un conjunt genèric o col−lecció de taules.

L'abstracció es fa també en concpetes d'alt nivell com podria ser la gestió d'una llista. Sabem que podem tenir llistes de qualsevol objecte, es a dir, que el terme llista és molt genèric, i per tant és lògic pensar a fer una abstracció de llista i crear una llista genèrica que pugui ser instanciada. Aquesta llista genèrica s'anomena classe *genèrica, contenidora o parametritzada*. És a dir, és una classe que definim completament però on un o més dels seua atributs estan indefinits o parametritzats a l'espera de l'instanciació o de la definició d'una classe concreta.

Un concepte proper és el de classe *abstracta, diferida o no instanciable*. A una classe abstracta li falta algun element, atribut o servei per definir i per tant, no es pot instanciar. Hi ha diverses raons per aquesta situació: s'ha creat una superclasse per reflectir el que de comú tenen unes subclasses, es vol forçar que tota subclasse tingui cert atribut o mètode definit, etc.

- ***Polimorfisme*** a través ***d'herència***: l'herencia és una analogia de software de l'herencia taxonòmica en el sentit biologic. Però si s'hereda d'un sol pare tenim *herència simple* i si s'hereda de més d'un pare tenim *herència multiple*. L'herència ofereix reusabilitat, extensibiltat i cost baix de manteniment. També ofereix un mètode per relacionar les classes d'una forma que sigui semànticament sensata. Es pot veure com una forma de compartir codi.

El que no es pot fer és anul−lar atributs o serveis d'un pare, ano ser que artificialment es buidin de contingut. En l'herència s'hereda tot: atributs, serveix i relacions.

Com que no tots els llenguatge ofereixen herència múltiple existeix un mecanisme per simular−la anomenat *delegació*. Llavors se le afegeix un atribut del tipus del segon pare i es crea un servei que tenia aquest segon pare. La implementació d'aquests serveis consisteix exclusivament a replicar la crida però enviant−la a l'atribut creat internament. És a dir que deleguem a l'objecte inclós totes les crides que ens fan.

El ***polimorfisme*** es pot definir com l'habilitat de les abstraccions de compartir propietats. També podem definir−lo com el fet que un nom pot denotar objectes de classes diferents però relacionades per una classe base comuna. Un concpete relacionat amb aquest és el de *sobre−càrrega* d'operadors. Això significa que podem redefinir el significat dels operadors bàsics del llenguatge, per tal que actuï de forma diferent segons el tipus d'objectes que intervenen en l'operació.

A continuació resumim els termes mes bàsics d'OO i el seu significat:

- *Classe*: encapsulació de dades i procediments.
- *Objecte*: instanciació d'una classe.
- *Atribut*: dad interna d'un objecte.

- *Servei/Mètode*: procediment accessible d'un objecte.
- *Missatge*: forma de comunicació entre dos objectes per demanar l'execució d'algun servei. Similar a una crida a funció o procediment.

Notació gràfica

La notació de la figura és la creada per Coad/Yourdon.

La classe_1 representa una classe abstracta (no instanciable). Pot tenir o no serveis definits. Internament te tres atributs i te una relació d'***inclusió (agregació)*** amb la classe_3. Això vol dir que la classe_1 inclou un objecte de la classe_3. Aquesta inclussió es pot implementar almenys de dues formes: tenir un atribut del tipus Classe_3 o mantenir un punter a un objecte autònom de tipus classe_3. La cradinalitat de la relació és [1..N]. Naturalment si la cardinalitat no és [1/0..1], sinó que és superior, ens obligarà a implementar la relació com una col−lecció (vector, llista, etc.) d'objectes de tipus classe_3.

La classe_2 te dos atributs (A, B) i dos serveis (X, Y) i també és abstracta com les anterior. Però aquesta té una subclasse classe_4 que, en canvi és concreta, és a dir, instanciable. Entre la classe 2 i la classe_4 hi ha una ***relació d'herència***.

La classe_4 te tres atributs (A, B', C). L'atribut A el rep del seu pare. El B també, però el redefineix de forma dierent. El C és propi.

La classe_3 accedeix a la classe_4 enviant−li un missatge perquè executi alguns dels seus serveis (***conexió de missatge***). Aquesta relació és puntual i variable en el temps.

Entre la classe_1 i la classe_2 hi ha, en canvi, una relació anomenada ***connexió d'instància***. Aquesta relació intenta reflectir un lligam fort i permanent entre dues classes.

TEMA 3.− ESPECIFICACIÓ DEL PROGRAMA

Introducció

En aquesta etapa es preparen (idealment ho fa el client) unes especifcacions de requeriments del software, un pla de projecte de software i un estudi de viabilitat econòmica.

La tasca d'anàlisi de requeriments és un procés de descobriment, refinament, modelatge i especificació.

L'anàlisi de requeriments és una atsca d'enginyeria del software que permet a l'enginyer especifcar la funcionalitat del software, el rendiment esperat, les interfícies amb altres sistemes i les restriccions de disseny que cal respectar.

Les tasques que s'han de realitzar es divideixen en cinc àrees:

- Identificació del problema.
- Avaluació i síntesi.
- Modelatge.
- Especificació.
- Revisió.

L'avaluació del problema i sintetitzar una solució és l'àrea següent a tractar. S'ha d'evaluar el flux d'informació, definir i elaborar totes les funcions del software, entendre el comportament que ha d'existir en el context dels events que puguin succeir, establir les interfícies del sistema i descobrir o explicitar les

restriccions que hi hagi.

Durant l'avaluació i síntesi d'una solució l'analista se centra principalment en el que i no en el com. ¿Quines dades produeix i consumeix el sistema, quines interfícies es defineixen, quines funcions s'han de realitzar i quines restriccions s'aplicaran?.

Un cop s'han establert les funcions bàsiques, els rendiments, etc. s'han d'estbalir uns criteris de validació que puguin demostrar l'èxit de la implementació del software. Aquests criteris seran la base per a les activitats de prova en les etapes posteriors del procés d'enginyeria del software. També es pot crear un manual de l'usuari preliminar quan no s'hagi creat un prototipus.

Formes d'especificació

A mesura que es vagi realitzant l'analisi s'hauran d'anar construint les especificacions. S'utilitzaran diverses notacions:

- diagrames de flux.
- Diagrames jeràrquics i de classes.
- Text lliure.

El text lliure es la font més important de generació dde problemes de comunicació, bona part del s 70% d'errors d'un programa.

Els errors mes freqüents segons Meyer son els següents:

- Soroll: informació innecessària o redundant.
- Silenci: manca d'nformàció.
- Sobreespecificació: detallar més del que es necesari.
- Contradicció: entre parts de l'especificació.
- Ambigüitat: per flta de precisió, per exemple amb l'ús de sinònims i homònins.
- Referencies endavant: utilitzar termes no definits previament.
- Remordiment: fer precisions o introduir restriccions sobre declaracions anteriors.

Especificació: principis, representació, requeriments

L'especificació del programa es crea al final de la tasca d'anàlisi.

Balzer i Goldman han proposat 8 principis per desenvolupar una bona espeficació:

- Separar funcionalitat d'implementació
- Es necessita un llenguatge d'especificació per a sistemes orientats als procesos (diagrames d'estat de transició).
- Una especificació ha d'englobar el sistema del queal el software és un component.
- Una especificació ha d'englobar l'entorn en què operarà.
- Una especificació de sistemes ha de ser un model cognitiu.
- Una especificació ha de ser operativa.
- L'especificació d'un sistema ha de ser tolerant a incompletiud i augmentable.
- Una especificació ha de ser localitzada i dèbilment acoblada.

A aquests principis hi afegim les quatre regles generals següents que caldrà seguir en tota especificació i, en general, en qualsevol tasca de documentació:

- El format de representació i el contingut han de ser rellevants al problema
- La informació continguda ha d'estar anidada.
- Els diagrames i altres formes de notació han de ser restringits en nombre i consistents en la utilització.
- Les representacions han de ser revisables.

Principis de l'anàlisi.

Tots els mètodes d'anàlisi tenen en comú uns principis bàsics:

- El domini de la informació d'un problema ha de ser representat i entès.
- S'han de desenvolupar models que descriguin la informació, la funció i el comportament del sistema.
- Els models i el problema s'han de partir de forma que mostrin el detall d'una forma jeràrquica.
- El procés d'analisi ha d'anar de l'essència de la informació a la implementació del detall: anàlisi descendent.

Particionar ens permet reduir la complexitat. L'anàlisi descendent és necessària per la inabordabilitat d'un problema complex com una unitat. En anar descomponent el problema i analitzar en nivells més profunds a cada etapa se simplifica l'anàlisi global.

Modelatge

Es creen models per entendre millor l'entitat que s'ha de construir. En no ser el software quelcom físic, el que s'ha de modelar és la informació que transforma el software, les funcions que ho permeten i el comportament que existeix quan es fa la transformació.

Els models que hem de construir han de centrar−se en allò que fa el sistema, no com ho fa. Normalment el model pren la forma d'una notació gràfica que representa la informació, el procés,el comportament, etc. usant icones reconeixibles. També s'utiltza informació textual en llenguatge natural o en algun de més epscialitzat o formal.

Per al domini de la informació s'utilitzaran els diagrames de flux de dades (DFD) per representar el fluxi la transformació de les dades, els diagrames de classe per representar el contingut i les connexions entre classes per represnetar l'estrutura.

El domini de la informació

Introducció

La informació es transforma (passa d'un estat a un altre) perquè hi ha algun esdeveniment que ho provoca. Aquests esdeveniments s'anomenen events i representen un aspecte del sistema de cotrol de l'aplicació. Una forma de representar els events és com una dada booleana: passa o no passa tal event.

El domini de la informació te tres punts de vista:

- Contingut.
- Estructura.
- Flux d'informació.

El contingut

El contingut de la informació representa les dades individuals i les agrupacions en entitats de nivell superior que formaran les classes bàsiques de la nostra aplicació.

És possible que reconeguem una agrupació de dades com una entitat que pugui formar part d'una altra de nivell superior. Aquest tipus de relació es podrà descriure a través de l'estructura, com veurem en el proper apartat, i s'anomena relació d'inclusió o agragació.

Tant les dades individuals com les agrupacions s'han de descriure en un diccionari de dades o repositori. Un repositori o plantilla de definicó de classe i objecte la podem trobar al llibre pàgina 57, al igual que la plantilla de definició d'atributs a la pàgina 58.

L'estructura

L'estructura de la informació representa les relacions entre les classes de la ostra aplicació. Les relacions que ens interessen en aquesta etapa són les d'inclusió o agragació i les de connexió d'instància.

Les relacions d'inclusió s'utilitzen per descriure la situació en què una classe forma part d'una o més classes.

Les connexions d'instancia s'utilitzen per descriure una relació especifca i duradora entre dues classes, molt més enllà d'una col−laboració puntual.

El flux

El flux d'informació representa la manera com dades i control canvien a mesura que es mouen a través del sistema.

El sistema interactua amb entitats externes d'on arriba tota la informació i on van a parar totes les sortides. Entitats externes poden ser l'usuari, altres programes, el istema operatiu, etc. Es a dir, qualsevol entitat que no està sota el nostre control i que ens subministra informació o a qui hem de proveir d'informació.

Les tranformacions que s'apliquen són funcions que el programa ha de realitzar. Es correpondran directament amb entrades del diagrama jeràrquic de funcionalitat i/o amb algun servei de les classes que existeixen en l'aplicació. Aquestes relacions s'hauran d'identificar documentalment en l'especificació.

Una avantatge dels diagrames de flux de dades és que són facilment seguibles per l'usuari i, per tant, una bona eina de comunicació.

La informació que entra i surt d'un procés provindrà sempre d'algun objecte. També pot ser un objecte complet que és passat al procés o que és instanciat en ell mateix. En qualsevol cas s'accedirà a ell sempre a través dels seux serveis.

El tractament que s'ha de realitzar en un procés pot ser senzill o molt complex. En aquest últim cas pot realitzar−se el que s'anomena una explosió i descompondre en un segon diagrama el procés en diversos subprocessos, cada un d'ells expressat amb la mateixa notació gràfica.

La funcionalitat

Els problemes habitualment són massa complexos per entendre'ls com una untitat. Per aquesta raó tendim a partir−los en components que podem entendre més fàcilment.

Bàsicament, particionar consisteix a descompondre un problema en les seves parts constituents. Hi ha dues formes de dur a terme aquesta expansió:

- Verticalment, aprofundint en el detall.
- Horitzontalment, descomponent funcionalment el sistema.

En analitzar l'enunciat de l'aplciació haurem d'obtenir, entre altres coses, la funcionalitat que s'espera d'ella. Quan s'expliqui la metodologia, mostrarem una forma d'anàlisi de l'enunciat que ajuda a extreure del propi enunicat part d'aquesta funcionalitat.

Els diagrames jeràrquics, en canvi, permeten l'anàlisi d'aquesta funcionalitat per àrees i aplicant una descomposició descendent.

Vistes essencials i d'implementació

Una visió essencial presenta les funcions que s'han d'aconseguir (l'essencia) i la informació que s'ha de processar sense detalls d'implementació. Per cada funció del diagrama de l'apartat anterior i per les bombolles els DFD desvcriurem què fa la funció sense preocupar−nos de com s'ha de fer o dur a terme.

El model de classes descriurà el significat de les dades, els seus valors possibles, rangs, etc. sense indicar l'estructura física o el tipus d'implementació.

