

Metodologia: especificació, anàlisi i disseny.

Una **metodologia** pretén oferir una forma sistemàtica de procedir cosa que ha esdevingut imprescindible de cara al treball en equip. Sovint es basen en l'experiència, però no són panacees, tot i que faciliten les coses no fa bo un tècnic poc competent. (Hi ha elements dels que fan bo un tècnic que són de difícil adquisició, com l'ordre mental, i d'altres que són personals i intransferibles, com l'experiència.)

La metodologia que es proposa a l'assignatura se sustenta sobre el cicle de vida clàssic i s'hi aplica:

- UML (llenguatge unificat de modelatge) tant en els diagrames de casos d'ús, com en els diagrames de classes.
- Directrius per a primers projectes orientats a objectes (Wirfs-Brock, Wilkerson i Wiener)
- Arquitectura en tres capes en l'etapa de disseny.
- Avancem a l'especificació la creació del model del domini, és a dir, de l'estructura de dades. Com que es treballa en OO aquesta estructura de dades fa de nexce d'unió entre les quatre primeres etapes (especificació, anàlisi, disseny i programació) i dóna suport a la transició transparent.

L'etapa d'**especificació** és bàsica per a l'obtenció d'un bon producte ja que consisteix en definir-lo, en establir què ha de fer amb el programa o programes resultants.

Es requereix de la intervenció o col·laboració del client o futur usuari del *software* resultant ja que n'és l'expert, és qui sap com funciona el negoci al qual es destinarà el producte final, mentre que normalment el tècnic d'informàtica (TI) ho ignora. Moltes vegades la figura de l'usuari (l'expert) i la del client (qui paga) no coincideixen en la mateixa persona. El client sempre té el màxim interès en l'èxit del projecte. Es tracta d'aconseguir que l'usuari o expert tingui ganes de col·laborar i interès en l'èxit del projecte (moltes vegades n'hi ha prou en fer-li veure que l'èxit del projecte és d'ell, en cedir-li el protagonisme).

Una primera dificultat és que avui dia no se sol partir de zero, el client acostuma a disposar de processos informàtics, moltes vegades força bons, que estan donant un servei prou satisfactori i es tracta de substituir-los i millorar-los afegint-hi funcionalitat i tecnologia. Però és força freqüent que els objectius no estiguin prou ben definits, per exemple, es planteja de millorar la "fidelització" de clients oferint serveis addicionals a la venda, però no es concreta quins poden ser aquests serveis.

Convé fer una crítica seriosa del sistema actual, buscant els aspectes no coberts i necessitats no satisfetes. També és convenient una actitud crítica amb les propostes que es puguin anar fent, no quedar mai completament satisfet del tot amb les decisions que es van prenent valorant sempre els punts a favor i en contra de cada una.

A més, convé valorar les limitacions del nou sistema proposat, els aspectes que no cobreix i aquells que es deixen com a possible ampliació.

Un altre factor de dificultat són les diferències en el llenguatge. La terminologia de l'usuari i del seu negoci no té perquè resultar familiar al TI, ni la de les tecnologies de la informació (informàtica i telecomunicació) resultar-li familiar a l'usuari ni al client. Això pot dificultar una comunicació correcta, cosa que es pot pal·liar amb l'es-

tabliment de glossaris. Un abús de la terminologia professional pot resultar pedant i, per tant, molestar. Convé evitar els defectes en el tracte amb el client com els que pot produir la prepotència que massa vegades caracteritza als TI, augmentada per aquest abús de terminologia de les tecnologies de la informació.

Un altre aspecte que sol col·laborar amb aquestes dificultats és l'ús imprescindible del text lliure. L'usuari sol tenir serioses dificultats per entendre els formalismes relacionats amb les tecnologies de la informació i només ser capaç d'entendre les descripcions que li puguin fer usant llenguatge natural, oral o escrit, del sistema de *software* que s'està definint. Un TI farà l'especificació del sistema amb els formalismes adequats, però, malauradament per a ell, també haurà de fer la descripció del sistema perquè pugui ser compresa i acceptada pel client (per anar bé, s'ha de tenir l'especificació acceptada pel client), és a dir, en llenguatge natural i sense formalismes tècnics, en tot cas amb terminologia professional de l'usuari, i per escrit.

Els principals vicis que pot tenir un **text lliure** en llenguatge natural, a banda dels de mala redacció (que resulti incompreensible), es podrien resumir així:

- Soroll, informació innecessària.
- Sobredefinició o sobreespecificació, massa detall (detall innecessari).
- Silenci, manca d'informació.
- Ambigüitat, manca de concreció o hi ha falses descripcions (ús de sinònims).
- Penediment, rectificar el que s'ha dit.
- Referències avançades, referir-se a aspectes encara desconeguts pel lector, tot i que s'expliquin més endavant.

Un text tècnic es redacta sempre en forma impersonal i ha de tenir una estructura precisa, consta com a mínim de tres parts: introducció, discussió o justificació i conclusions.

