Programari lliure

David Aycart Pérez
Marc Gibert Ginestà
Martín Hernández Matías
Jordi Mas Hernández

XP06/M2012/01486

Enginyeria del programari en entorns de programari lliure

Amb el suport de:

UOC
www.uoc.edu

Generalitat de Catalunya
Departament d'Universitats, Recerca
i Societat de la Informació

Jordi Mas Hernández

Coordinador

Enginyer de programari en l'empresa de codi obert Ximian, on treballa en la implementació del projecte lliure Mono. Com a voluntari, col·labora en el desenvolupament del processador de textos Abiword i en l'enginyeria de les versions en català del projecte Mozilla i Gnome. Coordinador general de Softcatalà. Com a consultor, ha treballat per a empreses com Menta, Telépolis, Vodafone, Lotus, eresMas, Amena i Terra España.

David Megías Jiménez

Coordinador

Enginyer d'Informàtica per la UAB. Magíster en Tècniques avançades d'automatització de processos per la UAB. Doctor en Informàtica per la UAB.

Professor dels Estudis d'Informàtica i Multimèdia de la UOC.

David Aycart Pérez

Autor

Director de desenvolupament formatiude Free Software Certification.Soci fundador de Free Software Certification.Soci fundador d'Esware Linux, SA.

Marc Gibert Ginestà

Autor

Enginyer d'Informàtica per la Universitat Ramon Llull. Soci fundador i cap de projectes de Cometa Technologies, empresa dedicada a donar solucions en tecnologies de la informació, basades en l'ús d'estàndards i eines de codi obert. Professor del màster de Seguretat en tecnologies de la informació a Enginyeria i Arquitectura La Salle i consultor del màster internacional de Programari lliure de la UOC.

Martín Hernández Matías

Autor

Enginyer tècnic informàtic de la Universitat Politècnica de Madrid.Director de formació de Free Software Certification.

La Fundació per a la Universitat Oberta de Catalunya agraeix el suport del Departament d'Universitats, Recerca i Societat de la Informació de la Generalitat de Catalunya per a la versió d'aquesta obra en català.

Primera edició: febrer 2006

Material realitzat per Eureca Media, SL

Dipòsit legal: B-9.313-2006
ISBN: 84-9788-411-6

Índex

Agraïments

Els autors agraeixen a la Fundació per a la Universitat Oberta de Catalunya (http://www.uoc.edu) el finançament de la primera edició d'aquesta obra, emmarcada en el màster internacional de Programari Lliure ofert per aquesta institució.

ANOTACIONS

1. Introducció a l'enginyeria del programari

1.1. Introducció

Aquest primer capítol del curs *Enginyeria del programari en entorns de programari lliure* proporciona les bases per a conèixer els conceptes principals involucrats en l'enginyeria del programari en general, i la seva aplicació al programari lliure en particular.

Durant el capítol, veurem que l'enginyeria del programari és gairebé tan antiga com el mateix programari, i també que part dels problemes que van fer néixer aquest sector de la indústria del programari segueixen vigents, la majoria sense una solució clara i definida. Malgrat això, s'han fet grans avenços en tots els camps, des de l'estimació del cost fins a la gestió dels equips de treball, documentació i molt notablement en proves, qualitat i seguretat.

Després d'introduir breument els conceptes més importants, comencem a examinar directament dues metodologies concretes molt populars. En primer lloc, veurem la metodologia programació extrema o *eXtreme Programming*, potser la més pròxima a la manera d'organitzar projectes de programari lliure i participar-hi. A continuació, farem un cop d'ull a Métrica v3, una metodologia clàssica que és imprescindible conèixer si s'han de realitzar projectes amb l'Administració espanyola.

Hi ha centenars de metodologies (públiques i particulars d'empreses), però els conceptes introduïts durant aquest mòdul ens proporcionaran els coneixements necessaris per a avaluar-ne la idoneïtat en el nostre àmbit de treball o projecte concret i per a aprofundir els aspectes de l'enginyeria del programari que més ens hagin cridat l'atenció.

ANOTACIONS

1.2. Objectius

- Adquirir consciència de la importància dels processos i activitats relacionats amb l'enginyeria del programari en projectes de tot tipus.
- Conèixer els conceptes bàsics involucrats que permetran documentar-vos més àmpliament quan calgui i avaluar metodologies concretes.
- Familiaritzar-vos amb la metodologia de la programació extrema, i amb els nous paradigmes de gestió, disseny i desenvolupament que planteja.
- Conèixer l'estructura general de la metodologia Métrica v3 i els seus principals processos, activitats i interfícies.

1.3. Una mica d'història

L'expressió *enginyeria del programari* es va començar a usar al final de la dècada dels seixanta per a expressar l'àrea de coneixement que es desenvolupava entorn de les problemàtiques que oferia el programari en aquell moment.

En aquella època, el creixement espectacular de la demanda de sistemes de computació cada vegada més i més complexos, associat amb la immaduresa del mateix sector informàtic (totalment lligat a l'electrònic) i amb la falta de mètodes i recursos, va provocar el que es va denominar *la crisi del programari* (en paraules d'Edsger Dijkstra) entre els anys 1965 i 1985.

Durant aquella època, molts projectes importants superaven amb escreix els pressupostos i dates estimats, alguns dels quals eren tan crítics (sistemes de control d'aeroports, equips per a medicina, etc.) que les seves implicacions anaven més enllà de les pèrdues milionàries que causaven.

La crisi del programari va passar, no tant perquè va millorar la gestió dels projectes, sinó en part perquè no és raonable estar en crisi més de

Nota

Algun dels projectes més representatius de l'època, com el desenvolupament del sistema OS/360 d'IBM, va trigar més d'una dècada a acabar-se, i va ser el primer que va involucrar més de mil programadors. Més endavant, el cap del projecte Mythical Man Month, Fred Brooks, va reconèixer que s'havien comès errors de milions de dòlars, i va pronunciar la coneguda llei de Brooks:

Assignar més programadors a un projecte ja endarrerit només endarrereix encara més el projecte.

ANOTACIONS

vint anys i en part perquè es feien progressos en els processos de disseny i metodologies.

Així doncs, des de 1985 fins avui dia, han anat apareixent eines, metodologies i tecnologies que es presentaven com la solució definitiva del problema de la planificació, previsió de costos i assegurament de la qualitat en el desenvolupament de programari.

Entre les eines, la programació estructurada, la programació orientada a l'objecte, als aspectes, les eines CASE, el llenguatge de programació ADA, la documentació, els estàndards, CORBA, els serveis web, i el llenguatge UML (entre d'altres) van ser tots anunciats en el seu moment com la solució dels problemes de l'enginyeria del programari, l'anomenada *bala de plata* (per *silver bullet*). I encara més, cada any sorgeixen idees i iniciatives noves encaminades a això.

Sens dubte, també hi ha hagut qui ha culpat els programadors per la seva indisciplina o anarquia en els desenvolupaments. La ignorància i alguns casos excèntrics van contribuir a crear una imatge falsa del programador, que avui dia encara perdura. Encara que moltes vegades ell és el "patidor" d'alguna d'aquestes metodologies o d'una implementació pobra d'aquestes, sembla lògic que, com a participant actiu en el projecte, les metodologies més modernes comencin a tenir-lo més en compte.

En combinació amb les eines, també s'han fet esforços perquè incorporin els mètodes formals al desenvolupament de programari, argumentant que si es prova formalment que els desenvolupaments fan allò que s'hi requereix, la indústria del programari serà tan predictible com les altres branques de l'enginyeria.

Entre les metodologies i processos, a més de Métrica v3 (promoguda per la Secretaria del Consell Superior d'Informàtica) i la programació extrema, que veurem detalladament més endavant, destaquen molts altres com ara RUP (*rational unified process*, desenvolupat per Rational Software Corp., ara una divisió d'IBM), SSADM (*structured systems analysis and design methodology*, promogut pel Govern britànic) o el mètode d'avaluació de la capacitat de desenvolupament dels equips o empreses conegut com a CMMI (*capability maturity model integration*).

ANOTACIONS

Paral·lelament, se solen usar també mètodes de predicció de costos com COCOMO o els punts de funció.

Les darreres iniciatives en aquest camp són múltiples i s'estenen per tot el procés relacionat amb el programari. Els més acadèmics s'inclinen per una estructura de components, serveis, amb orientació a l'objecte o a aspectes en la implementació; encara que també és igualment significatiu el desenvolupament de les eines que ens ajuden a representar i compartir aquests dissenys, i també a valorar l'esforç i el valor que afegeixen al producte acabat. És realment un camp fascinant, en què tant els seminaris com les grans consultores com els laboratoris petits innoven cada dia i presenten bones idees.

1.4. Conceptes

En aquest capítol, veurem els conceptes més comuns associats amb l'enginyeria del programari, aplicables tant a entorns de programari lliure com a la resta. Tots disposen de bibliografia abundant i d'alguns cursos i màsters especialitzats en el seu aprenentatge i aplicació als processos de desenvolupament. Aquí únicament els definirem per a tenir una base sobre la qual tractar els exemples de metodologia que veurem en els capítols següents.

1.4.1. Gestió de projectes

La gestió de projectes és la disciplina que agrupa i ordena el conjunt de tasques o activitats destinades a obtenir uns objectius. Això inclou la planificació, definició, ordenació i gestió de les activitats que formaran el projecte de programari.

En la seva primera expressió, la bona gestió d'un projecte és la que és capaç de reduir al mínim les possibilitats d'error durant tot el transcurs d'aquest projecte. Es pot entendre com a error la no-consecució dels requisits inicials del projecte, la inviabilitat econòmica d'aquest, un resultat que impedeixi mantenir-lo o li impedeixi evolucionar, etc.

Una altra visió més orientada a l'àmbit econòmic i de costos del projecte és la que defineix la bona gestió com la que optimitza l'ús dels

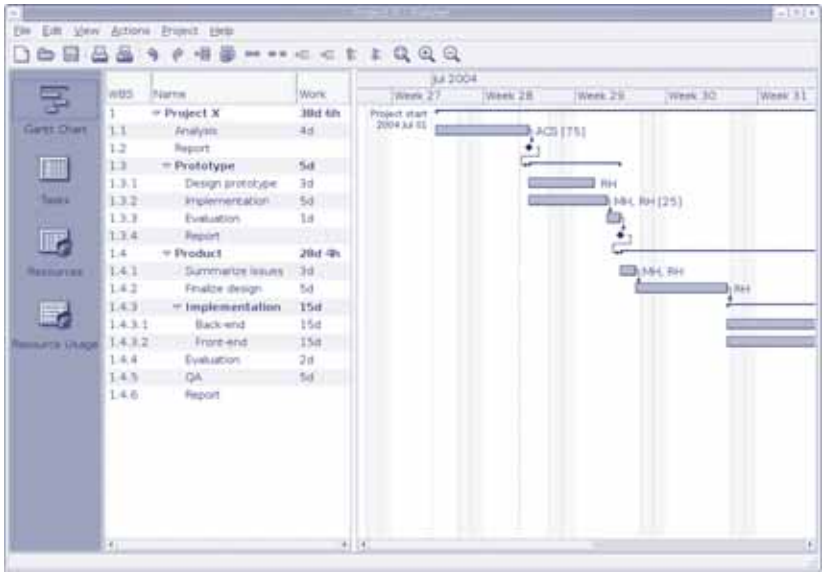
ANOTACIONS

recursos (temps, diners, persones, equips, etc.) en cada una de les seves fases.

Molt comunament, la gestió del projecte la duu a terme el director del projecte, normalment una sola persona que no té per què participar activament en les activitats d'aquest, però que sí que s'ocupa de monitoritzar-ne el progrés i de la interacció entre els diferents grups que hi intervenen per minimitzar el risc d'error del projecte.

A més de fulls de càlcul de control d'hores per tasca, ús de recursos, etc., els diagrames de Gantt són molt usats en aquests entorns per a mostrar d'una manera clara la successió de tasques, recursos involucrats i les seves dependències.

Figura 1. Captura de pantalla del programari GPL Imendio Planner que mostra un diagrama de Gantt



En els diagrames de Gantt, representem les activitats que cal realitzar en forma d'arbre (meitat esquerra de la figura) indicant-ne la data d'inici i la durada estimada. El programa representa l'activitat sobre un calendari (meitat dreta de la figura) i ens permet definir les dependències entre les activitats. Quan s'indica que una tasca ha de començar quan n'acabi una altra, o que han de començar alhora o acabar les dues simultàniament, el diagrama modifica automàticament la situació de les activitats en el temps.

ANOTACIONS

Nota

El camí crític d'un projecte es defineix com la seqüència de tasques que sumen l'interval de temps més llarg d'aquest. Així doncs, determina la durada mínima del projecte, ja que un retard en alguna de les tasques té un impacte directe en la durada total.

Depenent de la complexitat del programa que utilitzem per a representar el diagrama, podrem especificar també els recursos disponibles, els necessaris per a cada tasca, etc., i serem capaços de detectar conflictes en la planificació com ara intervals en què no disposarem dels recursos necessaris, o quin serà el camí crític del projecte (la successió de tasques que per les seves dependències determinaran la durada màxima del projecte).

Concretament, en entorns de programari lliure, no hi ha una manera "tradicional" de gestionar un projecte. Un bon exemple d'això és l'article d'Eric S. Raymond, "La catedral i el basar", en què es descriu la manera tradicional de gestionar un projecte com a analogia de construir una catedral, i es compara la manera de gestionar molts projectes de programari lliure amb el funcionament d'un basar.

Els grans projectes de programari lliure, com el desenvolupament del nucli Linux o el servidor web Apache, estan formats per un comitè de gestió del projecte (o una sola persona com a dictador benèvol) que decideix els passos següents que caldrà dur a terme en el projecte i deleguen immediatament en altres persones o equips els canvis que cal realitzar per a arribar als objectius. L'aprovació dels canvis realitzats o bé la incorporació de funcionalitats noves proposades pels usuaris o altres desenvolupadors segueix el procés invers fins a arribar al comitè d'aprovació que decideix si s'incorporen al projecte o no.

Nota

http://www.bugzilla.org/
http://sourceforge.net/

Sens dubte, en aquest tipus de gestió de projectes, tenen un paper fonamental les eines de suport i comunicació entre els diferents equips. Programes de gestió d'incidències com el conegut Bugzilla o entorns com Sourceforge que ajuden els gestors del projecte o els comitès organitzadors a conèixer l'estat del projecte, l'opinió dels usuaris, les aportacions dels desenvolupadors, etc.

Els projectes petits o mitjans de programari lliure no acostumen a seguir una metodologia tradicional i s'orienten més a una gestió àgil del projecte, seguint principis d'acord amb el programari lliure, en què la publicació de funcionalitats i versions noves del projecte preval enfront del compliment d'un calendari rígid d'activitats i fites. En aquests projectes, el director sol intervenir molt estretament en el desenvolupament i en la resta d'activitats.

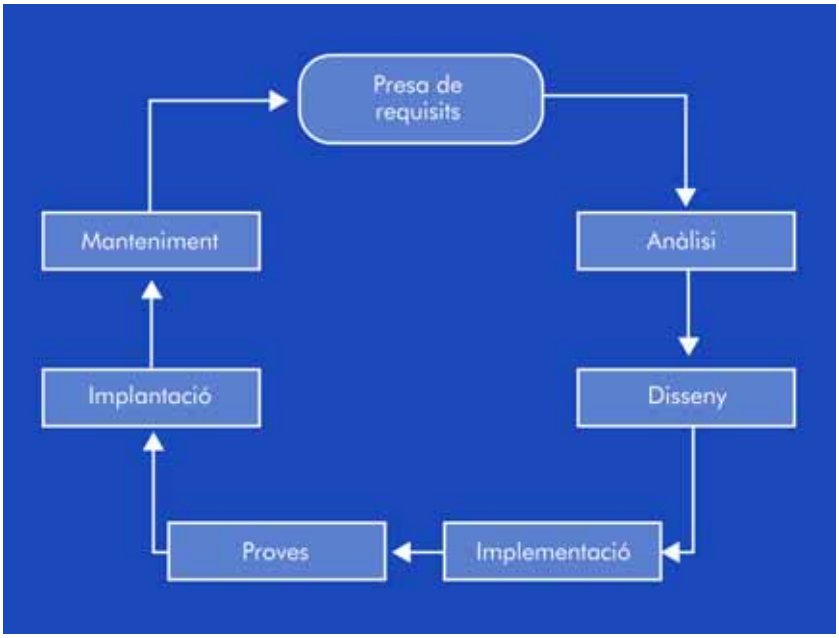
ANOTACIONS

Aquest darrer tipus de projecte se sol gestionar amb simples llistes de requisits i errors per resoldre, que per la naturalesa distribuïda dels projectes de programari lliure hauran d'estar disponibles perquè els desenvolupadors i usuaris les puguin consultar i modificar. El gestor o els desenvolupadors seran els qui donaran prioritat a les tasques i decideixin quines incorporaran en els nous llançaments del projecte.

1.4.2. Cicle de vida del programari

Es denomina *cicle de vida del programari* el conjunt de fases per les quals passa un projecte de programari des que és concebut fins que està llest per a ser usat. Típicament, inclou les activitats següents: presa de requisits, anàlisi, disseny, desenvolupament, proves (validació, assegurament de la qualitat), instal·lació (implantació), ús, manteniment i obsolescència.

Figura 2. Cicle de vida típic d'un projecte de programari



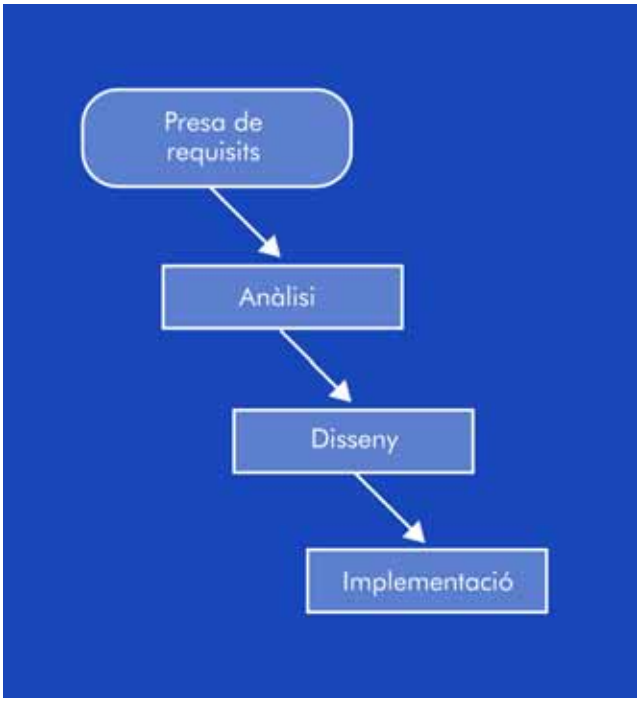
El projecte tendeix a passar iterativament per aquestes fases, en comptes de fer-ho linealment. Així doncs, s'han proposat diversos models (en cascada, incremental, evolutiu, en espiral o concurrent, per esmentar-ne alguns) per a descriure el progrés real del projecte.

ANOTACIONS

El model en cascada és el més simple de tots ells i serveix de base per a la resta. Simplement assigna unes activitats a cada fase que serviran per a completar-la i per a proporcionar els requisits de la fase següent. Així, el projecte no es dissenya fins que no ha estat analitzat, o no es desenvolupa fins que no ha estat dissenyat, o no es prova fins que no ha estat desenvolupat, etc.

Els models incremental i evolutiu són una variació del model en cascada en què aquest s'aplica a subconjunts del projecte. Depenent de si els subconjunts són parts del total (model incremental) o bé versions completes però amb menys prestacions (model evolutiu), n'estarem aplicant un o un altre.

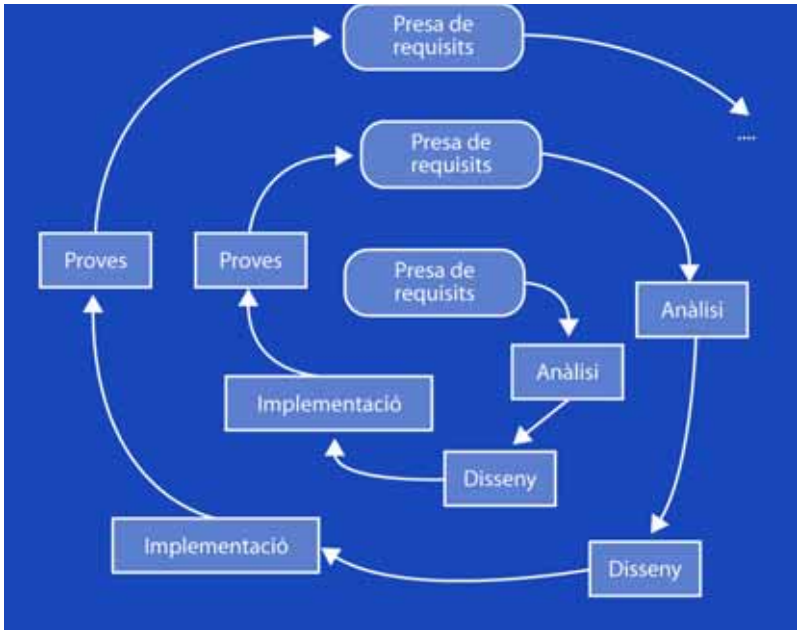
Figura 3. Model en cascada



El model en espiral es basa en la creació de prototips del projecte que passen per les fases anteriors i que s'apropen successivament als objectius finals. Així doncs, ens permet examinar i validar repetidament els requisits i dissenys del projecte abans d'emprendre fases de desenvolupament noves.

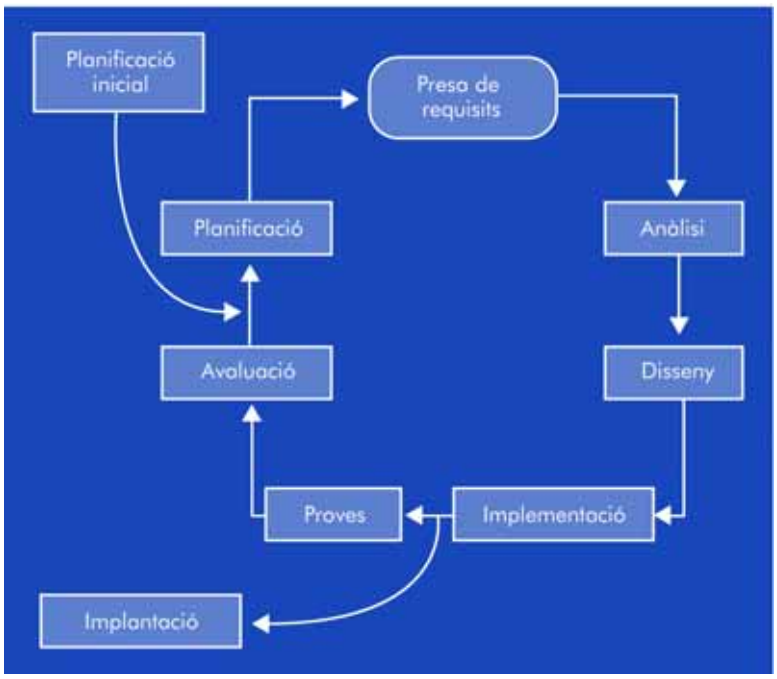
ANOTACIONS

Figura 4. Model en espiral



Finalment, el model iteratiu o incremental és el que permet que les fases d'anàlisi, disseny, desenvolupament i proves es retroalimentin contínuament i que comencin al més aviat possible. Permet atendre possibles canvis en les necessitats de l'usuari, o tenir en compte noves eines o components que els desenvolupadors descobreixin i que facilitin el disseny o proporcionin funcionalitats noves.

Figura 5. Model iteratiu



ANOTACIONS

Es tracta d'obtenir al més aviat possible una versió funcional del programari i afegir-hi prestacions a partir del que s'ha après en la versió anterior. L'aprenentatge prové tant del desenvolupament anterior com de l'ús del programari, si és possible. En aquest tipus de desenvolupament, és imprescindible establir una llista de control del projecte, en la qual registrarem les funcionalitats que falten per implementar, les reaccions dels usuaris, etc., i que ens proporcionarà les bases per a cada iteració nova.

Aquest darrer mètode és el més usat –encara que sigui involuntàriament– en projectes de programari lliure, per la mateixa naturalesa canviant dels requisits o l'aportació constant de col·laboradors nous.

1.4.3. Anàlisi de requisits

L'anàlisi de requisits és la primera fase de la vida d'un projecte. Caldrà recollir-hi tant les necessitats de l'usuari del producte, en el nivell més alt, com l'especificació dels requisits de programari del sistema.

Per a guiar-nos en aquest procés, hi ha multitud de plantilles, llibres i metodologies orientats a extreure de l'usuari (o client) allò que vol realment que faci el sistema. Sens dubte, no és una tasca senzilla. El projecte ReadySET, allotjat en l'adreça http://www.tigris.org, proporciona un conjunt de plantilles de documents per a gestionar un projecte de desenvolupament de programari extremadament útil i de codi obert.

Seguint aquestes plantilles, veiem que l'especificació de requisits del sistema inclou, com a mínim:

- Els casos d'ús: actors que intervindran en l'ús del producte i les seves possibles accions.
- Els seus requisits funcionals: prestacions del producte en la primera versió i possible planificació per a les versions futures.
- Els seus requisits no funcionals: de rendiment, d'usabilitat, de seguretat, etc.

ANOTACIONS

En entorns de programari lliure, el fundador (i gestor) del projecte en serà molt freqüentment també el primer usuari, però això no l'eximeix d'aquesta tasca, que determinarà l'abast del projecte almenys en les seves primeres versions i permetrà als desenvolupadors que s'incorporin al projecte que es facin una idea del seu progrés i objectius, i també on poden ajudar.

1.4.4. Estimació de costos

L'estimació de costos (recursos, equips i temps emprat) és una de les raons de ser de l'enginyeria del programari. Encara que no sempre aplicable en entorns de programari lliure, en què no sol ser possible fer una planificació de recursos disponibles, convé conèixer les mètriques i mètodes que ens permetran predir l'esforç que comportarà implementar un sistema o alguna de les seves prestacions.

L'estimació se sol fer basant-se en models matemàtics que parteixen de la "mida" estimada del projecte i de constants que l'ajusten segons les tecnologies usades, recursos de què disposem, etc. Els models ens permeten estimar l'esforç requerit (habitualment en hores/home o mesos/home) per a acabar el projecte.

Òbviament, la clau resideix en el fet d'estimar la "mida" del projecte. Fins i tot sabent que pot arribar a ser molt poc indicatiu, hi ha molts models que usen les línies de codi per a determinar la mida d'un projecte (COCOMO, COCOMO II). Altres models utilitzen els denominats *punts de funció*, que són una mesura de la complexitat de cada funcionalitat a partir de les entrades que rep, les sortides que aporta, la seva interacció amb altres funcionalitats o recursos, etc.

Hi ha també mètriques per a llenguatges orientats a l'objecte que tenen en compte factors com els mètodes per classe, les herències entre objectes, la interacció entre ells, el seu grau de dependència (que complica el manteniment), etc.

No és l'objectiu d'aquest capítol endinsar-nos en aquests conceptes, la qual cosa requeriria moltes pàgines de fórmules, taules i casos

ANOTACIONS

d'èxit o fracàs en l'estimació. Els estudiants interessats a aprofundir-los en trobareu molta informació en la bibliografia.

1.4.5. Disseny

La fase de disseny del sistema produirà un conjunt de documents que descriuran un sistema que compleixi amb els objectius de la fase d'anàlisi de requisits. Les decisions que es prenguin en aquesta fase s'hauran de basar en aquests requisits i en la comprensió de la tecnologia i els components disponibles.

S'ha de tenir en compte que molts dels documents que es crearan en aquesta fase els usaran els desenvolupadors del projecte (diagrames de components del sistema, arquitectura del sistema, etc.). Això implica que s'han de fer tan complets com es pugui i que la participació dels desenvolupadors en aquesta fase ajudarà a evitar revisions innecessàries.

Altres documents seran destinats als usuaris del projecte (per exemple, el disseny de la interfície d'usuari), i que s'aprovin i s'entenguin serà clau per a evitar desviacions.

Per tot això, és molt convenient disposar de llistes que ens permetin comprovar que no hem oblidat cap aspecte clau en aquesta fase. El projecte ReadySET té plantilles de gairebé tots els documents que podem necessitar, amb els qüestionaris i exemples corresponents.

Extraient el més significatiu, trobem:

- Diagrama estructural: disseny i notes sobre l'estructura detallada del sistema.
- Diagrama de comportament: disseny i notes sobre el comportament del sistema.
- Arquitectura del sistema: disseny dels seus components, de la seva implantació i integració.

ANOTACIONS

- Organització de codi font i compilació: directoris, opcions de compilació, sistemes de control de versions, etc.