Aquestes descripcions hauran de ser un text lliure amb tots els incovenients que això comporta. A la figura 3.6 de la pagina 63 tenim l'esquema de la plantilla de definició de serveis.

La vista d'implementació dels requeriments de software presenta la manifestació al món real de les funcions de processament i de les estructures de la informació.

També s'haurà de descriure les restriccions dels elements que hi intervinguin.

Requeriments no funcionals

El model descrit fins ara pretén reflectir els requeriments funcionals que haurà de complir l'aplicació. Aquests bàsicament es redueixen a quines sortides volem obtenir de les entrades al sistema. Hi ha una serie d'altres requeriments que el sistema ha decomplir, anomenats no funcionals, i que fan referència a qüestions alienes a les fucions o procesos de l'apliació. La figura 3.7 pàgina 65 del llibre de l'assignatura, fa una relació dels més comuns que s'han de preveure.

Els requeriments no funcionals han de ser mesurables i, per tant, han d'estar quantificats.

La descripció dels requeriments no funcinals es farà en llenguatge natural, excepte en interfícies amb altres sistemes, en què es podran utilitzar descripcions formals.

L'especificació

L'especificació és el document final d'aquesta etapa i haurà de recollir tota la documentació generada en cda una de les tasques descrites en aquest capítol. A continuació es decriu una versió d'aquests per a projectes orientats a objectes:

- Introducció
- Referència del sistema (dels requeriments no funcionals)
- Descripció global. (text lliure)
- Restriccions del projecte de software (dels requeriments no funcionals)
- Descripció de la informació
- Contingut i estructura (model del domini i repositori)
- Flux (DFD i plantilles processos/serveix)
- Descripció de la nterfície del sistema (dels requeriments no funcionals)
- Desripció funcinal

- Partició funcional (diagrames jeràrquics)
- Descripció funcional (plantilles processos/serveis)
- Criteris de validació
- Límits de rendiment (dels requeriments no funcionals)
- Classes de proves
- Resposta esperada del software (dels requeriments no funcionals)
- Consideracions especials (dels requeriments no funcionals)
- Requeriments no funcionals (els no descrits anteriorment)
- Manual preliminar de l'usuari.

TEMA 4.− LA METODOLOGIA: ESPECIFICACIÓ; ANÀLISI; DISSENY

Especificació

Identificació del problema

En primer lloc haurem d'estudiar les especificacions que ens hagi donat el client, amb els seus objectius i les seves restriccions.

Es essencial revisar sistemes semblants que existeixin al mercat. Si ens han encarregat la construcció d'un nou sistema pot ser per quatre raons:

- no n'hi ha cap al mercat.
- no n'hi ha cap que faci el que vol l'usuari.
- n'hi ha però no estan disponibles per a l'entorn físic de que disposem.
- no es coneixen els que existeixen.

Qualsevol de les tres últimes raons justifica invertir temps i esforç en la recerca d'allò que existeix i en la discussió amb l'usuari dels avantatges i inconvenients que hi veu. En tot sistema, hi ha dues classes de funcionalitst: les d'usuari i les de sistema. Aquestes últimes són funcionalitats que no et demana explícitament o implícita l'usuari però que és necessari que el sistema tingui, i que coincideixen amb alguns requeriments no funcionals. Funcionalitats d'aquest tipus poden ser sistemes de còpia de seguretat automàtica, recuperació de còpies de seguretat, sistemes de segureat en l'accés, etc.

Pel que fa a les d'usuari, també podem distingir dues classes de fncionalitats explícites i implícites. Les explícites seran exposades pel mateix usuari. Segurament són les que més li preocupen. però també n'hi ha una sèrie d'altres que considera tan evidents que formen part del domini que creu innecesari explicitar o que fins i tot, ni pensar a presentar.

Un cop hàgim acabat aquesta etapa de recerca convindrà crear un esborrany textual de les especificacions o reescriure les que ens hagués donat el client.

L'anàlisi del domini

El mètode per trobar les classes i els objectes d'un domini consisteix, en primer lloc, a analitzar les especificacions, creades en l'etapa anterior, i extreure'n les frases nominals. Per obtenir una primera llista:

- Subratlleu totes les frases nominals.
- Canvieu tots els plurals per singulars.
- Si s'usa més d'una paraula per concepte (sinònims), unifiqueu−les.

Aquesta primera llista s'ha d'ampliar amb models de:

- Estructures (generalitzacions, compoents).
- Sistemes amb els quals interaccionem.
- Objectes físics.
- Coses o esdeveniments que cal recordar.
- Rols participants (usuaris i externs).
- Procediments operacionals (nom, nivell d'autorització, descripció, etc.)
- Ubicacions.
- Unitats organitzatives.
- Entitats conceptuals que siguin una abstracció cohesionada.
- Interfícies conegudes al món exterior o al sistema operatiu o en latres programes.

Aquestes classes són només candidates; algunes desapareixaran i n'hi ha haurà de noves. Per fer un primer filtratge, intenteu classificar−les dividint−les en òbvies, il−lògiques i dubtoses.

Un cop s'ha fet aquesta classificació apliqueu les directrius següents:

- Dubte dels adjectius, moltes vegades només representen un atribut de la classe;
- Dubte de les frases en passiva o de les que el subjecte no formi part del sistema;
- Modeleu categories de classes sense preocupar−vos inicialment del fet que siguin abstractes o no.
- Si no entene alguna part del sistema tracteu−lo com un subsistema tancat inicialment.
- Identifiqueu classes que faltin normalment perquè són poc imortants o perquè no eren explícites a la especificació.

Els criteris per confirmar la validessa de le calsses són els següents:

- Necessitat de recordar quelcom dels objectes.
- Comportament necessari a través de serveix.
- Atributs múltiples.
- Més d'un objecte per classe usualment.
- Atributs sempre aplicables.
- Serveis sempre aplicables.
- Requeriments obligats pel domini.

Si d'una classe només hi ha un objecte mireu si n'hi ha d'altres amb atributs i serveis similars. Si hi són probablement els pugueu considerar la creació d'una jerarquia de classes (herència) unificant les dues.

Si els atributs i/o serveis d'una classe no són sempre aplicables llavors haurem de crear una classe de nivell superior amb la part sempre aplicable i afegir una especialització amb la part més específica.

Definir atributs

Els atributs i els serveis són coneixement que manté una classe. D'altra banda, els atributs seran accessibles exclusivament a través de serveis; per tant, des del punt de vista (d'implementació) de les altres classes, no hi ha diferència entre serveis i atributs. Però en l'analisi i el disseny si que serà necessari distingir−los.

A l'especificació ens interessa concentrar−nos especialment en els atribts perquè és la informació del nostre domini i serà manipulada en el nostre sistema.

Haver identificat una classe significa que ha de tenir responsabilitats; per tant, hem d'analitzar una una i intentar assignar−li els atributs que li corresponen. Moltes vegades, en descriur una classe ens estenem amb carcaterístiques de les seves responsabilitats i, per tant, és una bona directriu llegir−nos−les per intentar identificar més atributs o serveis.

S'han de mantenir sempre els criteris següents en assignar responsabilitats, siguin atributs o serveis a una classe:

- Manté operació amb informació relcionada. La manipulació d'informació s'ha de intentar mantenir en el mateix objecte que conté la informació.
- Manté informació d'una cosa en un sol lloc. Si més d'un objecte necessita mateixa informació: creeu un objecte nou com a repositori, o reassigneu responsbilitats, o col−lapseu n objecte en un de sol.

S'han d'identificar els següents casos especials en atributs:

- Comproveu cada atrbuts per casos no aplicabes: genralitzeu el que és bàsic. Si hi ha atributs que no són aplicables en totes les instàncies d'una classe podeu crear una superclasse amb menys informació però sempre aplicable. Després es pot crear una subclasse de l'anterior amb la informació addicional.
- Comproveu cada classe i objecte amb un sol atribut: moltes vegades és un atribut d'una altra classe que està mal assignat.
- Comproveu cada atribut amb valors repetitius: a vegades un atribut repetitiu indica la manca d'una classe que precisament ha de mantenir aquest atribut i que tindrà una connexió d'instància amb la que stem estudiant.
- Eviteu posar resultats derivats (edat, etc.). És una decisió que es pot posposar fins al disseny valorant temps versus memòria.

Identificar estructures: agregació, connexió d'instància

En primer lloc analitzarem les relacions d'**inclusió** o agregació. A vegades, la relació d'inclusió es diu que es una relació is−part−of. Per trobar−les podem fixar−nos en:

- Acoblaments − parts (un cotxe i les seves peces).
- Continen − continguts (un avió i la tripulació)
- Col.leccions − membres (un departament i els seus empleats).

En analitzar un objecte com una unitat s'ha de considerar pels seus components:

- Classes compostes usualment delegant responsabilitat.
- Les clases continents no.
- Està en el domini? Potser encara que la nclusió sigui raonable no és problema del domini tractar−la.
- És responsabilitat del sistema?
- Representa més que tan sol un valor d'estat? Si no, afegiu un atribut a la classe continent.
- És una abstracció útil en el domini?

S'ha de tenir en compte que entre dues classes pot haver−hi més d'una relació d'inclusió. Per exemple, una escola te alumnes, però també te grups d'activitats i aquests tenen alumnes.

Les relacions d'inclusió s'ha de qualificar amb la cardinalitat de la relació. Aquesta cardinalitat és semblant a la que s'utilitza en el model entitat−relació. És a dir, que hem d'indicar a cada classe continent quantes instàncies de la continguda pot tenir i si és obligatoria la inclusió.

Les relacions següents que hem d'analitzar són les connexions d'**instancia**. Aquesta relació entre objectes existeix quan una classe necessita d'unes altres per dur a terme les seves responsabilitats. Per trobar−les podem fixar−nos en:

- Que sap aquesta calsse?
- Qui necessita el que sap?

- Analitzeu cada parell de classes i considereu si: hi ha una relació en el domini?; i si podem sempre accedir d'una classe a l'altra.

De la mateixa manera que en les relacions d'inclusió, haurem de aulificar la relació amb la cardinalitat que hi ha entre les classes. S'utilitzarà la mateixa notació per les dues relacions.

La funcionalitat

El flux

Els diagrames de flux de dades són una de les formes de visualitzar la funcionalitat d'un sistema. Com s'ha dit cada bombolla s'ha de correspondre amb un servei o un objecte controlador. Si és un controlador sgurament s'explosionarà en més d'un subprocés on cada bombolla serà un servei del controlador. El que hem de fer, per tant, per construir els DFD, és trobar totes les funcions i/o reponsabilitats del sistema i assignar−les a algna classe.

Per identificar les funcionalitats el que s'ha de fer és:

- Buscar al document de les especificacions del requeriments les frases verbals.
- També a tot arreu on s'esmenti informació.
- També s'ha de buscar tota informació que algun objecte del sistema ha de mantenir i manipular.
- Tal com hem dit a la identificació d'atributs, els serveis són una responsabilitat de la classe i tota classe ha de tenir serveis. Per tant, reviseu cada classe i la seva sentència de propòsit. El mateix nom de la classe implica un rol i aquest alguna responsabilitat.

La descomposició jeràrquica

Tot sistema informàtic pot considerar−se de diferents punts de vista. Cada punt de vista veurà una part de la funcionalitat del sistema.

Els diferents punts de vista poden ser els dels usuaris (i amb més d'un punt de vista segons el tipus d'usuari), els que provenen dels requeriments no funcionals, els de manteniment del domini, els de les funcions de sistema, el dels altres sistemes amb qui s'ha d'interactuar o els de les altres activitats funcionals que ha de realitzar el sistema.

El primer pas consisteix a identificar el màxim depunts de vista. No tots els punts de vista seran vàlids. Uns criteris per seleccionsr els vàlids són:

- Els punts de vista no s'han de solapar. És a dir, no hi a d'haver funcions que se solapin entre puts de vista i cada funció s'ha de realitzar en un punt de vista.
- La font i el destí de tota la informació del sistema ha de ser un punt de vista.
- Les característiques no funcionals del istema (cost, memòria, seguretat, etc.) poden considerar−se com punts de vista separats.
- Cada punt de vista ha de realitzar algun processament d'informació.

Per un mateix sistema pot haver−hi diferents punts e vista segons l'enginyer que el dissenyi.

Els punts de vista funcionals es divideixen en dues cateories: límits i de deefinició. Els de límits representen les fots i els destins últims de tota la informació del sistema. Són punts de vista externs al sistema però que afecten o són afectats pel sisstema. Els de definició representen el processament localitzat de la informació i són part del sistema que es model. S'utilitzen per definir funcionalment el sistema.

Anàlisi

Definir serveis, identificar connexions de missatges

Tota funcionalitat que hem trobat a l'especifiació de programa s'ha d'assignar a algun servei de classe, és a dir, se n'ha de responsabilitzar alguna classe. En l'etapa d'anlaisi nmés hem d'assignar els sserveis que corresponen a les classes del model del domini. Aquests seran tots aquells que les classes bàsiques necessiten per tal que les classes que implementen la funcionalitat puguin dur a terme les seves responsabilitats.

Si se n'ha trobat alguna que no correspon a la funcionalitat del sistema i que no sabeu on assignar−la, es poden seguir les directrius següents:

- Distribuïu la intel.ligència uniformement.
- Definiu les responsabilitats de la forma més general possible, ja que això permet unificar més fàcilment les classes.
- Mantingueu comportament amb informació relacionada.
- Mantingueu informació d'una cosa en un sol lloc. Si més d'un objecte necessita una mateixa informació podeu crear un objecte nou com a repositori, o reasignar responsabilitats o col.lapsar n objectes en unde sol.
- Compartiu responsabilitat entre objectes relacionats
- Si encara no s'ha pogut assignar clarament una responsabilitat a una classe podeu analitzar les situacions següents: és massa complexa i convé partir−la per assignar parts més concretes a més d'una classe; falta alguna classe; potser creeu una classe encapsulant les responsabilitats no assignades.