La introducció serveix per situar el lector en el document, és a dir, explica quin tema s'aborda, "de què va". Pot tenir forma de memòria, recollint els antecedents que li donen sentit, no és més que una relació històrica de fets que duen al tema del document. Aquest tipus d'introducció té també molt de sentit quan es refereix a algun aspecte que forma part d'un conjunt més gran o complex. Una introducció pot limitar-se a ser una simple declaració dels objectius del document, per exemple: "en aquest document s'especifica del programa <tal> necessari <pel que sigui>".

La discussió o justificació és la part central i més complexa dels document, normalment, i si es prou gran, s'estructura en apartats i subapartats. Contindrà un exposició de la problemàtica, de les diferents alternatives de solució estudiades, si n'hi ha, i es justificarà als ulls del lector les conclusions que es plasmaran a l'apartat següent. L'estudi crític del sistema actual s'haurà d'incloure en aquest apartat de l'especificació, normalment serà la justificació d'algunes de les conclusions.

Finalment, les conclusions, ho són de tot el que s'ha discutit i justificat en el document, només hi han d'aparèixer els aspectes tractats en l'apartat de discussió. Les conclusions del document d'especificació seran la descripció detallada del nou sistema: funcionalitat i altres requisits, i es diferenciarà els aspectes que puguin ser opcionals, és a dir, que poden quedar sense fer perquè no són imprescindibles.

Tot document escrit, especialment si té més de tres planes, ha de disposar sempre d'un índex que permeti al lector conèixer l'estructura del document i accedir un a partat concret amb rapidesa.

Per començar a fer un text d'aquest tipus pot ser convenient començar amb un esquema, sobretot si es té poca experiència. Aquest esquema sovint servirà de sumari o índex del document. A vegades resulta més còmode començar aquest esquema pel final, per les conclusions on es vol arribar, sobre elles estructurar la discussió i justificació de cada una d'elles. Un bon esquema, a més, permet ordenar el discurs en la seqüència que es consideri més assimilable o que faciliti una millor compressió.

L'**especificació** està encara lluny de la implementació (malgrat que potser ara us pugui costar força de separar una cosa de l'altra) ha de definir el què ha de fer el programa, i no ha d'ocupar-se de com es farà, és a dir, dels detalls d'implementació. Per tant, no es pensa en tal o qual llenguatge de programació, sistema operatiu, sistema de fitxers o bases de dades, tot i que moltes vegades no es pugui evitar de tenir-ho al cap i d'anar pensant què s'hi adequa i què no.

En aquesta etapa, la d'especificació, el *software* s'ha de veure com una part d'un sistema d'informació i convé especificar el *software* com un component del sistema complet. Per tant, haurà de tenir en consideració l'entorn on operarà: els diferents usuaris, ubicacions físiques reals, els procediments complets, etc.

Com que els TI es mouen millor en el terreny dels llenguatges formals, se sol treballar amb llenguatges formals d'especificació per definir el programa i, després, muntar la descripció textual del sistema a partir d'aquesta descripció formal. De fet, el procés de construcció correcte és iteratiu: el TI estudia una especificació que presenta en forma textual al client, aquest li fa les observacions adients i el TI revisa el seu treball en conseqüència i torna a presentar-li, fins que s'arriba a un resultat prou satisfactori i consensuat.

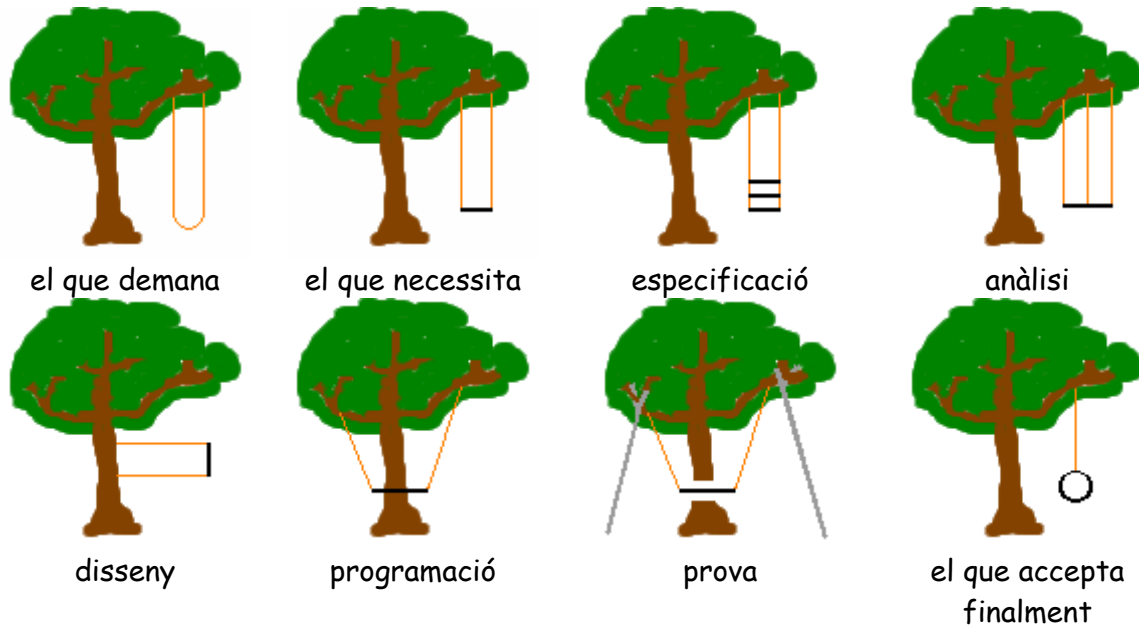
És força habitual que en aquesta etapa s'usin prototipus (muntatges que permeten navegar per les vistes del programa (menús, finestres, formularis, llistats, etc.) deixant veure com es comportarà aquest un cop fet) que donen al client una idea força aproximada de la realitat del producte final. Evidentment l'ús de prototipus requereix el disseny previ de les vistes. Anteriorment, per donar aquesta idea de com havia de ser el programa també s'havia fet ús d'un esquema o primer esborrany del manual de l'usuari.