- Interfície d'usuari: metàfores, estàndards que cal seguir, disseny dels contextos d'interacció amb l'usuari, etc.

- Sistema d'informació: bases de dades, abstracció d'objectes, emmagatzemament, persistència, etc.

- Seguretat.

1.4.6. Documentació

La documentació d'un projecte és molt important perquè tingui èxit. Ja des de la fase de disseny, com a part de la mateixa arquitectura del projecte, haurem de definir i escollir el sistema de documentació que usarem per al projecte tenint en compte factors com els següents:

- Formats dels documents, segons la seva tipologia i mètodes d'accés. Haurem de definir els formats i plantilles escollits per als diagrames de disseny, fulls de càlcul, fulls de seguiment del projecte, documents que registrin errors o canvis en les especificacions durant el desenvolupament, documents que defineixin la interfície d'usuari, etc.

- Mètode d'accés i flux de treball de cada tipus de document. Qui tindrà accés als diferents tipus de documents i amb quins privilegis; on es notificaran els canvis que es realitzin (en el mateix document?, en un sistema de control de versions?).

En el cas de la documentació del desenvolupament (del codi font del projecte), convé estudiar les eines que ens ofereixi el llenguatge per a generar la documentació, ja que en molts casos hi ha eines de generació de documentació a partir del codi font i comentaris inserits mitjançant una sintaxi determinada que ens ajudaran molt en el procés. En el capítol 7, examinarem exemples d'eines d'aquest tipus.

ANOTACIONS

En el moment de prendre decisions de format i fluxos de treball sobre la documentació, és molt important tenir en compte els estàndards de formats de document que hi hagi i evitar els formats propietari, sobretot en organitzacions heterogènies en què convisquin diferents sistemes operatius o en projectes de programari lliure, per a donar a la documentació la màxima accessibilitat possible. Sol ser una bona decisió escollir un format de documentació fàcilment convertible en d'altres (per exemple, XML) i així poder disposar de la documentació en HTML per a consultar-la ràpidament, en PDF per a agregar a la documentació del projecte, etc.

1.4.7. Proves i qualitat

La planificació de prototips, proves i tests per a assegurar-ne la qualitat és també un tema molt tractat en l'enginyeria del programari.

Perquè un projecte de programari tingui èxit, cal que el resultat tingui la qualitat esperada pel client o els usuaris. Així doncs, la qualitat del projecte s'haurà de poder definir en termes de prestacions, respostes esperades davant de determinades accions, o accessibilitat del producte en diferents condicions per a poder-lo provar posteriorment mitjançant uns tests de qualitat específics.

S'haurà de poder fer un pla de proves o d'assegurament de la qualitat que classifiqui les activitats relacionades amb la qualitat del producte segons la seva importància i que defineixi amb quina freqüència i quins resultats s'haurien d'obtenir de cada una per a passar a la següent o per a complir els requisits per a aquest llançament en particular.

En el procés d'assegurament de la qualitat no es tracta únicament que el producte passi tots els tests establerts, sinó que implicarà en molts casos aspectes com:

- L'ús de fulls d'estil aprovats pels usuaris.
- La confecció i repàs de llistes de control sobre funcionalitats.
- La revisió periòdica del producte amb els usuaris.

ANOTACIONS

- Les eines que posarem a disposició dels usuaris per a comunicar els errors o suggeriments.

- L'ús d'eines d'anàlisi per a mesurar la resposta del projecte a determinades situacions o per a simular un ús normal d'aquest.

Els models tradicionals de cicle de vida del programari incloïen la fase de proves com un procés que calia realitzar una vegada acabat el desenvolupament. S'ha demostrat que això és altament contraproduent, no només pel cost que implica arreglar errors o deficiències una vegada acabat el desenvolupament, sinó per la naturalesa evolutiva de molts projectes (sobretot en entorns de programari lliure), en què, estrictament parlant, la fase de desenvolupament no acaba mai.

Així doncs, es tendeix a incorporar les proves –o els mecanismes per a dur-les a terme de manera automatitzada– en el desenvolupament des del primer moment. Tecnologies com les proves unitàries o models com la programació per parelles (o *pair programming*) o la comprovació de punts (o *peer testing*) ens ajudaran a mantenir una disciplina en el desenvolupament i a augmentar la qualitat del resultat del projecte.

Tractarem dels tipus de proves, les eines de gestió de les proves i els plans de qualitat detalladament en el capítol 3.

1.4.8. Seguretat

La seguretat en projectes de programari ha estat un factor clau des de l'inici de l'enginyeria del programari. Igual que altres conceptes com la qualitat, la seguretat no pot ser un factor que no es tingui en compte durant el disseny, ni un procés que es comprovi en finalitzar el desenvolupament, sinó que es tracta d'un aspecte del projecte que s'ha de tenir en compte i s'ha de planificar des de la fase de disseny i durant tot el cicle de vida del projecte.

Els principis bàsics de la seguretat d'un sistema són els següents:

- Confidencialitat: els recursos (o funcionalitats sobre aquests) són accessibles només per als usuaris (o processos) autoritzats.

ANOTACIONS

- Integritat: els recursos poden ser modificats només pels usuaris autoritzats.

- Disponibilitat: els recursos accessibles estan disponibles.

Molt freqüentment, la seguretat d'un projecte anirà més enllà de manera que afectarà el sistema en què s'implantarà i, per tant, haurem d'assegurar que la seva implantació deixa el sistema final en un estat segur.

Per aquesta raó, en la fase de disseny del projecte, s'ha de tenir en compte la seva seguretat, primer de manera general, analitzant riscos i activitats destinades a mitigar-los, i més endavant, ampliant els casos d'ús segons els principis bàsics de la seguretat comentats anteriorment.

L'anàlisi de riscos consistirà, molt resumidament, en les activitats següents:

- Recopilar els recursos que han de ser protegits: la informació, tot un sistema, un únic dispositiu, etc.

- Classificar els actors del projecte: en quins casos s'usen i quins rols representen.

- Recopilar requisits legals i de negoci: certificacions que cal emplenar, restriccions de xifratge per a exportar a determinats països o bé regles de negoci més específiques com l'acceptació o no de la repudiació per part dels usuaris.

- Amb la informació recopilada, hauríem de ser capaços de construir una taula en què, per a cada risc, estiméssim el cost que té per incident i, a partir d'una estimació d'incidents per any, ens permetés decidir sobre l'estratègia que cal implantar (que accepti el risc tal com és i defineixi un pla de contingència en cas que es produeixi, o bé que el mitigui amb desenvolupaments addicionals, altres eines o canvis en el disseny que ho permetin).

ANOTACIONS

Figura 6

ID risc	Recurs	Valor recurs	Cost per incident	Incidents/ any	Cost/ any	Estratègia
	Informació del client	6.000 €				
1. Pèrdua d'informació			6.000 €	2	12.000 €	Acceptar
2. Robatori d'informació			60.000 €	1	60.000 €	Mitigar

Sense entrar en detalls, enunciarem a continuació un conjunt de bones pràctiques genèriques que ajudaran a mitigar els riscos associats amb qualsevol projecte:

- Assignar el mínim privilegi possible a cada actor en el sistema.
- Simplicitat. La seguretat per ofuscació no dóna bons resultats.
- Disseny obert. Sempre tindrem alternatives per a millorar o assegurar més el sistema.
- Seguretat per defecte. El sistema ha de ser tan segur com sigui possible per defecte; si es poden relaxar les restriccions, s'ha de tractar d'una acció addicional que es faci amb el projecte ja implantat.
- Fallada segura. Si el sistema falla o pot fallar, hem d'evitar que ho faci quedant-se en una modalitat insegura.
- Minimitzar l'ús de recursos compartits.
- Treballar a favor de la usabilitat del projecte redunda en un ús més bo d'aquest i en menys probabilitats d'errors de seguretat si s'usa malament.

ANOTACIONS

1.5. Exemples de metodologies

Fins ara, hem repassat breument la història i els conceptes relacionats amb l'enginyeria del programari en general. A continuació, examinarem una mica més detalladament dues metodologies concretes per veure com intenten resoldre alguns dels problemes comentats, i així podrem prendre decisions en la gestió de projectes.

Es pot comentar que no sempre convé escollir i aplicar una metodologia de manera estricta. És important entendre-la i conèixer què pot aportar al nostre projecte per a aplicar-la en aquelles fases o processos en què el nostre equip o els nostres usuaris estiguin més còmodes amb aquesta, i no a l'inrevés.

En entorns de programari lliure, no és habitual que els programadors que col·laboren de manera altruista en el projecte ho vulguin fer amb una metodologia massa estricta. Tanmateix, valoraran enormement les facilitats de gestió que ofereixi el projecte, l'accés a la seva documentació, la qualitat d'aquesta, etc., per la qual cosa estaran molt més motivats i passaran més temps col·laborant en el projecte que intentant entendre'l. De la mateixa manera, determinats col·laboradors o clients rebutjaran una metodologia massa moderna o flexible per desconeixement.

1.5.1. Metodologia pròxima al programari lliure: programació extrema

La programació extrema és una de les metodologies anomenades *àgils* per a desenvolupar projectes de programari. Es basa en els principis de la simplicitat, la comunicació, la retroalimentació i el coratge per a implicar tot l'equip –i els usuaris o clients– en la gestió del projecte. El 1996, Kent Back i Ward Cunningham van posar en pràctica una metodologia nova fent prevaler la simplicitat i evitant els hàbits que convertien les coses fàcils en difícils durant el desenvolupament d'un projecte a DaimlerChrysler. El resultat va ser la metodologia programació extrema o eXtreme Programming (XP) –que sens dubte no té res a veure amb el programari de la companyia Microsoft.

En la seva forma més genèrica, les metodologies àgils proposen una implicació total del client en el projecte, i porten al límit el model de

ANOTACIONS

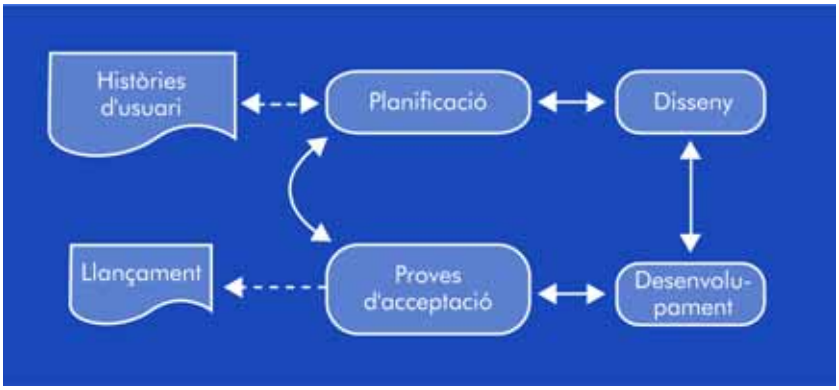
desenvolupament evolutiu en espiral. És a dir, realitzar un pla de projecte basat en versions del producte acordades a partir de funcionalitats concretes, i realitzar el desenvolupament per a aquestes funcionalitats concretes. Una vegada lliurada la versió del projecte que compleixi amb els requisits –no un prototip, sinó una versió que funcioni–, el procés es torna a iniciar amb un conjunt de funcionalitats més ampli.

Els processos i pràctiques d'aquesta metodologia estan basats en l'experiència d'equips de desenvolupament i en els errors comesos o trobats repetidament quan s'utilitzen metodologies més tradicionals. Tanmateix, veurem que algunes de les seves propostes són força xocants i d'altres són incompatibles amb organitzacions o projectes grans.

La proximitat d'aquesta metodologia a l'entorn del programari lliure està determinada pels seus principis bàsics, més que per la seva aplicació concreta. En els apartats següents, examinarem aquests principis i veurem que alguns són molt propers al programari lliure, mentre que d'altres –sobretot els que tenen a veure amb el client o amb la programació per parelles– no solen ser massa aplicables.

La programació extrema es pot dividir en quatre principis sobre els quals s'itera fins que el projecte ha acabat (el client aprova el projecte). Aquestes fases o principis són **planificació**, **disseny**, **desenvolupament** i **proves**. Encara que a primera vista sembla que no hem afegit res de nou a les metodologies tradicionals, és en els detalls de cada fase i en els objectius que ens marcarem en cada una d'elles (iteració rere iteració) on hi ha les diferències més accentuades.

Figura 7



ANOTACIONS

Planificació

La planificació comença amb la confecció de les "històries d'usuari". De manera semblant als casos d'ús, es tracta de frases breus escrites pel client (no pas més de tres línies) en què es descriu una prestació o un procés sense cap tipus de tecnicisme (és l'usuari o el client qui les escriu).

Aquestes històries d'usuari serviran per a planificar els llançaments, i també per a fer els tests d'acceptació amb el client. Hauríem de poder estimar el temps ideal de desenvolupament de cada història, que hauria de ser d'una a tres setmanes com a màxim. Si el temps de desenvolupament és més llarg, haurem de partir la història en trossos que no excedeixin aquestes estimacions.

A continuació, podem començar a planificar el pròxim (o primer) llançament del projecte. En la reunió de planificació, haurem d'implicar el client, el gestor del projecte i els desenvolupadors. L'objectiu serà planificar els llançaments següents ordenant les històries d'usuari que falten per desenvolupar. Haurà de ser el client qui dicti l'ordre de les històries d'usuari, i els desenvolupadors qui estimin el temps que trigarien idealment a desenvolupar-lo –*idealment* significa aquí sense tenir en compte dependències, ni altres treballs o projectes, però sí incloent-hi les proves.

Les històries d'usuari se solen reflectir en targetes o trossos de paper que s'agrupen i classifiquen sobre la taula durant la planificació. El resultat haurà de ser un conjunt d'històries que tinguin sentit i que es puguin dur a terme en poc temps. Les podem planificar a partir de dos criteris: basant-nos en el temps fins al llançament següent o en l'abast (entès com el nombre de funcionalitats) que volem que tingui.

Aquí introduïm un concepte nou, la *velocitat* del projecte. Aquesta velocitat ens servirà per a decidir quantes històries d'usuari inclourem en el pròxim llançament (si planifiquem a partir del temps ja fixat), o bé quant trigarem a fer el llançament (si planifiquem per abast ja fixat). La velocitat del projecte és simplement el nombre d'històries d'usuari completades en la iteració anterior. Si es tracta de la primera iteració, caldrà fer una estimació inicial.

ANOTACIONS

La velocitat es pot augmentar si en el transcurs del desenvolupament s'acaben abans d'hora les històries d'usuari previstes. Llavors els desenvolupadors demanaran històries d'usuari que s'havien planejat per al llançament següent. Aquest mecanisme permet als desenvolupadors recuperar-se en els períodes durs i accelerar després el desenvolupament si és possible. Recordem que les històries han de tenir totes una durada màxima i que com més semblant sigui el volum de treball estimat en cada una d'elles, més fiable serà la mesura de velocitat del desenvolupament.

Aquest mètode de planificació ràpid i adaptatiu ens permet fer un parell d'iteracions per a tenir una mesura fiable de la velocitat mitjana del projecte i estimar així més detalladament el pla de llançaments –a més d'haver començat ja a desenvolupar-lo– en l'interval de temps en què altres metodologies trigarien a documentar, planificar i realitzar una estimació completa, que potser no seria tan fiable.

En la reunió de planificació, a l'inici de cada iteració, també caldrà incorporar les tasques que han generat els tests d'acceptació que el client no ha aprovat. Aquestes tasques se sumaran també a la velocitat del projecte, i el client, que és qui escull les històries d'usuari que cal desenvolupar, no podrà triar un nombre més gran que el de la velocitat del projecte.

Els desenvolupadors convertiran les històries d'usuari en tasques (aquestes sí en llenguatge tècnic) d'una durada màxima de tres dies com a ideal de programació per a cada una. S'escriuran en targetes i es posaran sobre la taula per a agrupar-les, eliminar les repetides i assignar-les a cada programador. És molt important que s'eviti afegir més funcionalitats que les que la història d'usuari requereixi estrictament. Aquesta tendència dels gestors de projectes o analistes acostumats a les metodologies tradicionals s'ha d'evitar en models iteratius com aquest, ja que desvirtuen les estimacions i el principi de llançaments freqüents.

Amb les tasques resultants, es tornarà a comprovar que no superem la velocitat del projecte i s'eliminaran o afegiran històries d'usuari fins a arribar al seu valor. És possible que històries d'usuari diferents tinguin tasques en comú, i aquesta fase de la planificació ens permetrà filtrar-les per a poder augmentar la velocitat del projecte afegint més històries d'usuari en aquesta iteració.

ANOTACIONS

En el moment de planificar l'equip de treball encarregat de cada tasca o història, és important la mobilitat de les persones; hem d'intentar que cada dues o tres iteracions tothom hagi treballat en totes les parts del sistema. Conceptes com la programació per parelles, que veurem en apartats següents, també poden ajudar.

Disseny

Durant el disseny de la solució, i de cada història d'usuari que ho requereixi, la màxima simplicitat possible és la clau per a l'èxit d'aquesta metodologia. Sabent que un disseny complex sempre triga més a desenvolupar-se que un de simple, i que sempre és més fàcil afegir complexitat a un disseny simple que treure-la d'un de complex, sempre haurem de fer les coses tan senzilles com puguem evitant afegir funcionalitats no previstes en aquesta iteració.

Així mateix, és important i s'ha demostrat que és molt útil trobar una metàfora per a definir el sistema. És a dir, un procés o sistema que tots coneguin (el client, el gestor del projecte, els desenvolupadors) i que puguin identificar amb el projecte que es desenvolupa. Trobar una bona metàfora ajudarà a tenir:

- Visió comuna: tothom estarà d'acord a reconèixer on és el nucli del problema, i com funciona la solució. Ens ajudarà a veure com serà el sistema o què podria arribar a ser.

- Vocabulari compartit: la metàfora ens ajudarà a suggerir un vocabulari comú per als objectes i processos del sistema. Depenent de la metàfora escollida, el vocabulari pot ser altament tècnic o, al contrari, molt comú.

- Generalització: la metàfora pot suggerir idees o solucions noves. Per exemple, la metàfora "el servei d'atenció al client és una cadena de muntatge" ens suggereix que el problema passa d'un grup de gent a un altre. Però també ens fa pensar què passa quan el problema arriba al final de la cadena...

- Arquitectura: la metàfora donarà forma al sistema, identificarà els objectes clau i ens suggerirà característiques de les seves interfícies.

ANOTACIONS

Per al disseny del sistema, se suggereix fer reunions entre tots els desenvolupadors implicats, en què es prenguin les decisions usant targetes CRC (***class***, ***responsabilities*** i ***collaboration*** –'classe, responsabilitats i col·laboració'–). Cada targeta representa un objecte del sistema: el nom apareix escrit en la part superior, les responsabilitats en la part esquerra i els objectes amb què col·labora en la part dreta.

Figura 8

El procés de disseny es realitza creant les targetes –a l'inici només hi escriurem el nom, la resta ja l'anirem completant– i situant-les a prop de les que comparteixen interfícies o crides. Les targetes corresponents a objectes que hereten o que són interfícies d'unes altres es poden situar a sobre o a sota.

Aquest mecanisme ajuda a fer que tothom participi i aporti les seves idees en el disseny movent les targetes sobre la taula a mesura que aquest progressa. Si s'arriba a un punt mort i es decideix tornar a començar, serà tan simple com "netejar" la taula i tornar a col·locar les targetes. No haurem d'esborrar diagrames ni fer treball extra dissenyant alternatives. Quan tinguem el disseny definitiu, també serà més fàcil que els participants el mantinguin en la memòria i és llavors quan entrarem minuciosament en cada objecte i farem els diagrames necessaris.

Finalment, és clau també el concepte de refactorització, és a dir, el fet de fer els canvis necessaris en les parts de codi que ho requereixin sense modificar-ne la funcionalitat o la interacció amb la resta del sistema. A mesura que avancem en iteracions en el projecte, ens veurem obligats a modificar o ampliar parts de codi ja escrites anteriorment. En aquell moment, en comptes de deixar el que funcionava sense tocar-ho i desenvolupar el mòdul addicional per a la

ANOTACIONS

funcionalitat nova, haurem de fer l'esforç de refactoritzar el mòdul que tenim deixant-lo igual de simple però afegint-hi la funcionalitat nova. Serà sempre molt més fàcil de provar, d'explicar i de comprendre per a la resta de l'equip.

Aquest concepte provocarà canvis en el disseny que havíem fet en iteracions anteriors, però això no és perjudicial, sinó al contrari. Ens haurem d'adonar que el disseny evoluciona iteració rere iteració, per la qual cosa el disseny anterior ja serà obsolet a partir d'aquell moment, i ens hi haurem d'acostumar.

Codificació

Una primera diferència important entre aquesta metodologia i les anomenades *tradicionals* és la disponibilitat del client, que ha de ser total. En lloc de limitar-se a escriure durant setmanes un full de requisits, el client ha de participar en les reunions de planificació, ha de prendre decisions, i ha d'estar disponible per als desenvolupadors durant els tests funcionals i d'acceptació.

A causa de la mobilitat dels desenvolupadors durant el projecte, que participen en cada iteració en parts diferents, és extremadament important acordar uns estàndards de codificació i respectar-los en el desenvolupament. Cada llenguatge de programació té suggeriments o regles més o menys detallades sobre això. Haurem de ser tan concrets com sigui possible, no deixant al lliure albir temes com la sagnia del codi, la sintaxi i els noms de variables, etc. En gairebé tots els casos, hi ha unes convencions recomanades pels creadors del llenguatge que cobreixen aquests aspectes.

Sempre convé tenir en compte aquestes recomanacions i utilitzar-les tal qual o bé adaptar-les a les preferències dels nostres programadors, però en tot cas és imprescindible definir les decisions preses sobre això.

Pel que fa a la mateixa codificació, la programació extrema ens proposa que anteposem la creació dels tests unitaris al desenvolupament de les funcionalitats. Malgrat que explicarem els tests unitaris detalladament en el capítol corresponent, us avancem que bàsicament són trossos petits de codi que proven les funcionalitats d'un ob-

ANOTACIONS

jecte de manera que aquesta prova es pugui incorporar a un procés de proves automatitzat. La majoria dels llenguatges tenen avui dia biblioteques per a crear i executar proves unitàries.

La idea que es forja darrere d'aquesta proposta és que, creant primer les proves que haurà de passar el nostre codi, tindrem una idea més clara del que haurem de codificar, i per a això ens guiarem implementant únicament allò que ens permeti passar la prova. També tindrem avantatges addicionals si sabem quan hem acabat d'implementar la funcionalitat requerida, o si la comprenem.

Una altra característica diferencial de la programació extrema és la programació per parelles. S'ha demostrat que dos programadors, treballant conjuntament, ho fan al mateix ritme que cada un per la seva banda, però el resultat obtingut és de molta més qualitat. Simplement el fet que tots dos seguin davant el mateix monitor i es passin el teclat cada cert temps provoca que mentre un està concentrat en el mètode que està codificant, l'altre pensa com aquest mètode afectarà la resta d'objectes. La interacció, els dubtes i les propostes que sorgeixen redueixen considerablement el nombre d'errors i els problemes d'integració posteriors.

En una metodologia incremental com aquesta, la integració amb el que ja s'ha desenvolupat en iteracions anteriors és clau. No hi ha una solució única per a aquest problema. Depenent del projecte i de l'equip de treball, es poden proposar solucions com per exemple designar "amos" per a cada objecte que coneguin tots els tests unitaris i d'integració que ha de passar i s'encarreguin d'integrar-lo a cada llançament. Una altra alternativa és que cada parella de programadors es faci responsable d'integrar quan tots els tests de la tasca que estaven realitzant passin al 100%. D'aquesta manera, afegim al paradigma *release-often* (distribueix o implanta sovint) l'*integrate-often* (integra sovint), amb la qual cosa reduïm enormement les possibilitats de problemes d'integració.

Proves

Ja hem parlat de les proves unitàries en la fase anterior, i les veurem detalladament en l'apartat corresponent. És important insistir en aquest tema, ja que, fins i tot en el cas que tinguem una data de lliu-

ANOTACIONS

rament molt pròxima, la construcció de les proves unitàries ens estalviarà molt més temps del que invertim programant-les, buscant petits errors i protegint-nos contra ells permanentment durant la resta del desenvolupament.

Com més complicada sigui la prova, més necessària és per a assegurar que després el desenvolupament farà el que es requereix.

De vegades, hi ha la tendència a pensar que les proves unitàries es poden fer durant els darrers tres mesos de desenvolupament. Això és erroni, ja que sense aquestes proves el desenvolupament s'allarga aquells tres mesos, i potser més.

Els tests unitaris ajudaran també a la refactorització, ja que asseguraran que els canvis que hàgim introduït en la iteració actual no afectin la funcionalitat.

Quan trobem un error o *bug* en els tests d'acceptació amb el client, o durant l'ús, haurem de crear un test unitari que el comprovi. Així assegurem que no torna a sorgir en iteracions següents.

Els tests d'acceptació es crearan a partir de les històries d'usuari. Una història pot tenir un o diversos tests, segons la funcionalitat que calgui provar. El client és responsable de definir els tests d'acceptació, que haurien de ser tan automatitzables com fos possible. Aquests tests són del tipus "caixa negra", en el sentit que únicament ens defineixen el resultat que ha de tenir el sistema davant d'unes entrades concretes. Els tests d'acceptació que no tinguin èxit generaran noves tasques per a la pròxima iteració, amb la qual cosa afectaran la velocitat del projecte i proporcionaran, a més, una puntuació de l'èxit o el fracàs de cada història d'usuari, o de cada equip de treball.

Conclusions

Hem vist de manera ràpida i introductòria la metodologia de programació extrema, i n'hem comentat les particularitats respecte de les metodologies tradicionals. Sense cap mena de dubte, introdueix

ANOTACIONS

alguns conceptes nous i fins i tot discutibles com la programació per parelles, però també usa tècniques més convencionals com les targetes CRC.

Prescindint dels detalls, el que proposa XP és un canvi de paradigma durant tota la metodologia. Les eines concretes, com les proves unitàries, les històries d'usuari, la refactorització, etc. no són res més que recursos que també podríem utilitzar en altres metodologies. El que veritablement és destacable d'XP és la seva manera d'ordenar el cicle de vida del projecte i el fet d'involucrar-se amb el client. Sense això, no fem XP.

Les seves característiques principals fan que sigui molt adequada per a projectes de programari lliure, encara que, com hem vist en la seva breu història, no va ser concebuda amb aquest objectiu. Altres aspectes com la programació per parelles o el disseny col·laboratiu amb targetes CRC no són tan aplicables en aquest àmbit.

1.5.2. Metodologia clàssica: Métrica v3

Métrica v3 és la metodologia de planificació, desenvolupament i manteniment de sistemes d'informació promoguda per la Secretaria del Consell Superior d'Informàtica del Ministeri d'Administracions Públiques, que és l'òrgan interministerial responsable de la política informàtica del Govern espanyol.

Encara que el seu àmbit inicial és el de les administracions públiques, les millores introduïdes en la versió 3 i l'ús més adequat d'estàndards i normes d'enginyeria del programari fan que el seu abast es pugui ampliar a les administracions autonòmiques, locals i a la resta d'empreses i organitzacions.

Entre les millores introduïdes a la versió 3.0 (publicada l'any 2000), destaca la incorporació de nous mètodes i tecnologies (client/servidor, interfície gràfica d'usuari, orientació a l'objecte) i també la incorporació d'aspectes de gestió (que la metodologia denomina *interfícies*) per a millorar aspectes que no pertanyen a una sola fase, sinó que intervenen durant tot el projecte, com ara la seva gestió, qualitat i seguretat, entre d'altres.