Analitzant els objectes identifiqueu també diversos serveis necesaris:

- Identifiqueu estats dels objectes: si un objecte es pot trobar en estats diferents cada canvi d'estat requerirà un servei.
- Tot objecte ha de tenir uns serveis anomenats aritmèticament simples que són el 80/95% dels que existeixen: crear un objecte, connectar un objecte amb un latre, accés per obtenir/modificar un atribut, donar de baixa, etc.
- Identifiqueu els serveis complexos: calcular un resultat a partir dels atributs, monitoritzar dispositius externs, etc.

Les connexions de **missatge** són aquelles que ha d'establir amb altres per poder dur a terme les seves responsabilitats. També s'anomenen col.laboracions o una relació depens−on. L'estrategia per identificar co.laboracions es basa a fer−se dues preguntes per ada classe:

- Quins objectes necessito?
- Quins obejctes em necessiten?

I per cada responsabilitat haurem e preguntar:

- És capaç la classe per si sola de complir?
- Si no, que necessita?
- De quines altres classes necessita?.

Identificar estructures:herència

L'activitat següent conisteix a identificar les relacions d'herència entre objectes. Es crearan diagrames jeràrquics distingint les classes abstractes de les concretes. S'hauran d'examinar les relacions:

- is−a
- is−analogous
- kind−of

Les classes abstractes es creen com una classe que conté la part comuna d'altres classes agrupant−les.

Per crear una bna jerarquia és necessari:

- Posar responsabilitats comunes al nivell més alt possible.
- Tota classe abstracta hauria de tenir almenys dues subclasses.
- Assegureu−vos que una classe abstracta no hereti d'una concreta.
- Elimineu classes queno afegeixen funcionalitat.

Per últim s'haurà de tenir en consideració:

- Estan en el domini del problema?
- Tenen sentit les especialitzacions?
- És necessari reconèixer les diferències?
- H haurà herència?
- Totes les especialitzacions compleixen les consideracions generals d'objectes?

Identificar temes i subsistemes

Pensant en la reusabilitat els temes ofereixen una clusterització de classes que no permet localitar els objectes més fàcilment en futures aplicacions. D'altra banda, els clusters ofereixen la possibilitat de tractar−los com un sistema en si mateix amb el seu propi cicle de vida.

També h ha una altra raó que és la organització modular del sistema. Alguns llenguatges de programció orientada a objectes suporten, amés a més d'objectes, el concepte de modul.

El procés de creació es basa a convertir la classe superior de cada estructura en un tema. És a dir, en un primer pas deixar només les clases més altes dins les jerarquies d'herència i d'inclusió. A continuació s'entrarà en un procés de refinament minimitzant interdependències (estructures) i interaccions (missatges). Aquest refinament signfica que també haurem d'unir aquells dos temes candidats si tenen forts lligams d'instancia o misstage.

Disseny

Component domini del problema

La primera tasca que haurem de realitzar en el disseny és incorporar en el component domini del problema el model objecte que s'ha acabat de desenvolupar en l'analisi.

Aquest model pot experimentar en aquesta etapa diverses modificcions. Les raons de modificar el model són els següents:

- Reutilització de la llibreria de classes.
- Per agrupar les classes de la nostra aplicació que siguin específiques d'un domini en un cluster s'acostuma a crear una classe arrel abstracta que les agrupi a tote.
- Adaptar l'herència al nivell suportat pel llengutage
- Millorar el rendiment.
- Incrementar la velocitat reduint el trnasit de missatges mitjançant l'agrupació de classes.

- Millorar la velocitat aparen, creant classes temporals per emmagatzemar resultats intermedis.
- Suport per al component gestor de dades.

Component interfície

El paradigma MVC

MVC és l'acronim de M;odel−View−Controller. Aquest paradigma consisteix a separar la funcionalitat de la interfície. És a dir, si tenim una funció que ha de realitzar un mantenment de clients hi haurà tres objectes col.laborant entre si. El Controlador és el responsable d'interpretar l'entrada del client i actualitzar l'objecte model client. Per accedir a `usuari ho farà per mitjà d'un objecte Vista que serà el que mantindrà la interfície.

El paradigma ha resultat mlt eficaç perquè independitza la interfície molt volàtil de la funció. A més a més, permet fer l'aplicació molt més transportable en haver de preocupar−se només dels objectes Vista. D'altra banda, això també ha permés que aquestes vistes es poguessin generar molt més fàcilment si no hi havia intercalada la funcionalitat. Per tant, s'estan creant molts generadors d'interfícies que permeten generar−les per qualsevol entorn gràfic o textual.

Amb aquesta divisó de responsabilitats mai no s'han d'incorporar en el model del domini les funcions que no li siguin pròpies, com ara el manteniment amb pantalles, perque aquestes són especificques d'una aplicació concreta en què s'està utilitzant un objecte i no al revés.

En dissenyar les pantalles i/o els llistats convé aplicar els mateixos criters que s'han usat per al model del domini. Si somparem, per exemple, les pantalles de manteniment de diferents objectes, veurem que totes són molt semblants quan a l'operativa. Convé, per tant, fer una abstracció d'aquestes característiques comunes i crear unes classes abstractes dels diferents prototipus de pantalles que tenim i per mitjà de l'herència crear les específiques de cada funció.

Incorporar la funcionalitat

La part de les funcionalitats que no era funció específica d'un objecte del model del domini del problema s'incorpora en controladors que s'hauran de crear en aquest punt.

Si hi ha responsabilitats que no sabem a qui asignar haurem d'aplicar tots els criteris que s'han descrit anteriorment. Definr serveis, identificar connexios de missatges.

Component gestor de dades

El component gestor de dades serà el rsponsable de recuperar la informació de la memòria persistent. Aquesta es pot organitzar en forma de fitxers plans o clàssics, bases de dades relacionals o orientades a objectes.

Hi ha dues formes de realitzar la gestió de dades. Una és que cada objecte sap com salvar−se a si mateix i se li demana directament. L'altra és que l'objecte gestor de dades és el responsable d'accedir al disc, l'objecte l'únic que fa és passar−se elll mateix en algun format al gestor de dades i aquest ´s qui fa l'emmagatzemament.

La recuperació d'objectes la farà sempre el gestor, el qual haurà de tenir un servei per cda classe que s'ha de recuperar.

TEMA 5.− LLENGUATGES ORIENTATS A OBJECTES (EIFFEL)

Inroducció

Eiffel està considerat el llenguatge orientat a objectes més ben dssenyat i més complet. En realitat és un preprocessador que tradueix a C.

Classes i objectes

Una classe està composta d'atributs i serveix. El conjunt dels dos s'anomena features en Eiffel. Els atributs poden ser simples o complexos. Els simples són els de tipus INTEGER, BOOLEAN, CHARACTER i REAL. Tots els altres (array's, strings's, etc.) han d'estar definits per una classe. El llenguatge incorpora una llibreria de les classes més freqüents.

Per definir una clase PERSONA escriuríem:

Class PERSONA **feature**

Nom, cognoms: STRING;

Any_naixement, any_defunció: **INTEGER**

Casat: **BOOLEAN**

End −− clasee persona

L'última línia inclou un comentari que s'indica amb el doble guió. Les paraules amb negreta són paraules reservades.

La relació referencia s'implementa com un punter, definir un atribut d'un tipus d'una classe. En Eiffel no existeixen punters, tot són referències. La declaració d'un atribut com un altre objecte no implica la seva creació automàtica. És a dir, que aquest camp està inicialment buit o **void**. És responsabilitat del programador instanciar l'objecte referenciat.

Els atributs de classe queden inicialitzats als valors següents en el moment de la instanciació d'un objecte:

Tipus	**Valor inicial**
INTEGER	0
BOOLEAN	fals
CHARACTER	caràcter nul
REAL	0.0
tipus de classes	referència buida

Si ens fixem en l'última línia veurem que el valor incial és una referencia buida. Això és perque en Eiffel sempre s'ha d'executar un *create* per crear un objecte.

Hi ha alguns serveis predefinits (amb nom fixat) per Eifel. Tots es poden redefinir per l'usuari per tal d'adaptar−los a les seves necessitats. Si tenim un atribut b d'alguna classe els serveis i la seva crida són els següents:

- Create: per instaciar un objecte: b.create
- Forget: per destruir un objecte, tornar la referència buida: b.Forget
- Void: per conèixer l'estat de la referència d'un atribut: b.void ens retornarà un booleà.
- Clone: per crear un objecte amb el mateix contingut d'atributs que un altre: a.Clone(b).
- Equal: per comparar si dos objectes tenen el mateix contingut: e.Equal(b) retorna un booleà.

Per referenciar un atribut internament en una classe ho farem amb la mateixa notació utilitzada habitualment si és de tipus simple:

any_naixement:=1939

Per referenciar un atribut visible d'un objecte des d'un altre:

autor.any_naixement

També pot usra el terme **Current** per referir−se a la instància actual de la classe continent però normalment no és necessari.

Els serveis s'anomenen rutine i se subdivideixen en procediments i funcions. EL procedimet no té retorn i modifica l'estat d'un objecte. Les funcions retornen algun valor calculat de l'estat d'un objecte. La forma d'accedir a un servei és:

entitat.operació (arguments)

En les funcions es retorna el que es trobin despres de la paraula **result**, assignant−li un valor.

La clàusula *export* ens permet especificar la visibilitat d'un objecte. Només els atributs i serveis que relacionem a *export* seran visibles pels altres objectes.

Per definir variabes locals en un servei es pot fer servir la clausula **local** i després declarar les variables locals que volguem. S'hi accedei x de la mateixa forma que si fos un atribut.

Eiffel pot redefinir els serveis bàsics, veiem com:

create (a, b REAL) **is** − inicialitzar un punt amb dues coordenades

do

x:=a;

y:=b;

end − crear un objecte

i la utilització seria: p.Create(10.4, 15.7)

Genericitat

La creació de classes genèriques o parametritzables s'especifica afegint a la capçalera de la classe generica una llista de parametres formals, i a continuació, utilizant aquests paràmetres dins de la definició de la classe.

Assercions

Eiffel permet amb molta senzillesa la construcció de software robust mitjançant la incorporació d'assercions en les definicions de les classes. les asercions ens permetran introdui pre i postcondicions, invariants. També te dues construccions més, una per bucles la variant, i l'altra general, el check.

Les assercions són expressions simples booleanes amb algunes extensions. La primera extensió és el punt i

coma ; per separar−les i és equivalent a un **and** però amb l'avantatge que permet identificar millor cada component individual i fins i tot etiquetar−les.

Per definir les precondicios s'utilitza la clausula **require** i les postcondicions **ensure**.

Dues extensions adicionals són la **old** i la **Nochange**. La **old** es refereix al valor anterior que tenia una variable en el moment d'entrar a la rutina. La Nochange ens indica si l'estat de l'objecte actual no ha estat canviat des de la crida.

Els invariants es poden utilitzar en dues situacions: classe i bucles. Per definir invariants globals de la classe, és a dir, unes condicions que s'han de mantenir en toto moment dins d'una classe, es poden especificar assercions no solament sobre atributs sino també entre serveis de la classe.

La definició d'invariant requereix de la clausula **invarant**.

El variant és una expressió entera el valor de la qual és no negatiu després d'inicialització i que ha de decrèixer almenys per u a cada cicle. Això permet assegurar que el bucle acabi ja que no pot ser negatiu. La denició dels variants o fites es realitza amb la paraula **variant**.

També es pot utilitar la construcció check en qualsevol punt d'una rutina per assegurar que s'estan complint certes condicions. L'estructura de check és la següent:

Check

Asserció 1;

Asserció 2;

...

asserció n;

end

Instruccions

Eiffel suporta les instruccions següents: crides a procediments, assignació, condicionals, bucles, cntrol (**check**) i debug.

Les crides a procediments tenen la forma:

P (sense arguments), o

P (x, y, ...) (amb arguments)

Si la crida és a un objecte tindrà l'identificador de l'objecte com a prefix seguit d'un punt, com ja hem vist. Poden encadenar−se diverses referències en una única crida.

p.q.create

L'assignació té la forma: x:=e

On x és una entitat (un atribut de classe, una variable local, etc),

Els condicionals són de les formes conegudes:

if ... **then** ...[**else** ...] **end**

if .... **then** ..... **elsif** ....**then** ...**else**.... **end**

L'estructura del bucles és la següent:

from instruccions d'inicialització

invariant invariant

variant variant

until condició de sortida

loop instruccions de bucle

end

on el variant i l'invariant són opcionals.

Pel **check** el bucle és el que hem vist anteriorment:

check

asserció; asserció; ...; asserció

end

També hi ha un debug de complació condicional

debug instrucció; instrucció; ... ;instrucció

Operadors i constants

Els operadors de diferents tipus que tenm són els següents: =, /=, <, >, >=, <=, and then, or else, and , else.

Els **and then** i **or else** són no conmutatiu (forcen la seqüència d'execució i eviten avaluacions no desitjades).

La classe STRING també està definida i està previst poder escriure cadenes de caràcters en més d'una línia, com es mostra a continuació:

ABCDEFGHIJKLM\

|NOPQRSTUVWXYZ

Eiffel te dues classes per a les entrades i sortides bàsiques: FILE i STD_FILE.