Per a la identificació del problema que ens presenta l'usuari, en primer lloc, cal tenir ben presents les seves especificacions, que seran el punt de partida. Convindrà incloure-hi l'estudi del sistema actual (el que es fa manualment i la part mecanitzada). D'aquí n'extraurem el que vol el client de forma explícita. En aquest punt voldria fer una reflexió basada en les diferents visions del problema:

1. El que vol el client.
2. El que en realitat necessita.
3. El que interpreta el TI que cal fer.
4. El que realment és el producte final.

Evidentment una diferència entre la visió del TI del producte (3) i el que realment és el programa final (4) només posaria de manifest incompetència professional, per tant, hem de considerar que 3 i 4 han de coincidir sempre. La dificultat és encertar a

fer coincidir el que necessita (2) amb el producte final (4). És a dir, que l'objectiu de l'especificació és esbrinar què necessita realment. No oblidem que també convé deixar content al client amb el que volia (1) fent-ho quadrar amb el que li calia (2). Un acudit clàssic:



Per complementar la informació subministrada pel client i amb l'objectiu de saber que necessita convindrà estudiar tota la documentació existent que es pugui aconseguir, tan normativa (legislació, estàndards, etc.) com descriptiva (bibliografia).

En segon lloc, convé consultar altres experts per conèixer altres formes de fer.

Finalment, convé revisar sistemes semblants existents en el mercat que si més no serviran per treure idees. Podem trobar que el producte ja existeixi i deixi de tenir sentit fer-lo de nou, o que n'hi hagi que cobreixin parcialment el que busquem i que és puguin adaptar, o no, al problema plantejat. També pot passar que no se'n trobi cap relacionat amb el sistema que s'està plantejant cosa que avui és força estranya i convindria veure'n la raó.

Òbviament, aquests aspectes els heu de tenir presents en el vostre projecte. Un cop us hagi explicat que volem us convindria esbrinar altres possibilitats pel programa buscant documentació, altres experts i productes al mercat.

L'especificació del programa es fa amb 4 documents, tres d'expressió formal i un text descriptiu per al client. Un dels documents d'expressió formal és un diagrama de casos d'ús que descriu en UML la funcionalitat del sistema, un altre és el diagrama de classes també en UML, i completa la terna una llista de requisits no funcionals (RNF) que recull tots els requisits del sistema no recollits amb la seva funcionalitat, com pot ser el sistema operatiu on s'executarà o el temps de resposta.

El **diagrama de casos d'ús** recull la funcionalitat del sistema, és a dir, tot allò que els usuaris del sistema han de poder demanar-li o fer-li fer.

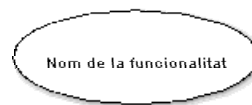
Quan parlem de funcionalitat cal tenir present que tot sistema té dues classes de funcionalitats: les de l'usuari i les del sistema. Les primeres són les que l'usuari ne-

cessita per al seu treball i que demana explícitament o implícita, mentre les segones són aquelles que es necessita que tingui el sistema per satisfer aspectes de funcionament no lligats a les funcionalitats que requereix l'usuari. Aquestes a vegades coincideixen amb els RNF. Un exemple de funcionalitat del sistema pot ser una funcionalitat per fer de còpies de seguretat específica de l'aplicació, un altre, la gestió de la seguretat en l'accés (control d'accés dels usuaris).

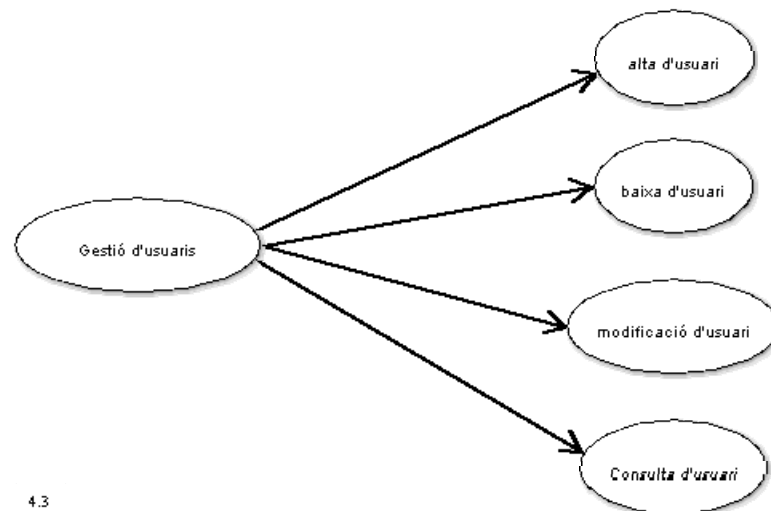
En el diagrama es plasmen totes les funcionalitats que haurà de tenir el sistema i a quins usuaris se'ls permetrà fer-ne ús.