ANOTACIONS

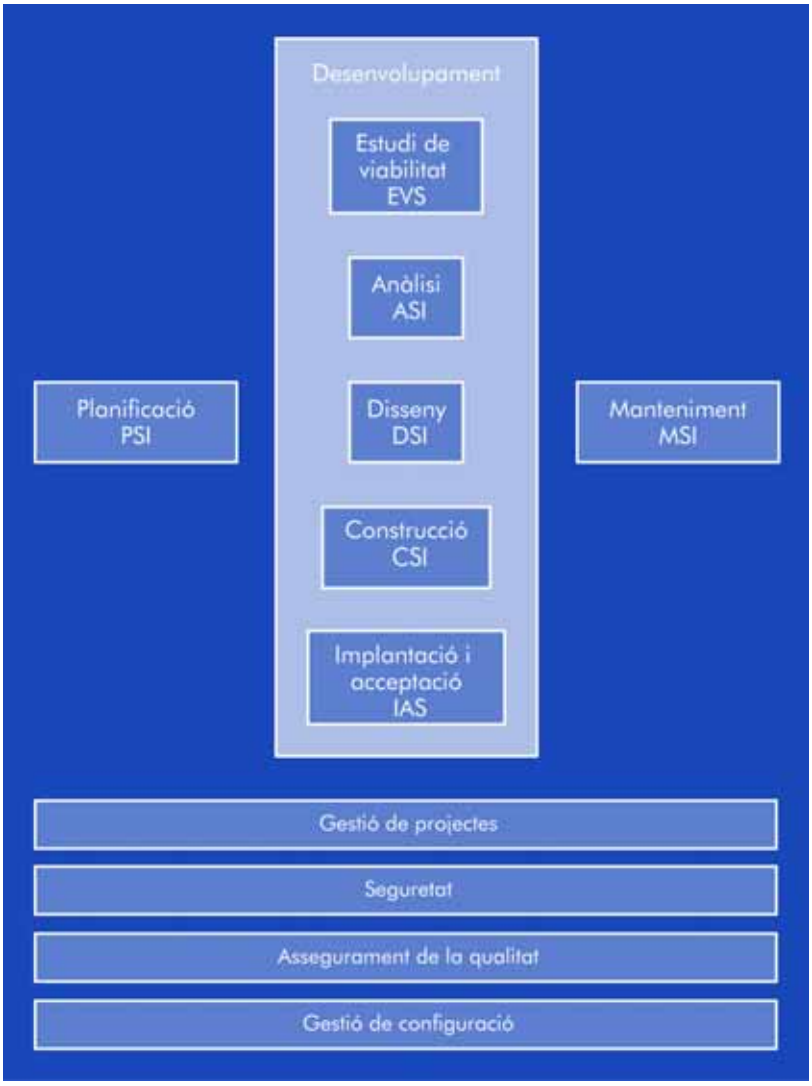
L'estructura de la metodologia segueix el mètode clàssic en cascada basat en els processos següents:

- Planificació
- Desenvolupament
- Manteniment

Cada procés dels anteriors detalla les activitats i tasques que cal realitzar, de manera que per a cada tasca s'indiquen:

- Les tècniques i pràctiques que cal utilitzar.
- Els responsables de realitzar-la.
- Els seus productes d'entrada i sortida.

Figura 9



ANOTACIONS

L'aspecte més destacable d'aquesta metodologia no és tant el que pugui aportar com a innovació a l'enginyeria del programari en si mateixa com l'esforç que s'ha fet per a posar a disposició del públic una metodologia completa, més o menys actualitzada, i que representa un marc inicial de referència per a presentar projectes a l'Administració pública –que ho exigeix com a requisit–, però que podem adaptar a la nostra empresa o projecte en el sector privat, si la nostra organització se sent més còmoda amb models de desenvolupament clàssics.

Métrica v3 defineix molt bé els documents d'entrada de cada procés, activitat i tasca, i també el resultat que genera. Durant aquest apartat i els següents, destacarem els més rellevants. Si voleu ampliar informació, la documentació disponible és molt extensa, i teniu exemples, cursos d'autoformació, i també programes auxiliars d'ajuda i selecció d'eines compatibles amb la metodologia.

Planificació de sistemes d'informació

Aquest procés té com a objectiu últim crear el pla de sistemes d'informació (PSI) de l'organització. Adaptant el marc i els objectius, en podem utilitzar les activitats per a generar el pla del projecte concret en què treballem.

Entre les activitats que haurem de realitzar, destaquen:

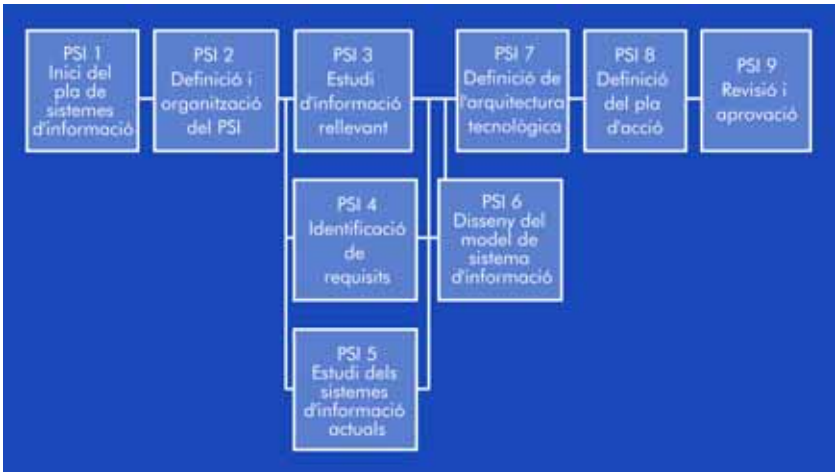
- Descripció de la situació actual.
- Conjunt de models que constitueixen l'arquitectura de la informació.
- Priorització i calendari dels projectes que s'han de desenvolupar.
- Avaluació dels recursos necessaris.
- Pla de seguiment i compliment sota una perspectiva estratègica i operativa.

ANOTACIONS

El pla ha de ser realitzat en un nivell alt, sense tecnicismes i amb una perspectiva estratègica i operativa. Així mateix, és fonamental que la direcció s'impliqui en el seu desenvolupament. Per a descriure la situació actual, el nivell de detall dependrà de la documentació de què es disposi i de la predisposició de l'organització a una substitució total o parcial del sistema d'informació actual.

El quadre complet d'activitats que cal dur a terme durant aquesta fase és el següent:

Figura 10



Activitats

1. Inici del pla de sistemes d'informació

L'objectiu d'aquesta primera activitat és obtenir la descripció general del pla de sistemes d'informació, identificant els objectius generals en què se centra i l'àmbit que afecta dins l'organització.

S'hauran d'identificar també els participants en l'elaboració del pla, que en definiran els seus factors crítics d'èxit.

El mètode per a obtenir aquesta informació és participar en sessions de treball del comitè de direcció fins que nomeni els gestors del projecte.

ANOTACIONS

2. Definir i organitzar el pla

Una vegada que s'han determinat els responsables del projecte i els seus objectius, haurem de detallar l'abast del pla, organitzar l'equip de persones que el duran a terme i elaborar un calendari d'execució. Aquest calendari haurà d'incloure una valoració en termes econòmics a partir d'estimacions que permetin prendre decisions quant a la seva aprovació.

Una vegada definit el pla de treball per a realitzar el pla, s'haurà de comunicar a la direcció perquè l'aprovi definitivament.

3. Estudiar la informació rellevant

La primera activitat, una vegada aprovat el pla, haurà de ser recopilar i analitzar tots els antecedents generals que puguin afectar els processos i recursos previstos en el pla, i també els resultats que van presentar. Seran especialment interessants els estudis realitzats anteriorment relatius tant als sistemes d'informació de l'àmbit del pla com al seu entorn tecnològic.

La informació que obtinguem serà útil per a incloure requisits en activitats posteriors.

4. Identificar requisits

La presa de requisits es farà estudiant els processos en l'àmbit del pla. El model d'aquests processos, juntament amb les seves activitats i funcions, la informació implicada en aquests i les unitats organitzatives o recursos que hi participen, s'obtindrà de reunions de treball amb usuaris experts o tècnics implicats.

Una vegada contrastades les conclusions, s'elaborarà el model de processos implicats unificant tant com es pugui els que tinguin relació entre ells, amb l'objectiu de tenir una visió com més general millor.

ANOTACIONS

A continuació, s'hauran d'analitzar les necessitats d'informació de cada procés modelat anteriorment, i s'elaborarà un model d'informació que reflecteixi les entitats principals i les relacions que hi ha entre elles en termes d'informació d'entrada/sortida, les seves activitats i les seves funcions.

Finalment, elaborarem el catàleg de requisits a partir de la informació obtinguda en activitats anteriors i de les necessitats d'informació i procés obtinguts prèviament. És important prioritzar els requisits sobre la base de les opinions dels usuaris i els objectius del pla.

5. Estudiar els sistemes d'informació actuals

A partir dels sistemes actuals afectats pel pla, s'haurà de valorar la situació actual basant-se en criteris com la facilitat de manteniment, documentació, flexibilitat, facilitat d'ús, nivell de servei, etc. Els usuaris aportaran aquí els elements de valoració més importants.

És important obtenir una valoració tan objectiva com sigui possible, ja que aquesta influirà sobre la decisió de millora o substitució de cada procés o sistema.

6. Dissenyar el model de sistemes d'informació

En aquest punt, tindrem prou informació per a decidir en quins sistemes apliquem millores o bé quins sistemes substituïm, i en cada cas, quin haurà de ser el sistema resultant.

Una vegada preses aquestes decisions, hem d'obtenir el model de sistemes d'informació, que inclourà un diagrama de representació de tots ells amb les seves connexions i interfícies i una descripció de cada sistema amb el conjunt de processos i requisits que cobreix.

ANOTACIONS

7. Definir l'arquitectura tecnològica

En aquesta activitat, hem de proposar una arquitectura tecnològica, a alt nivell, que doni suport al model de sistemes d'informació, i que pot incloure, si és necessari, opcions. Per a aquesta activitat, tindrem en compte sobretot els requisits de caràcter tecnològic, encara que pot ser necessari comprendre el catàleg complet de requisits.

La definició i elecció entre les alternatives possibles s'haurà de realitzar sobre la base de l'entorn actual, els estàndards, i basant-nos en l'anàlisi cost/benefici i l'impacte en l'organització de cada alternativa.

8. Definir el pla d'acció

El pla d'acció serà el que definirà els projectes concrets que cal dur a terme per a implantar els sistemes i models d'informació definits en les activitats anteriors.

Dins del pla d'acció, s'inclourà un calendari de projectes amb possibles alternatives i una estimació de recursos. Per a elaborar aquest pla, serà important tenir en compte les prioritats que haurem marcat en matèria de requisits i els sistemes implicats en cada un d'ells.

Finalment, també serà important fer un pla de manteniment i control de l'execució dels projectes.

9. Revisar i aprovar el pla

Finalment, hem de presentar l'arquitectura d'informació dissenyada i el pla d'acció als responsables de la direcció. Millorarem la proposta si cal i obtindrem l'aprovació final.

ANOTACIONS

Desenvolupament de sistemes d'informació

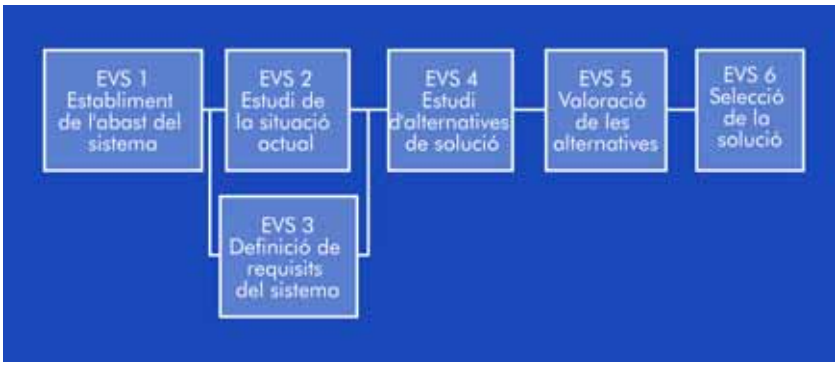
El pla de sistemes tenia com a objectiu proporcionar un marc estratègic que servís com a referència. Una vegada completat el pla d'acció, començarem a desenvolupar cada projecte.

En el projecte, algunes activitats seran rèpliques d'activitats realitzades en el pla de sistemes (presa de requisits, anàlisi de la situació actual, etc.). Si el pla s'ha realitzat amb prou nivell de detall, o es refereix únicament a un projecte concret, aquestes activitats no seran necessàries. En cas contrari, seran les primeres que es duguin a terme en aquest procés.

Estudi de viabilitat

Si el pla de sistemes ens ha deixat amb diverses alternatives per a un projecte en concret, en la primera fase haurem d'estudiar la viabilitat de cada una en termes d'impacte en l'organització i de la inversió que cal realitzar.

Figura 11



En tot cas, la primera activitat que Métrica v3 defineix en la fase de desenvolupament és l'estudi de la viabilitat del projecte, que hauria de generar un o diversos documents (segons les alternatives considerades) amb un índex com el següent:

Solució proposada:

Descripció de la solució

Model de descomposició en subsistemes

ANOTACIONS

Matriu processos / localització geogràfica
Matriu dades / localització geogràfica
Entorn tecnològic i comunicacions
Estratègia d'implantació global del sistema
Descripció de processos manuals

Si l'alternativa inclou desenvolupament:

Model abstracte de dades / model de processos
Model de negoci / model de domini

Si l'alternativa inclou un producte de programari estàndard de mercat:

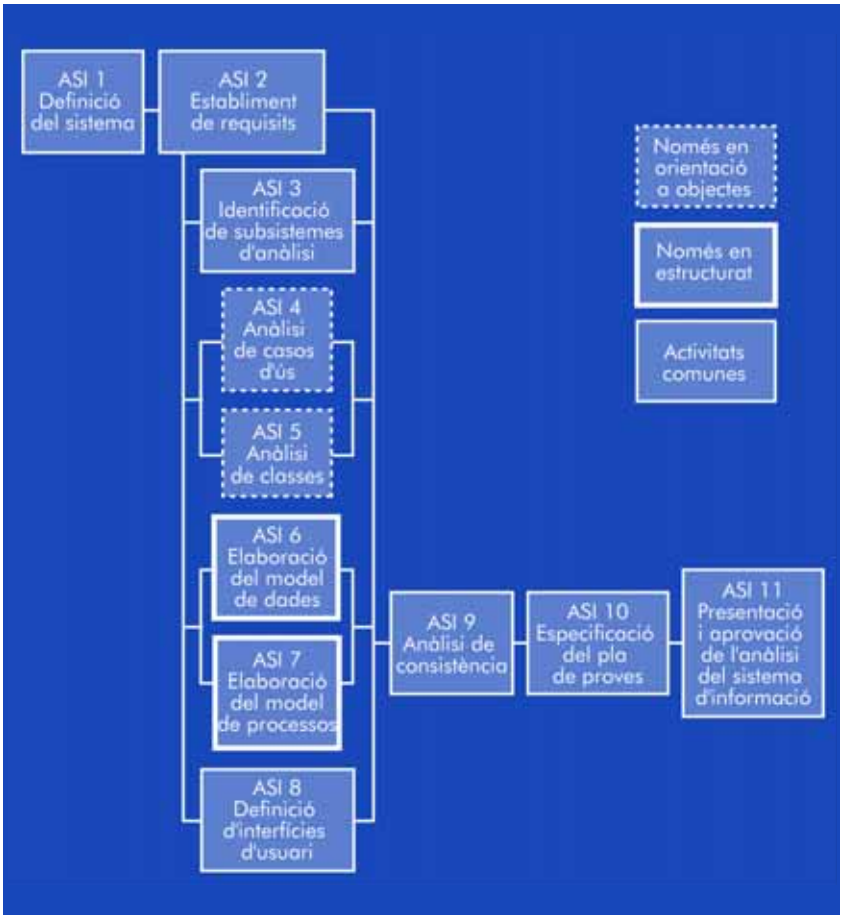
Descripció del producte
Evolució del producte
Costos causats pel producte
Estàndards del producte
Descripció d'adaptació (si s'escau)
Context del sistema (amb la definició de les interfícies)
Impacte en l'organització de la solució
Cost / benefici de la solució
Valoració de riscos de la solució
Enfocament del pla de treball de la solució
Planificació de la solució
Anàlisi del sistema d'informació

L'objectiu d'aquest procés és obtenir una especificació detallada del sistema d'informació que satisfaci les necessitats dels usuaris i serveixi de base per al disseny posterior del sistema.

Métrica v3 suporta el desenvolupament amb llenguatges tant estructurats com orientats a l'objecte, però les activitats particulars en cada cas s'integren a una estructura comuna.

ANOTACIONS

Figura 12



Les dues primeres activitats seran recuperar i aprofundir la definició i presa de requisits del sistema amb l'objectiu de detallar al màxim el catàleg de requisits i de descriure amb precisió el sistema d'informació.

Tot seguit, començarà el cos de l'anàlisi, formada per quatre activitats que es realimentaran entre elles fins a analitzar completament el sistema. De les quatre activitats, dues són comunes per a desenvolupaments estructurats o orientats a l'objecte, mentre que les altres dues són diferents en cada cas.

Les activitats són:

- Identificar subsistemes d'anàlisi: facilitar l'anàlisi descomponent el sistema en subsistemes. Activitats posteriors poden obligar a revisar aquesta descomposició.

ANOTACIONS

- Analitzar casos d'ús (elaborar el model de processos en llenguatge estructurat).

- Analitzar classes (elaborar el model de dades en llenguatge estructurat).

- Definir interfícies d'usuari.

Finalment, haurem de verificar i validar els models per a assegurar que són:

- Complets: cada model conté tota la informació necessària.

- Consistents: cada model és coherent amb la resta.

- Correctes: el model segueix les normes de qualitat, estàndards i nomenclatura determinats en activitats anteriors.

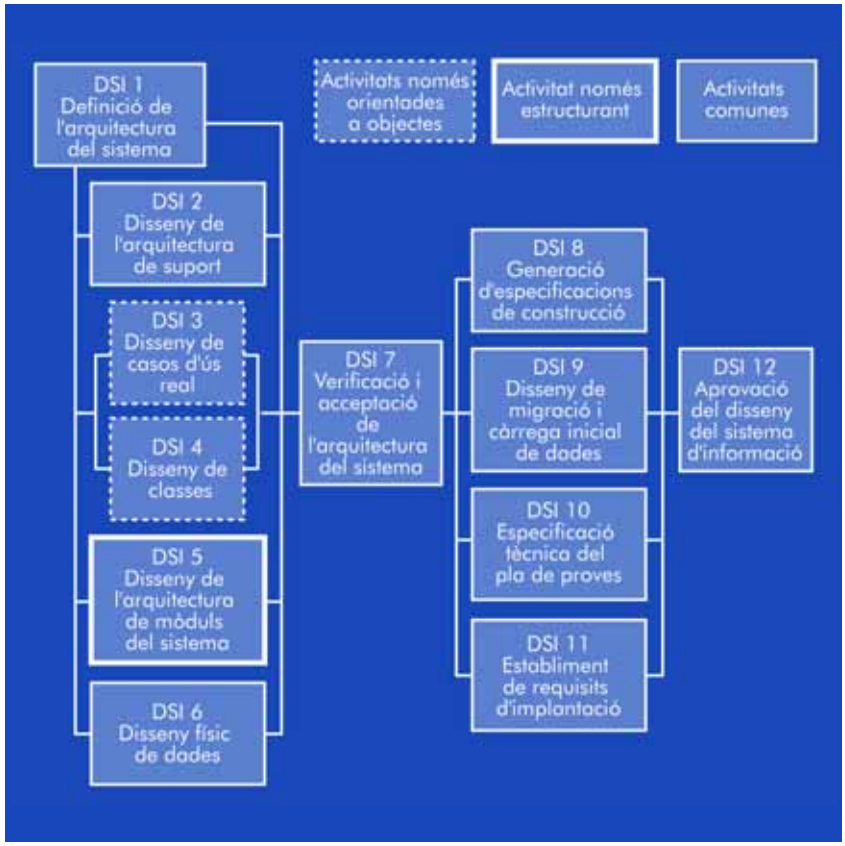
Com a darrera activitat de l'anàlisi, s'haurà d'especificar el pla de proves del sistema. Es tracta únicament d'iniciar-ne l'especificació, que es detallarà en activitats posteriors: n'hi ha prou de preveure'n l'abast, els requisits de l'entorn de proves i la definició de les proves d'acceptació.

Dissenyar el sistema d'informació

L'objectiu d'aquesta activitat és especificar detalladament l'arquitectura del sistema i de l'entorn tecnològic que li donarà suport juntament amb els components del sistema d'informació.

ANOTACIONS

Figura 13



A partir d'aquesta informació, especificarem la construcció del sistema i també farem la descripció tècnica del pla de proves, definirem els requisits d'implantació i dissenyarem els procediments de migració i la càrrega inicial si s'escau.

Les activitats d'aquest procés es divideixen en dos blocs:

El primer bloc d'activitats es realitza paral·lelament, i comprèn:

- Definició de l'arquitectura del sistema d'informació: s'estableix la partició física del sistema d'informació i la seva correspondència amb els subsistemes de disseny. També en definirem els requisits d'operació, administració, seguretat i control d'accés. Els subsistemes de disseny s'hauran de classificar en els tipus següents:

– Subsistemes de suport: contenen elements o serveis comuns al sistema i a la instal·lació. Generalment originats per la interacció amb la infraestructura tècnica o per la reutilització.

ANOTACIONS

– Subsistemes específics: contenen els elements propis del sistema d'informació, com a continuïtat dels vistos en l'activitat d'anàlisi del sistema.

- Disseny de casos d'ús reals: és el disseny detallat del comportament del sistema per als casos d'ús, juntament amb la interfície d'usuari.

- Disseny de classes: detallat amb atributs, operacions, relacions, mètodes i l'estructura jeràrquica de tot el model.

- Disseny físic de dades una vegada obtingut el model de classes.

En el segon bloc d'activitats, es generen totes les especificacions necessàries per a la construcció:

- Generació d'especificacions de construcció que fixen les directrius per a construir els components del sistema.

- Disseny de la migració i càrrega inicial de dades.

- Especificació tècnica del pla de proves. Realitzar un catàleg d'excepcions permetrà establir un conjunt de verificacions relacionades amb el disseny o amb la mateixa arquitectura.

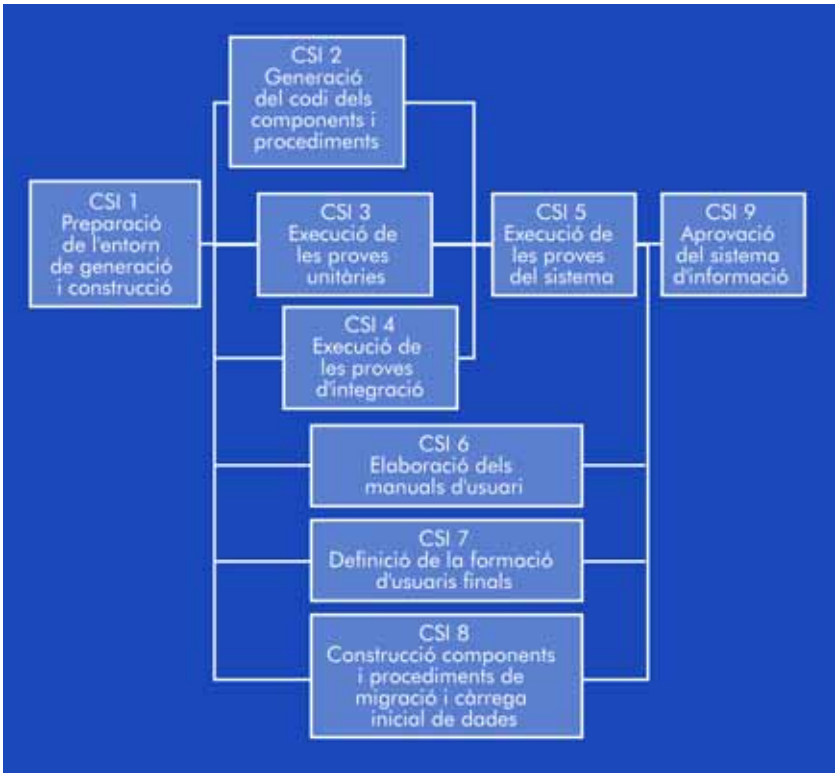
- Establiment de requisits d'implantació.

Construcció del sistema d'informació

En aquest procés, es genera el codi dels components del sistema d'informació, es desenvolupen tots els procediments d'operació i seguretat i s'elaboren els manuals d'usuari i d'explotació.

ANOTACIONS

Figura 14



Un objectiu clau en aquesta fase serà assegurar el funcionament correcte del sistema perquè s'accepti i s'implanti posteriorment. Per a aconseguir-ho, en aquest procés es faran les proves unitàries, les proves d'integració dels subsistemes i les proves de sistema d'acord amb el pla establert.

La base per a construir el sistema és l'especificació de construcció obtinguda en el procés de disseny anterior. Una vegada configurat l'entorn de desenvolupament, es farà la codificació i les proves d'acord amb les activitats següents:

- Generar el codi: d'acord amb les especificacions de construcció i amb el pla d'integració dels subsistemes.

- Executar les proves unitàries: d'acord amb el pla de proves dissenyat.

- Executar les proves d'integració: verificacions associades als components i subsistemes.

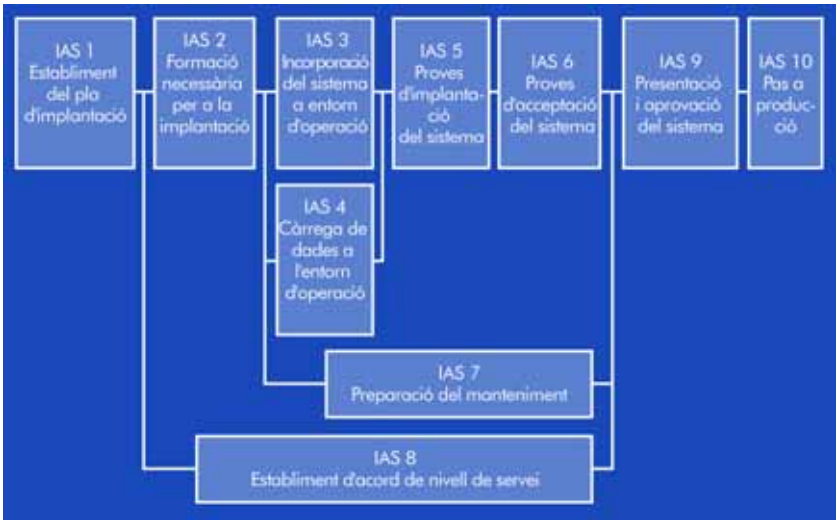
ANOTACIONS

- Executar proves de sistema: integració final del sistema d'informació, comprovant tant les interfícies entre subsistemes i sistemes externs com els requisits.
- Elaborar el manual d'usuari: documentació d'usuari final i d'explotació.
- Formar usuaris finals.
- Construir els components i procediments de migració i càrrega inicial de dades.

Implantació i acceptació del sistema

L'objecte d'aquest procés és lliurar el sistema i que s'accepti totalment, i també realitzar les activitats necessàries perquè es comenci a produir.

Figura 15



En aquest procés haurem de fer el següent:

- Revisar l'estratègia d'implantació determinada en l'estudi de viabilitat del sistema.
- Preparar la infraestructura necessària, la instal·lació dels components, l'activació dels procediments manuals i automàtics i la migració o càrrega inicial de dades.

ANOTACIONS

- Realitzar les proves d'implantació i acceptació del sistema en la seva totalitat.

- Preparar el manteniment.

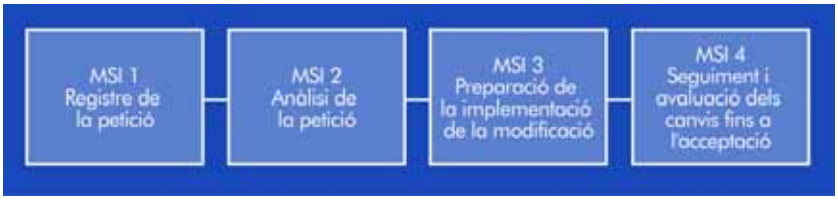
- Determinar els requisits dels serveis que requereix el sistema. Caldrà distingir entre serveis d'operacions (seguretat, comunicacions, etc.) i serveis al client (atenció a l'usuari, manteniment, etc.).

- El pla d'implantació pot definir aquest procés de manera iterativa, posant en marxa els subsistemes i realitzant aquestes activitats per a cada un d'ells.

Manteniment de sistemes d'informació

El procés de manteniment té com a objecte posar en marxa una versió nova del sistema o projecte a partir de les peticions dels usuaris amb motiu d'un problema detectat en el sistema o per una necessitat de millorar-lo.

Figura 16



Una vegada rebuda una petició, s'incorpora a un catàleg de peticions i es diagnostica de quin tipus de manteniment es tracta:

- Correctiu: canvis precisos per a corregir errors.

- Evolutiu: modificacions necessàries per a cobrir l'expansió o canvi en les necessitats.

- Adaptatiu: modificacions que afecten l'entorn en què opera el projecte, canvis de configuració, de maquinari, bases de dades, comunicacions, etc.

ANOTACIONS

- Perfectiu: accions dutes a terme per a millorar la qualitat interna dels sistemes. Reestructuració de codi, optimització de rendiment, eficiència, etc.

El pas següent que defineix Métrica v3 és determinar la responsabilitat en atendre la petició. La petició pot ser acceptada o rebutjada: si es rebutja, es registra i acaba el procés; si s'accepta, caldrà estudiar la viabilitat del canvi, verificar i reproduir el problema i estudiar l'abast de la modificació.