Les constants simbòliques de tipus simples es tracten com atributs que tenen el valor fixat per totes les

instancies d'una classe:

cero: INTEGER **is** 0

Per crear constants e tipus de classes resulta una mica més complicat. La forma és definir dins d'alguna classe un servei. On es substitueix **do** per **once** , aquest canvi assegura que només s'executi una sola vegada la creació i així s'estalvïi memòria i tmeps d'execució. Cal anar molt amb compte perquè, realment, el que hem aconseguit és una referència constant i un objectecompartit, no un objecte constant, i per tant, si s'utilitzen els serveis per modificar els seus atributs quedarà modifict.

Herència

Eiffel permet tant l'herència simple com la múltiple. Vegem en primer lloc un exemple d'herència simple en el cas d'un rectangle que hereta de polígon:

class RECTANGLE **export**

vèrtex, perimetre, trasllada, rotar, ...

inherit

POLIGON **redefine** perimetre

feature

En aquest cas, RECTANGLE ho heretarà tot de POLIGON excepte el servei perímetre, que es redefineix a RECTANGLE perquè interessa una implementació diferent del seu antecessor.

En una relació d'herència s'hereta tot excepte el servei create, i els seus paràmetres poden ser diferents en nombre i tipus. Malgrat que no s'hereti mai directament el servei Create, indirectament sí que pot fer−se en poder utilitzar l'algorisme del Create del pare.

un tipus especial de classes es presenta, especialment en el proces d'abstracció de subclasses a superclasses, quan una superclasse té un o mé serveis no definits. En aquest cas s'espera que cada subclasse que hagi de ser instanciable concreti (defnició) els serveis enunciats però no definits en el seu antecessor. El que es fa en aquests casos és declarar en el pare un servei com diferit. Suposem que tenim una classe abstracta FIGURA, que ens és útil per recollir el que tenen en comú totes les figures i el que volem que totes ofereixen. Naturalment en aquesta classe no podrem definir tots els serveis, però podem fer el següent:

trasllada (a, b : REAL) **is** − moure la figura horitzontalment a, verticalment b

deferred

end; −− trasllada

L'herència multiple és una opció que no és necessaria sovint però que en cas de que el llenguatge no suporti la funcionalitat, s'ha de resoldre amb dificultat mitjançant delegació. En Eiffel s'ha implementat de la forma següent:

class C **export** .....

inherit

A **rename** f **as** A_f;

B **rename**

g **as** B_g, h **as** B_h

redefine g, B_h

feature

....

Tecnques d'implementació (Objectes estàtics i subsistemes)

Freqüentmet a l'hora de construir sistemes ens interessa tenir objectes i paràmetres globals. En el cas dels objectes a vegades s'anomenen objectes estàtics. La forma de resoldre aquesta situació és englobar−los en una clsse ENTORN i aplicar la clàusula **once** com hem vist a l'apartat de constants.

Aquesta tècnica també es pot utilitzar en funcions d'inicialització. Per no haver−nos de preocupar de si ja s'han reaitzat o no les inicialitzacions, podem també utilitzar el **once** dins de procediments.

TEMA 6.− ORGANITZACIÓ DEL PROCÉS DE PRODUCCIÓ

Introducció

Existeixen problemes de gestió: terminis de lliurament ajustats, mitjans tècnics i humans molt sovint insuficients, distribució temporal de les tasques, etc.

Tot això fa que l'organització del procés de producció del software sigui un punt important i que les deficiències en aquest procès puguin portar a fer inviable un projecte.

L'organització del proces de producció es basa en quatre elements:

- Un pla de desenvolupament del projecte realista, adequat als requisits que s'exigeixen i als mtjans de què es disposa.
- Una assignació raonable de mitjans humans i tècnics al projecte. Una organització efectiva de tals mitjans.
- Un seguiment estricte del pla de treball que detecti divergències i permeti la redistribució de recursos, l'assiggnació de recursos adicionals i, eventualment, l'actualització del pla.
- Una documentació adequada de totes les etapes de desenvolupament del projecte.

Planificació del projecte

En primer lloc, després dels estudis de viabilitat corresponents, cal preparar un pla del projecte que es vol realitar. El pla es prepararà a partir de les especificacions funcionals i els requeriments no funcionals que hagin estat deinits i aprovats. EL pla tindrà l'estructura següent:

- Estimacions: S'hauran d'incorporar els conceptes següents:

Calendari.

Organització i assignació de personal.

Pressupost.

Anàlisi de benefici/cost.

Anàlisi de risc.

Documents que s'han de lliurar.

Pla de documentació.

Eines de software necessaries.

Necessitats d'instalacions i hardware.

Pla de formació.

Pla d'instal.lació.

- Procediments: aquesta etapa haurà d'inclourea:

Mdel de procès.

Metodologies i convencions de notació.

Monitorització de controls comptables.

Pla de revisions del desenvolupament.

Informes de desenvolupament.

Control de producte.

Pla de control de modificacions i versions.

Proves i fonts de dades de prova.

Pla de proves.

Criteris d'acceptació i forma de pagament.

- Referència

Documents utilitzats en el desenvolupament del plà.

Glossari de termes.

Contracte proposat.

Organització, etapes, fites i control

Una de les tasques principals de tot responsable de projecte és la preparació d'un calendari de tasques, amb la distribució de les mateixes entre els tècnics disponibles, i el seguiment del compliment d'aquest calendari.

El control adequat del desnvolupament és, per tant, un requeriment essencial de tot projecte informàtic i la detecció oportuna de desviacions pot pemetre posr mitjans adicionals que tot i que representin un cost no previst, permetin salvar la data final.

L'eina principal de eguiment de projecte és l'anomenat diagrama de GANTT. L'esquema es pot veure a la figura 6.1 pagina 156 del llibre.

Si en qualsevol cas, la tradicional mesura d'assignació de persones a activitats indueix a l'errada i és poc fiable, això s'accentua en les darreres fases de desenvolupament del projecte, quan se solen fer patents les divergències irrecuperables i es cau en la temptació de fer una assignació massiva de nou personal que pocs problemes resoldrà.

Una tècnica que molt sovint s'ha utilitzat amb èxit és la del PERRT−CPM (Critical Path Analysis). La figura 6.2 de la pagina 157.

La tècnica utilitzada inicalment en el control i la monitorització de projectes d'enginyeria i construcció d'un graf dirigit els nodes del qual són fites temporals i els arcs del qual estan etiquetats per les activitats que s'han de realitzar.

El graf implementa, d'aquesta manera, les relacions de precedència entre les diferents activitats: si una activitat emergeix d'un node, volem indicar què requereix per iniciar−se la finalització de totes les activitats convergents en aquest node.

Un cop dissenyat el graf i etiquetats els seus arcs podem passar a la segona etapa. Es tractarà d'assignar cost a cada una de les activitats.

Molt sovint, el cost s'estableix en terminis de la duració de l'activitat. És habitual assignar tres mesures de cost a cada activitat: el cost mínim, el màxim i el mitjà o més probabe.

Un cop assignats costos als arcs podem localitzar en el graf els camins crítics: es tracta dels camins tals que qualsevol desviació en el cost de qualsevol del seus arcs es reflectiria en la duració total del projecte.

Els camins crítics permeten identificar activitats crítiques i sobre elles han d'establir−se controls i seguiments més forts.

Organització humana. Mecanismes de comunicació i cooperació.

És estrany que un projecte de desenvolupament de software recaigui en una sola persona, el normal és que s'hagi de recórer a un equip de diverses persones per abordar qualsevol projecte.

Això planteja problemes d'organització de l'equip i de comunicació entre els seus membres i amb l'entorn (usuari).

Existeixen dues formes bàsiques d'organització dels equips de desenvolupament de software: una orientada a les funcions i ua altra als projectes.

En la primera, s'aïllen una sèrie de funcions o de perfils de tasca, s'organitzen en grups de treball i s'assignen a programadors a cada un dels grups.

És més corrent l'organització d'equips de desnvolupament per a porjectes concrets. En aquesta forma d'organitacio un grup es forma per abordar el desenvolupament d'un projecte concret i despaareix quan el projecte ha conclòs.

Dins d'aquesta forma d'organització caben, a la seva vegada, dues estructures: la forma jeràrquica (CPT: Chief Programming Team) i l'homogènia (que Teasley denomina democràtica).

En la primera s'organtitza l'equip de forma altament estructurada i jerarquitzada. El cap es converteix en interlocutor únic de l'exterior, en responsable de compliment de l'especificació i els requisists, en distribuidor de les tasques dins del grup i en garant de l'exitència de mitjans adequats.

Bell i Morrey suggereixen aleshores dos tipus de grups de desenvolupament: uns orientats als programes d'aplicació i un altres a les classes o als components. Els primers, que qualifiquen de Consumidors, haurien d'implementar les classes especifiques de l'aplicació i reutilitzar les existents en les llibreries. Els segons, Productors, actuarien sobre les classes de components de reutilització del material.

Una activitat estretament lligada als grups de treball organitzats, però no exclusiva d'ells, es la de les reunions de recorreguts (walktrough) de programes per part del membres del grup.

Documentació

Introducció

Cada tasca en un procés de desenvolupament d'un producte informàtic requereix la creació i el manteniment de certa documentació.

A més d'aquesta documentació específica de les diferents parts del desenvolupament del sistema informàtic, hem d'incloure una documentació general de l'aplicació. Aquesta documentació ha de cobrir els aspectes d'utilització del producte (documentació d'usuari) u o els de compresió i manteniment del mateix (documentació tècnica o de sistema).

La documentació d'usuari ha de proporconar una visió precisa i realista de les capacitats del producte que permeti un apropament progressiu i personalitzat de l'usuari a aquest producte.

La documentació tècnica ha de proporcionar una visió precisa i competa de l'estructura interna del producte, com també del seu procés de fabricació i dels controls i verificacions a què ha estat sotmès.

L'estil en la documentació

Sommerville cita les normes següents:

- És preferible en la redacció del text l'ús de la ve activa enfront de la passiva.
- Són preferibles les frases curtes.
- Convé evitar les referències a unitats del text simplement amb la seva referència numérica.
- Si unadescripció és compexa o difícil d'entendre convé parafrasejar−la en ulterirs utilitzacions.
- És preferible l'enumeració o presentació en forma de taula de diferents fets que la confecció d'una frase llarga que els nclogui.
- És important la concreció.
- Convé prestar atenció a la precisió terminològica.
- Utilitzar paràgrafs curts, separacions entre paràgrafs, marges amplis.
- Prestar atenció especial als títols i als subtítols.
- Prestar atenció a la correció gramatical, ortogràfica i lèxica del text.

Documentació tècnica

L'objectiu de la documentació tècnica és facilitar una comprensió global del programa.

Especial importància, des del punt de vista de documentació tècnica té la documentació del programa, és a dir, de l'anomenada carpeta de programa.

Per cada programa s'haurà de crear una carpeta o dossier on s'inclourà tota la documentació relacionada amb el programa. Aquesta informació és la següent:

Especificacions.

Disseny modular.

Interfícies

Informació operativa (notes i plans, instruccions, implantacions alternatives, viabilitat, decisions).

Codi font, preferent amb referències creuades.

Proves.

Problemes sorgits.

Full de notificació de problema, anàlisi i acció.

La utilització d'estandars

Introducció: necessitat d'un estandard

En primer lloc ens podem preguntar per la necessitat d'un estàndar de documentació a nivell de programa. Les raons sónmultiples, però la més important és la de reduir el cost de manteniment d'un programa.

La documenatació com a únic lligam en les diferents etapes d'un programa serà l'eina fonamental per garantir un cost mínim del manteniment i de la fiablitat del programa modificat. La uniformitat en la forma de codificar els noms com també unes normes mínimes d'estructurar programes, facilitaran aquesta comunicació entre tècnics.

La utilització d'estandars de documentació en els programes es refereix a una quants aspectes alguns lligats a la documentació en si mateixa i altres a la codificació dels programes.

Respecte als prmers, podem tenir en compte les exigències de documentació del programa i dels seus components: Data, Autor, Versió....

Respecte als segons podem citar:

- Limitacions en el joc d'instruccions usat.
- Limitacions sobre la dimensió del programa o d'algun dels seus elements.
- Limitacions sobre l'ús d'identificadors tant de variables com de tipus, funcions, procdiments, etc..
- ...

La notació hongaresa

La notació hongaresa és un estàndard de documentació/codificació de programaes que s'està imposant a les empreses mes importants d'informàtica.

Ha estat desenvolupada per Charles Simonyi, hongarès de naixement, l'any 1972.

La notació hongaresa dóna una estructura als noms que augmenta la informació que es comunica pel nom d'una variable a qualsevol que llegeix el codi.

La idea bàsica de la notació consisteix a prefixar al nom uns identificadors de tipus i qualificadors. D'aquesta manera, l'identficador d'una variable ens aportarà informació tant del tipus com, per mitjà del qualificadors, del seu contingut semàntic.

El tipus

El que la notació hongaresa propsa és incloures dintre de la identificació de la variable informació sobre el tipus al qual peratny. És a dir, que un nombre per una finestra és diferent que el d'un fitxer. La diferencia ve determianada per les operacions que es realitzen sobre ells.

Per identificar el tipus s'han de cosiderar primer la representació de les dades i a continuació els valors possibles.

La costrucció de tipus

Com hem dit, tots els noms comencen per un tipus seguit d'un qualificador. Posar el tipus primer facilita que el càlcul de tipus sigui un acte reflex. La maúscula serveix per separar les prts del nom i també per fer alstres distincions.