Un diagrama de casos d'ús consta de tres tipus d'elements: funcionalitats, actors i les relacions entre ells.

Una funcionalitat o cas d'ús en UML es representa:



En principi n'hi ha una per cada funcionalitat, però a vegades es creen elements d'aquest tipus per agrupar les funcionalitats per categories o famílies. Per exemple:



Aquest seria l'ABM (alta, baixa i modificació) de l'usuari.

L'especificació del cas d'ús es completa amb el diàleg, que explica el comportament de la funcionalitat. La descripció del diàleg complet és objecte de les assignatures d'enginyeria del software, aquí només us demanem un resum del diàleg. Aquest consisteix en explicar el comportament normal o regular (diàleg típic) del cas d'ús i d'altra banda els comportaments excepcionals o irregulars (errors possibles i cursos alternatius). Aquest diàleg ha de donar una idea de com es comportarà el cas d'ús, a l'assignatura no es demana molt detall, però cal que doni una idea mínima del com funciona. Per exemple, en l'alta d'usuari podria ser:

- Comportament: un cop l'actor ha triat fer una alta, el sistema demana les dades següents: el nom i cognoms de l'usuari, el codi d'usuari (*username*), la contrasenya (*password*) -no es mostra, es demanarà dos cops- i el tipus d'usuari -que s'haurà de triar d'una llista desplegable inclosa en el formulari d'entrada. El sistema valida els valors i coherència de les dades entrades, és a dir, que no

existeixi en el sistema un usuari amb aquest codi d'usuari, i que les dues contrasenyes entrades coincideixen. Per acabar, el sistema enregistra les dades.

- Errors possibles i cursos alternatius:
 1. Quan existeixi en el sistema un usuari amb el codi proposat: avisarà i tornarà a demanar un codi d'usuari, oferint l'opció d'abandonar la funcionalitat sense fer l'alta d'usuari.
 2. Quan les dues contrasenyes entrades no coincideixin, traurà un avís i les tornarà a demanar de nou dues vegades.

Si ho voleu, es podria sintetitzar més:

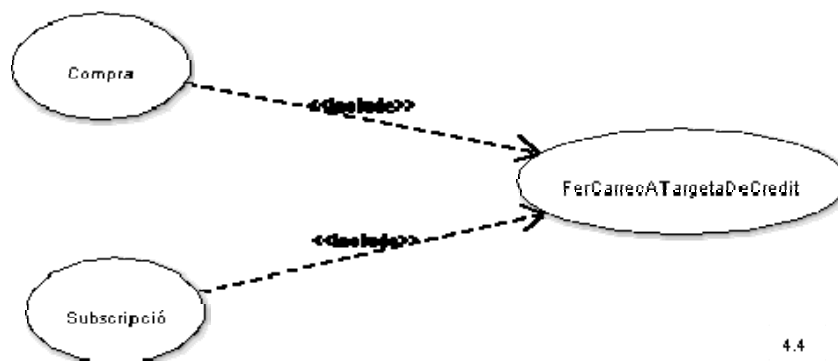
- Comportament: l'actor tria fer una alta, ha d'entrar el nom i cognoms de l'usuari, el codi d'usuari (*username*), la contrasenya (*password*) -dos cops- i el tipus d'usuari -que es tria d'una llista. El sistema valida valors i coherència de les dades, i les enregistra.
- Errors possibles i cursos alternatius:
 1. Aquest codi d'usuari ja existeix: canviar-lo o abandonar.
 2. Les dues contrasenyes no coincideixen: tornar a entrar-les.

La informació d'un cas d'ús es completa amb una petita descripció i amb la relació dels actors que poden usar-lo. Els que ja coneixíeu aquestes tècniques convé que tingueu present que en aquesta assignatura no s'usen els diagrames de seqüència.

Al web de l'assignatura, dins del document enllaçat a "[Informació sobre els diagrames del primer lliurament i Rational Rose](#) (si es fa servir una altra eina, s'ha d'incloure la mateixa informació)", es descriu el que és **normatiu adjuntar als diagrames del primer lliurament**. Se us demana un llistat fet amb una eina de tractament de textos que ha de contenir per a cada cas d'ús:

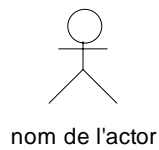
- **Nom** del cas d'ús
- **Actors** que hi intervenen
- **Descripció** (una o dues línies)
- **Diàleg típic** entre els actors i el sistema, i canvis que s'operen en el sistema
- **Errors possibles i cursos alternatius** del cas d'ús

A vegades, es defineix un cas d'ús en funció d'un altre. UML defineix diverses formes de fer-ho, però en el nivell d'aquesta assignatura només té sentit una d'elles que correspon a quan un cas d'ús n'usa un altre (*include*), és a dir, una funcionalitat crida a l'altre dintre del flux normal d'execució, el cas d'ús cridat sol ser una funcionalitat d'utilitat. Per exemple:

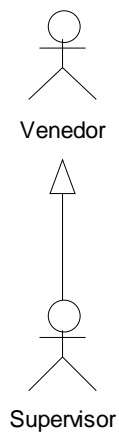


4.4

L'altre element important del diagrama de casos d'ús és l'actor, n'hi ha d'haver tants com tipus d'usuari diferents tingui el sistema. Es representa:

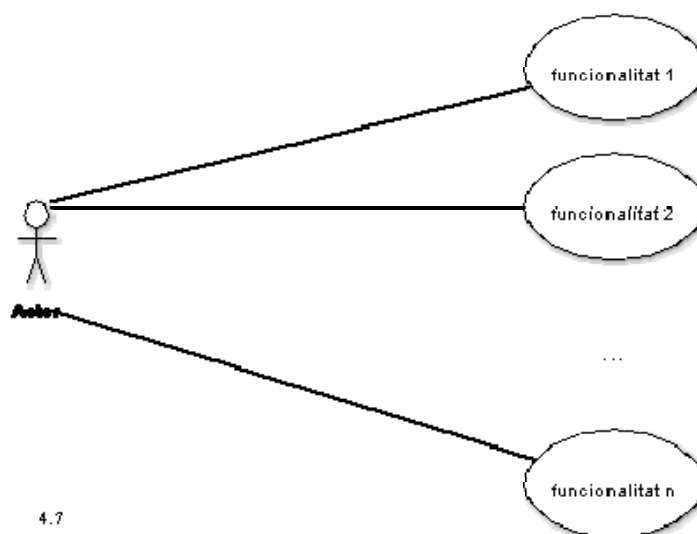


Podem aplicar herència: un actor n'hereta un altre quan pot fer tot el que pot fer l'altre i unes quantes coses més. Per exemple:



La tipologia dels usuaris es descriu als RNF, per cada tipus d'usuari hi ha d'haver un actor en el diagrama de casos d'ús.

Cada actor s'associa a les funcionalitats a les quals li és permès d'accedir en el sistema cosa que defineix el seu menú:



A vegades ens convé que determinades funcionalitats no hi tingui accés cap usuari, és a dir, que siguin automàtiques, que les dispari o posi en marxa el sistema en deter-

minades circumstàncies. Per exemple, un sistema de còpies de seguretat automàtic. En aquests casos se sol crear un actor amb el nom "rellotge" que té la missió de fer-se càrrec dels casos d'ús d'execució automàtica (és com una mena d'*scheduler*).

Com que del diagrama de classes ja en vam parlar amb la OO anem a comentar que són els **RNF**. S'entén que són RNF tots aquells requisits del sistema que no es recullen en la definició de la funcionalitat, tot i que alguns d'ells puguin tenir repercussions en el diagrama de casos d'ús o en el de classes. Cal definir cada requisit de forma que pugui mesurar-se i comprovar si el producte acabat el satisfà o no. Per exemple, en lloc de dir "el sistema ha de ser fàcil d'aprendre" convé especificar "el sistema s'ha de poder aprendre en dos dies de dedicació completa per part d'un administratiu".

A la pàgina de l'assignatura (<http://www-assig.fib.upc.edu/~prop/>) hi trobareu un document amb una relació dels RNF més comuns a preveure en un projecte (<http://www-assig.fib.upc.edu/~prop/rnf.pdf>). En aquesta relació hi trobo a faltar alguns aspectes com els relacionats amb l'entorn d'execució: sistema operatiu, si s'executarà en xarxa, client/servidor, distribuït, etc. Si hi ha un sistema gestió de fitxers o un SGBD decidit prèviament per a la gestió de la persistència. La importància i necessitat de transportabilitat. És força probable que se'n pugui trobar a faltar algun altre.

De cara al projecte només heu de tenir en consideració allò que són realment requisits, coses que l'assignatura us obliga a fer, per exemple, usar determinades eines de desenvolupament, el tipus d'equip o el sistema operatiu. En realitat a la vostra relació hi haurà d'haver uns pocs RNF, moltes vegades per sota de la mitja dotzena, depenent del nombre de tipus d'usuaris definits.

No s'hi ha d'afegir elements de collita pròpia que si bé poden ser molt recomanables no se us exigeixen (cosa que teniu tendència a fer). Heu de tenir present que aquesta especificació serà d'obligat compliment pel vostre programa.

Finalment, hi ha algun d'aquests RNF que encara no esteu en condicions d'incloure en la relació que lliurareu ja que no els haureu desenvolupat, com els estàndards de programació que haureu de definir de cara a la documentació en el segon lliurament.

Un ordre lògic de treball que pot ser recomanable és el següent: un cop hagueu recollit tota la informació que us ha de permetre especificar (petició del client, documentació existent, consultats altres experts i vist el mercat) començar pels RNF, ja que poden tenir incidència sobre els diagrames de casos d'ús i de classes. En segon lloc convé confeccionar una primera versió del diagrama de classes. Aquest només contindrà l'estructura de dades, és a dir, classes, atributs i relacions (només calen associacions i composicions), treballant especialment les associacions. A continuació treballar el diagrama de casos d'ús on es defineix què sabrà fer el programa. Un cop hagueu confeccionat i revisat aquests tres documents: relació de RNF, diagrama de classes i diagrama de casos d'ús, esteu en condicions d'escriure la descripció textual del programa per al vostre client, recollint-hi tota la informació dels tres documents formals. No oblideu que es tracta d'un procés iteratiu, sempre convé tirar endarrera i revisar el que s'ha fet, sobretot quan hi ha canvis.