El termini i urgència de la petició s'ha d'establir segons els paràmetres de l'estudi anterior. Mitjançant aquesta anàlisi, la persona encarregada del manteniment haurà de valorar el cost i l'esforç necessaris per a implementar la modificació.

Les tasques de processos de desenvolupament corresponen a les dels processos d'anàlisi, disseny, construcció i implantació. Finalment, és important establir un pla de proves de regressió que asseguri la integritat del sistema afectat.

El registre de les peticions rebudes es pot utilitzar també per a fins estadístics (peticions ateses, sistemes afectats, etc.), ja que un registre minuciós de les activitats realitzades i una documentació extensa dels canvis incorporats al sistema repercutiran directament en la qualitat dels sistemes resultants i en el fet de tenir controlat el cost del manteniment.

Interfícies

Com que Métrica v3 és una metodologia fortament estructurada, i malgrat que la seva aplicació proporciona sistemes amb qualitat i seguretat, s'han definit unes interfícies que els reforcen i altres aspectes durant tots els seus processos. Les interfícies descrites en la metodologia són:

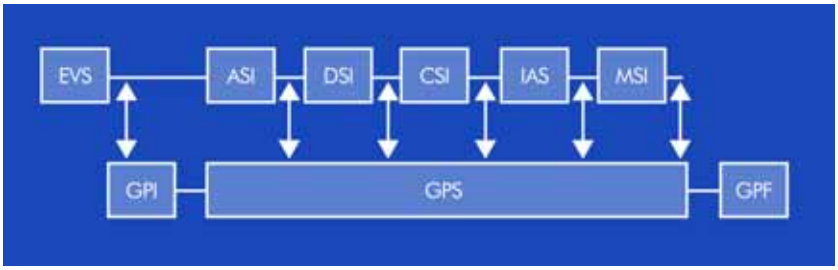
- Gestió de projectes
- Seguretat
- Assegurament de la qualitat
- Gestió de la configuració

ANOTACIONS

Gestió de projectes

La gestió de projectes té com a finalitat principal planificar, seguir i controlar les activitats i els recursos humans i materials que intervenen en el desenvolupament del sistema d'informació o del projecte. L'objectiu d'aquest control és identificar en tot moment els problemes que es produeixin i resoldre'ls o ser capaços de mitigar-los immediatament.

Figura 17



- **GPI**: s'inicien les activitats del projecte en concloure'n l'estudi de viabilitat, que consistiran a estimar l'esforç i a planificar el projecte. Per a estimar l'esforç, partirem de la descomposició en subsistemes obtinguts de l'estudi de viabilitat i dels elements implicats (funcions, entitats i dades en desenvolupaments estructurats o classes, propietats i mètodes en desenvolupaments orientats a l'objecte).

  Per a calcular l'esforç, Métrica v3 proposa utilitzar tècniques conegudes com el *mètode Albrecht* o els *punts de funció* (en desenvolupament estructurat), o bé la mètrica Staffing Size en el cas de desenvolupament orientat a l'objecte.

  Métrica v3 no fa recomanacions quant a l'estratègia de desenvolupament, simplement enumera les que hi ha (en cascada, per subsistemes, per prototip o híbrida) i deixa l'elecció en mans del cap de projecte.

  Quant a la descomposició detallada d'activitats i l'assignació de recursos, Métrica v3 recomana usar el mètode PERT i els diagrames de Gantt, juntament amb taules d'assignació de recursos.

ANOTACIONS

- **GPS**: Les activitats de seguiment i control comprenen des de l'assignació de tasques fins a la seva acceptació interna, incloent-hi la gestió d'incidències, els canvis en els requisits i la vigilància del desenvolupament correcte de les tasques i activitats establertes en la planificació.

  Per a això, Métrica v3 enumera les activitats que han de tenir lloc paral·lelament a l'anàlisi, disseny, construcció, implantació i manteniment del projecte. Les resumim en la figura següent.

Figura 18

- **GPF**: finalment, en concloure el projecte, es realitzen les activitats per a tancar-lo i es registra la documentació que inclou.

  El tancament del projecte consisteix a resumir totes les dades que es considerin interessants perquè poden servir de referència a projectes futurs, aprofitant les experiències passades i mirant de no incórrer en els mateixos errors.

Seguretat

L'objectiu d'aquesta interfície és incorporar mecanismes de seguretat addicionals als que es proposen com a part de la mateixa metodologia establint un conjunt d'activitats que tenen lloc durant tot el procés.

ANOTACIONS

Dins la interfície, hi ha dos tipus d'activitats diferenciats:

- Activitats relacionades amb la seguretat intrínseca del sistema que es desenvoluparà. Són activitats d'anàlisi detallada dels requisits de seguretat que tenen lloc durant la planificació, l'estudi de viabilitat i l'anàlisi i disseny del sistema.

- Activitats que s'encarreguen de la seguretat del mateix procés de desenvolupament.

Si en l'organització ja hi ha un pla de seguretat o metodologia d'anàlisi i gestió de riscos, s'haurà d'aplicar per a detectar aquelles necessitats que no estiguin cobertes i per a prendre les decisions a partir dels riscos que es vulguin assumir o mitigar.

Si no hi ha un pla de seguretat, caldrà desenvolupar-lo des del principi. El pla haurà de recollir les mesures de seguretat actives, preventives i reactives per a respondre a situacions en què es pugui produir un error, reduint-ne l'efecte, tant si tractem amb el sistema d'informació o projecte en si mateix com durant el procés de desenvolupament.

Assegurament de la qualitat

L'objectiu d'aquesta interfície és proveir un marc de referència per a definir i posar en marxa plans específics d'assegurament de la qualitat del projecte. El que es pretén és donar confiança que el producte compleix amb els requisits.

Métrica v3 recomana que el grup d'assegurament de la qualitat sigui totalment independent del de desenvolupament, i tindrà com a missió identificar les possibles desviacions en els estàndards, requisits i procediments establerts, i també comprovar que s'han dut a terme les mesures preventives o correctores necessàries.

Per a un resultat òptim d'aquesta interfície, s'haurà d'aplicar des de l'estudi de la viabilitat del sistema i durant tot el desenvolupament en els processos d'anàlisi, disseny, construcció, implantació i manteniment del projecte.

ANOTACIONS

Gestió de la configuració

L'objectiu d'aquesta interfície és crear i mantenir un registre de tots els productes creats o usats durant el projecte. Les seves activitats tenen lloc durant tot el projecte, s'encarregaran de mantenir la integritat dels productes, d'assegurar-nos que cada persona involucrada disposa de l'última versió de tots ells i que disposem de tota la informació sobre canvis produïts en les seves configuracions, requisits, etc.

Quan parlem de productes, no ens referim únicament a codi font i executables, sinó també a tot el material i documentació generats durant el projecte com diagrames, manuals, estudis i anàlisis que s'hagin revisat, etc.

Ens ajudarà a valorar l'impacte de cada canvi en el sistema i a reduir-ne el temps d'implementació. El compliment i cost d'implementació d'aquesta interfície no s'han de menysprear; si l'organització ja disposava d'un sistema de gestió de configuracions, les seves activitats es desenvoluparan amb més facilitat i rapidesa.

La informació que es podria sol·licitar al sistema de gestió de configuració és molt àmplia:

- Informació sobre una fase del projecte concreta, anàlisi, construcció, etc.
- Informació sobre la "vida" d'un producte, les seves versions, persones involucrades, etc.

Les seves activitats es distribuiran durant les diferents fases. Una vegada determinada la selecció de la solució en l'estudi de viabilitat, se seleccionarà l'entorn tecnològic en què es registrarà la configuració del projecte, i a partir d'aquest moment en endavant es posarà en marxa el procés.

En cada una de les fases següents –anàlisi, disseny, etc.– s'aniran registrant els canvis en els productes, i al final de cada fase s'anotaran en el registre global.

ANOTACIONS

Evidentment, en el procés de manteniment, aquesta interfície adquireix una importància especial, ja que la informació de què disposarem quan valorem cada canvi nou reduirà i compensarà el que hem invertit durant les altres fases. El treball també serà intens, ja que se succeiran versions noves més ràpidament i caldrà incorporar-les i notificar-les en el registre de configuracions.

Conclusions

Tant pel seu llenguatge com pel tipus i nombre d'activitats plantejades, es detecta ràpidament que Métrica v3 és una metodologia impulsada per una administració pública. Tal com hem comentat a l'inici, el seu mèrit resideix en la mateixa elaboració de la metodologia i no tant en els seus plantejaments, que són molt clàssics.

Si ens sentim còmodes amb aquest tipus de metodologies, un aspecte molt interessant és la documentació que proporciona sobre les tècniques (documents, plantilles, llistes de comprovació, etc.) que es poden aplicar en cada activitat. També és molt convenient la definició concreta que realitza sobre les entrades i sortides de cada activitat. Encara que la nostra metodologia basada en Métrica v3 no segueixi al peu de la lletra totes les activitats plantejades, un estudi dels documents que haurem d'obtenir i utilitzar en cada fase pot ser de gran ajuda per a assegurar-nos que no ens deixem res abans de passar a la fase següent del projecte.

1.6. Resum

En aquest capítol, hem fet un repàs general de l'enginyeria del programari. Hem vist els problemes que van tenir els primers gestors de grans projectes i hem conegut els conceptes que s'han de tenir en compte en el moment que es planteja un projecte de programari de qualsevol envergadura.

Els dos exemples de metodologies plantejats, encara que molt diferents, intenten solucionar els mateixos problemes, i organitzacions o projectes diferents es trobaran més pròxims a una o a l'altra indistintament.

ANOTACIONS

En la resta de capítols, veurem l'aplicació pràctica de molts dels conceptes introduïts aquí en l'àmbit concret del programari lliure i els estàndards.

1.7. Altres fonts de referència i informació

Albrecht, A.J.; Gaffney S.H. (1983). *Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation*

Beck, K. (1999). *Extreme Programming Explained*. Addison-Wesley Pub Co.

Brooks, F.P. (1995). *The Mythical Man-Month*.

Consejo Superior de Informática. *Métrica v3*. Ministerio de Administraciones Públicas. http://www.csi.map.es/csi/metrica3/

CSE. Center for Software Engineering COCOMO II. http://sunset.usc.edu/research/COCOMOII/

Development Support Center. http://www.functionpoints.com/

Djikstra, E. *The Humble Programmer*. http://www.cs.utexas.edu/users/EWD/ewd03xx/EWD340.PDF

Extreme Programming: A gentle introduction. http://www.extremeprogramming.org/

Libre Software Engineering. http://libresoft.dat.escet.urjc.es

Lorenz, M.; Kidd, J. (1994). *Object-Oriented Software Metrics: A Practical Guide*. Prentice Hall Inc.

Naur, P.; Randell, B. (1968). *Software Engineering*. Informe presentat en una conferència patrocinada pel the Nato Science Comité.

Pair Programming. Wikipedia. http://en.wikipedia.org/wiki/Pair_programming

ANOTACIONS

PERT. http://en.wikipedia.org/wiki/Program_Evaluation_and_Review_Technique

Pressman, R.S. (2004). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Science.

Raymond, E.S. (1997). "The Cathedral and the Bazaar". http://www.catb.org/~esr/writings/cathedral-bazaar/

ReadySET. http://readyset.tigris.org/

The Future of Software Engineering. http://www.softwaresystems.org/future.html

Wikipedia. The Free Encyclopedia. http://www.wikipedia.org

ANOTACIONS

2. Disseny de programari orientat a l'objecte amb UML

2.1. Introducció

L'anàlisi i disseny del programari constitueix una part fonamental en qualsevol projecte, independentment de la seva mida. Totes les metodologies, en un grau més o menys elevat, o amb diferents abastos, donen molta importància a aquestes fases com a pas intermedi entre la presa de requisits i el desenvolupament del projecte.

Com a part del que es considera enginyeria del programari, el modelatge i disseny ha evolucionat durant el temps mitjançant tècniques aportades tant per especialistes de l'àmbit acadèmic com per empreses especialitzades en consultoria i desenvolupament.

Hi ha moltes tècniques orientades a modelar un aspecte dels sistemes d'informació, bases de dades, interfícies d'usuari, components, fluxos de dades, etc., però poques han aportat un enfocament global del problema.

Al final dels anys noranta, una empresa en particular (Rational Corp.) va començar una iniciativa per a desenvolupar un estàndard de modelatge al qual es van afegir científics i altres empreses del sector. Així va néixer UML (*unified modeling language*), que avui dia continua essent el mètode de modelatge més complet i acceptat en la indústria.

El mètode de modelatge està basat en el paradigma de programació orientat a l'objecte, que en aquell moment es començava a popularitzar i avui dia continua essent, amb alguna variació i revisió, el més usat en tot tipus de projectes.

Per això, en aquest capítol començarem comentant aquest paradigma de programació i els seus aspectes fonamentals. No s'ha pretès fer un tractat complet sobre l'orientació a l'objecte, sinó proporcio-

ANOTACIONS

nar una base per a comprendre els conceptes que suportarà cada tipus de diagrama.

A continuació, presentarem un exemple pràctic que anirem seguint durant el capítol i en cada apartat corresponent aplicarem un tipus de diagrama UML.

Finalment, comentarem una altra utilitat d'UML, la generació de codi a partir de certs tipus de diagrames. Presentarem tres eines de codi obert molt populars i en veurem les prestacions en aquest àmbit.

2.2. Objectius

Els objectius que els estudiants haureu d'haver aconseguit en finalitzar el capítol de "Disseny de programari orientat a l'objecte amb UML" del material ***Enginyeria del programari en entorns de programari lliure*** són els següents:

- Tenir clars els conceptes més importants relacionats amb l'orientació a l'objecte tant en l'àmbit de la seva anàlisi i disseny com en l'àmbit d'implementació.
- Conèixer els diferents tipus de diagrames UML, les seves comeses i particularitats.
- Disposar dels coneixements necessaris per a afrontar l'anàlisi i disseny d'un projecte de programari i representar-ne el model mitjançant UML.
- Conèixer algunes eines de modelatge i generació de codi i identificar la més idònia per a un projecte concret.

2.3. Revisió de conceptes del disseny orientat a l'objecte i UML

Aquest apartat té com a objectiu introduir els conceptes de l'anàlisi, el disseny i la programació orientada a l'objecte necessaris per a poder assimilar la resta del capítol i entendre el cas pràctic que plantejarem.

ANOTACIONS

2.3.1. Introducció a l'orientació a l'objecte

L'orientació a l'objecte és un paradigma més de programació en què un sistema s'expressa com un conjunt d'objectes que interactuen entre ells. Un paradigma de programació ens proporciona una abstracció del sistema real en alguna cosa que podem programar i executar, i es pot dir que el tipus d'abstracció està directament relacionat amb els problemes que pot resoldre o almenys amb la facilitat amb què els podrem resoldre. Mentre que el llenguatge d'assemblador és una abstracció del processador, podríem dir que altres llenguatges de programació, com BASIC o C, són abstraccions del mateix llenguatge d'assemblador. Malgrat que han significat un avenç important sobre aquest, encara obliguen els programadors a pensar en termes de l'estructura de l'ordinador en lloc de l'estructura del problema que intenten solucionar.

Altres llenguatges han intentat modelar el problema presentant visions diferents del món (tots els problemes són llistes en LISP, tots els problemes són una cadena de decisions en PROLOG). Aquestes visions són bones solucions per als tipus de problemes que s'hi ajusten, però quan no és així, el paradigma comença a no ser tan útil.

L'orientació a l'objecte fa un pas més i ens proporciona les eines per a representar elements en l'espai del problema concret. No ens imposa cap restricció ***a priori***, de manera que el programador no està limitat a una classe de problemes. Els elements en l'espai del problema i la seva representació és el que anomenem ***objectes***.

Alan Kay va resumir les cinc característiques bàsiques de SmallTalk, el primer llenguatge de programació purament orientat a l'objecte, en què estan basats tant C++ com Java:

- **Tot és un objecte**: pensem en un objecte com una "gran variable"; emmagatzema dades, però també li podem fer peticions perquè faci operacions sobre ella mateixa. En teoria, podem escollir qualsevol component conceptual del problema que volem resoldre (estudiants, edificis, serveis, etc.) i representar-lo com un objecte en el nostre programa.

ANOTACIONS

- **Un programa és un conjunt d'objectes en què uns diuen als altres què han de fer mitjançant missatges**: per a fer una petició a un objecte, li enviem un missatge. Podem pensar en el missatge com la petició per a cridar un mètode que pertanyi a un objecte (enviarNota, activarAlarma, imprimirInforme).

- **Cada objecte té la seva memòria pròpia feta d'altres objectes**: dit d'una altra manera, crearem un nou tipus d'objecte empaquetant altres objectes que ja existeixen. Així podrem construir programes de molta complexitat amagant-la en objectes simples.

- **Cada objecte té un tipus**: o dit d'una altra manera, cada objecte és una instància d'una classe. Aquí utilitzem ***classe*** com a sinònim de ***tipus***. La característica més important d'una classe és "quins missatges li podrem enviar?".

- **Tots els objectes d'un tipus particular poden rebre els mateixos missatges**: com que un objecte de tipus "cercle" és també un objecte de tipus "figura", un cercle podrà rebre missatges per a figures. Això significa que podem escriure codi que parli amb "figures" i automàticament estarem preparats per a parlar amb qualsevol objecte d'aquest tipus.

La definició del paradigma no s'ha arribat a consensuar mai, però en la seva forma més general podríem dir que expressa el fet de veure el programa en termes de les coses (els objectes) que manipula, en lloc de les accions que realitza. Avui dia, entenem per *llenguatge orientat a l'objecte* el que ens proporciona les prestacions següents:

- **Objectes**: empaqueten les dades i la funcionalitat conjuntament. Són la base per a l'estructura i la modularitat en un programari orientat a l'objecte.

- **Abstracció**: l'habilitat d'un programa per a ignorar certs aspectes de la informació que manipula. Cada objecte del sistema és un model d'un "actor" que pot treballar en el sistema, informar o canviar d'estat i comunicar-se amb els altres objectes sense revelar com s'han implementat aquestes prestacions.

ANOTACIONS

- **Encapsulació**: assegura que els usuaris d'un objecte no en poden canviar la informació o l'estat de formes permeses. Només els mateixos mètodes que ofereix l'objecte n'han de poder canviar la informació. Cada objecte ofereix una interfície que especifica com la resta d'objectes hi han de treballar. L'objectiu d'aquesta encapsulació és mantenir la integritat de l'objecte.

- **Polimorfisme**: diferents objectes poden tenir la mateixa interfície per a respondre al mateix tipus de missatge i fer-ho apropiadament segons la seva naturalesa.

- **Herència**: organitza i facilita el polimorfisme de manera que permet als objectes definir-se com a especialitzacions d'uns altres, que poden compartir i estendre la seva funcionalitat sense haver de reimplementar-lo de nou. Això se sol fer agrupant els objectes en classes i definint les altres classes com a extensions d'aquestes, creant arbres de classes.

També es pot dir que la definició de l'orientació a l'objecte prové de l'"objecte gramatical" d'una acció. Vegem alguns exemples de com una funcionalitat s'expressa de manera diferent en programació estructurada (orientada al subjecte) o en programació orientada a l'objecte:

- **Orientada al subjecte**: l'aplicació de vendes guarda la transacció.

- **Orientada a l'objecte**: la transacció se salva a partir d'un missatge de l'aplicació de vendes.

- **Orientada al subjecte**: l'aplicació de vendes imprimeix la factura.

- **Orientada a l'objecte**: la factura s'imprimeix a partir d'un missatge de l'aplicació de vendes.

Com veiem, associem l'acció amb l'objecte que afecta, no amb qui la inicia o la demana.

ANOTACIONS

2.3.2. Història

El concepte d'objecte en programació va aparèixer el 1967, quan es va presentar el llenguatge Simula 67 dissenyat per a programar simulacions. La idea d'agrupar dades i funcionalitat en una única estructura es va originar per a respondre a la complexitat de representar totes les combinacions de diferents atributs que els elements participants en una simulació podien presentar. Més endavant, va arribar SmallTalk com un refinament del mateix concepte desenvolupat en el XeroxPARC, que permetia crear i usar objectes en temps d'execució.

Però la popularització del paradigma va tenir lloc durant la dècada dels vuitanta amb l'arribada de C++, una extensió del llenguatge C. I la seva consolidació va arribar amb l'èxit de les interfícies d'usuari gràfiques, a les quals aquest paradigma s'ajusta perfectament.

Des de llavors, s'han afegit característiques de l'orientació a l'objecte a llenguatges de programació ja existents com Ada, BASIC, Lisp, Perl, etc., cosa que ha comportat problemes d'incompatibilitat i dificultats en el manteniment de codi. Al contrari, als llenguatges orientats a l'objecte "purs", els mancaven prestacions a les quals els programadors s'havien acostumat.

En les darreres dècades, el llenguatge Java ha tingut un gran èxit no només per la seva implementació molt sòlida del paradigma, sinó també per la seva semblança amb C i C++ i per l'ús d'una màquina virtual que permetia executar el codi en plataformes múltiples. La iniciativa .NET de Microsoft és semblant en aquest aspecte, ja que suporta també diversos llenguatges de programació, tant de nova creació com variants de llenguatges ja existents.

Més recentment, han aparegut llenguatges que són principalment orientats a l'objecte però compatibles amb metodologies estructurals com ara PHP, Python i Ruby.

De la mateixa manera que la programació procedimental va evolucionar cap a la programació estructurada, l'orientació a l'objecte ha anat incorporant millores com els patrons de disseny i els llenguatges de modelatge com UML.

ANOTACIONS

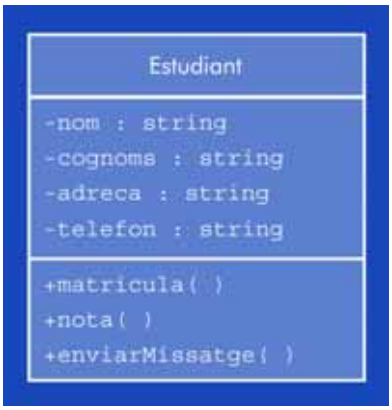
2.3.3. Classes i objectes

Com hem comentat en apartats anteriors, tot objecte és instància d'una classe, entenent aquesta com l'abstracció d'un tipus de dades dins l'espai del problema que volem resoldre.

Com que una classe descriu un conjunt d'objectes que tenen característiques idèntiques (elements de dades o atributs) i comportaments idèntics (funcionalitats o mètodes), una classe és un tipus de dades amb caràcter general. Cada objecte de la classe dins el nostre programa tindrà uns valors concrets per a les seves dades (i per això anomenem ***instància*** l'objecte de la classe). Una vegada definida la classe que representa un element del nostre problema, podrem crear tants objectes d'aquesta com vulguem i manipular-los perquè es comportin com ho han de fer en el problema que resolem.

Cada objecte satisfarà o acceptarà un cert tipus de peticions, tant segons la seva comesa com segons la responsabilitat en el problema. Les peticions que podem fer a un objecte es defineixen en la seva "interfície". Els atributs propis de l'objecte juntament amb el codi que satisfà aquestes peticions és la seva "implementació".

Figura 1. Representació gràfica d'una classe



El nom de la classe és "Estudiant", els atributs de la qual són el seu "nom", "cognoms", "adreça" i "telèfon" i les peticions que li podem fer són "matrícula", "nota" i "enviarMissatge".

Nota

Aquesta figura segueix la notació definida en el llenguatge unificat de modelatge (UML), en què la classe es representa amb una caixa dividida en tres zones. El nom es representa en la part superior; les dades o atributs que volem mostrar, en la part central, i les peticions o mètodes públics, en la part inferior.

A l'hora de desenvolupar o entendre un problema que solucionarem amb orientació a l'objecte, sol ser útil pensar en els objectes en termes de "proveïdors de serveis". El nostre programa prestarà serveis als usuaris, i ho farà utilitzant serveis que ofereixin els objectes. L'objectiu és crear (o reutilitzar) els objectes que proporcionin els serveis més ajustats a les necessitats del problema.

Continuant amb l'exemple de l'estudiant, si volem que el nostre programa imprimeixi el seu expedient imaginarem un objecte "estudiant" semblant a l'anterior, un objecte que calcularà el seu expedient i un objecte que crearà l'informe amb el format adequat per a imprimir-lo (potser mitjançant un altre objecte).

Si pensem d'aquesta manera, inconscientment estem creant un sistema ben cohesionat. Això és fonamental en la qualitat del disseny del programari, significa que els objectes (i les seves interfícies) encaixen bé els uns amb els altres. En un bon disseny orientat a l'objecte, cada un fa una cosa bé i no intenta fer-ne massa. En l'exemple, probablement descobriríem que l'objecte que ens proveirà el servei d'impressió, l'haurem de dividir per a poder proveir serveis d'impressió genèrics, comunicar amb models específics d'impressores, etc. També aquest exercici és útil per a identificar objectes susceptibles de ser reutilitzats o que els trobem ja desenvolupats en catàlegs públics d'objectes.

2.3.4. Encapsulació

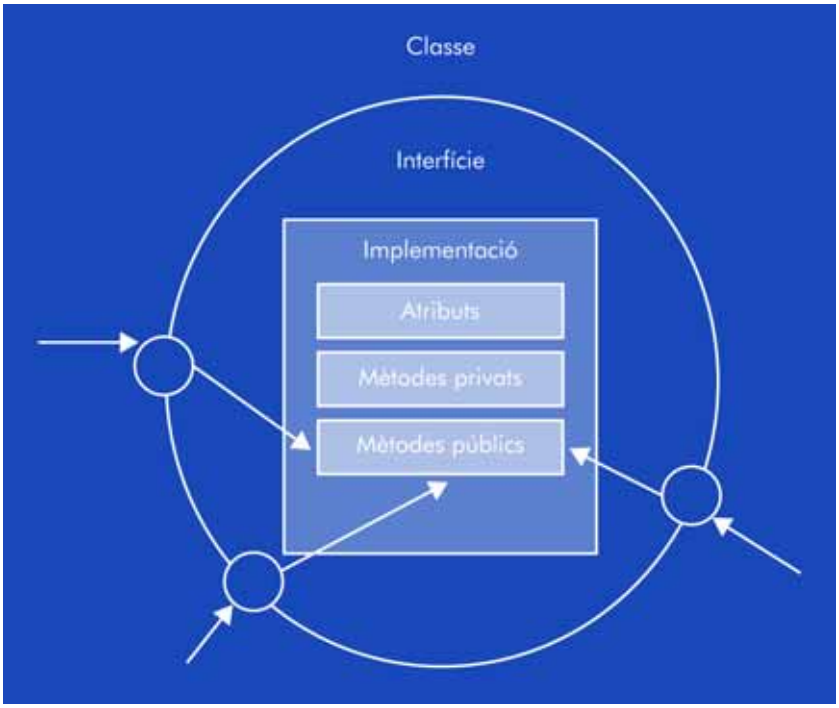
Quan dissenyem els objectes de la nostra aplicació, necessitem tenir en compte els tipus de consumidors d'aquests. D'una banda, hi ha els programadors que poden crear classes (creadors de classes), i de l'altra, els que només les usaran en la mateixa aplicació o en d'altres (consumidors de classes).

Els programadors consumidors de classes buscaran disposar d'unes eines en forma d'objectes per a desenvolupar funcionalitats o aplicacions noves ràpidament a partir de les nostres classes, mentre que l'objectiu dels creadors de classes serà construir-les de manera que la seva interfície només sigui visible per als consumidors de classes i que per a la resta estigui oculta.

ANOTACIONS

El motiu és que construint les classes d'aquesta manera, el creador de la classe en podrà canviar la implementació sense preocupar-se de l'impacte que aquests canvis tenen en els consumidors de la classe, amb la qual cosa es redueixen els possibles errors d'integració. Evitem que els programadors consumidors de les nostres classes toquin parts d'aquestes que són necessàries per al funcionament intern però que no formen part de la interfície de la classe. Facilitem, a més, la seva tasca, ja que només s'hauran de concentrar en allò que usaran sense haver de preocupar-se de com està fet.

Figura 2. Representació gràfica dels elements que componen una classe



L'encapsulació és la propietat de les classes i els mecanismes que tindrem en cada llenguatge de programació que ens permet definir quins mètodes o atributs són públics, privats o tenen algun altre tipus de mecanisme de protecció. Separant la interfície pública de la implementació interna (privada), facilitarem també les tasques de reescriptura de processos interns de la classe (per a millorar-ne el rendiment, per exemple).