S'han de definir els tipus base com a abreviacions de la descripció del tipus. Millor crear molts tipus diferents que no reduir−ne el nombre per la dificultat de trobar una etiqueta. Si realment ens hem de referir als tipus bàsics de la màquina (byte, paraula, paraula sense signe, paraula llarga, etc.) utiltzem les etiquetes b, w, u, l.

Uns exemples de tipus són:

- f: Es tracta d'un tipus booleà
- ch: caracter, 1 byte.
- Sz: string acabt en caracter zero.

Partint dels tipus b`sics es pden construir tipus complexos. La construcció de nous tipus significa que elnom d'un nou tipus està relacionat amb altres tipus ja existents.

Alguns exmeple de prefixos de tipus són:

- p: punter.
- h: handle (punter doble).
- mpT1T2: taula indexada per quantitats del tipus T1 i que conté elements del tipus T2.
- rgT2: una taula d'elements del tipus T2, que és el range del mapa.
- i: index a una taula.
- c: comptador d'instancies de determinat tipus.
- d: diferencia entre dues instàncies d'un tipus.

Si en anar concatenant tipus es formés un preix excessivament llarg o dificil d'entendre, es pot crear un nou tipus per a tot ell o un dels seus components.

Pels noms d'etiquetes d'adreces i procediments hi ha regles especials. Els noms de procediments comencen amb majúscules per diferenciar−les de les variables i les constants.

Qualificadors

Els qualificadors serveixen pe distingir variables amb el mateix tipus en un mateix àmbi. També són útils per documentar altrespropietats. A diferència dels tipus es poden utiltzar paraules per als qualificadors. Si s'usa més d'una paraula se separen posant en majúscula la primera lletra de cada paraula.

Els criteris per triar qualificadors depenen del tipus al qual pertany la variable que es vol identificar.

Per variables booleanes (f): descriure la condició de certesa.

Per valors en conjunts enumerats descriure l'element particular.

Existeixen una sèrie de qualificadors estandards que es detallen a continuació:

- Temp. Una variable temporal.
- Sav: una variable temporal d'on es recuperarà el valor.
- Prev: un valor salvat que va un lloc darrere de l'actual en una iteració.
- Cur: valor actual d'una enumeració.
- Next: valor següent d'una enumeració.
- Dest, Src: destí i font en alguna relació client/poductor.
- Nil: valor especialil.legal disntigible de tots els legals.
- 1,2: també podeu usar nombres per distingir valors.

Per tipus que indexen taules els estandards són:

- Min: minim index vàlid.
- Mac: màxim actual.
- Max grandària assignada màxima.
- First: primer element d'un interval.
- Last: últim element d'un interval.

Procediments

Les regles basades en els tipus de les variables i les constants no serveixen directament pels procediments. La raó és que n'hi ha que no retornen cap valor. Un espera que elnom d'un procediment digui el que fa, no el que retorna. A més a més, els noms de procediments han de ser únics en tot el programa.

[Tipus][Acció(ns)][Paràmetre(s)]

Cada paraula comença amb majúscula. La majúscula inicial distingeix els noms de procediments del de variables. El tipus quan apareix indica el retorn. S'escriu primer com en el cas de les variables per ajudar en el càlcul de tipus.

Altres normes

- Grandaria màxima mòdul.
- Presentació: identar per a cada estructura de control un nombre fix de posicions (3 o 5).
- Per mòdul: descripció, autor, data, paràmetres, retorn, altres mòduls usats, pre i postcondicions.
- Documentació interna: per variable, constant i tipus no auto−descriptiva; per bucle; per condicional.
- Utilització de constants definides.
- Normes de programació: per exemple admetre un sol RETORN per mòdul i mai dins un bucle.
- Normes d'utilització de recursos: utiltzar al màxim la biblioteca per definicions, declaracions, tipus, etc.

La documentació d'usuari

Per tot programa s'ha de crear un manual d'usuari que li permeti la utilització correcta del programa sense requerir assistència per part del programador.

Per construir el manual d'usuari la primera exigència és un coneixement adequat de les característqiues de l'usuari a qui va dirgir el manual.

El contingut bàsic d'un manual d'usuari és el de les capacitats funcionals que el sistema ofereix. El manual ha de descriure aquestes capacitats i ha de relacionar−les amb els objectius que l'usuari tindrà en utilitzar el sistema.

El sistema li oferirà unes funcionalitats i li proposarà uns mitjans d'utilitzar−les per satisfer aquests objectius.

La descripció d'aquesta relació entre els objectius de l'usuari i les capacitats del sistema s'haurà d'expressar tant a nvell conceptual com a nivell sintàctic.

Segons el tpus d'usuari i el que es pretengui documentar existeixen diferents formes de documentació d'usuari:

- Guia de referència: adreçada al tècnic i que contindrà la referència exhaustiva de les capacitats del sistema.
- Guia d'usuari: que proporcionarà un accès ampli i poc profund a les funcionalitats del sistema.
- Manual d'aprenentatge i tutoriala: guia d'aprenentatge de le funcionalitats bàsiques del sistema. Documentació d'iniciació al producte.
- Targetes de referència rapida: recordatoris, bàsicament sintàctics, per a usuaris experimentals.
- Diccionaris i glosaris.

TEMA 7.− LA INTERFÍCIE PERSONA−MÀQUINA

Introducció

La interficie entre sistema informatic i usuari (HCI: Human Computer Interface) és simplement el mecanisme de comunicació, possiblement en les dues direccions, entre ambdos elements.

Shneiderman assenyala com a objectiuineludible d'un sistema interactiu, que tingui una funcionalitat adequada, ja que sense ella, encara que la interfície etigui ben dissenyada l'usuari se sentirà frustrat. El pas següent es assegurar−se que el sistema sigui fiable. Només quan aquests rrequisits funcionals es compleixen té sentit considerar els criteris d'habilitat que permeten definir la qualitat de l ainteracció per a una comunitat especifica d'usraris i une classes de tasques concretes:

- Temps necessari per aprendre a utilitzar−la.
- Rapidessa amb què permet resoldre les tasques que l'usuari ha de realitar.
- Percentatge d'errades que els usuaris cometen.
- Factor de retenció.
- Satsfacció subjectiva de l'usurai.

Aquest criteri d'independencia es basa en la definició formal de la comunicació entre la interfície i els programes d'aplicació, de forma que les decisions que afecten la interacció estiguin aïllades de les que afecten a la resta.

Nombroso autors advoquen pe una separació radical entre els aspectes funcionals del software i els aspectes comunicatius. En una aplicació informatica per a ús humà els primers definirien les capacitats funcionals del producte (el que fa el producte) mentre que els segons es referirien a la forma d'accedir a aquestes capacitats.

Teasley assenyala una sèrie d'avantatges de tal procedir:

- Els avantatges inherentsa tota partició d'un problema en subproblemes.
- Una majorfacilitat en el manteniment i la millora de qualsevol de les dues parts.
- Una major capacitat de reutilització.
- La possibilitat d'utilització d'eines d'ajuda al disseny d'interfícies.

La funcionalitat de la interfície ha de basar−se en l'analisi cognitiva de les tasques que van a realitzar els diferents tipus d'usuaris, i a prtir d'aquesta, en l'establiment dels requisits que hande guiar tant l'estructuració de la interfície com el disseny de la interacció per tal que aquesta pugui realitzar−se d'una forma productiva a tots els nivells: concpetual, sintactic, funcional i lèxic.

Dues són les metàfores d'interacció amb els sistemes informàtics que trobem avui en dia: el dàleg, en forma de menus , formularis, ordres o llenguatge natural i la manipulació directa. Tots ells han de complir certs requisits:

- Consistencia
- Realimentació informativa.
- Maneig i recuperació simple de les errades.
- Possibilitat de desfer certes accions.
- No sobrepassar la capcitat de la memoria a curt termini de l'usuari.
- Permetre a l'usuari experimentat formes reduïdes d'interacció.
- Cessió del control

Principis generals de la interficie

El primer pas per determinar de forma correcta la funcionalitat d'un sistema ha de ser realitar una analisi de les tasques que l'usuari ha de resoldre i estudiar si s'ajusten a les possibilitats que ofereix el sistema que es condueix per realitzar−les.

Shneiderman assenyala que un dels errors més corrents sol ser la sobrefuncionalitat del sistema que s'ofereix, ja que l'exces de característiques lluny de simplificar la tasca sol sobrepassar l'usuari i fer més dificil la comprensió i elmaneig del sistema, a més complica el manteniment i la documentació.

El model sintàctica−semàntic

Aquest model, del Shneiderman, afirma que els usuaris tene dos tipus de coneixement, un sintactic, sobre els dispositiu amb el qual interaccionen i sobre el mode d'interacció, i un altre semàntic, que engloba al seu torn dos tipus de conceptes: sobre les tasques que cal realitzar en el domini de l'aplicació, fonamentalment sobr els objectes i les accions, i sobre el propi sistema informàtic.

El coneixement sintàctic es dependent del dispositiu, s'adquireix per memorització i s'oblida fàcilment, mentre que el semàntic es estructurat, el seu procés d'aprenentatge és més profund i el coneixemnt adquirit és més estable.

El GOMS (Goals, Operators, Methods, Selection)

GOMS te per proposit construir una representació explícita del coneixement procedimental que ha de tenir un usuari per poder interaccionar amb un sistema.

En aquest model es destaquen quatre elements fonamentals:

- Objectius que l'usuari ha de realitzar.
- Mètodes o procdiments per realitzar aquests objectius.
- Operadors que l'usuari por portar a terme.
- Regles de selecció que especifiquen els mètodes adequats en un determinat context.

A més poden estalir−se criteris qualitatius tals com:

- Naturalitat del disseny.
- Completitud.
- Clardat.
- Consistencia
- Eficiencia.

La component comunicativa

Un cop establerta la funcionalitat del sistema s'ha de separar les decisions que afecten el diàleg d'aquelles lligades a les característiques funcionals de l'apliació.

El disseny d'una interficie requerirà doncs, modelitzar l'estructura i a naturalessa dels intercanvis entre usuari i sistema i descriure'ls amb algun esquema notacional.

La modelització estructural de la interficie té com a objectiu descriure els procesos de comuniació de forma que serveixin per guiar ek disseny del diàleg.

Existeixen dues alternatives, els models lingüístics (com MIKE) i els no lingüístics.

L'interlocutor humà

Bona part de l'èxit d'una interficie entre persona i màquina depèn de la forma com la interfície incorpora les característiques concretes que un interlocutor humà té.

Pressman cita tres nivells per definir aquests factors humans:

- Nivell bàsic, en què hauran de tenir−se en compte la percepció visual, la psicologia cognitiva en la lectura, la memòria humana i el raonament deductiu i inductiu.
- Nivll d'usuari, en què s'hauria de tenir en compte el comportament concret de l'usuari davant l'aplicació.
- Nivell d'aplicació, en qué s'haurien de tenir en compte aquestes característiques ja que condicionaran el disseny posterior de la interfície.

Coneixement que l'usuari poseeix

No té sentit informar l'usuari d'alguna cossa que ell ja sap o donar−li explicacions a un nivekll inadequat. Existeixen dues aproximacions possibles per utilitzar aquest coneixement:

- Filtrar les intervencions del sistema de forma que no es comuniqui a l'usuari res que ja conegui.
- Planificar les intervencions del sistema tenint en compte el tipus de coneixement que l'usuari ja poseeix.

Les comunicacions a l'usuari han de ser suficientment informatives sense recarregar el receptor.

Objectius i actituds de l'usuari

Per tal que el contingut del missatge adreçat a l'usuari sigui acceptat per aquet, no ha de contenir simplement informació nova i coherent amb el coneixement previ que l'usuari poseeix sinó que ha de contenir elements que siguuin reconeguts per l'usuari com a rellevants per als seus propis objectius.

Si l'objectiu de l'usuari es limita a l'adquisició de nova informació, no sembla que es plantegin problemes.

La formació del model de l'usuari

Un dels primers sistemes que utilitza un model d'usuari per guiar el diàleg es l'SCHOLAR, el sistema tutor desenvolupat per J.Carbonell. En aquest sistema, els conceptes apareixen simplement marcats com a coneguts o no per l'usuari. Al llarg del diàleg nous conceptes eren marcats com a coneguts a mesura que eren previsiblement assimiilats per l'usuari.

En general ham de distingir dos aspectes, un d'estatic, que fa referencia a la classificació prèvia de l'usuari i un altre de dinàmic, que afecta els canvis que el model experimenta a mesura que es desenvolupa el diàleg. Wahlster i Kobsa clssifiquen les possibles fonts d'informació per a la construcció del models d'usuari de la forma següent:

- Raonament per defecte a partir d'estereotips.
- Models inicials a partir de sessions anterirs.
- Inferències directes a partir de les expressions de l'usuari.
- Inferències indirectes a partir de les expressions de l'usuari.
- Contingut de les intervencions del sistema.

El metode més utilitzat, en allò que respecta a la modelització estàtica, és, sens dubte, l'ús d'ESTEREOTIPS. La idea es predeinir una sèrie d'estereotips (descripcions típiques de comportament) i assignar cada usuari a una d'aquestes categories.

La classificació d'un usuari en una categoria determinada pot efectuar−se a instàncies del propi usuari, mitjançant alguna estructura de dades que stableixi l'associació entre usuari i categoria o mitjançant algun tipus de diàleg que permeti al sistema etablir aquesta associació.