Pels **primers projectes** d'OO és bo de disposar d'ajudes per trobar candidats a ser funcionalitats, casos d'ús, i per construir l'estructura de dades.

Sempre es parteix d'un primer enunciat o text descriptiu, que cas de no existir es pot confeccionar a partir de les entrevistes o converses que es mantinguin amb l'usuari. La proposta és senzilla, partim d'una lectura de l'enunciat, tota acció que se sol expressar en forma verbal apunta a les funcionalitats explícites del sistema, tota frase nominal apunta a classes o als seus atributs.

Caldrà sempre eliminar plurals (passar-los a singular) i eliminar sinònims. Cal vigilar a recuperar el subjecte de les frases on pugui haver estat omès, especialment important quan la frase està en passiva. Els adjectius moltes vegades representen atributs.

Els atributs són junt amb els mètodes les responsabilitats de la classe, són el coneixement que aquesta manté. Avui en dia se sol accedir als atributs exclusivament a través de mètodes, és a dir, que els atributs són privats de la classe.

En els atributs s'emmagatzema la informació de la classe i són l'objecte de les seves operacions (mètodes). Una classe sense responsabilitats no té sentit en aquesta fase. Una classe amb un sol atribut sol correspondre a un atribut d'una altra classe mal aplicat. Han d'evitar-se les redundàncies en la informació i evitar de tenir atributs amb valors derivats, com l'edat d'una persona si es té també la data de naixement.

El diagrama de classes es completa amb les relacions entre classes. Les realment importants en l'especificació són les d'associació, ja que són les de més influència en la definició del comportament del sistema.

Pel que fa al diagrama de casos d'ús, a més de les expressions verbals poden donar lloc a funcionalitats la informació que s'esmenti ja que caldrà obtenir-la i validar-la, especialment aquella que calgui mantenir i manipular. Finalment el propi rol de cada classe en el sistema pot donar lloc a responsabilitats.

Una manera de revisar si la funcionalitat definida pel sistema és completa és recollir de l'usuari una relació de situacions en les quals voldria obtenir informació del sistema. Per aquesta revisió també s'usen els *walk-troughs*, que consisteixen en repetir pas a pas els processos que fa l'usuari i veure si disposa dels casos d'ús i les dades que li permeten fer-los.

De cara al projecte cal fer els diagrames de casos d'ús i de classes amb *Rational Rose* i documentar els diagrames seguint les instruccions que se us dona a "Informació sobre els diagrames del primer lliurament i Rational Rose" (<http://www-assig.fib.upc.edu/~prop/InfoRatRose.html>)

Per a l'etapa d'**anàlisi** queda poca feina, ja que una part de la que tradicionalment calia fer ha estat avançada a l'etapa anterior. En primer lloc es tracta de completar les classes definides amb l'assignació de mètodes. Com que encara no anem a construir el programa es fa una distribució més estratègica que real. Es tracta de transformar els casos d'ús en mètodes de les classes que s'han definit.

A vegades pot resultar poc clar on assignar com a mètode un determinat cas d'ús, podeu optar per fer una assignació provisional, per partir el cas d'ús en diversos mètodes o per crear una classe específica per a aquest cas d'ús (o per a tots els que no sabem on assignar).

Durant aquesta tasca és quan convindrà definir les dependències, ja que és el moment en què es posen de manifest les crides entre mètodes de les diferents classes.

Un cop el diagrama de classes està complet amb els mètodes, totes les classes tenen les seves responsabilitats, és el moment d'estudiar les generalitzacions (herències), és a dir, de crear classes abstractes. Es tracta de buscar en el diagrama classes anàlogues o similars i veure si es pot crear una classe abstracta amb la part comuna. També podem tenir la sort que hi hagi alguna herència clara entre dues de les classes que haguem definit, però això no és gaire habitual.

Per acabar es defineixen temes i subsistemes, això és agrupar les classes en famílies de cara a estructurar la biblioteca de classes, és a dir, es tracta d'organitzar-les. Per analogia podeu pensar en l'estructura de directoris convenient per emmagatzemar els fitxers font de les classes (en Eiffel, Java —que en diu *package*—, etc.) Normalment els projectes de l'assignatura no donen gaire de si per muntar temes o subsistemes.

El **disseny** és la primera de les etapes d'implementació del programa. Ara preocupa com s'ha de implementar el programa. És el moment de fer-ho.

Com que convé fer el mínim treball possible, interessarà intentar reutilitzar tan com es pugui.

De cara a la reutilització, si no es troba a les biblioteques exactament la classe que es necessita, es pot intentar reutilitzar a base de l'herència, l'agregació (composició) o combinar herència i agregació. Es tracta de buscar a les biblioteques classes que puguin ser abstraccions de les que cal escriure i aprofitar el mecanisme d'herència per només haver d'escriure'n una part. O bé, es buscaria classes que es poguessin agregar a la classe que cal confeccionar. O una classe que pugui heretar una altra que serà agregada, etc.