ANOTACIONS

2.3.5. Reusar la implementació. Composició

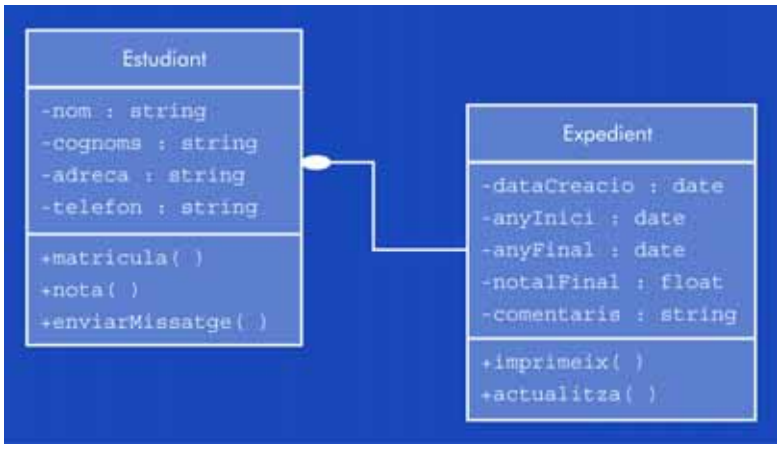
Una vegada que tenim una classe desenvolupada i provada, hauria de representar un element de codi útil en si mateix. No sempre és així, i es necessita experiència i intuïció per a produir classes fàcilment reutilitzables. La manera més simple d'utilitzar una classe és instanciar un objecte d'aquesta, però també podem posar aquesta classe dins d'una altra, amb la qual cosa creem el que s'anomena un ***objecte membre*** de la classe nova. Aquesta classe nova pot estar formada per tants objectes com vulguem i en qualsevol combinació destinada a solucionar el problema que afrontem.

Com que componem una classe nova a partir de les altres ja existents, aquest concepte es diu ***composició*** (si la composició és dinàmica, se sol dir ***agregació***). La composició se sol referir a relacions del tipus "té-un", com per exemple "un alumne té un expedient".

Nota

Aquesta figura segueix la notació definida en el llenguatge unificat de modelatge (UML), en què la composició es representa amb el diamant sòlid. L'agregació es representaria amb la silueta del diamant.

Figura 3. Representació gràfica d'una composició



Típicament, l'objecte membre de la classe nova és d'accés privat. Això permetrà al programador d'aquesta canviar-los per d'altres si és necessari sense canviar altres parts del codi, o instanciar-los en temps d'execució per a canviar el comportament de la classe.

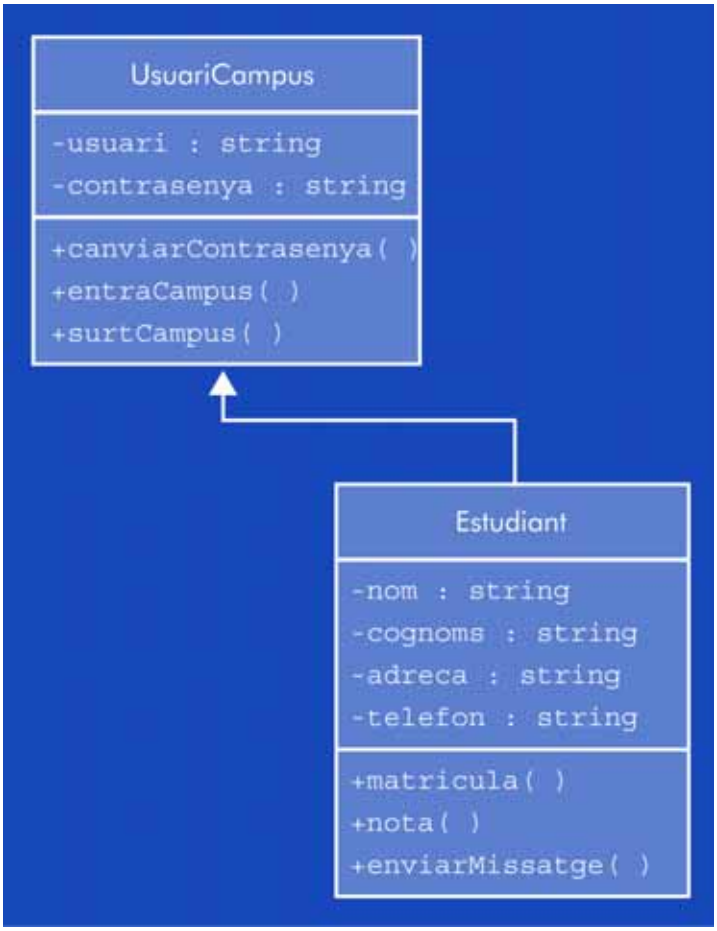
2.3.6. Reusar la interfície. Herència

Una de les característiques més conegudes de l'orientació a l'objecte és la possibilitat d'escollir una classe existent, crear una còpia i des-

ANOTACIONS

prés afegir o modificar les seves prestacions a la còpia creada. Això és el que s'entén per *herència* en aquest paradigma, amb la característica addicional que si la classe original (denominada *classe base*, *superclasse* o *classe pare*) canvia, també canviarà la seva còpia (denominada *classe derivada, heretada, subclasse* o *classe filla*).

Figura 4. Representació gràfica de l'herència



Nota

La fletxa d'aquest diagrama UML apunta de la classe derivada a la superclasse.

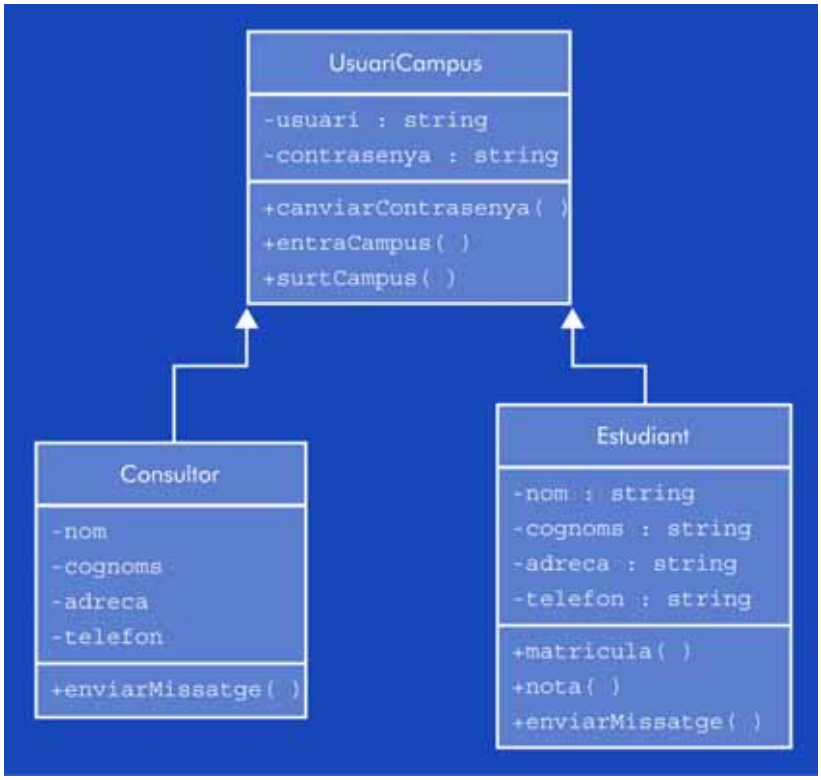
Quan heretem d'una classe base, creem un tipus nou. Aquest tipus nou no solament té tots els atributs i mètodes de la classe pare –malgrat que els que eren ocults continuaran essent-ho–, sinó que en duplica completament la interfície. És a dir, tots els missatges que podíem enviar a un "UsuariCampus", els podem enviar a un "Estudiant" i, com que la classe és determinada per la interfície, deduïm que un "Estudiant" és també un "UsuariCampus".

ANOTACIONS

Ara bé, la classe "Estudiant" s'ha diferenciat de la classe "UsuariCampus", ha afegit propietats i mètodes nous que "UsuariCampus" no tenia. En altres paraules, hem "estès" la classe "UsuariCampus".

En disseny orientat a l'objecte, sempre hem d'estar disposats a reestructurar les nostres classes, i precisament gràcies a les propietats d'encapsulació i abstracció, aquest redisseny no comportarà massa canvis en el codi. Vegem la situació següent:

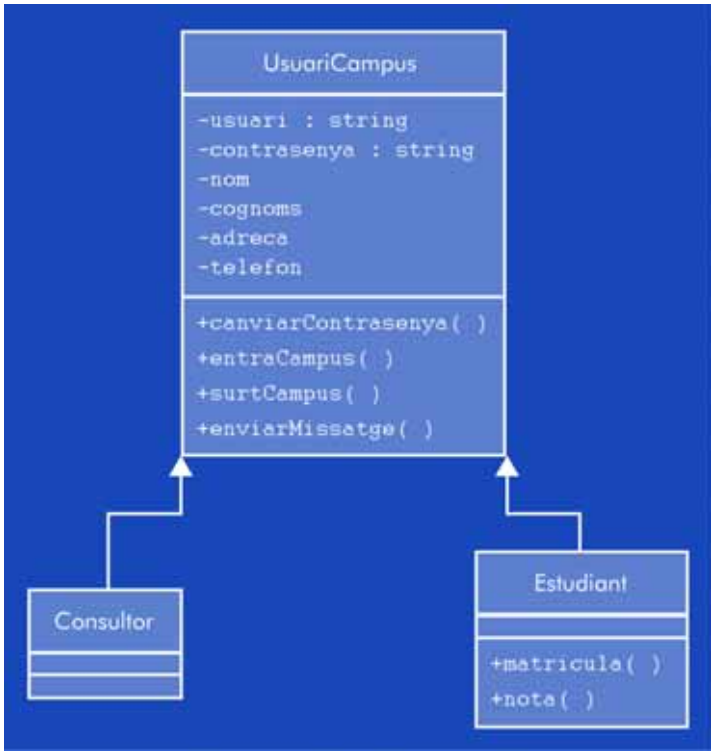
Figura 5. Representació gràfica de dues classes heretades



A primera vista, veiem que totes les classes que hereten d'"UsuariCampus" comparteixen alguns atributs i mètodes; a més, conceptualment aquests atributs també tenen sentit per a un "UsuariCampus". Així doncs, el més natural seria situar aquests atributs i mètodes en la superclasse.

ANOTACIONS

Figura 6. Representació gràfica de dues classes heretades després de reestructurar-les



Ara ens podríem plantejar si té sentit mantenir la classe "Consultor" o bé si ja no és necessària. I aquí és on apareix l'altra manera de diferenciar la superclasse. Per a un "Consultor", els mètodes `entraCampus`() i `surtCampus`() realitzaran operacions diferents que per a un "Estudiant" (el seu accés quedarà registrat en estadístiques diferents, etc.), per la qual cosa, en lloc d'afegir funcionalitat d'"UsuariCampus", el que fa "Consultor" és modificar la funcionalitat existent (sobreescriu una funcionalitat).

Pel que hem vist fins ara, l'herència ens permet crear relacions del tipus "és-un" i del tipus "és-com" segons si sobreescrivim o estenem funcionalitats respectivament.

2.3.7. Polimorfisme

Quan treballem amb jerarquies de classes d'aquest tipus, ens adonarem que de vegades voldrem treballar amb la classe derivada com si fos la superclasse. Això ens permetrà escriure codi que parli amb "UsuariCampus" i afecti "Estudiant" o Consultor" sense haver d'escriure codi per a cada un.

ANOTACIONS

Si més endavant estenem encara més la classe "UsuariCampus", tampoc no haurem de canviar el nostre codi. Això provoca el que s'anomena en termes de compilació el ***late binding***. El compilador no sap quina classe cridem en temps d'execució, només pot comprovar que els paràmetres de la crida són correctes, però no sap exactament quin codi executarà.

Ja hem vist que la implementació del mètode `entraCampus()` serà diferent per a "Consultor" i per a "Estudiant", però l'objecte que cridi aquest mètode ho farà sobre un objecte de la classe "UsuariCampus" (ja que tant "Consultor" com "Estudiant" ho són).

Suposem que tenim un codi que treballa amb "UsuariCampus", d'aquesta manera:

```
void treballaUsuaris(UsuariCampus uc) {
   uc.entraCampus();
   ...
   ...
   uc.canviaContrasenya();
   uc.surtCampus();
}
```

Aquest codi parla amb qualsevol "UsuariCampus" i, per tant, és independent del tipus específic d'usuari. Si en una altra part del programa usem aquest mètode,

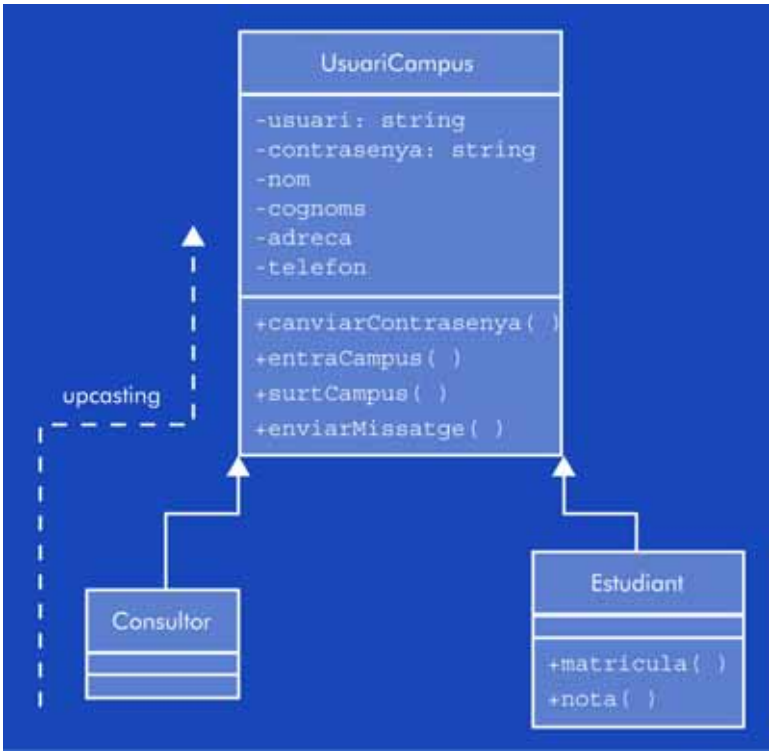
```
Consultor co = new Consultor();
Estudiant es = new Estudiant();
treballaUsuaris(co);
treballaUsuaris(es);
```

aquestes crides són perfectament lògiques. El que ocorre és que un "Consultor" es passa com a paràmetre a un mètode que espera un "UsuariCampus", però com que un "Consultor" és un "UsuariCampus", pot ser tractat com a tal per `treballaUsuaris()`.

El procés de tractar una classe derivada com la classe base se sol dir *upcasting*, perquè fa un *cast* (canvia el tipus d'una variable o objecte) cap amunt en la jerarquia d'objectes:

ANOTACIONS

Figura 7. Camí que segueix la conversió de classes en un upcasting



De la mateixa manera, veiem que el codi de la funció treballaUsuaris() no preveu tots els tipus d'"UsuariCampus", sinó que crida directament el mètode i internament succeeix el que és correcte. Tant si "Consultor" o "Estudiant" han sobreescrit aquest mètode en particular com si no, es crida el mètode apropiat. Això és polimorfisme en el sentit que les classes poden actuar tant com les seves derivades o com les seves classes base segons les circumstàncies en què es trobin o els missatges que rebin.

2.3.8. Superclasses abstractes i interfícies

En algunes ocasions, ens pot interessar que la superclasse representi únicament una interfície per a les seves classes derivades. És a dir, no volem que ningú instanciï un objecte de la superclasse, només necessitarem que facin *upcasting* per a usar-ne la interfície. Això és el que s'anomena *definir una classe "abstracta"*. De la mateixa manera, un mètode definit en una superclasse pot ser abstracte en el sentit que aquesta superclasse no l'implementi i l'haurem de sobreescriure afegint codi en cada classe derivada.

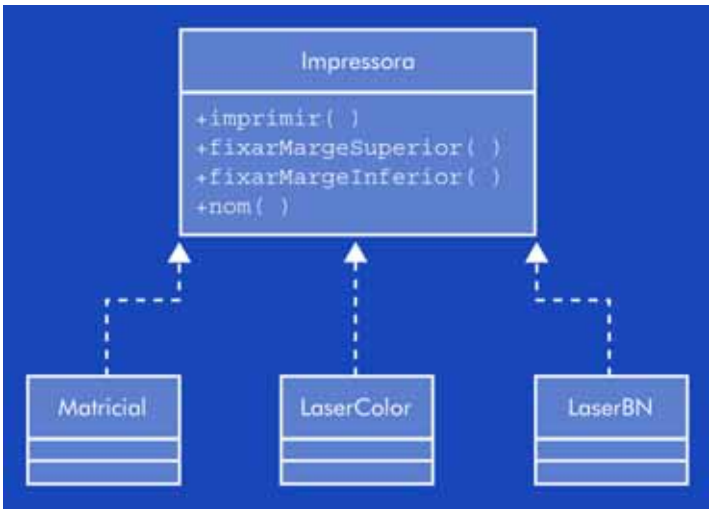
ANOTACIONS

Si una classe té mètodes abstractes, ja és una classe abstracta. Si tots els mètodes són abstractes, llavors haurem separat totalment la implementació de la interfície i tindrem una superclasse "interfície". Pensem, per exemple, en una classe que representi les impressores del nostre sistema. Aquesta classe tindrà mètodes per a imprimir, modificar els marges, etc., però no tindrà implementació. Per a cada impressora del nostre sistema haurem de crear una classe filla que implementi aquests mètodes en cada model d'impressora.

Nota

La fletxa discontínua d'aquest diagrama UML apunta des de la classe que implementa la interfície cap a la classe definida com a interfície.

Figura 8. Representació gràfica de la implementació d'una interfície



2.3.9. L'orientació a l'objecte i la notació UML

Paral·lelament als avenços en paradigmes de programació, també han evolucionat els mètodes i notacions per a modelar i dissenyar els sistemes abans d'implementar-los. L'orientació a l'objecte no és una excepció i la manera de presentar la solució al problema plantejat en forma d'objectes que representen les entitats involucrades és molt susceptible de ser modelada visualment, és a dir, mitjançant diagrames fàcilment comprensibles.

El Dr. James Rumbaugh va ser un dels pioners en tècniques de modelatge de classes, jerarquies i models funcionals, que segons ell "capturaven les parts essencials d'un sistema".

ANOTACIONS

El modelatge visual d'un sistema ens permetrà el següent:

1) Identificar i capturar els processos de negoci.

2) Disposar d'una eina de comunicació entre els analistes de l'aplicació i els coneixedors de les regles de negoci.

3) Expressar la complexitat d'un sistema de manera comprensible.

4) Definir l'arquitectura del programari i els components implicats (interfície d'usuari, servidor de bases de dades, lògica de negoci) independentment del llenguatge d'implementació que usem.

5) Promoure la reutilització en identificar més fàcilment els sistemes implicats i els components.

Així doncs, a partir de tècniques desenvolupades per James Rumbaugh en modelatge d'objectes, per Ivar Jacobson en casos d'ús, per Grady Booch en la seva metodologia de descripció d'objectes i amb la participació d'empreses com HP, IBM, Oracle i Microsoft entre d'altres, es va crear la notació UML amb el patrocini de l'empresa Rational (recentment adquirida per IBM), que va ser aprovada per l'OMG (Object Management Group) el 1997.

La sigla *UML* és l'abreviatura de *Llenguatge Unificat de Modelatge* (en anglès, *Unified Modeling Language*), que combina en una única notació els models següents:

- Modelatge de dades (semblant a un diagrama entitat-relació)
- Modelatge de regles de negoci i fluxos de treball
- Modelatge d'objectes
- Modelatge de components d'un sistema

En pocs anys, s'ha convertit en l'estàndard per a visualitzar, especificar, construir i documentar els elements que intervenen en un sistema de programari de qualsevol mida. Es pot usar en qualsevol procés

ANOTACIONS

durant tot el cicle de vida del projecte i independentment de la implementació.

Cal tenir en compte que UML no és una metodologia, sinó que és simplement una notació per a modelar el nostre sistema. Per això, tampoc no està pensat per a descriure la documentació d'usuari, ni tan sols la seva interfície gràfica. Així doncs, en començar un projecte de programari, haurem d'escollir primer la metodologia amb la qual treballarem per a després usar UML durant el cicle de vida que ens marqui la metodologia escollida. UML tampoc no en recomana cap en concret, encara que potser el mètode iteratiu vist en el capítol anterior és el més usat i el que s'adapta millor als projectes dissenyats amb el paradigma d'orientació a l'objecte i modelats en UML.

UML ha estat i continuarà essent l'estàndard de modelatge orientat a l'objecte dels pròxims anys, tant per l'aprovació dels experts en metodologies i enginyeria del programari com per la participació de les grans empreses de programari, l'acceptació de l'OMG com a notació estàndard i la quantitat d'eines de modelatge que el suporten.

2.3.10. Introducció a UML

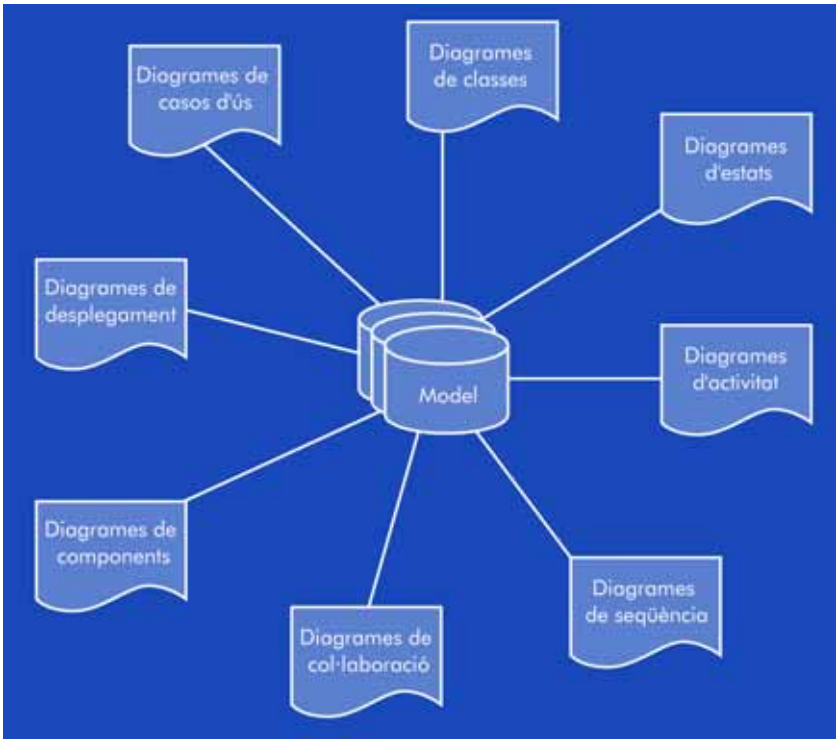
Un model és una abstracció d'un sistema o d'un problema que cal resoldre considerant un cert propòsit o un punt de vista determinat. El model ha de descriure completament els aspectes del sistema que són rellevants per al seu propòsit i amb un nivell de detall determinat.

El codi font és també una expressió del model, la més detallada i la que n'implementa a més la funcionalitat, però no és còmode com a eina de comunicació. A més, per a arribar-hi convé desenvolupar abans altres representacions.

Un diagrama ens permetrà representar gràficament un conjunt d'elements del model, de vegades com un graf amb vèrtexs connectats i altres vegades com a seqüències de figures connectades que representin un flux de treball.

ANOTACIONS

Figura 9. Diagrames UML de representació dels models d'un sistema



Cada punt de vista del sistema –i cada nivell de detall– es podrà modelar i aquest model es podrà representar gràficament. El que UML proposa és una notació i un conjunt de diagrames que inclouen les perspectives més rellevants del sistema:

- Diagrama de casos d'ús
- Diagrama de classes
- Diagrames de comportament
  - Diagrama d'estats
  - Diagrama d'activitat
  - Diagrames d'interacció
    - Diagrama de seqüència
    - Diagrama de col·laboració
- Diagrames d'implementació
  - Diagrama de components
  - Diagrama de desplegament

Aquests diagrames responen a les vistes d'un sistema de programari. Des de la definició del problema (casos d'ús), la vista lògica (classes,

ANOTACIONS

objectes), la vista de processos (comportament) i la vista d'implementació i distribució.

La darrera especificació d'UML disponible és la versió 2.0, que va ser adoptada a l'octubre de 2003– i que incorpora conceptes com la descripció d'infraestructures, l'intercanvi de diagrames i l'expressió de restriccions en objectes (*constraints*).

2.3.11. Conclusions

En aquest apartat, hem fet un repàs ràpid dels conceptes relacionats amb el paradigma de l'orientació a l'objecte. Hem intentat descriure'n les característiques més importants i proporcionar exemples il·lustratius, alhora que hem introduït alguns elements de la notació UML que veurem detalladament en apartats posteriors.

També hem introduït la notació UML, la seva història i les seves característiques principals.

Lectura recomendada

Per a aprofundir aquest paradigma o adquirir experiència en programació orientada a l'objecte, us recomanem que consulteu els textos següents:

B. Eckel (2003). Thinking in Java (3.ª ed.). Upper Saddle River: Prentice Hall. http://www.mindview.net/Books/TIJ/

Java Technology. http://java.sun.com

E. Gamma; R. Helm i altres (1995). Design Patterns. Reading, Mass: Addison Wesley.

Object Management Group. http://www.omg.org/

2.4. Definició del cas pràctic

Per a entendre una notació de modelatge com UML, podríem llegir simplement l'especificació que publica l'OMG i en tindríem la visió més completa possible. Però tractant-se d'un estàndard que indica com hem de modelar un sistema, és molt més còmode i fàcil entendre'l per mitjà d'un cas pràctic en què puguem aplicar cada concepte a un cas concret i a la visió que en tenen els seus participants, i reflectir en un diagrama cada una d'aquestes visions i casos.

El cas pràctic que plantegem és el següent:

Una empresa de desenvolupament de programari té problemes per a gestionar les seves operacions amb els clients, tant si es tracta de clients futurs als quals ha presentat una proposta (o està pendent de fer-ho) com si es tracta de clients que han contractat un projecte i cal fer-ne el seguiment, no solament en termes de tasques pendents, hores inver-

ANOTACIONS

tides, etc., sinó també en termes de facturació, imports pendents de pagament, etc.

Per a això, es podria decidir per implantar un ERP basat en programari lliure i parametritzar-lo segons les seves necessitats, però després d'avaluar algunes de les alternatives, va optar per desenvolupar un sistema complet de gestió de projectes i clients a mida del seu mètode de treball. A més, si el sistema és prou flexible, el podrà alliberar sota alguna llicència aprovada per l'OSI i obtenir beneficis prestant-ne suport, formació i actualitzacions.

És obvi que no descriurem aquí el projecte, ja que aquesta és la tasca que realitzarem mitjançant UML en els apartats posteriors, sinó que simplement enumerarem algunes de les seves característiques més representatives per a tenir conceptes i casos amb què començar a treballar.

El sistema de gestió haurà de permetre:

1) Disposar d'una llista completa de contactes de l'empresa, tant si són clients o possibles clients com proveïdors.

2) Disposar d'una llista completa de projectes en què treballa l'empresa. De cada projecte haurem de disposar d'informació sobre:

- El client a qui fa referència.
- L'estat actual del projecte.
- L'estimació que es va fer en el seu moment (el pressupost que es va presentar al client).
- Les tasques que implica el projecte, el seu estat, durada estimada i real.
- Els costos que ha tingut el desenvolupament del projecte.

ANOTACIONS

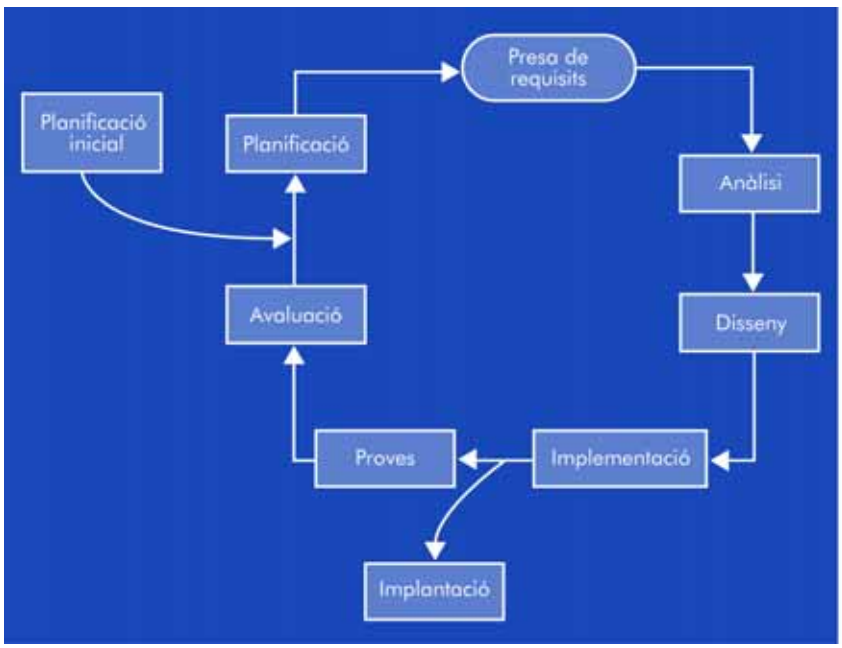
3) Disposar d'una llista de desenvolupadors que puguem associar a cada projecte. Aquests hauran de registrar les hores treballades en cada tasca, els seus avenços, etc.