User Modeling Front End (UMFE)

L'UMFE el va crear Sleeman. En aquest sistema, la informació sobre el coneixement que l'usuari poseeix és derivada de tres fonts:

- A partir de preguntes explicites formulades pel sistema.
- A partir d'inferències basades en el nivell d'experiencia de l'usuari i la dificultad dels conceptes implicats.
- A partir d'inferèncis basades en l'estructura conceptual del domini, d'una manera especial en les relacions taxo nòmiques entre conceptes.

Es tracta, com veiem, d'una aproximació en la qual elmodel de l'usuari es va actualitzant a mesura que el tipus de coneixements posseïts per l'usuari es va incrementant.

Estils d'interacció

Hi ha dues metàfores per a la interacció: la conversa i el model del món, que es corresponen amb dos tipus de diàleg, el seqüèncial i l'asincron, respectivament. El primer evoluciona d'una manera predictible, vsualitzant seqüències lògiques de comportament, mentre que el segon facilita l'accés directe als objectes del domini, per això s'anomena manipulació directa. Asociat a la manipulació directa hi ha el diàleg multi−traça, que indica la

multiplicitat de tasques que pot realitzar un usuari en l ainteracció. En diàlegs aincrons, diverses tasques poden realitzar−se en qualsevol moment, per això s'anomenen dirigits pels events. En dialegs conversacinals l'usuari interacciona mitjançant el teclat, mentre que en models del domini l'usuari maneja objectes en la pantalla (ratolí).

Un aspecte important en la definició de la interfície persona/maquina es efereix a la iniciativa de la comunicació. Existeixen formes de comunicació dirigides pel sistema mentre que altres són dirigides per l'usuari.

Entre les primeres podem citar:

- les seqüències de menús.
- Els sistemes basats en preguntes i respostes.
- Les comunicacions unidireccionals del sistema com l'emissió de misstages d'error o de misstages d'estat.

Entre les formes de comunicació dirigides per l'usuari podem citar:

- L'ús de llengutages d'ordres.
- La utilitació de formularis.
- Els diàlegs asíncrons.

Diàleg seqüèncial

Seqüència de menús

Cada menú presenta una sèrie d'alternatives i l'usuari n'ha d'escollir una d'elles mitjançant un mecanismes de selecció. En funció d'aquesta elecció s'ofereix un altre menú a continuació o s'executa directament l'acció seleccionada.

Ofereix una interacció simplificada, que redueix la possibilitat d'errades, i guia l'usuari per estructurar la seva interacció.

És important tenir en compte que un menú no és un element aïllat sinó que forma part d'una estructura complexa. Habitualment l'estructura de menús és un arbre i la seqüència de menus que constitueix el diàleg una de les branques de l'arbre. Habitualment, també, les accions que el sistema ha d'efectuar s'associen a lesfulles de l'arbre.

La utilització de seqüències de menús és especialment adequada quan els usuaris tenen poca experiència.

L'organització d'un sistema de menus ha de permetre una comunicació àgil i efectiva. Teasley proposa dues fases:

En la primera s'estructuraria semànticament l'espai de les funcions que l'aplicació és capaç d'efectuar i que l'usuari haurà de poder activar a trevés del menú. Les funcions es poden agrupar en forma diversa: en forma línial, jerarquica, en xarxa, etc.

En una segoa fase s'establiria l'estructura de menus i les connexions entre les funcions per executar.

Han de tenir−se en compte diverses consideracions:

- Agrupar les funcions pròximes d'acord amb algun criteri clar i consistent.
- La correspondència ha de ser exhaustiva.

- No han d'admetre's interseccions.
- L'etiquetatge de les opcions es fnamental.
- Dins de cada menú, les opcions han d'organitzar−se d'acord amb algun criteri consistent

Llenguatge d'ordres (Command Language)

Les ordres son frases d'un llenguatge formal que permeten a l'usuari expressar les diferents operacions que desitja realitzar amb el sistema. A diferencia de la seqüència de menús, l'usuari te l'iniciativa, i ha de recordar la forma d'expressar−se. Per això, un punt important en el disseny de les instruccions es tenir en compte una estrategia que faciliti el seu aprenentatge.

La definició d'un llengutage d'ordres ha de tenir en compte el tipus d'usuari que ha de interaccionar amb el sistema. S'han de tenir en compte diferents aspectes per tal que l'actuació de l'interfície sigui efectiva:

- S'ha d'estudiar acuradament el nombre d'ordres que el llenguatge ha de poseeir com també els paràmetres que cada ordre haurà d'admetre.
- S'hurà de definir clrament, per cada una de les ordres, els paràmetres que s'hi han d'incloure, si són o no obligatoris, si prenen valors per defecte, etc.
- Les ordres hauran de poseeir identificadors mnemotècinis.
- No hi ha d'haver ambigüitat ni cap confusió en la nomenclatura.
- La nomenclatura haurà de ser consistent per a tot el llenguatge.
- És aconsellable l'admissió d'abreviatures, macroordres i ordres i alies definibles per l'usuari.
- És aconsellable accentuar els mecanismes d'ajuda oferint a l'usuari menús d'ordres utilitzables, sintaxi admissible, etc.

Reemplenament de formularis

El mecanisme de comunicació basat en el reemplenament de formularis és molt convenient en operacions d'entrada massiva d'informació.

Es tracta d'un metode d'iniciativa mixta en el sentit que el sistema presenta el marc i l'usuari reemplena els camps que jutja necessaris en l'ordre que considera adequada.

També en aquest tipus d'interfícies necessitem tenir en compte una serie de criteris de disseny:

- Agrupar d'acord amb algun criteri semàntic, la informació que s'ha d'incloure (o sol.licitar) en cada un de les pantalles.
- Tenir molt presents les qualitats de presentació de les pantalles.
- Tenit en compte la quantitat d'informació present en la pantalla i la forma de presntar−la.
- Avaluar per a cada entrada les possibilitats d'adquisició línia a línia enfront l'adquisició pantalla a pantalla.
- Característiques físiques del dispositiu: grandaria, facilitats tipogràfiques, font, color, etc.
- Forma de visualització dels camps d'entrada: cixes, video invers, lluïsor, subratllat, etc.
- Etiquetatge dels camps.
- Moviments de cursor, ús de tabuladors, ratolí, etc.
- Ú de missatges d'error pertinents i positius.
- Senyalització gràfica de la obligatorietat o opcionalitat de cada camp.
- Possibilitat d'obtenció d'ajudes lligades a cada camp.
- Formes de validació dels camps.
- Possibilitat de correció a nivell camp i a nivell de caràcter.
- Possibilitat d'adquisició i/o validació incremental de les dades entrades.

Maneig directe

Aquest mode d'interacció es fonamenta en la simulació del domini, de forma que tant els objectes com les accions que poden realitzar−se sobre ells siguin visibles, i l'usuari els pot manejar de forma directa. El lema és allò que es veu és allò que s'obté (WYSIWYG).

El maneig directe convé per a tasques que involucren un nombre limitat d'objectes, que són relativament simples, iper als quals es pugui trobar una bona representació icònica del domini. Els avantatges que ofereix quan el disseny és adequat són la seva facilitat d'ús i d'aprenentatge, accions ràpides i reversibles, efectes visibles inmediatament i, per tant, ecessitat menor de missatges d'error.

Metodologia de la construcció d'interfícies

RAPID/USE, creat per Wasserman en 1985, es basa en el principi d'independencia mencionat prèviament i propugna la participació de l'usuari des dels inicis del procés de desenvolupament per avaluar les progressives maquestes d'interfície que es dissenyin.

La interfície que Rapid/Use ofereix per realitzar el disseny de les maquetes és un editor gràfic (de diagrames de transició) que genera el codi corresponent.

DMS (Dialogue Management System, creat per Ehrich i Harston) veu el desenvolupament d'un sistema interactiu com la creació de tres components: el dialeg, encarregat de la comunicació usuari/sistema, el component computacional que conté els programes de l'aplicació i un component de control global que supervisa i dirigeix la relació entre el component de diàleg i el computacional.

Actualment es disposa de software i hardware per ajudar en l'automatització del desenvolupament d'interfície. Podem distingir dues aproximacions.

La primera, anomenada UIMS (User Interface Management Systems), que consisteix en un conjunt de programes interactius per dissenyar, prototipar, executar, avaluar i mantenir interfícies d'usuaris, tot intregat sota una única interfície de desenvolupament de diàlegs. La segona aproximació es denominada pels autors caixa d'eines (toolkits).

L'aproximació caixa d'eines consisteix en una biblioteca de programes executables, per implementar les característiques de baix nivell, i que pot ser utilitzada des d'una UIMS o des de qualsevol altre programa.

Les UIMS, malgrat tot, solen tenir una metodologia, un model estructural, notació per al manteniment del cicle de vida, com també prototipatge ràpid.

Els requisits que han d'exigir−se a l'eina pel disseny d'interfícies són els següents: amplia funcionalitat, usabilitat, completessa, integració, extensibilitat, compatibilitat, maneig directe, etc.

Sistemes basats en diàlegs

El dialeg donarà lloc a una serie d'intervencions: isualitzacions per pantalla, emissió de sons, moviments del cursor, apunts amb el ratolí, pulsació de les tecles, etc.

Podem considerar tres factors fonamentals en l'organització dels diàlegs: l'estructura, el tipus i el control.

L'estructur del diàleg implica la definició del components i de les formes vàlides de composició. Existeixen, per descomptat, nombroses alternatives. Podem considerar com a components possibles les classes Conversa, Dialeg, Intercanvi i Intervenció. La intervenció respondria a l'esquema bàsic d'un missatge que un emissor

envia a un receptor. Una sèrie coherent d'intervencions donaria lloc a un intercanvi. Els diàlegs engloben diferents intercanvis referents a un tema. La conevrsa, finalment, pot consistir en un o diversos diàlegs entre diversos interlocutors.

D'aktra banda s'ha de considerar també la tipologia dels diàlegs que reflectiria les seves carcterístiques funcionals.

La forma del diàleg

Un primera classificació faria referencia a les modalitats bàsqiues d'expressió: mode gràfic, mode acústic, interacció amb un terminal, etc.

Dins de cada un d'aquests modes bàsics podríem classificar formes diverses, de les quals destaquem algunes característiques importants:

- Llenguatge natural
- Llenguatges artificials
- Imatges, icones.
- Gest
- Llenguatge musical.
- Coordinació de formes

La connectivitat del diàleg

Ens referim a la qualitat de desenvolupar el diàleg d'una forma fluïda, enllaçant els seus diferents elements en estructures més o menys complexes que incrementin la qualitat final de la coversa.

Podem distingir una connectivitat local que afectaria els elements més proxims en l'espai o en el temps i una connectivitat global que afectaria porcions més amplies de l'estructura del diàleg.

Dins de la connectivitat local podem situar les característiques de seqüència o simultaneïtat, la capcitat de rectifiació, de modificació de la intervenció anterior, etc.

Problemes del disseny d'interfícies

Pressman assenyala una sèrie de criteris pràctics que cal tenir en compte durant el procés de disseny de la interfíce. Es refereixen a la interacció en general i als processos concrets de visualització d'informació per part del sistema i d'entrada de dades per part de l'usuari.

Interacció general

- Consistencia
- Confirmació
- Admetre tornar a enrere en la majoria de les accions.
- Reduir la quantitat d'informació que l'usuari ha de meoritzar entre accions.
- Cercar eficiencia en el diàleg, el moviment i el pensament.
- Protecció enfrot d'errors.
- Categorització de les accions per la seva funcionalitat.
- Organització de les pantalles d'acord amb aquesta classificció.
- Facilitats d'ajuda.
- Temps de resposta. Durada i viabilitat. Missatges informatius sobre l'estat.
- Nomeclatura dels identificadors de les ordres i opcions de menú.

Presentació de sortida de la informació

- Només presentació de la informació rellevant en el cntext actual.
- Ús de presentacions en un format que permeti una assimilació ràpida de la informació.
- Etiquetes consistents, abrujaments estàndards.
- Context visual.
- Missatges d'error.
- Ús de les facilitats tipogràfiques del dispositiu.
- Consistencia en la construcció de les finestres.
- Geografia de la pantalla. Organització de la informació que hi apareix.
- Icones. Forma. Associació a l'objecte evocat. Consistencia.

Entrada de dades

- Minimitzar el nombre d'accions d'entrada que l'usuari ha de realitzar.
- Mantenir la consistencia entre la informció introduïda i la presentada.
- Permetre a l'usuari personalitza la seva forma d'entrada. Ús d'abreviatures.
- Flexibilitat. Multimodalitat. Tecles de funció. Associació menús/ordres.
- Desactivar les opcions no adequades en el context actual.
- Donar facilitat a l'usuari per controlar el flux de diàleg.
- Facilitar ajudes per a totes les operacions d'entrada.
- Facilitar l'entrada de dades (decimals, unitats, informació que el sistema pugui calcular, etc.).
- Facilitat d'aprenentatge

TEMA 8.− PROVES DE PROGRAMES

Introducció

La construcció de programes inclou una serie d'activitats productives en les quals les possibilitats d'incloure errades humanes són enormes. Les errades poden apareixer en els estadis incials de l'activitat de producció de software o més endavant en les fases de disseny o implementació. Es fa, doncs, necessari incloure processos que provin el producte que s'està construint i, si cal, garanteixin el nivell de qualitat desitjat.

El 50% dels recurso totals destinats al desenvolupament d'un sistema e dediquen a la tasca de prova i depuració de programes.