D'altra banda convindrà adaptar la implementació al llenguatge de programació i les seves característiques. Així, Eiffel implementa la genericitat i l'herència múltiple, però Java no.

Quan ens convingui herència múltiple en un llenguatge que no hi dóna suport podem usar el mecanisme conegut com de *delegació*. Consisteix en heretar directament la classe que més ens convingui (en general, la més significativa) i la resta de classes replicar-ne el codi font dins dels codi font de la classe que l'hereta. Això es pot implementar simplement definint a la classe filla un atribut del tipus de la classe pare, substituint l'herència per una agregació. Per completar-ho es pot crear un mètode per cada mètode de la classe pare que es limiti a cridar-lo (`pare.meto-de_de_la_classe_pare(arguments)`) així es poden simular millor els efectes de l'herència sobre el mètodes. En aquest cas haurem de vigilar quan afegim o traiem mètodes a la classe pare.

Per implementar la genericitat quan no ho fa el llenguatge s'usa algun dels mecanismes de substitució disponibles com el "define" en el compilador de C o simplement usant un editor. Es tracta de substituir el paràmetre o paràmetres pels tipus que interressi en cada cas. De fet, procedint així acabarem tenint a la biblioteca totes les classes concretes d'aquesta genèrica que haguem necessitat implementar. També es pot transformar el paràmetre de creació en argument de les operacions creadores.

Finalment, convé aplicar el model arquitectònic adequat. En el cas de l'assignatura aplicarem un model en capes, en concret en 3 capes: presentació, domini i gestió de dades. Aquest model el teniu descrit en el document "Apunts de l'arquitectura en 3 capes" (<http://www.lsi.upc.edu/~ibarz/A3COOj.pdf>), el veurem més endavant. Fonamentalment es tracta de separar els components de gestió de la interfície i els de la gestió de la persistència dels que són la base lògica del programa o domini. Hi ha diverses raons que fan recomanable aquesta disgregació, destacarem el fet que l'evolució tecnològica sol afectar en un ritme diferent als elements de gestió de dades i de la interfície que els de la lògica del programa. Si no els separéssim per capes cada canvi tecnològic en qualsevol aspecte —interfície o gestió de dades— podria afectar directament a la majoria, per no dir totes, les classes del programa. També ho fa recomanable el fet que avui en dia en el treball en xarxa es pot pensar distribuir les capes en equips diferents (client, servidor d'aplicacions i servidor de dades). De fet és l'arquitectura que ha adoptat Microsoft en la seva plataforma ".net".

Aquest model arquitectònic és caracteritzat perquè facilita la canviabilitat (que vol dir: extensibilitat, portabilitat, mantenibilitat i reestructurabilitat) i la prova, per contra dificulta l'obtenció de l'eficiència òptima en afegir al sistema feina innecessària o redundant.

L'aplicació de la **reutilització** en el vostre projecte: no disposem d'una biblioteca (*repository* o dipòsit) de classes on buscar classes per reutilitzar. D'una banda es pot optar per buscar biblioteques existents (a internet o on sigui), però en tal cas us caldrà l'autorització del tutor per usar-les. El que és normatiu a l'assignatura és trobar classes per compartir entre les del *cluster*. Aquestes operacions solen ser complicades i fins i tot conflictives i per això plantegem un esquema de com fer-ho per intentar que tot plegat funcioni prou adequadament. L'esquema del vostre comportament hauria de ser:

1. Cada grup treballa per separat el seu projecte i li aplica l'arquitectura de 3 capes. (Convé deixar les classes ben especificades, amb les responsabilitats ben definides, això és força útil pel treball en equip a nivell del propi grup i del *cluster*.)
2. Es posen en comú les classes dels tres grups del *cluster* (totes) per buscar classes abstractes i agregacions (composicions) que es puguin compartir. Us comentaré algunes possibilitats.
3. Es munta una relació de les classes que es poden compartir i dels grups interessats en utilitzar-les (aquesta relació compromet, vol dir que serà obligatori utilitzar-les). No cal que tots els grups del *cluster* reutilitzin totes les classes compartides, potser que algunes només interessin a dos d'ells.
4. Es consensua l'especificació de cada una de les classes a compartir. S'haurà de formalitzar l'especificació per lliurar-la al tutor. És molt important l'acord dels grups que utilitzaran la classe en la seva especificació, és a dir, que no convé deixar aquesta tasca d'especificació en mans de qui la implementi.
5. Es reparteix, tan equitativament com sigui possible, el desenvolupament de les classes que s'ha decidit compartir, tenint en compte que:
 - El grup que desenvolupa una classe ha d'estar interessat en usar-la.