4) Disposar d'informes en què puguem veure quins projectes s'han endarrerit, quin ha estat el resultat d'un projecte respecte a la seva planificació inicial, etc.

Amb aquestes dades, resta plantejat el cas d'estudi. No entra dins els objectius d'aquest capítol resoldre'l, per la qual cosa l'especificació no és tan completa com ho hauria de ser en realitat, encara que és suficient per a començar a treballar.

Tal com hem comentat en apartats anteriors, és fonamental determinar la metodologia que usarem abans de començar el procés de modelatge. En el nostre cas, com que es tracta d'una empresa de desenvolupament de programari moderna i amb una inclinació clara envers el programari lliure, el nostre mètode de desenvolupament i gestió de projectes s'ajusta perfectament al que caracteritza un cicle de vida de tipus iteratiu.

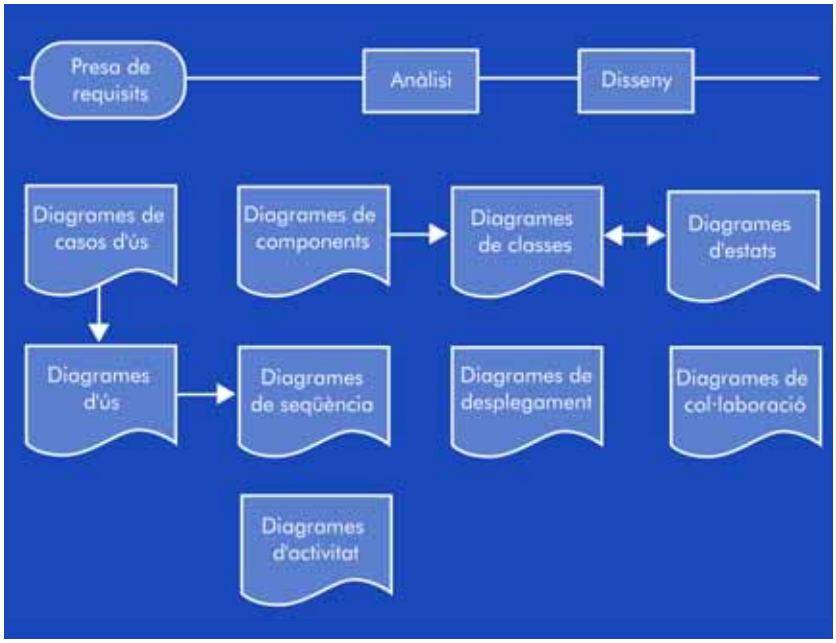
Figura 10. Cicle de vida iteratiu



ANOTACIONS

Encara que ens centrarem en les fases de presa de requisits, anàlisi i disseny, alguns dels diagrames es basaran en models de les fases d'implementació i implantació. No és possible establir una seqüència perfecta entre els diagrames, ni una correspondència total amb les diferents fases del cicle de vida, ja que depenent de les persones involucrades en cada fase i de la seva experiència i coneixement, alguns diagrames poden quedar completats en fases molt primerenques mentre que d'altres estaran sotmesos a revisions contínues. Tenint en compte això, una correspondència possible és la següent:

Figura 11. Correspondència dels diagrames UML amb fases d'un projecte



Encara que la figura mostra alguns dels diagrames units per fletxes, gairebé tots ells tenen relació amb els altres, de manera que un canvi en algun pot afectar molts altres. Només hem indicat les correspondències més evidents, encara que cal esperar que un canvi en un cas d'ús, per exemple, comportarà canvis en molts altres diagrames, o que un canvi en el diagrama de components pot afectar els diagrames de classes i de desplegament.

Com que la lectura del material ens imposa un ordre, seguirem, tant com puguem, el mateix de la figura, de dalt a baix i d'esquerra a dreta, avançant per les fases del projecte.

ANOTACIONS

2.5. Diagrames de casos d'ús

Els casos d'ús són una eina essencial en la presa de requisits del sistema. Ens permeten expressar gràficament les relacions entre els diferents usos que té i els seus participants o actors. El resultat és un conjunt de diagrames molt fàcils d'entendre, tant pel client com pels analistes del projecte.

No defineixen tots els requisits (per exemple, tipus de dades, interfícies externes, rendiment, etc.), però sí que representen el fil conductor que els vincula tots amb els actors del sistema.

Es componen dels elements següents:

- **Actors**: representen els rols que exerceixen els usuaris o altres sistemes en el sistema del problema. Identificar els actors d'un cas d'ús passa per esbrinar qui està involucrat en cada requisit concret, qui es beneficiarà de cada funcionalitat o qui proveirà o usarà la informació.

Figura 12

- **Cas d'ús**: són les accions que poden tenir lloc en el sistema que volem modelar. Per a identificar-les, pot ser útil preguntar-se quines són les tasques i responsabilitats de cada actor, si hi haurà actors que rebran informació del sistema, etc.

Figura 13

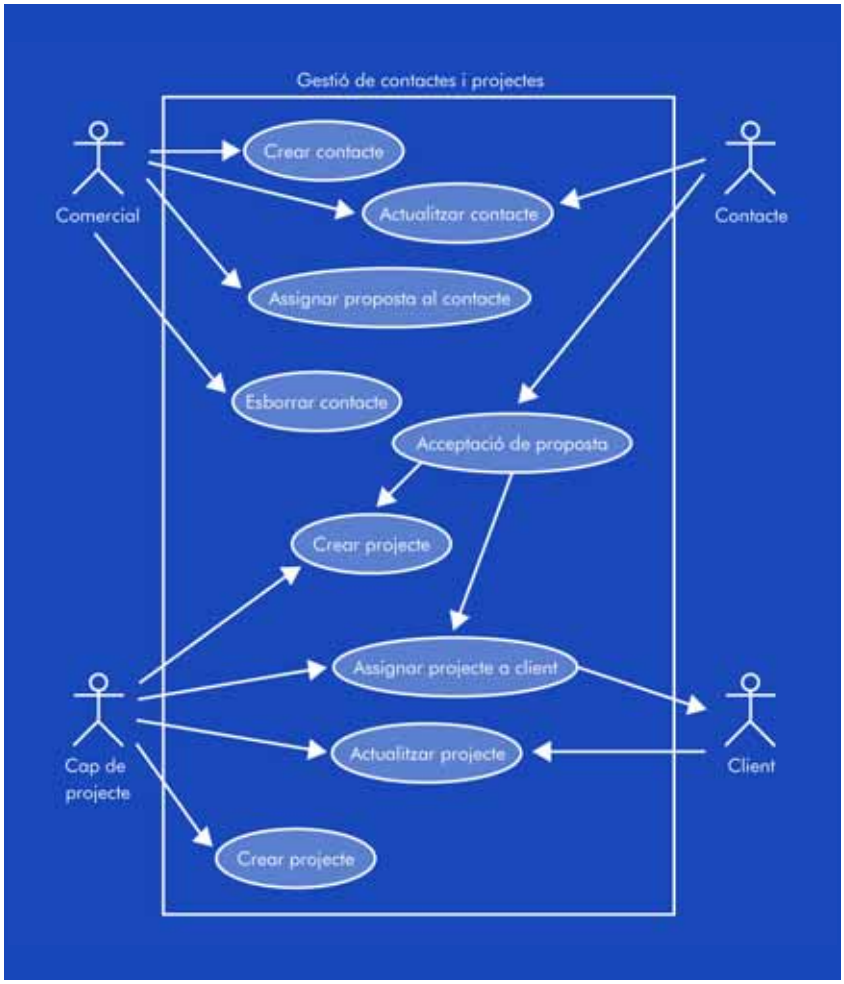
ANOTACIONS

- **Relacions**: indiquen activitat o flux d'informació.

- **Límit del sistema**: defineix l'àmbit en què es produeix el cas d'ús que representem i que serà tractat pel sistema. Els actors no són part del sistema i, per tant, estan fora dels seus límits.

A continuació, vegem una proposta de cas d'ús en l'àmbit de la gestió dels contactes i projectes.

Figura 14. Cas d'ús per a la gestió de contactes i projectes. Realitzat amb Umbrello

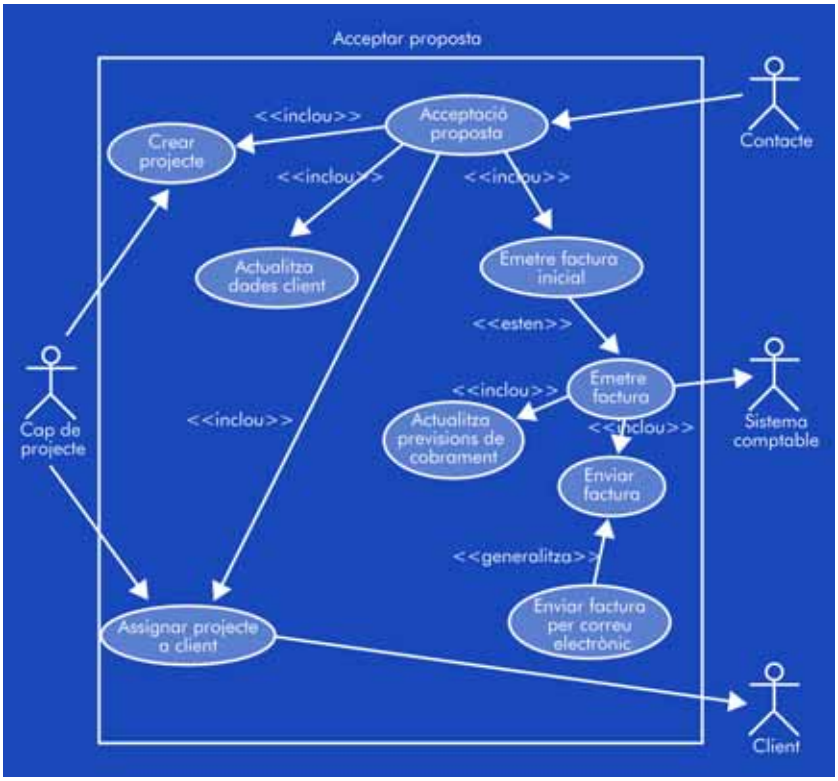


En primer lloc, veiem que modelem en un nivell molt alt, traslladant el que s'ha expressat en els requeriments a una representació gràfica. Les relacions entre els actors i els casos d'ús indiquen si proporcionen o reben informació (segons el sentit de la fletxa).

ANOTACIONS

En canvi, les relacions entre casos d'ús poden tenir significats diferents. Cada cas d'ús és susceptible de ser representat més detalladament en un altre diagrama. Ampliem-ne un per mostrar-ho.

Figura 15. Cas d'ús per a l'acceptació de la proposta



L'etiqueta <<inclou>> entre dos casos d'ús indica que un té la funcionalitat d'un altre com a part integrant seva. En l'exemple, l'acceptació de la proposta inclourà la funcionalitat per a actualitzar les dades del client –ara tenim un client en lloc d'un simple contacte–, emetre la factura inicial, potser –no forçosament– crear el projecte, etc.

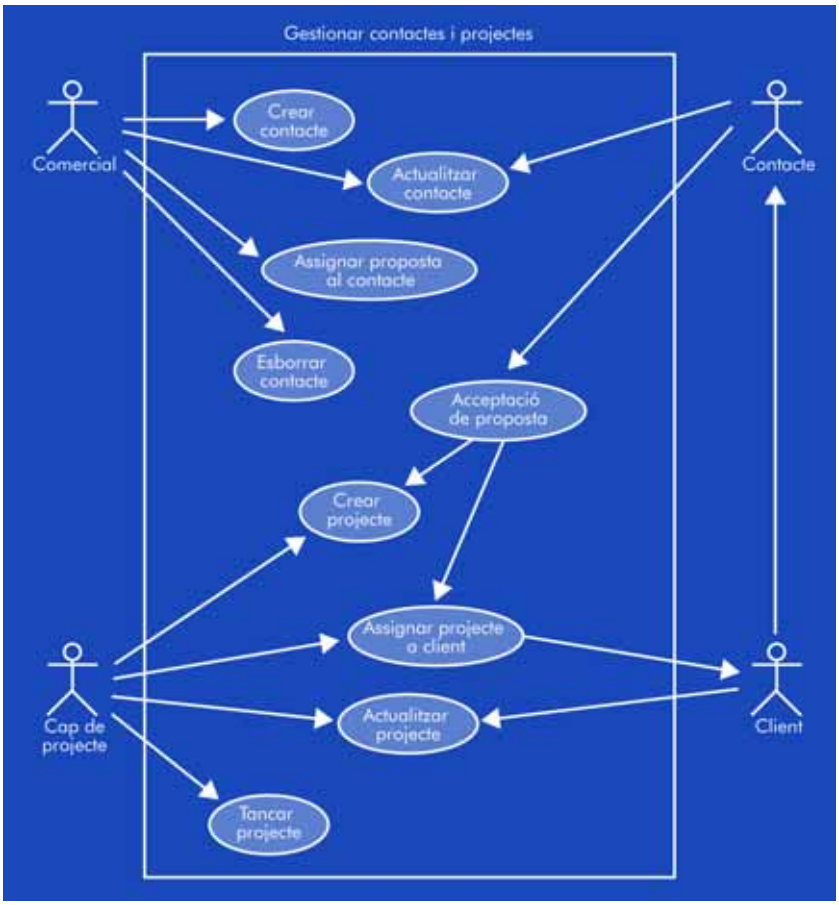
L'etiqueta <<esten>> indicarà que un cas d'ús amplia la funcionalitat d'un altre. En l'exemple, l'emissió de la factura inicial no és l'emissió d'una factura qualsevol, implicarà comprovacions, com per exemple si tenim totes les dades del client, etc., que no es produeixen en l'emissió d'una factura normal. Ara bé, és clar que hi haurà una part de funcionalitat que compartiran. En altres paraules, la creació de la factura inicial és un cas particular de l'emissió d'una factura.

ANOTACIONS

L'etiqueta <<generalitza>> indica també una relació pare-fill semblant a l'extensió, amb la diferència que actuen exactament igual quant a regles de negoci. Dit d'una altra manera, en un moment donat podríem substituir el cas d'ús pare pel fill i el sistema no es veuria afectat. En canvi, en una relació d'extensió això no ocorre, no podem substituir l'emissió de qualsevol factura per la de la factura inicial.

Les relacions de generalització també es poden donar entre els actors. Tornant al diagrama inicial:

Figura 16. Cas d'ús per a la gestió de contactes i projectes, amb generalització en els actors



Veiem que generalitzem un client en un contacte; és completament lògic si considerem que tots els clients van ser contactes en un instant determinat i que han de poder mantenir les seves mateixes funcionalitats.

ANOTACIONS

Atès l'alt nivell en què modelem el sistema en aquesta fase, és important tenir presents algunes bones pràctiques respecte a la interpretació i creació de diagrames de casos d'ús:

- S'han de començar els noms dels casos d'ús amb un verb.
- Encara que no hi ha cap manera d'indicar l'ordre en què un actor aplicarà els casos d'ús, sol ser més intuïtiu representar-los en ordre descendent, situant els més importants en la part superior del diagrama.
- S'han de situar els actors principals en la part superior del diagrama també ajuda a comprendre'l i a generalitzar de manera més comprensible.
- S'han de denominar els actors amb substantius relacionats amb les regles de negoci, d'acord amb els rols que representen, i no pas amb el seu càrrec o posició en el sistema.
- S'ha d'anteposar "Sistema" o <<sistema>> als actors que siguin processos externs.
- Per a representar esdeveniments que ocorren de manera programada, podem introduir un actor de sistema "Temps" per a modelar-ne els casos d'ús.
- L'etiqueta <<inclou>> no és obligatòria. És aconsellable incloure-la només si en un punt específic la lògica del cas d'ús inclòs és necessària.
- No s'ha d'abusar de l'etiqueta <<esten>>, ja que dificulta la comprensió del cas.
- La generalització de classes se sol identificar perquè una sola condició del cas d'ús (en l'exemple, el mètode de tramesa) en canvia completament la lògica interna, però no les regles de negoci del sistema.

ANOTACIONS

- S'ha d'evitar representar més de dos nivells d'associacions de casos d'ús en un mateix diagrama. Si ens trobem en aquesta situació, hi ha moltes probabilitats que fem una representació funcional del que ocorre en el cas d'ús. Ens hem de concentrar només en els requisits d'ús, ja que la descomposició funcional és part de la fase de disseny.
- Situar els casos inclosos a la dreta del cas que els inclou ajuda a comprendre millor el diagrama. De la mateixa manera, és més intuïtiu situar els casos que estenen a sota del cas pare, igual que els casos que hereten o generalitzen.
- És útil intentar expressar amb "és com" la generalització d'actors per a comprovar si els modelem correctament.

2.6. Diagrames de seqüència

Els diagrames de seqüència modelen el flux de la lògica dins el sistema de manera visual, amb la qual cosa permeten documentar-la i validar-la. Es poden usar tant en anàlisi com en disseny i proporcionen una bona base per a identificar el comportament del sistema.

Típicament s'usen per a modelar els escenaris d'ús del sistema, descrivint de quines maneres es pot utilitzar. La seqüència pot expressar tant un cas d'ús complet com, potser, un cas concret d'aquest preveient algunes alternatives.

També són una bona eina per a explorar la lògica d'una operació complexa o els elements implicats en la prestació d'un servei. Ens poden ajudar a identificar colls d'ampolla en la fase de disseny, detectar quines seran les classes més complexes d'implementar i decidir quines d'aquestes necessitaran diagrames d'estats –que veurem més endavant– per a facilitar-ne la implementació.

Es componen dels elements següents:

- **Objecte**: instància d'una classe que podem començar a identificar com a participant en la seqüència d'operacions que representa aquest cas d'ús.

ANOTACIONS

Figura 17

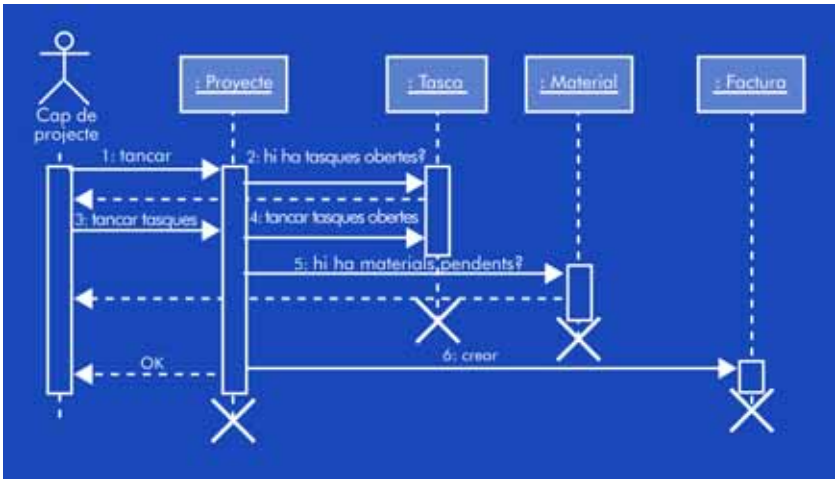
- **Actor**: els actors es poden comunicar amb els objectes i, per tant, formaran part d'aquest diagrama.
- **Vida de l'objecte**: indiquem l'existència d'un objecte durant el temps amb una línia discontínua, el final de la qual s'indica amb una aspa.
- **Activació**: indiquem quan l'objecte realitza una tasca concreta.

Figura 18

- **Missatge**: la comunicació entre objectes i les seves activacions.

A continuació, vegem un exemple de diagrama de seqüència per al cas d'ús implicat en el tancament d'un projecte:

Figura 19. Diagrama de seqüència per al cas d'ús "Tancar un projecte". Realitzat amb Dia



ANOTACIONS

Una de les característiques dels diagrames de seqüència és que gairebé no necessiten aclariments quant a la seva notació. En la part superior, veiem els participants en la seqüència (també denominats *classificadors que la implementen*). En general, sempre serà un actor qui inicii la seqüència, i la resta de participants poden ser objectes (representats amb una caixa en què escriurem el nom de l'objecte), altres actors o, potser, un cas d'ús complet.

Cada participant té un interval en què aquest està actiu en la seqüència, la qual cosa s'indica amb un rectangle sobre la línia discontínua que representa la vida de l'objecte. El rectangle comença quan l'objecte rep un missatge (representat per una fletxa que incorpora el nom de la crida) i acaba quan aquest torna la seva darrera resposta (representada per una fletxa discontínua).

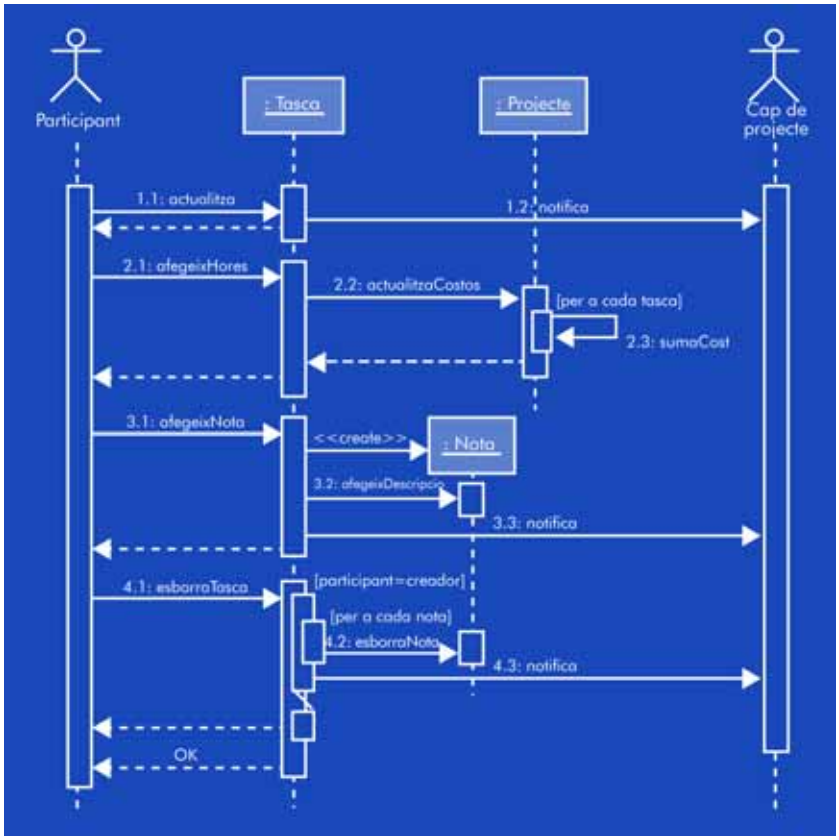
Els missatges solen incloure números de seqüència que faciliten la comprensió del diagrama i el seguiment de l'ordre en què es produeixen els missatges. De vegades, un error pot provocar una línia discontínua de retorn fins al primer participant, de manera semblant a com es propagaria una excepció. Així doncs, les línies de retorn poden incloure també etiquetes per a indicar si representen un error o no.

Finalment, una aspa al final de la vida de l'objecte indica que es pot destruir.

Segons la notació, és possible expressar iteracions i condicionals en un diagrama de seqüència. Molts experts no ho recomanen, ja que implica incloure lògica dins un diagrama que només hauria de representar missatges entre participants en la seqüència, però de vegades és imprescindible fer-ho per a reflectir correctament la seqüència de missatges. Vegem un altre exemple a partir del cas d'ús relacionat amb l'actualització de les tasques en la gestió de projectes.

ANOTACIONS

Figura 20. Diagrama de seqüència corresponent a la gestió d'un projecte, amb iteracions i condicions

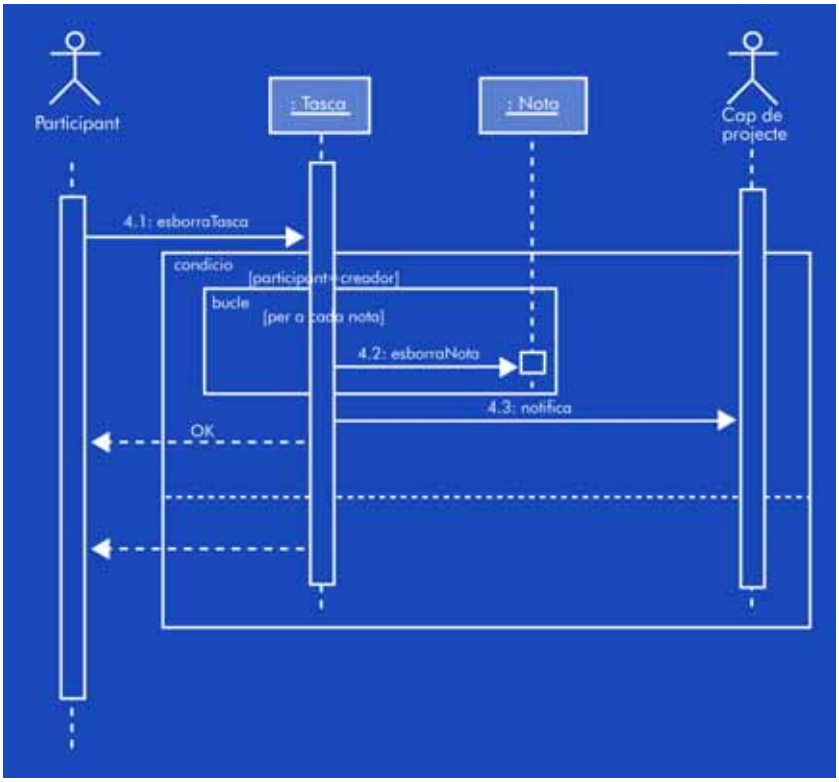


Les guardes de les condicions o dels bucles s'expressen entre claudàtors. Per a indicar que actuem sobre el mateix objecte, pintarem un rectangle desplaçat lleugerament a la dreta. En general, els bucles s'identificaran pel text entre claudàtors, que indicarà sobre què s'itera (per exemple "per a cada tasca"), mentre que la guarda de la condició haurà de ser tan explícita com sigui possible.

En cas de necessitar una acció alternativa (participant!=creador en l'exemple), creuarem una línia sobre un altre rectangle desplaçat per indicar l'activitat que tindrà lloc si no es compleix la condició. Es pot apreciar fàcilment que la tendència d'introduir lògica complexa dins els diagrames de seqüència no ajuda a esclarir-lo, més aviat al contrari. La versió 2.0 d'UML introdueix el concepte dels marcs d'interacció, que ajuden a millorar la representació en aquests casos:

ANOTACIONS

Figura 21. Diagrama de seqüència amb iteracions i condicions en notació UML2.0

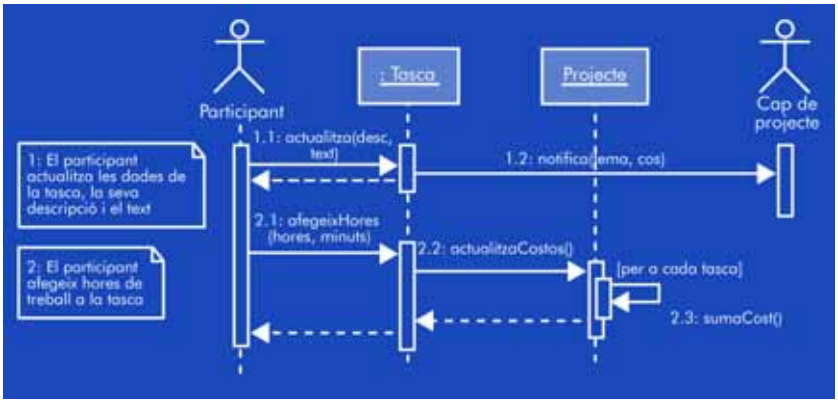


Els marcs d'interacció es representen amb caixes transparents que envolten la condició o bucle. En la cantonada superior esquerra s'escriu el tipus d'interacció (bucle o condició), i en la línia d'activitat de l'objecte, la condició entre claudàtors. En el cas de condicions amb alternativa, aquesta se separa mitjançant una línia de punts.

Quan aquests tipus de diagrames s'usen en fase de disseny, se sol disposar de més informació sobre els objectes i els seus missatges. Si el diagrama es realitza en aquella fase, la seva representació pot ser més completa:

ANOTACIONS

Figura 22. Diagrama de seqüència realitzat en fase de disseny



Els missatges poden incloure la informació que necessitarà l'objecte receptor per a realitzar la seva activitat. També hem inclòs unes notes en cada seqüència per indicar textualment l'activitat que representen.