El proces de prova d'un programa

Teasley defineix la prova de programes com el procès d'execució i avaluació, manualment o automàticament, d'un sistema informàtic per comprovar si satisfà determinats requisits o per identificar diferències entre els resultats que produeix i els que s'esperaven.

És doncs, important una revisió acurada de les quatre fases bàsiques de la construcció del software:

Especificació.

Analisi

Disseny

Codificació.

Malauradament, només a nivell de codificació se sol produir una prova més o menys sistemàtica del software. La detecció dels errors d'especificació i disseny es basa gairebé sempre en un seguiment manual dels documents que descriuen el producte en aquests nivells.

Un fet important en relació amb la prova de programes és que la prove només serveix per detectar errors, no per garantir la seva absencia. En aquest sentit, cal distingir els concpetes de prova i verificació. La verificació ens permet garantir que un programa és correcte, és a dir, que és consistent amb una especificació igualment formal.

Existeixen diverses etapes en la prova d'un programa. Si el programa es compon de diverses peces serà necessari provar les diferents peces per separat per comprovar que funcionen integrades.

Per a cada una de les proves que s'han d'efectuar caldrà definir:

- L'abast de la prova.
- El moment d'efectuar la prova.
- El responsable de la prova.

Una practica molt estesa és provar per separat els diferents components i realitzar−ne després una integració massiva (big bang).

Mes bon resultat ens donarà una integració incremental i una prova dels components paral.lela amb la codificació.

En general cal parlar de tres nivells de prova:

- Prova de components.
- Prova d'integració.
- Prova d'aceptació.

Les dues primeres es poden solapar entre si i amb el procés de codificació. En elles, independentment de la intervenció d'altres persones, l'activitat principal la portarà a terme el programador. La prova d'acceptació, en canvi, se sol portar a terme en l'usuari o client.

La prova de components

Cada peça de software ha de ser provada de forma aïllada abans de ser integrada, totalment o parcialment, en el sistema que s'està desenvolupant.

La prova ha de començar per una anàlisi estàtica del codi escrit. Existeixen llenguatges de programació i entorns de desenvolupament que faciliten una codificació i una prova incrementals i/o en paral.lel.

A continuació, i un cop lliure el programa dels errors detectats pel compilador, passem a l'analisi dinàmica del programa.

Dues aproximacions se solen utilitzar: l'anomenada de la caixa blanca i l'anomenada caixa negra. La primera es basa en l'examen de l'estructura interna del programa per deduir el seu comportament (continuació natural de l'anàlisi estàtica). La segona tracta el programa com una caixa negra que només s'examina a través del seu comportament en dierents situacions. Les dues aproximacions no són excloents sinó, fins a cert punt, complementaries.

Tecnica de la caixa blanca

La idea és que, molt sovint, els errors lògics corresponen a interpretacions errònies del propi programadr que hauran de ser posades de manifest. Si tots els camins possibles que la lògica del programa admeti són recorreguts durant la prova podem estar raonablement convençuts que aquestes interpretacions dolentes sortiran a la llum. Igualment l'existència d'errors tipogràfics (no detectats pel compilador), que en principi tindrien una distribució uniforme en el text del programa, es pot detectar usant aquest criteri.

Pressman, per exemple, proposa les normes següent:

- Garantir que tots els camins independents possibles es recorren almenys una vegada.
- Garantir que davant una decisió tots els camins se segueixen.
- Garantir que tots els bucles s'executen, tant en els seus límits com en el seu interior.
- Garantir que totes les estructures de dades es proven completament d'acord amb la seva casuística.

La confecció de joc de proves

Denominem joc de proves un conjunt de casos que desitgem provar. Cada cas correspon a un conjunt de valors d'algunes de les variables del programa. Les variables i els valors es trien de forma que en executar el programa es recorrin determnats camnis i s'executin determinades instruccions.

Partirem d'un diagrama de flux simplificat en el qual s'assenyalen els elements bàsics implícits en les normes que presentàvem abans (opcions, bucles, seqüències). Podem trobar un exemple a pagina 212.

Quna serien els camins que cal seguir en cada cas?

En la instrucció **if** C1 **then** S1 és evident que hi ha dos camins possibles: el corresponent a fer certa la condició C1 i el corresponent a fer−la falsa.

Un cop obtingut el graf i determinats els cammins de flux que cal recórrer hem d'abordar la tasca de derivar el joc de proves. Per descomptat podem operar a base d'assignar a cada camí possible un cas, és a dir, un element del joc de proves. Es tracta d'una possibilitat acceptable i potser la mes sistemàtica.

Conjunt base (Basis set)

Tom McCae proposa un metode interessant per implementar aquesta tècnica. Es denomina Basis Path Testing. El metode consta de les etapes següents:

- Construir un graf (flow path) del programa, simlar al que hem vist, a partir del seu codi.
- Aplicar una mesura de la complexitat lògica del graf resultant. La complexitat ciclomàtica (CC) es pot calcular de diverses formes.

CC = nombre de regions aïllades del graf.

CC = nombre d'arcs−nombre de nodes +2

CC = nombrre de nodes lògics +1

- Seleccionar un conjunt de camins (Basis set) en el graf des del node inicial al final de foma que es recorrin tots els arcs. El nombre de camins serà CC.
- Construir el joc d'assaig de manera que tots els camins del conjunt anterior s'executin.

Flux de dades

Un mètode de caixa blanca diferent a l'anterior i en certa mesura complementari seu és el de flux de dades. Es tracta de localitzar quins punts en el programa donen valor a alguna de les variables i quins consumeixen o utilitzen aquesta variable. Així, per a cada instrucció S del programa definiríem un conjunt DEF de variables que prenen valor en S i un conjunt USE de variables que són usades en S.

A continuació es defineixen cadenes DU (Definició−Us) de la forma següent: (x, S1, S2) és una DU per a la variable X si:

- X pertany a DEF (S1)
- X pertany a USE (S2)
- La deinició de X donada en S1 és encara vàlida en S2.

El mètode consisteix a obligar a cobrir almenys un cop cada una de les cadenes DU existents.

Prova de bucles

Pressman distingeix 4 classes de bucles:

- Bucles imples.
- Bucles niats.
- Bucles concatenats.
- Bucles no estructurats.

Pels bucles simples Pressaman proposa provar el pas a través del bucle sense executar el seu cos, els passos fronterers (primer i segon, ultim i penúltim) i un pas intermedi. El problema és que de vegades no és trivial trobar aquests passos i executar−los de forma aïllada. Si el bucle tracta un vector per mitjà d'una instrucció for llavors és senzill, si es tracta d'un while o un repaet no ho és.

El procediment de prova dels bucles niats consisitiria en un proces de dins enfora. En la prova de cada bucle es considerarien provats els bucles interior a ell assignant−los valors típics mentre que els bucles més externs prendrien valors inicials.

Tècnica de la caixa negra

Aquesta aproximacó no té en compte l'estructura interna el mòdul que s'ha de provar sinó només la funcionalitat que persegueix. Es tracta de provar per un ombre suficient de casos que totes les funcions que el modul ha de cobrir ho són a satisfacció.

El mètode consta de quatre etapes:

- Partició del domini de definició de cada variable en un conjunt quocient en el qual cada classe reflecteixi pautes de comportament uniformes.
- Selecció d'un representant de cada classe.
- Combincació de les classes per formar jocs de prova.
- Execució de la prova, comprovació de resultats i altres activitats.

La primera fase, doncsm consisteix a classificar, es a dir, establir una relació d'equivalència sobre el domini de cada variable de forma que cada un dels valors possibles quedi assignat a un dels elements del conjunt quocient.

Teasley apunta la possibiltat d'utilitzar taules de decisió com una eina d'ajuda a la planificació dels casos que s'han d'examinar en l'aproximació de caixa negra.

Característiques de la prova de programes en temps real

La prova d'un programa en temps real implica tot el que hem vist fins ara i a més algun dels elements següents:

- Seqüenciació temporal (timing) de les dades.
- Problemes de bloqueig entre els recursos utilitzats per cada procés.
- Paral.lelisme dels processos. Problemes derivats de la interacció entre processos.
- Relació entre el software en temps real i el seu entorn d'execució.

L'estratgia de la prova de programes

Hem vist en les seccions anteriors un repertori de tècniques per a la prova d'unitats de software. Teasley es refereix a aquest tipus de proves com testing in the small per al.ludir al caracter atomic de les unitats que es consideren. Donat que un sistema complex consta d'un conjunt de tals unitats amb relacions molt sovint complexes es fa necessri integrar els processos de prova individuals en una serie d'etapes correctament planificades. L'àmbit de la prova és ara tota l'estructura que s'ha de provar (testing in the large).

Criteris generals de l'estrategia de prova

Una serie de consideracions generals es poden esmentar:

- la prova comença a nivell de les unitats i finalitza amb la integració d'questes en el producte final.
- No sol existir un metode o una aproximació que pugui utilitzar−se sistemàticament en totes les fases del procés de construcció del programa.
- La prova del sistema implica l'actuació del programador però, també, d'altres persones.
- La prova i la depuració són activitats diferents i molt sovint són desenvolupades per diferents persones.

Criterirs per a les proves de validació i acceptació

La prova del programa no es limita a la prova de les unitats i a la seva integració.

Pressman, parla de verifiació versus validació (V&V) i matisa els dos termes. El primer respondria a la pregunta:

Hem construit el producte correctament?

Mentre que el segon respondria a l pregunta

Hem construït el producte correcte?

És a dir, la validació faria referencia a la confrontació entre el producte construït i el que s'esperava d'ell. Els instruments bàsics per aquesta validació serien:

- El producte software complet.
- L'especifiació de requisits funcionals i operacionals.
- La documentació d'usuari.

Una última prova, abans de sotmetre el producte al test d'acceptació per part del client, é la prova en el sistema on ha de funcionar.

Altres proves de validació serien les següents:

- Prova de recuperació (es produeix la fallida del sistema i mirar si aquest es recupera be).
- Prova de seguretat
- Prova de potència (Stress Testing), sotmetre el programa a condicions de uncionament extremes (però possibles).
- Prova d'eficiencia.

Quan acaba la prova del programa?

Pressman cita alguns casos que podríem denominar aproximació cínica a la prova de programes:

La prova no acaba mai. El que succeix és que el provador deixa de ser el programador per passar a ser el client. La prova acab quan s'acaben el temps o els diners que l'hi són assignats.

Les proves d'acceptació

L'últim pas de la prova el constitueix el test d'acceptació. La diferencia essencial entre la prova d'acceptació i les anteriors és que és el client, o usuari final, qui la realitza.

Es tracta d'una prova del tipus de caixa negra sobre tot el software integrat. La documentació bàsica per aquesta prova serà la d'especificció i requisits i el manual d'usuari.

En cas de construir software dirgit a un nombre potencialmet alt d'usuaris se sol parlar de proves alfa i beta. Es tracta de proves d'acceptació controlades. El software en distribueix a un nombre limitat de provadors que amb (proves alfa) o snse (proves beta) presencia del programador fan proves lliurement i en comuniquen els resultats.

Les proves d'integració

La unitat de prova

Quan tractem de provar un mòdul o una unitat de programa ham de tenir en compte que aquest mòdul estarà connectat amb altres amb els quals intercanviarà informació.

Si l'estructura modular és de tipus arborescent, o podem establir una jerarquia modular, elmodul que tractem de provar serà utilitzat per un o més situats en nivells més alts de la jerarquia i alseu torn podrà utilitzar mòduls situats en nivells inferiors. Mentre no estiguin provats els mòduls superiors se substitueixen per conductors (DRIVERS) l'única missió dels quals és proporcionar la interfície amb el modul actual. Els mòduls inferiors també se substitueixen per maquestes (STUBS) amb identica missió.

Si l'stub substitueix una funció que torna un resultat únic a partir d'unes dades d'entrada, podem implementar−la sense problemes greus. Una consulta a l'usuari per tal que aquest proporcioni aquest valor és també una implementació acceptable.

L'estrategia de les proves d'integració.

El que es pretén amb les proves d'integració es determinar l'ordre en què es van a construir i provar els mòduls i garantir que la integració sigui correcta.

El més raonable és que la integració es produeixi de forma icremental (enfront al Big Bang) i que la prova d'integració es porti a terme en cada etapa, utilitzant mòduls ja provats individualment. És evident que laprova individual dels mòduls no garanteix el que el funcionament conjunt sigui correcte.

Existeixen dues estratègies principals d'integració, la descendent (Top Down) i l'ascendent (Bottom Up).

Estrategia descendent

En aquesta estrategia el procés de prova es porta a terme en forma descendent, començant pel model de control que molt sovint existeix en el nivell més alt de la jerarquia.

No existeixen drivers en aquesta aproximació. Al principi el mòdul de control assumeix el paper de driver i tots els mòduls anomenats per ell se substitueixen per stubs. Es realitza aleshores la prova unitària del modul en qüestió.

A continuació el proces consisteix a anar substituint progressivament els stubs per modul eals, provant en cada pas la unitat corresponent. L'ordre de selecció dels stubs que substituïm no és arbitrari. Podem utilitzar una estratega de recorregut en profunditat (DFS) o de recorregut en amplada (BFS).

El procés conclou quan tots els stubs han estat substituïts per moduls reals.

El principal problema amb que ensopega aquest sistema és la construcció d'stubs.

Un avantatge evident és que, en disposar des de l'inici d'un sistema visible complet, el client o l'usuari poden, des d'estadis inicials del desenvolupament del producte examinar i criticar les seves funcionalitats i interfície.

Un avantatge és la d'una distribució més uniforme dels recursos de computació al llarg del temps de desenvolupament.