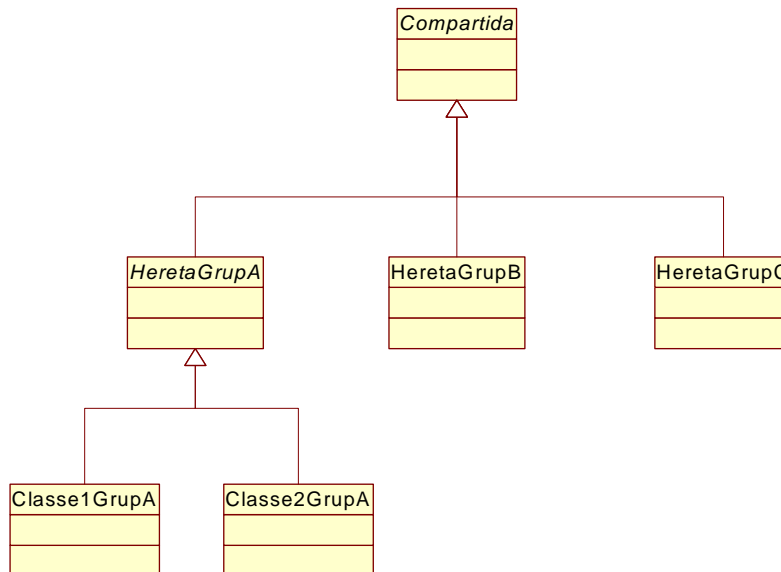
- Els grups receptors, que les reutilitzen, hauran de validar-les, és a dir, comprovar que satisfan les especificacions.
- El repartiment ha de tenir present que el treball de desenvolupament ha de ser disjunt, no pot dependre ningú de ningú altre, i menys encara si són de grups diferents. El desenvolupament de les classes compartides s'ha de poder fer totalment per separat. Cas d'haver-hi alguna dependència (dos classes que no es puguin fer per separat) només es podrà admetre si se'n pot fer càrrec la mateixa persona.

Us recomanem que:

1. Centreu la recerca de classes a compartir a la capa del domini. De les capes de presentació i de gestió de dades compartiu només les classes més abstractes i les genèriques, si n'hi ha.
2. Les reunions de més de 4 o 5 persones solen ser poc operatives, per aquestes tasques us convé fer les reunions amb només un representant de cada grup.
3. Esforceu-vos en especificar bé les classes compartides, evita problemes. (És possible fixar condicions de rendiment o qualitat de la classe, però vigileu que això comporta riscos addicionals.)

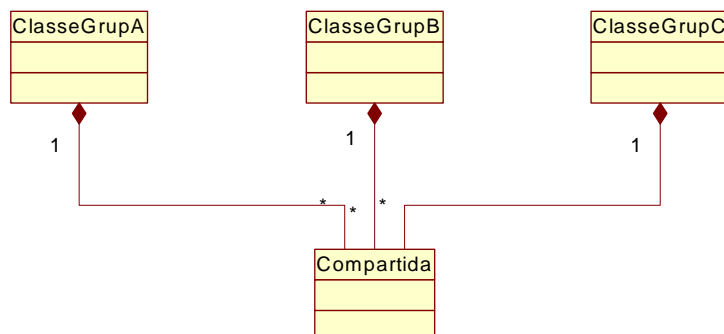
Algunes possibilitats per **trobar classes per compartir**:

1. Abstraccions de classes concretes o abstractes:

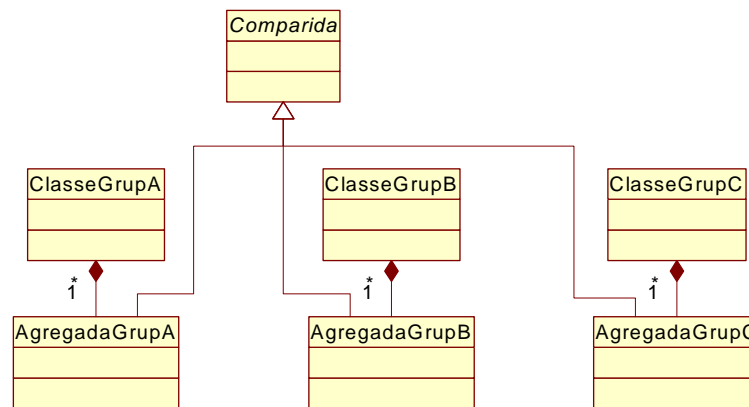


(On A, B i C poden valer 1, 2 o 3, però entenent que $A \neq B \neq C$)

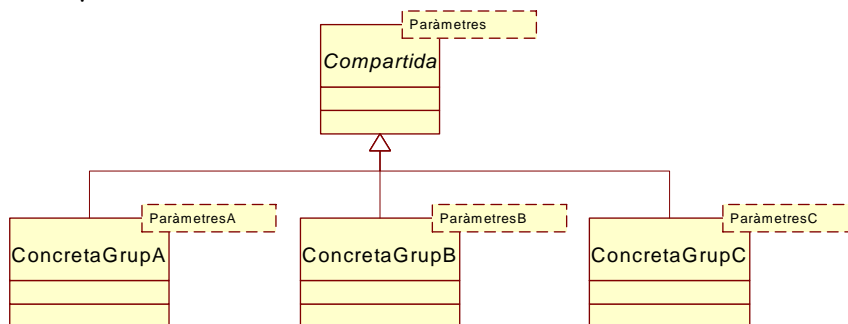
2. Composicions:



3. Abstraccions de les composicions:



4. Classes genèriques:



5. D'altres que deriven d'aquestes bàsiques.