Finalment, enumerem un conjunt de bones pràctiques en la representació de diagrames de seqüència:

- L'ordre entre els missatges i els participants ha de ser sempre d'esquerra a dreta i de dalt a baix per a facilitar la comprensió del diagrama.

- El nom dels actors ha de ser consistent amb els casos d'ús.

- El nom dels objectes ha de ser consistent amb els diagrames de classes.

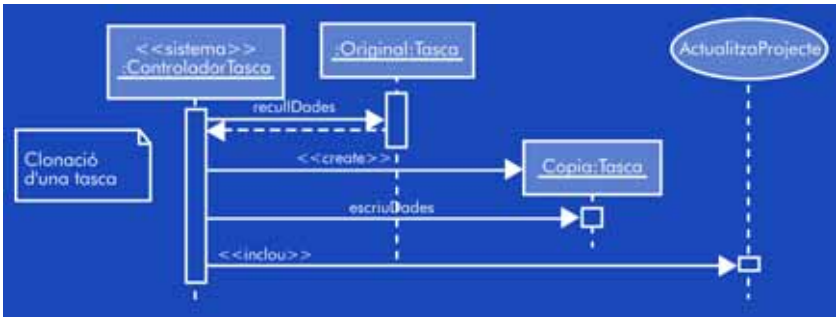
- S'han d'incloure notes en les seqüències.

- S'ha d'incloure l'aspa de destrucció de l'objecte només en casos en què proporcioni informació sobre quan s'ha de destruir; en cas contrari, "embrutarem" el diagrama innecessàriament.

- Si la seqüència treballa amb diversos objectes de la mateixa classe però amb rols diferents, podem etiquetar-los de la manera següent:

ANOTACIONS

Figura 23. Diagrama d'activitat amb rols



Veiem que tenim dues instàncies de l'objecte Tasca, una per a l'original i una altra per a la còpia.

- S'han d'usar els estereotips de manera consistent. En l'exemple anterior, el sistema controlador de tasques és qui inicia la clonació, i ho indiquem mitjançant la paraula <<sistema>>.
- En els paràmetres en missatges, és més convenient usar noms clars que no pas els seus tipus.
- En les crides a casos d'ús, s'ha d'usar l'estereotip <<inclou>>.

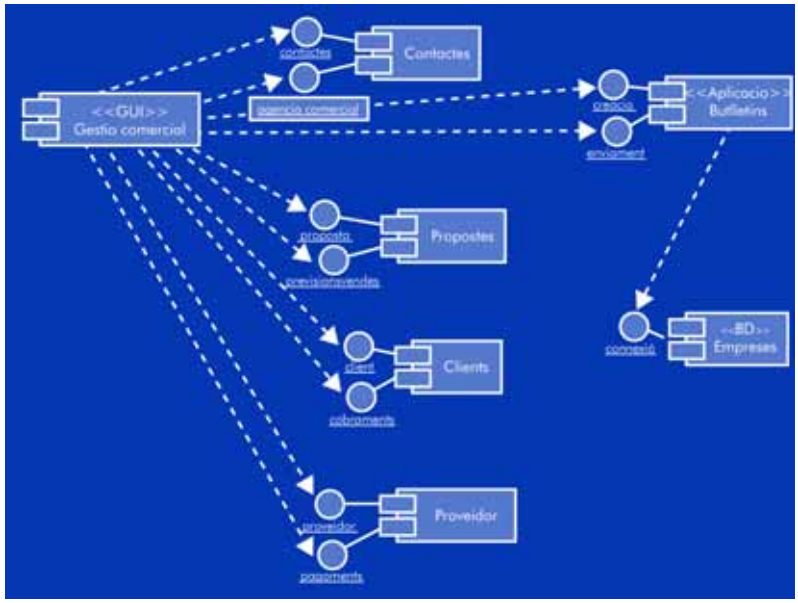
2.7. Diagrames de components

El desenvolupament orientat a l'objecte està molt relacionat amb el desenvolupament basat en components, en el sentit que les classes amb les seves propietats d'encapsulació i abstracció es veuen en moltes ocasions com a components tancats d'un sistema. Així doncs, la notació UML inclou un diagrama de components que segons la definició de l'OMG "mostra les dependències entre components de programari, incloent-hi els classificadors que els especifiquen (per exemple, les classes d'implementació) i els artefactes que els implementen, com els fitxers font, binaris, seqüències, etc.".

En molts casos en què es podrien usar diagrames de components, s'usen els diagrames de desplegament –que veurem més endavant–, ja que aquests ens permeten modelar a més on s'implantarà cada component i amb quina configuració o paràmetres. Així doncs, el diagrama de components ha quedat tradicionalment relegat a modelar l'arquitectura del sistema en un nivell lògic o d'entorn de negoci.

ANOTACIONS

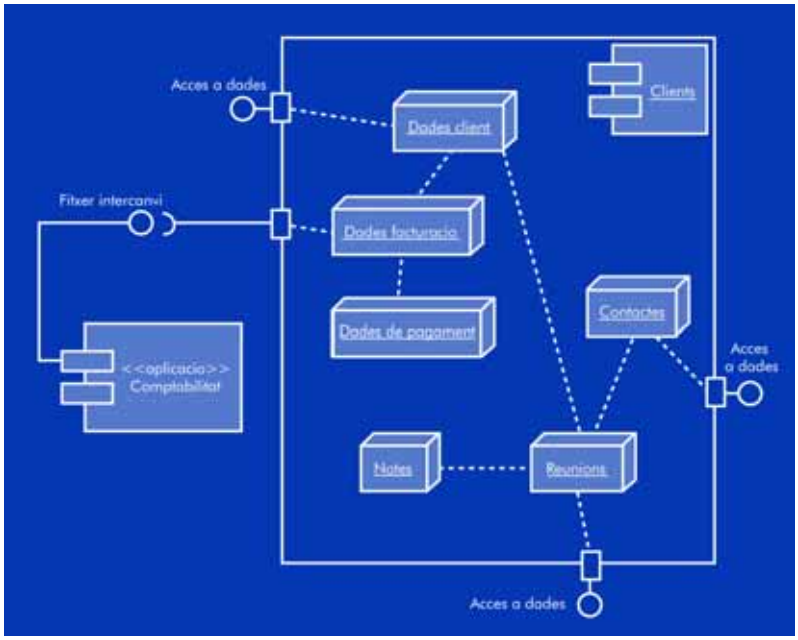
Figura 24. Diagrama de components del sistema



Aquí hem modelat l'estructura dels components de programari que estan involucrats en l'aplicació de gestió comercial. Els components incorporen ports cap a les dades o serveis que proporcionen (en l'exemple, "contacte", "agenda comercial", etc.) o ports per a rebre dades o resultats de peticions.

Cada port pot tenir una o més interfícies. Ampliem un component per a mostrar-ho més detalladament:

Figura 25



ANOTACIONS

Hem desenvolupat el component "Clients" en les seves parts constituents (classes, molt probablement) i n'hem definit les interfícies en un nivell alt. Veiem que bàsicament ofereix interfícies d'accés a les seves dades (d'entrada i sortida) i una interfície d'exportació de dades de facturació per a comunicar-se amb l'aplicació de comptabilitat. Aquí hem utilitzat parts de la notació UML2.0, que diferencia entre interfícies que el component ofereix (cercle tancat) i interfícies que el component necessita (cercle obert).

Els conceptes d'interfície i port deuen ser molt familiars als estudiants que tinguin experiència en desenvolupament basat en components (CORBA, COM, IDL, etc.), però en la majoria dels casos no serà necessari arribar a aquest nivell de detall en el diagrama en aplicacions orientades a l'objecte convencionals.

En uns altres entorns on sigui habitual utilitzar patrons de disseny, els diferents objectes, interfícies i la comunicació entre si es poden representar en els detalls del component.

Finalment, la llista de bones pràctiques en la representació de components és la següent:

- Aplicar els estereotips de manera consistent.
- Encara que usem UML 1.5, la representació de les interfícies dels components amb cercles és molt més clara que una simple fletxa entre si.
- Com que la interfície és un conjunt de mètodes, intentem no representar massa interfícies; és aconsellable agrupar-les sota un mateix nom i crear després un diagrama detallat del component.
- Els components poden heretar uns dels altres. En aquest cas, la fletxa que utilitzem en els casos d'ús per a expressar generalització pot servir perfectament.
- Per a millorar la comprensió d'un diagrama de components, és preferible connectar-los sempre per mitjà d'interfícies. És més consistent i evita confusions o dubtes sobre la seva interpretació.

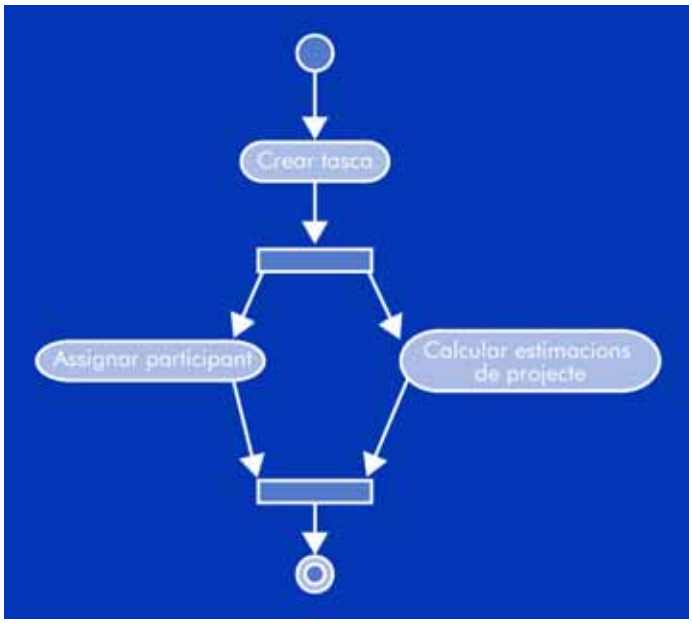
ANOTACIONS

2.8. Diagrama d'activitats

En molts aspectes, els diagrames d'activitats són l'equivalent orientat a l'objecte dels diagrames de flux i els DFD del desenvolupament estructurat. El seu ús és principalment l'exploració i representació de la lògica compresa en operacions complexes, regles de negoci, casos d'ús o processos de programari.

De manera semblant als diagrames de flux tradicionals, tot diagrama d'activitats té un punt de partida i un final. Les activitats representaran cada pas important que es produeix en el procés que estem modelant (pot representar un cas d'ús o bé un conjunt d'aquests).

Figura 26. Diagrama d'activitats. Representat amb ArgoUML

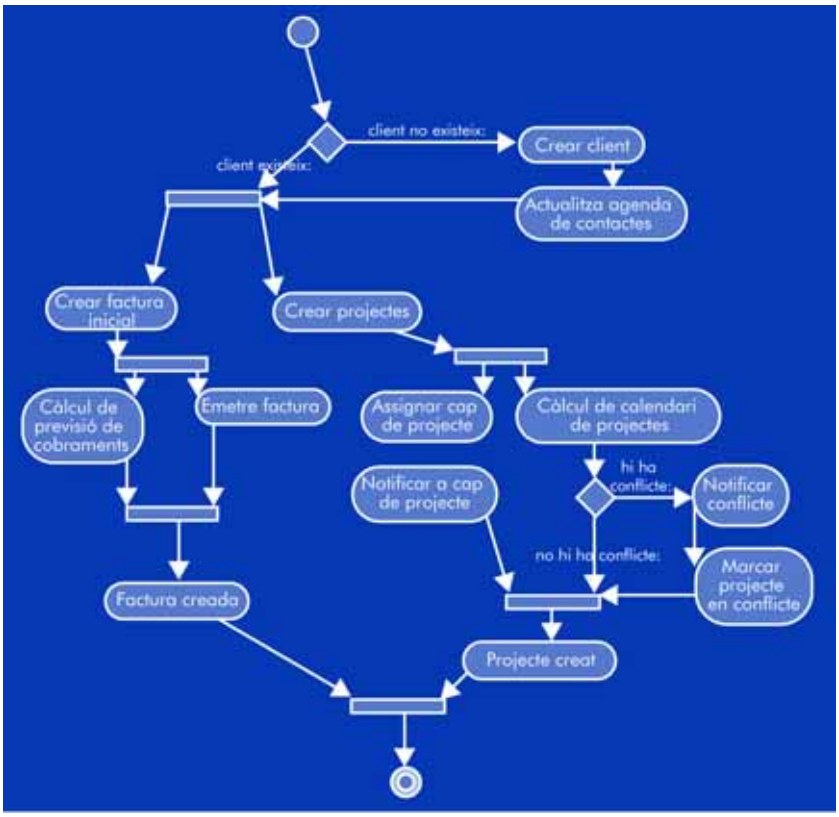


Les transicions són la representació del flux d'informació o procés que avança entre activitats. A diferència dels diagrames de flux, el diagrama d'activitats permet modelar accions en paral·lel. Per a dividir el procés o bé recuperar un únic flux, s'utilitzen les barres de sincronització que permeten diversos fluxos d'entrada o diversos de sortida.

ANOTACIONS

Vegem a continuació un exemple més complet:

Figura 27. Diagrama d'activitats corresponent a la creació d'un projecte

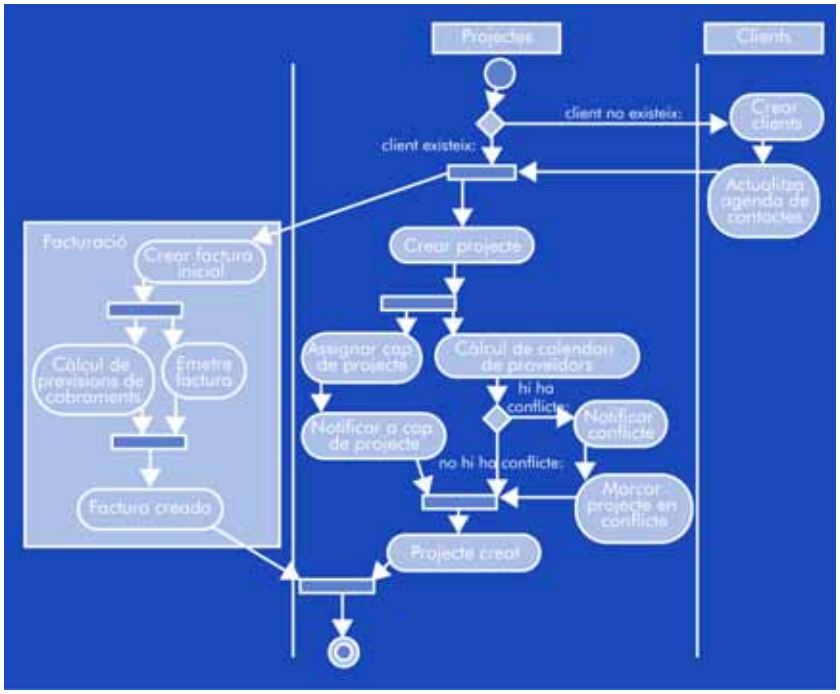


Es pot apreciar que els elements condicionals són semblants als dels diagrames de flux: un rombe amb les transicions etiquetades segons el resultat de la condició avaluada.

Alguns autors consideren una bona pràctica marcar els àmbits d'actuació de classes o components en el diagrama per a donar més informació sense afegir complexitat. Per a això, es poden usar ombres baix dels objectes o bé organitzar les activitats en columnes etiquetades.

ANOTACIONS

Figura 28. Diagrama d'activitats amb separació de l'àmbit en què es produeixen



El diagrama d'activitats, per incorporar mecanismes de sincronització, permet expressar quines activitats es poden realitzar en paral·lel i quines s'han de realitzar en sèrie. La implementació podria optar per no paral·lelitzar sempre que sigui possible, però hem de tenir present que representem el model, no n'expressem la implementació.

També és molt important representar correctament la sincronització quan dos o més fluxos d'activitat convergeixen en un punt. En l'exemple, hem preferit expressar dues activitats "Projecte creat" i "Factura creada" per a evitar sincronitzar totes les activitats finals en un mateix punt. És un concepte que aclareix la representació i en el moment de la implementació no afectarà la qualitat del resultat, ja que si l'activitat simplement torna un valor o recull els resultats de les activitats anteriors, el diagrama continuarà essent igual de correcte. Si a més es fan tasques de consolidació de dades, alliberament d'objectes, etc., aquestes es podran realitzar també en paral·lel.

El diagrama d'activitats d'UML 1.5 no va més enllà. En UML 2.0, s'afegeix un bon nombre d'elements a la seva notació per a expressar el temps, el tipus d'activitat (una simple crida, una transformació

ANOTACIONS

de dades, la transmissió d'un objecte i quin), o la representació d'esdeveniments externs que afecten el flux d'activitats.

Finalment, vegem algunes bones pràctiques relacionades amb aquests diagrames:

- Situar el punt d'inici en la part superior del diagrama.
- Encara que alguns autors el consideren opcional (una activitat pot ser el final del diagrama), sempre convé situar un punt de final de l'activitat per a apreciar ràpidament on acaba el flux d'activitat.
- Si tenim activitats que no tenen entrada però sí sortides, o a l'inrevés, hem de considerar la nostra representació, alguna cosa fem malament...
- El rombe que indica una decisió no té la condició escrita dins a diferència dels diagrames de flux. És en les activitats que en surten on escriurem la condició avaluada i el resultat obtingut per a seguir per aquesta transició.
- Les guardes dels rombes de decisió han de preveure tots els casos possibles. És correcte afegir una transició amb una condició [en cas contrari].
- La paral·lelització (*fork*) i la sincronització (*join*) no poden ocórrer simultàniament. Una paral·lelització només pot tenir una entrada i una sincronització una sortida.

2.9. Diagrama de classes

La realització d'un diagrama de classes és a la frontera entre l'anàlisi i el disseny. Probablement és el diagrama UML més conegut –amb permís dels casos d'ús–, i ens permet identificar l'estructura de classes del sistema incloent-hi les propietats i mètodes de cada classe.

ANOTACIONS

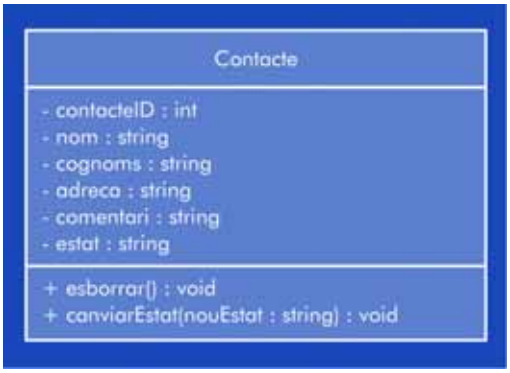
També representarem les relacions que hi ha entre les classes com ara herència, generalització, etc., mantenint la mateixa notació vista en diagrames anteriors per a casos semblants.

Gran part de la popularitat d'aquest tipus de diagrama és que nombroses eines de desenvolupament suporten la generació de codi a partir d'aquesta representació visual, cosa que facilita molt el treball i evita molts errors en les fases inicials del projecte. A més, algunes d'aquestes eines no solament suporten la generació inicial de codi, sinó que són capaces d'actualitzar el diagrama a partir del codi font (enginyeria inversa) o actualitzar el codi a mesura que introduïm canvis en el model encara que aquest hagi estat modificat ja pels desenvolupadors (sempre sota un entorn i unes condicions especials, és clar).

Els elements presents en aquest diagrama són únicament les classes i les seves relacions:

- **Classe**: es representa amb un rectangle dividit en tres seccions. En la part superior, n'haurem d'indicar el nom, a continuació, les propietats o atributs i, en la tercera secció, els mètodes. Alguns elements auxiliars ja vistos com els estereotips (per exemple, <<interficie>>) també poden aparèixer al costat del nom de la classe. Els atributs i els mètodes poden incorporar informació addicional com per exemple el tipus d'accés (públic, privat, protegit), el tipus de dades dels atributs i els paràmetres dels mètodes, etc.

Figura 29



ANOTACIONS

En la classe representada, tots els atributs són privats i els mètodes, públics.

Es poden veure indicats també els tipus de dades dels uns i dels altres.

- **Associació**: representa una relació genèrica entre dues classes, i la seva notació és simplement una línia que les uneix, on podem indicar la multiplicitat de la relació en cada extrem (un a un, un a *n*, *n* a *m*).

- **Composició, agregació**: tal com vam veure en el capítol d'introducció, si una classe és composta per unes altres, i aquestes altres no poden existir sense la primera, tindran una relació de composició amb la classe pare. Quan una classe simplement n'inclou una altra, però la inclosa té entitat per si mateixa, parlarem d'*agregació*.

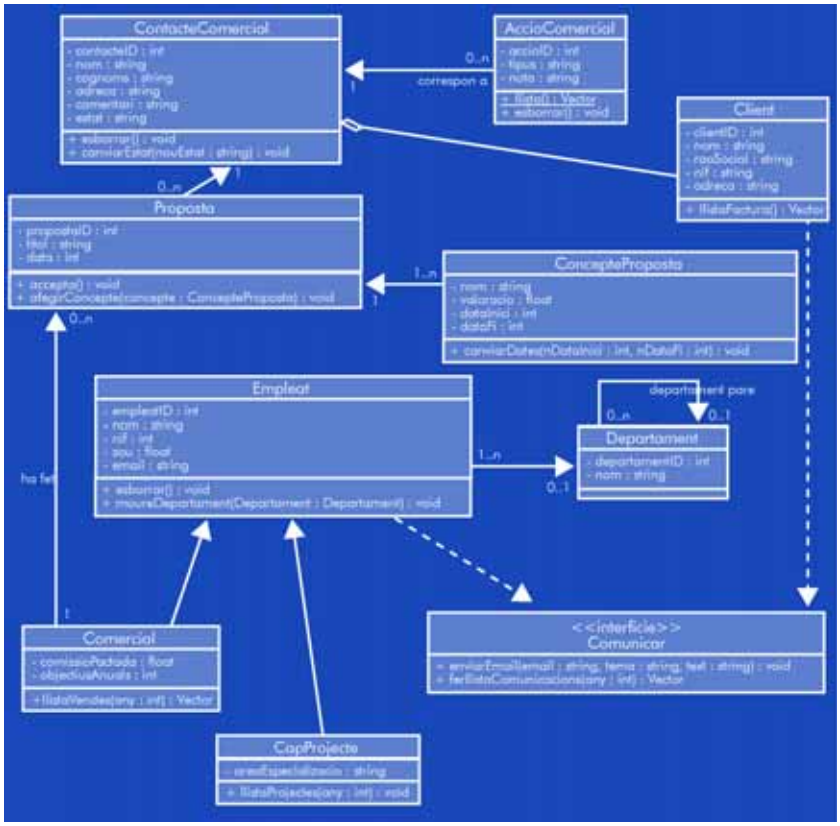
- **Dependència**: quan una classe depèn d'una altra en el sentit que l'usa com a atribut o paràmetre d'algun mètode, es pot expressar mitjançant una relació de dependència.

- **Generalització**: és l'equivalent de l'herència o extensió, tal com hem vist en els altres diagrames.

ANOTACIONS

Vegem un exemple:

Figura 30. Diagrama de classes



Es pot veure que les relacions de generalització i agregació usen la mateixa notació vista en els altres diagrames, la qual cosa contribueix a la coherència entre els models i facilita la seva comprensió i representació. En alguns casos, hem cregut necessari incloure etiquetes en les relacions, ja que aclareixen el diagrama. En uns altres casos –com veurem en el capítol de generació de codi–, pot ser interessant representar també el rol que exerceix cada classe en la relació.

La relació d'implementació de la interfície, l'hem expressada amb la notació de dependència.

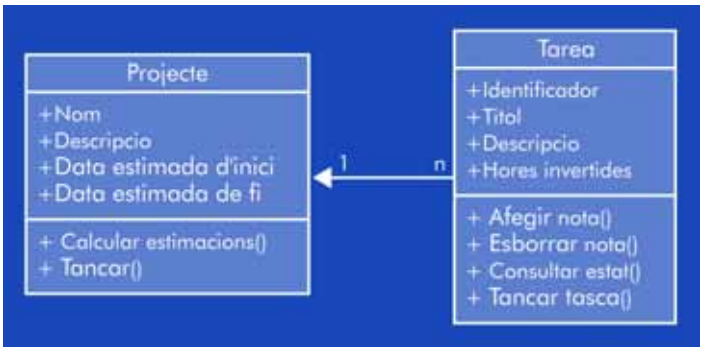
El diagrama expressa també perfectament les classes que es tenen a si mateixes com un dels seus atributs, com és el cas de "Departament".

ANOTACIONS

El modelatge de classes no presenta cap dificultat quant a la representació per si mateixa, la dificultat rau en el fet d'identificar les relacions que hi ha entre les classes que componen el sistema. Per a això, és imprescindible tenir molt clars els conceptes vistos en l'apartat corresponent a la introducció a l'orientació a l'objecte i consultar la nombrosa bibliografia que hi ha.

En un nivell més alt, és possible representar els diagrames de classes sense informació relativa a la seva implementació d'una manera semblant a la següent:

Figura 31



Aquesta representació pot ser molt útil en fases inicials, i si es fa, facilita molt crear el diagrama de classes detallat quan arribi el moment.

A continuació, veurem algunes de les bones pràctiques que cal tenir en compte a l'hora de representar diagrames de classes, encara que algunes d'aquestes fan referència també al disseny de les mateixes classes:

- La visibilitat dels atributs (públic: +; protegit: #; privat: –), és recomanable usar-la només en fase de disseny. Es tracta d'un aspecte important en el disseny de l'objecte i no s'hauria d'obviar, però en diagrames conceptuals no és necessari, ja que potser en fases posteriors hi haurà mètodes que es converteixin en atributs i al contrari. El mateix argument es pot usar per als tipus de dades.

- Els noms dels mètodes i els atributs haurien de reflectir les convencions quant a denominació del llenguatge de programació amb

ANOTACIONS

què anem a implementar el sistema. És important no només per coherència, sinó per a aprofitar els avantatges que ens oferiran els generadors de codi.

- Quan una associació té mètodes o atributs, l'haurem de modelar com una classe que s'associa amb les dues anteriors.

- No cal incloure els mètodes d'accés i modificació dels atributs (típicament getXXX i setXXX). Es poden donar per suposats en el model i la majoria de generadors de codi els generen automàticament o sota demanda.

- Si hem de deixar alguna llista de paràmetres incompleta, ho hem d'indicar amb una el·lipsi (...).

- Els atributs i operacions estàtiques (representades amb el nom subratllat en el diagrama) haurien d'aparèixer abans que les d'instància (la resta). Seguint el mateix criteri, convé ordenar els mètodes i atributs segons la seva visibilitat descendent: públics → protegits → privats.

- No convé abusar dels estereotips. Algunes eines inclouen estereotips del tipus <<constructor>> o <<getter>> en els mètodes corresponents. Això només contribueix a afegir complexitat al diagrama sense proporcionar informació valuosa.

- Si el nostre model inclou interfícies múltiples, pot ser útil incloure-les en un altre diagrama i utilitzar la notació per a la classe que implementi la interfície del diagrama de components en el diagrama de classes (el cercle tancat). Evidentment, es desaconsella repetir els atributs i mètodes de la interfície que implementa una classe.

- És aconsellable indicar sempre la multiplicitat d'una relació. Ens serà útil per a entendre millor el tipus de relació i ajudarà també el generador de codi. Convé ser escrupolós a l'hora d'especificar-lo, i concretar si la relació és 0..1, 1, 0..n, 1..n, o n..m.

ANOTACIONS

- No cal modelar totes les dependències. Això dóna com a resultat diagrames completament saturats d'associacions que no aporten informació. És més convenient modelar només les que aportaran alguna cosa al model.

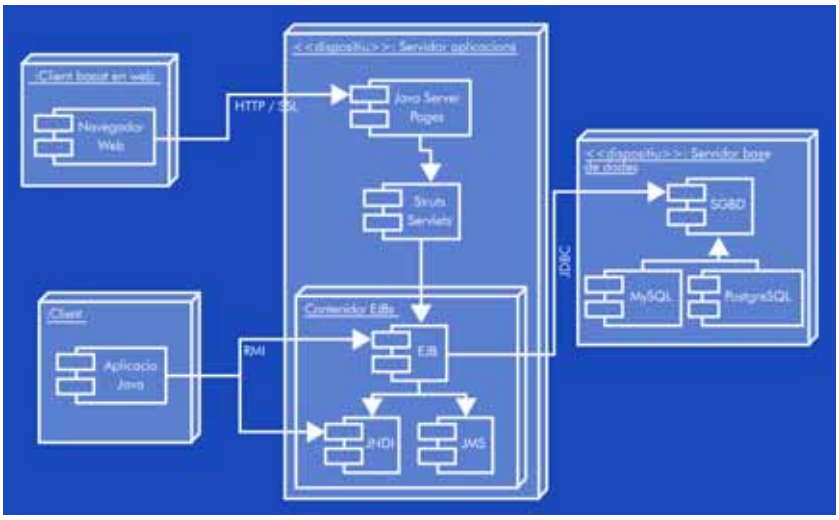
- Si hi ha diverses relacions entre dues classes, pot ser interessant indicar el rol que ocupen en cada una. El rol s'indica al costat de la multiplicitat mitjançant el subjecte de l'oració que contindria el verb de l'associació (per exemple, "nova proposta", "antic client", etc.).