L'avantatge més clar és la detecció més ràpida dels errors més greus que, molt sovint, apareixen en els moduls de nivell més alt.

Un problema evident és que, en figurar les operacions d'entrada/sortida en mòduls de baix nivell. L'ús d'una estrategia descendent dificulta la comunicació tant d'entrada com de sortida amb l'usuari.

Estratègia ascendent

En aqueta estratègia es parteix de la prova de mòduls terminals que s'agrupen en unitats més complexes. Es construeix aleshores un driver que coordia la prova de les unitats agrupades.

El proces continua de forma ascendent eliminant els drivers que coordina la prova de les unitats agrupades.

Si en el cas amterior el poblema esra la construcció d'stubs, ara ho és la construcció de drivers.

Sol ser més fàcil, en aquesta aproximació, la detecció i la correcció dels errors descoberts, es a dir, la localització en el text del programa dels errors que han pogut motivar els simptome que hem apreciat.

Un altre avantatge és que els moduls de nivell baix, en els quals hi sol haver més errors passen més proves que els de nivell alt.

Aproximacions híbride

Existeixen metodes híbrids que combinen ambues aproximacions. Es tracta d'una solució aconsellable quan hi ha diferencies grans en la complexitat del mòduls i convé començar provant quelles que poden resultar crítics.

Teasley suggereix l'agrupació de les unitats de prova en forma de paquets de mòduls`lligats per relacions

d'utilització que implementin funcionalitats clarament definides.

Les proves d'integració i l'orientació a objectes

Les unitats han de ser provades individualment i després integrades.

Quan la unitat usada no havia estat construïda o no havia estat provada, esr substituida per un stub. Quan la unitat que usava no havia estat construïda o provada esra substituïda per un driver. Les estrategies presentades, ascendent, descendent o híbrida, només ens definiem criteris per dur a terme aquest procés.

Si ens situem en un entorn d'analisi, disseny i programació orientats a objectes la situaciócanvia radicalment. La unitat de prova en aquest cas l'objecte i les relacions entre els dierents objectes poden ser molt més variades.

La mateixa metodologia ens imposa una classificació dels objectes d'acord amb la seva funció d'aplicació:

- Classes bàsiques.
- Objectes controladors.
- Objecte suport d'interfícies.
- Objectes gestors de disc.

Les classes bàsiques haurien de ser construides i provades en primer lloc. L'ordre lògic de construcció i prova de la resta de classes seria el presentat més amunt (controladors, interfície, gestió de disc) encara que sempre hem de tenir en compte el grau d'acoblament i la utilització dels diferents objectes.

Les relacions d'agregació han de ser considerades de forma prioritaria a l'hora de construir i provar els objectes.

L'ordre lògic de prova seria la construcció dels agregats i la seva prova mitjançant un driver i posteriorment la construcció i la prova de la classe contenidora.

Pel que fa a l'estrategia d'implementació i prova dels objectes lligats per relacions d'herencia, l'estrategia bàsica consistirà a començar la prova per un descendent. A continuació passariem a l'ascendent buidant el descendent de tot allo, atributs i serveix, que li es especifica. Un cop provat l'ascendent podríem provar la resta dels seus descendents afegint en cda cas les propietats que els son pròpies.

La depuració (el dubugging)

Podem recorrer a tres formes d'abordar la localització dels errors.. En totes elles ens recolzem en tècniques com l'execució simbòlica del programa, el seguiment del codi, etc. A vegades recorrirem a proves adicioals, traces, modificacions del codi incloent−hi missatges en lguns punts, abocats de memòria, etc.

La primera forma (Brute Force Approach) es basa en un seguiment sistemàtic del funcionament del programa per tractar de descobrir l'anomalia.

La segona és la ècnica del Backtracking. Es tracta de retrocedir en el programa des del punt en què es va descobrir el símptoma confiant que l'error no vagi lluny. El problema és que, molt sovint, el símptoma detectat no és conseqüència immediata de l'error sinó d'algun efecte lateral d'aquest error.

La tercera aproximació tracta de localitzar punts en el programa on es poguessin produir les causes que motivaren els efectes detectats. En aquest cas es tractaria de detectar les dades associades amb el símptoma i cercar els punts de possible assignació errònia de valors a aquestes dades.

Un altre punt important en la depuració és que malgrat que l'error i la seva correció siguin local el procés pot tenir conseqüències més amplies.

Una tècnica interesant quan es tracta de loclitzar un error dins d'un modul concret és substituir tota o part de la seva interfície per drivers o stubs que es poden fer servir per dur a terme proves addicionals.

TEMA 9.− LA QUALITAT DEL SOFTWARE

Introducció

Tots seríem capaços d'expresar tota una llarga llista de qualitats que un programa ha de poseeir: que sigui correcte!!, que sigui eficient, que sigui fiable, que estiguiben documentat, que sigui facilment intel.ligible i modificable, etc. Potser altres qualitats ens siguin més llunyanes, i en canvi, més properes alsàmbits de gestió: que el sistema estigui construit amb el mínim de mitjans, que sigui atractiu des del punt de vista comercial, que estigui protegit correctament, etc.

L'única exigencia de qualitat que sovint e planteja a un programa es que funcioni, l'única queixa es que no funciona adequadament.

La qualitat del software

Per a Pressman existeixen tres factrs bàsics que cal utilitzar per caracteritzar la qualitat del software:

- El patró bàsc per contrastar la qualitat del software resideix en l'especificació dels requisits, tant funcionals com operatius.
- Es important comprovar la qualitat no només del producte final sinó també del seu procés de construcció.
- Existeixen una sèrire de factors de qualitat que normalment no s'expliciten (formen part del sentit comú del constructor o del client o d'ambdos). Que aquests factors no s'explicitin ni vol dir que no s'hagin de tenir en compte.

Els factors de qualitat del producte no són els mateixos des del punt de vsta del client que ho compre, de l'usuari que haurà d'utilitzar−lo o del tècnic que eventualment haurà de mantenr−lo o modificar−lo.

Així Teasley assenyala que per al client els aspecte economics, de facilitat d'adaptació o de seguretat seran els més importants. L'usuari, valorarà més els aspectes lligats a la utilització del producte. El tècnic que ha de mantenir el programa en valorarà, en canvi, l'absencia d'errades, documentació, llegibilitat del codi, qualitats de disseny.

D'altra banda, no totes les qualitats són exigibles en la mateixa mesura: Així, la transportabilitat d'un programa és una qualitat en si però deixa de tenir importancia si el programa no hade ser previsiblement transportat a un entorn diferent d'aquell per al qual ha estat construït.

Factors de la qualitat del software

Podem considerar dues dimensions en les quals classificar els factors de qualitat: una estatica i una dinàmica.

Podríem definir la dimensio estàtica com la cpacitat de funcionament del programa d'acord amb les especificacions funcionals i operacionals que en el seu moment es formularen. La dimensió dinàmica faria referència a les qualitats que afavoririen els canvis del producte.

McCall classifica en tres grans grups els factors de qualitat del software d'acordamb:

- La seva operació.
- La seva modificabilitat.
- La seva adaptabilitat a altres entorns.

El primer correspondria a la dimensió estàtica i els altres dos a la dimensió dinàmica.

Factors lligats a l'operativitat del producte

- Correcció: grau de satisfacció dels requisits funcionals.
- Fiabilitat: grau de confiança en un funcionament correcte.
- Eficiència: quantitat de mitjans que el sistema necessita per acomplir la seva funció.
- Integritat: nivell de control sobre l'accés a les dades o les funcions per part de persones no autoritzade.
- Usabilitat: esforç que l'usuari ha de fer per aprendre a usar, utilitzar, preparar les entrades o nterpretar les sortides el programa.

Factors lligats a la modificabilitat del producte

- Mantenibilitat: esforç necessari per localitzar i eliminar un error.
- Flexibilitat: esforç necessari per modificar el programa.
- Facilitat de prova (Testability): esforç necessari per provar el programa.

Factors lligats a la transportabilitat del producte

- Portabilitat: esforç necessari per transportar el programa a un entorn diferent d'aquell per al qual va ser construït.
- Reusabilitat: capacitat de reutilització del programa en altres aplicacions o per a d'altres programes.
- Interoperabilitat: esforç necessari per acoblar el sistem amb un altre.

Mesures de la qualitat del software

McCall proposa una sèrie de mesures de qualitat que poden utilitzar−se com a base per a l'avaluació dels factors de qualitat. S´n els següents:

- Auditabilitat: facilitat en la confrontació del producte amb una normativa.
- Precisió: nivell de precisió en els càlculs i el control
- Estandarditació de la comunicació: nivell d'ús d'interfíce o protocols etàndars.
- Completessa: grau de completessa d'implementació de les funcionalitats previstes.
- Consistència: uniformitat en el disseny i la documentació al llarg de tot el procés de desenvolupament del projecte.
- Estandardització de les dades: és de tipus i d'estructures de dades estàndards.
- Tolerancia a errors: grau de danys produït en trobar un error.
- Eficiencia en l'execució: quantitat de recursos utilitzats en executar−se
- Expandibilitat: grau d'extensibilitat del disseny arquitectònic, de dades o e procediments.
- Generalitat: amplitud de les possibles aplicacions dels components del programa.
- Independencia del hardware.
- Instrumentació: grau de monitoritació de l'operació per part del programa.
- Modularitat: independecia funcional dels components
- Operabilitat: facilitat d'operació.
- Seguretat: diponibilitat de mecanismes de protecció de dades i programes.
- Autodocumentació.
- Simplicitat: grau d'intel.ligibilitat.
- Independencia del software.

- Traçabilitat.
- Facilitat d'entrenament: grau d'assistència del programa al nou usuari.

Aspectes de qualitat del software lligats a l'orientació a l'objecte

Ja hem assenyalat repetidament que l'objectiu fonamental de l'orientació a l'objecte és la reutilització dels objectes. Qualsevol mesura de la qualitat en relació amb la modelització del domini o amb l'analisi, el disseny i la programació orientades a objectes ha de partir d'aquest objctiu bàsic.

Des del punt de vista de les classes definides hem de tenir en compte:

- El nombre de classes. Un model amb un gran nombre de classes pot dificultar la comprensió global del sistema però facilita la reutilització.
- La nomenclatura utilitzada.
- El nombre de classes abstractes i les seves relacions. En general és preferible una estructura profunda, amb moltes classes intermedies que facilitin la reutilització que una estructura aplanada.
- La informació continguda a les classes. Hem d'evitar un ombre excessiu d'atributs.

Pel que fa als serveis, podem descriure una serie de criteris:

- Serveis coherents, petits i consistents. (Coad obté una grandaria mitjana de 5,5 linies de codi per servei)
- Els protocols de comunicació entre objectes han de ser tan senzills som sigui possible. Coad recomena limitar a tres el nombre de paràmetres.

Pel que fa a la organització general del model, també podem proposar una serie de sriteris:

- Convé afavorir la utilització d'una nomenclatura consistent entre les classes com també la definició d'un serveis generals comuns a totes les classes.
- El nombre de subsistemes que es detectin entre les classes és important. Un sistema amb molts subsistemes i poques classes aïllades sol ser millor per encapsular les funcionalitats de l'aplicació.
- S'ha d'afavorir la creació de frameworks. Un framework és un conjunt de classes que proporcionen esquelets per families d'aplicacions, lligades a un domdini especific o a funcionalitats sembants.

Control de la qualitat del software

El proces de control de qualitat (SQA: Software Quality Assurance) es relitza al llarg del procés de producció tot i que molt sovint les tasques que criden més l'atenció, com la prova d'acceptació dell producte, tenen llocjus abans de lliurar−lo.

Allen Macro distingeix tres conceptes diferents en relació amb aquest tema:

- El control de qualitat (QC: Quality Control)
- La inspecció de la qualitat (QI: Quality Inspection)
- La garantia de qualitat (QA: Quaity Assurance).

Els dos primers es refereixen a operacions i activitats que es porten a terme al llarg de tot el procés de desenvolupament del software mentre que la QA representa un control final de la qualitat del proucte lligat a la prova d'acceptació.

Del dos primers controls, el primer agrupa als mètodes subjectius i poc quantificables. El segon es refereix a mètodes objectius de control de qualitat: revisions tècniques formals, verificació de l'aplicació d'estàndards,

verificació del seguiment de la metodologia pertinent, etc.

Mesures objectives de la qualitat s'han intentat amb resultats incert. Pressman, per exemple, cta el DSQI (Design Structure Quality Index) utilitzada per la Força Aèria dels EUA.

El DSQI proporciona una xifra basada en dades objectives: nombre de mòduls del programa, nombre de mòduls el funcionament correcte dels quals depèn de processos anterior, nombre d'ítems de la BD, nombre de mòduls amb una sola entrada i una sola sortida, etc.

En funció d'aquestes dades en calculen una sèrie de relacions que hàbilment ponderades, ens donen a la fi un índex de qualitat (una xifra).

PP−5

SISTEMA ORIENTAT A L'OBJECTE

Polimorfisme

Encapsulació

Abstracció

Triangle orientat a objectes

CLASSE 1

Atribut 1

Atribut 2

Atribut 3

CLASSE 3

CLASSE 4

Atribut A

Atribut B'

Atribut C

Servei X'

Servei Y

CLASSE 2

Atribut A

Atribut B

Servei X

Servei Y

Conexió d'instància

Inclusió o agregació

Conexió de missatge

Classe 4 hereda de classe 2

Classe 4 és subclasse de classe 2

Notació gràfica per a models orientats a objectes

[1..4]

[1..1]

1

N