2.10. Diagrama de desplegament

El diagrama UML de desplegament representa una vista estàtica de la configuració en temps d'execució dels nodes que intervenen en el procés i dels components que s'executen en aquests nodes. Pot mostrar el maquinari del sistema i els paquets de programari que hi ha instal·lats en ell, el programari intermediari que els connecta, etc.

També són útils per a explorar l'arquitectura de sistemes embeguts perquè mostren els components de maquinari i de programari, i també els seus protocols o maneres de comunicar-se.

Figura 32. Diagrama de desplegament del sistema



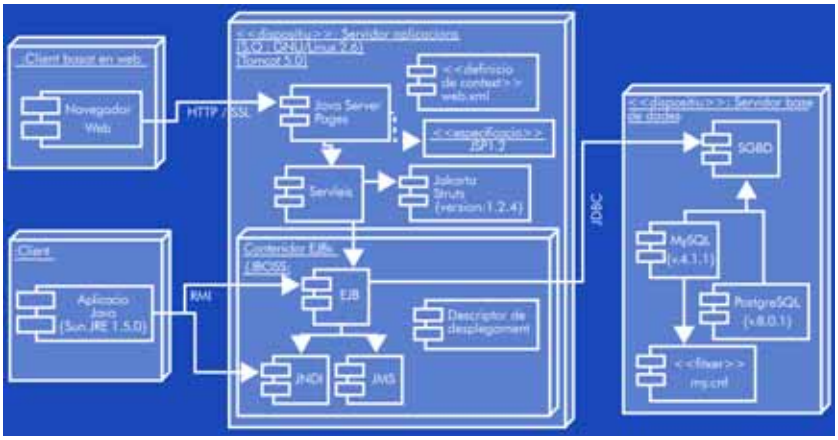
En el diagrama de desplegament anterior, les caixes tridimensionals representen els nodes del sistema, tant de programari com de maquinari. Podem aclarir de què es tracta exactament cada un mitjançant estereotips, sempre que aquests ajudin a entendre el diagrama.

Les connexions entre nodes del sistema es poden etiquetar amb el protocol de comunicació en què s'implementen. Els nodes poden contenir subnodes o components de programari. Els components de programari usen la mateixa notació que en els diagrames de components, per la qual cosa podríem afegir la notació sobre les seves interfícies si fos necessari, encara que en el nivell en què es representen els diagrames de desplegament no ho sol ser.

En configuracions complexes, els diagrames de desplegament e poden estendre ràpidament si indiquem tots els fitxers de configuració, especificacions d'arquitectures i versions de cada component implicat. Aquesta és la informació que seria realment útil per als desenvolupadors que han d'implantar o mantenir el sistema, però no sembla que UML proporcioni les eines per a representar-la de manera compacta.

Vegem el mateix exemple més detallat per comprovar-ho.

Figura 33. Diagrama de desplegament amb alguna informació addicional sobre la configuració del sistema



Veiem que la inclusió de fitxers de configuració, versions, especificacions de desplegament, etc. comença a complicar el diagrama quan, potser, un simple full de requisits i unes instruccions de desplegament textuals juntament amb el primer diagrama serien més útils.

ANOTACIONS

Així doncs, quan treballem amb diagrames de components és recomanable seguir aquestes pràctiques:

- Indicar només els components de programari clau per al projecte.
- És important expressar els nodes amb noms descriptius que siguin útils per als desenvolupadors.
- Intentar no abusar dels estereotips i mantenir una coherència en la seva nomenclatura amb la resta de diagrames.
- Les comunicacions també poden incorporar estereotips que indiquin, per exemple, si la comunicació és síncrona o asíncrona, si es tracta d'un servei web, etc.

No cal modelar totes les dependències dels components dins d'un node. S'entén que, per exemple, els fitxers de configuració o especificacions d'entorn d'execució seran llegits pel node o un altre component d'aquest. El desenvolupador ho sabrà i en la majoria dels casos no serà necessari representar associacions d'aquests components amb la resta.

2.11. Diagrama d'estats

El propòsit dels diagrames d'estats és documentar les diferents modalitats (els estats) per les quals una classe pot passar i els esdeveniments que provoquen aquests canvis d'estat. A diferència dels diagrames d'activitats o de seqüència que mostren les transicions i interaccions entre classes, el diagrama d'estats mostra habitualment les transicions dins d'una mateixa classe.

Normalment l'usarem en combinació amb els casos d'ús, per tenir delimitats els casos que provocaran canvis d'estat en un objecte. No totes les classes necessitaran un diagrama d'aquest tipus i normalment s'usaran com a complement dels diagrames d'activitats i dels de col·laboració.

ANOTACIONS

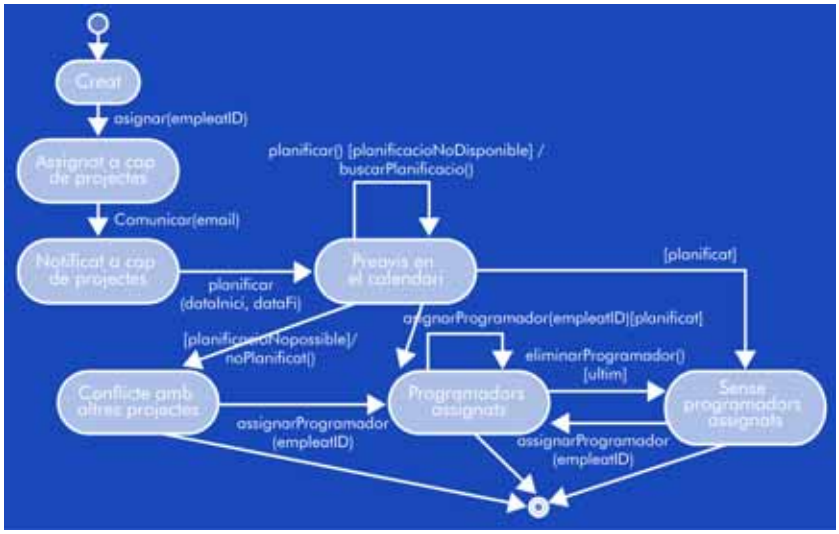
Quant a la notació, comparteixen molts elements amb els altres diagrames que representen el comportament del model, com els diagrames d'activitat i col·laboració ja esmentats.

- **Estat**: representa l'estat d'un objecte en un instant de temps. Tindrem tants símbols d'estat en el diagrama com estats diferents per a cada objecte calgui modelar. La seva aparença és semblant a la representació d'una classe, però amb les cantonades arrodonides.

- **Estats inicial i final**: són pseudoestats que mostraran el punt d'inici i final del flux d'activitat. La seva condició de pseudoestat ve donada pel fet que no té variables ni accions definides.

- **Transicions**: una fletxa indicarà la transició entre estats. Hi descriurem l'esdeveniment que ha disparat la transició i l'acció que provoca el canvi. Hi ha transicions en què no hi ha un esdeveniment que les provoqui (per exemple, ha acabat una activitat que s'estava realitzant).

Figura 34. Diagrama d'estats de la classe Projecte



L'exemple mostra les transicions que ocorren en l'objecte "Projecte" durant la seva creació. En aquest cas, la transició pot mostrar únicament l'acció que es realitza, l'esdeveniment que ocorre o la condició

ANOTACIONS

perquè es realitzi l'acció en concret. En la seva notació més completa, es representa de la manera següent:

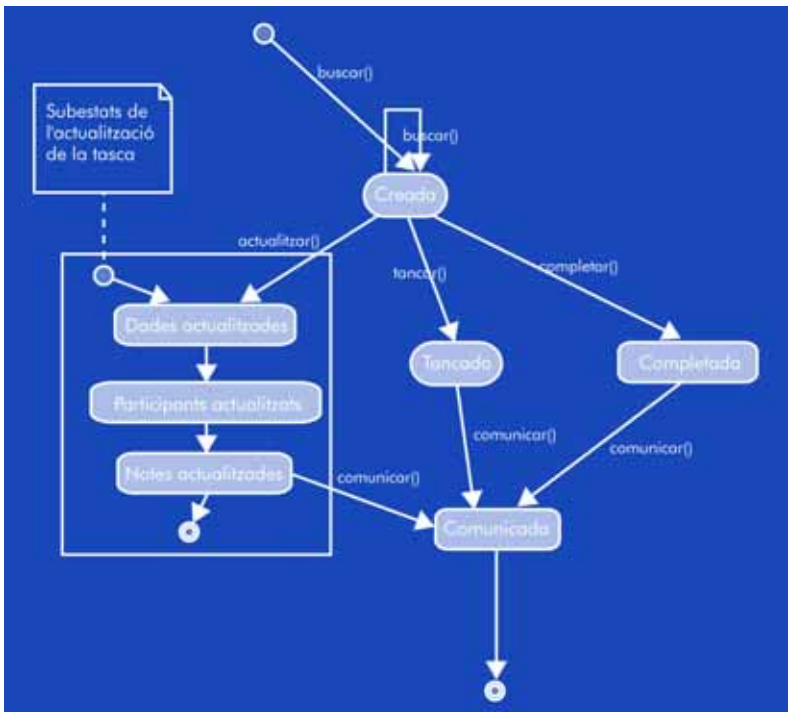
```
esdeveniment() [condició] acció ()
```

Veiem que una acció pot anar iterant sobre el mateix estat segons una condició. En l'exemple, podem anar assignant programadors al projecte mentre l'estat de l'objecte continua reflectint que té programadors assignats. En el moment en què eliminem l'últim programador, aquest passa a l'estat "Sense programadors assignats" del qual pot sortir amb l'acció `assignarProgramador()`.

En examinar el diagrama, ens adonem ràpidament del que comentàvem a l'inici: el diagrama d'estats és una representació molt detallada i només serà necessari quan en els diagrames de casos d'ús en què intervé un objecte o els diagrames de seqüència en què intervé no proporcionin tota la informació necessària.

Igual que en els altres diagrames, pot ser interessant encapsular els estats en zones que aclareixin l'acció global que estem realitzant o en subestats d'aquesta. Vegem un altre exemple:

Figura 35. Diagrama d'estats amb un subestat



En aquest cas, s'aprecien algunes incoherències en la representació, producte de l'eina que hem utilitzat. Teòricament, la transició cap al subestat s'hauria de fer cap a l'estat inicial d'aquesta, però l'eina impedeix realitzar una transició cap a un estat inicial. El mateix ocorre amb l'estat final del subestat, hi hauria d'haver la transició cap a l'estat "Comunicada", però l'eina impedeix que hi hagi transicions des d'un estat final.

Vegem les bones pràctiques relacionades amb els diagrames d'estats:

- Ordenar de dalt a baix els estats i les seves transicions (en concret l'estat inicial i final) ajuda a comprendre el diagrama.

- Per a incrementar el valor de comunicació del diagrama, és important parar atenció als noms dels estats. S'aconsella usar verbs en infinitiu (Actualitzar, Presentar) o en participi (Actualitzat, Presentat).

- Igual que en els diagrames d'activitat, caldrà vigilar els estats que només emeten transicions però no provenen de cap, i els estats que només reben transicions però no expressen un canvi d'estat de l'objecte.

- Si és possible, convé anomenar les accions amb els mètodes que faran canviar l'estat de l'objecte. D'aquesta manera s'incrementa la coherència del diagrama amb la resta.

- Si en un estat totes les seves transicions de sortida o d'entrada corresponen a la mateixa acció i només varien en la seva condició, és correcte situar el nom de l'acció dins l'estat i només indicar la condició en cada transició.

- De manera semblant als diagrames d'activitat, haurem de repassar les condicions de les transicions per a assegurar-nos que no hi ha un cas en què hi pugui haver dues transicions.

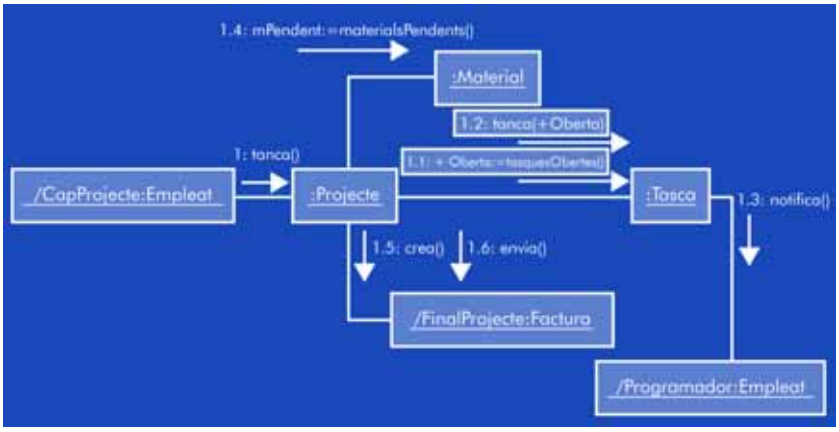
ANOTACIONS

2.12. Diagrama de col·laboració

Igual que altres diagrames UML basats en el comportament del sistema, els diagrames de col·laboració modelen les interaccions entre objectes. Molts autors els consideren una variació dels diagrames de seqüència en què els objectes no apareixen en files i columnes, sinó distribuïts lliurement i amb els missatges numerats per a seguir les possibles seqüències de missatges.

Els elements que intervenen són objectes, actors i missatges, en la mateixa notació que en diagrames anteriors.

Figura 36. Diagrama de col·laboració que representa els missatges entre objectes en tancar un projecte



Veiem ràpidament que el nivell d'informació és pràcticament idèntic al d'un diagrama de seqüència, amb les excepcions següents:

- No es preveuen els retorns ni els errors, simplement els missatges que intercanvien els objectes entre ells.

- La seqüència és més difícil de seguir, cal mirar l'etiqueta dels missatges.

Per claredat, la notació d'un diagrama de col·laboració és més simple que en un diagrama de seqüència, i pot ser convenient utilitzar-lo quan l'ordre dels missatges no és important. Aprofitant-ne la semblança, en podem ampliar també la notació amb elements dels diagrames de seqüència (les condicions entre [ ], la crida

ANOTACIONS

a casos d'ús, etc.) i obtenir un diagrama més complet i segurament més comprensible per personal no tècnic que un diagrama de seqüència.

Un aspecte important dels diagrames de col·laboració és que mostren els rols que pren cada classe en la comunicació. En el diagrama, es poden apreciar dues instàncies de la classe "Empleat" que ocupen els rols "CapProjecte" i Programador".

Una variació d'aquest diagrama pren el nom, de vegades, de *diagrama de comunicació*, en què especifiquem els objectes que es comuniquen entre ells, el rol que ocupa cada un en la comunicació i la multiplicitat dels uns respecte dels altres.

Es podria dir que aquest diagrama és una versió preliminar del diagrama de classes, en què reflectim també la comunicació entre elles. La notació dels missatges és particular per a aquest tipus de diagrames i té la forma següent:

```
[numeroSequencia:] nomMetode(parametres) [:valorRetorn]
```

o de vegades la següent:

```
[numeroSequencia:] [valorRetorn:=]nomMetode(parametres)
```

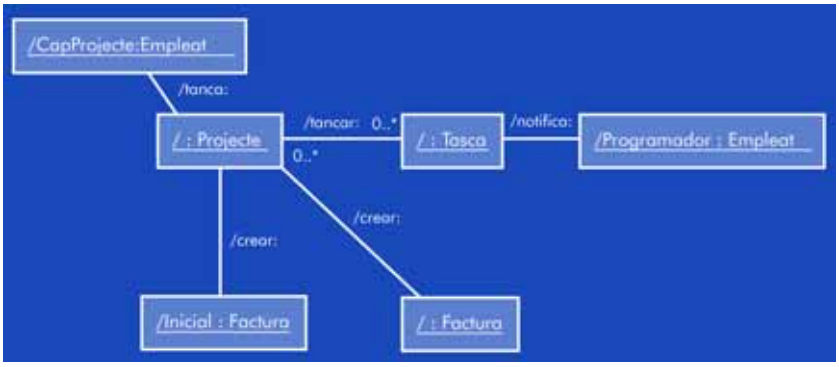
- Vegem algunes bones pràctiques relacionades amb ells:

- Els diagrames de col·laboració en l'àmbit d'instància (com el de l'exemple) són útils per a explorar possibles problemes en el disseny de les classes.

- Hi ha una variant d'aquest diagrama en un nivell més alt (en l'àmbit d'especificació) que se sol usar per a explorar els rols que ocuparan les classes en el sistema. Aquesta notació no és gaire habitual, ja que és gairebé una còpia de la notació del diagrama de classes però sense representar-ne els mètodes i atributs.

ANOTACIONS

Figura 37. Diagrama de col·laboració en l'àmbit d'especificació



- Cal recordar que els diagrames de col·laboració no modelen el flux de procés, únicament les interaccions entre objectes, i aquesta interacció es produeix per mitjà dels missatges que intercanvien. Si volem modelar el procés o les dades que intercanvien, hem d'utilitzar un diagrama d'activitats.

- Quan sigui clau la seqüència en què es produeixen els missatges per a comprendre'n la interacció, és molt més convenient utilitzar un diagrama de seqüència.

- Quan representem els missatges, és important pintar una fletxa per a cada missatge i incloure només les dades necessàries per a comprendre la comunicació. Si els paràmetres o els valors de retorn no aporten res a la comprensió d'aquest, no són necessaris.

2.13. Generació de codi

Fins al moment, hem parlat de la representació del model que ens proporciona UML, de la seva notació i de les eines que ens ajuden a obtenir diagrames i documentació per als nostres projectes basant-nos en aquest estàndard.

En els primers passos d'UML, va ser una empresa (Rational Corp.) la que va patrocinar el projecte, i posteriorment van ser moltes altres les que van participar i participen en el comitè que revisa les aportacions i idees. De la mateixa manera que els teòrics en modelatge i representació van aportar les seves idees per a la notació, les empreses es van preocupar perquè aquesta fos prou completa perquè les seves

ANOTACIONS

eines de modelatge i desenvolupament poguessin generar codi a partir dels models.

La majoria de les empreses que participen en la revisió de l'estàndard comercialitzaven entorns de desenvolupament que ja disposaven d'eines de generació de codi i van veure el potencial d'UML i les possibles millores de les seves eines. Lamentablement, cap d'aquestes eines no és de codi obert, i per tant no les veurem aquí, però com que UML és un estàndard obert, moltes altres eines de modelatge de codi obert (també excel·lents) han anat incorporant aquesta prestació al llarg dels anys.

Les tres eines que hem utilitzat durant el capítol per a representar els diagrames tenen, en un grau més o menys alt, suport per a generar codi a partir dels models.

2.13.1. Dia amb Dia2Code

El programari de modelatge Dia, que trobareu disponible a http://www.gnome.org/projects/dia/, no incorpora directament una eina de generació de codi, però hi ha diverses eines que treballen conjuntament amb aquest, sia per a generar diagrames en el seu format o sia per a extreure informació dels seus diagrames i treballar-hi. Aquest és el cas de Dia2Code.

Dia2Code està disponible a http://dia2code.sourceforge.net/ i l'última versió estable és la 0.81. És un programa executable des de l'indicador d'ordres que pren els paràmetres següents:

```
Usage: dia2code [-h|--help] [-d <dir>] [-nc] [-cl <classlist>]
       [-t (ada|c|cpp|idl|java|php|python|shp|sql)] [-v]
       [-l <license file>] <diagramfile>
  -h --help          Print this help and exit
  -t <target>        Selects the output language. <target> can be
                     one of: ada,c,cpp,idl,java,php,python,shp or sql.
                     Default is C++
  -d <dir>           Output generated files to <dir>, default is "."
  -l <license>       License file to prepend to generated files.
  -nc                Do not overwrite files that already exist
  -cl <classlist>    Generate code only for the classes specified in
                     the comma-separated <classlist>.
                     E.g: Base,Derived.
```

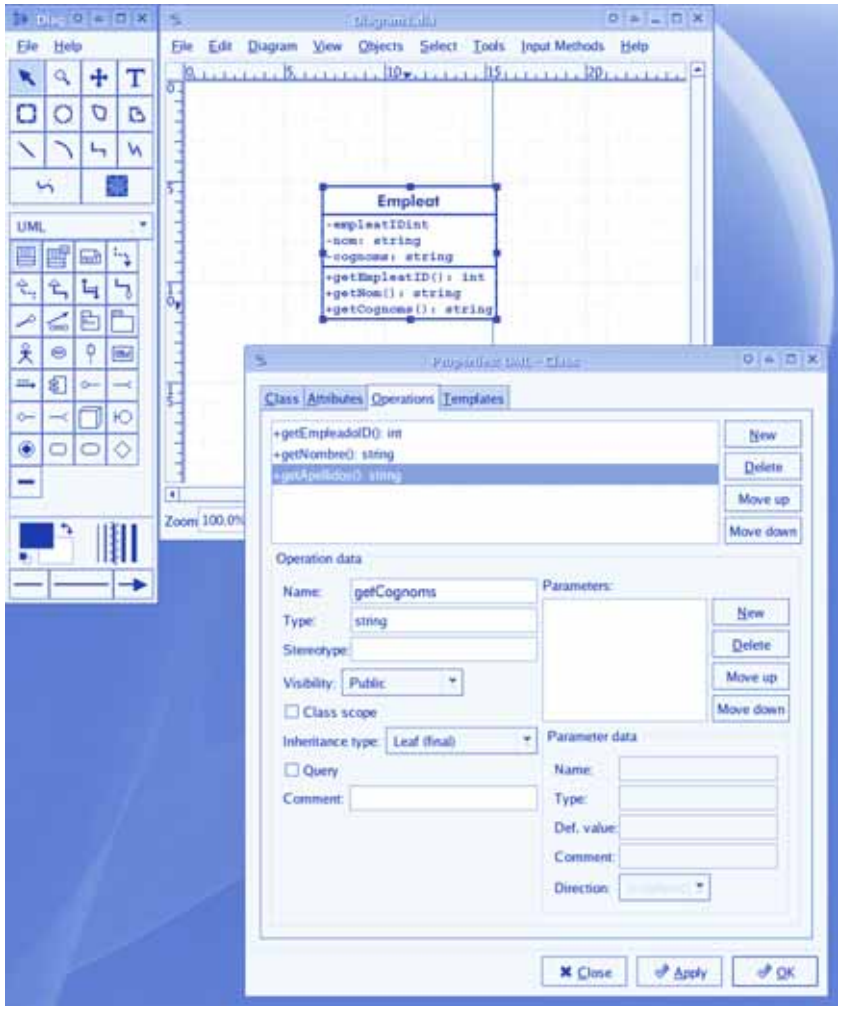
ANOTACIONS

```
-v                  Invert the class list selection. When used
                    without -cl prevents any file from being created
<diagramfile>       The Dia file that holds the diagram to be read
Note: parameters can be specified in any order.
```

A primera vista, veiem que el nombre de llenguatges de programació suportats és notable. Particularment interessant és el suport de Python i PHP, no massa habitual en aquestes eines, ni tan sols en les comercials.

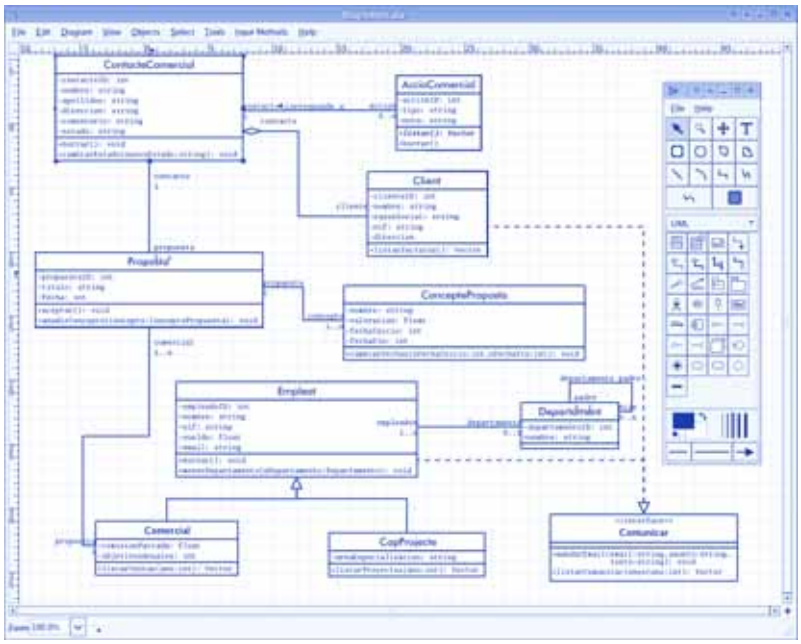
Es pot dir que, perquè Dia2Code interpreti correctament el diagrama UML, aquest s'ha de realitzar amb les eines específiques UML que proporciona Dia, i no pintant els mateixos símbols. Això sembla evident però és important perquè Dia és una eina genèrica de diagrames i es podria obtenir el mateix resultat representant els elements amb altres formes de la seva biblioteca:

Figura 38. Captura de pantalla de Dia que mostra les propietats d'una classe



El diagrama sobre el qual generarem codi és el que presentem en el capítol corresponent al diagrama de classes. Perquè la generació de codi sigui realment útil, és convenient proporcionar tota la informació sobre les classes, mètodes, atributs i cada una de les associacions. Així doncs, conceptes com el rol que exerceix cada extrem de les associacions són importants, ja que és presumible que el programa generador les utilitzi per a generar l'atribut corresponent a l'associació. Vegem com queda representat el diagrama complet en Dia:

Figura 39. Diagrama de classes representat en Dia



I ara llancem el programa generador de codi sobre el fitxer del diagrama. Per a aquest exemple, generarem codi Java:

```
[marcg@lynx src]$ dia2code -t java -d /tmp/uocFiles/ Diagrama26.dia
```

Comprovem que s'han generat els fitxers per a les classes. A continuació, vegem els casos més interessants i quin ha estat el codi obtingut:

```
public class Proposta {
   /* * Attributes */
   private int propostaID;
   private string titol;
   private int data;
```

```
   /* * Associations */
   private Comercial proposta;
   /**
   * Operation
   *
   */
   public void acceptar ( ){
   }
   /**
   * Operation
   *
   * @param concepte
   */
   public void afegirConcepte ( ConcepteProposta concepte ){
   }
}

import Empleat;

public class Comercial extends Empleat {
   /* * Attributes */
   private float comissioPactada;
   private int objectiusAnuals;
   /**
   * Operation
   *
   * @param any
   * @return Vector
   */
   public Vector ferllistaVendes ( int any ){
   }
}

import Comunicar;

public class Client implements Comunicar {
   /* * Attributes */
   private int clientID;
   private string nom;
   private string raoSocial;
   private string nif;
   private direccio;
   /**
   * Operation
   *
   * @return Vector
   */
   public Vector llistarFactures ( ){
   }
}


import Proposta;
import Client;
import AccioComercial;

public class ContacteComercial {
   /* * Attributes */
```

ANOTACIONS

```
    private int contacteID;
    private string nom;
    private string cognoms;
    private string adreca;
    private string comentari;
    private string estat;
    /* * Associations */
    private Proposta proposta;
    private Client client;
    private AccioComercial accio;
    /**
    * Operation
    *
    */
    public void esborrar ( ){
    }
    /**
    * Operation
    *
    * @param nouEstat
    */
    public void canviarEstat ( string nouEstat ){
    }
}
```

Podem veure que s'han generat correctament les importacions entre classes, els atributs i els mètodes. Destacable és també la inclusió de comentaris a l'estil JavaDoc.

Pel que fa a les particularitats del model, el generador ha entès perfectament les relacions d'herència (extends), la generalització i la implementació de la interfície per part de la classe "Client". En tots els casos, ha inclòs la declaració d'atributs amb els noms de rols que hem representat en el diagrama. Pel que fa a la multiplicitat en les associacions, és lògic que la generació no inclogui atributs per a les associacions amb multiplicitat més gran que un, ja que aquestes s'implementaran mitjançant mètodes (per exemple, `afegirContacte()`, `esborrarContacte()`) i no amb vectors o altres estructures de dades estàtiques (encara que altres generadors tenen enfocaments diferents, com veurem).

Dia ha modelat tots els mètodes de cada classe, però no ha inclòs mètodes getXXX i setXXX per a cada atribut. Si volem que el generador els inclogui en el codi font, els haurem de representar en el model malgrat afegir informació irrellevant al diagrama.

ANOTACIONS

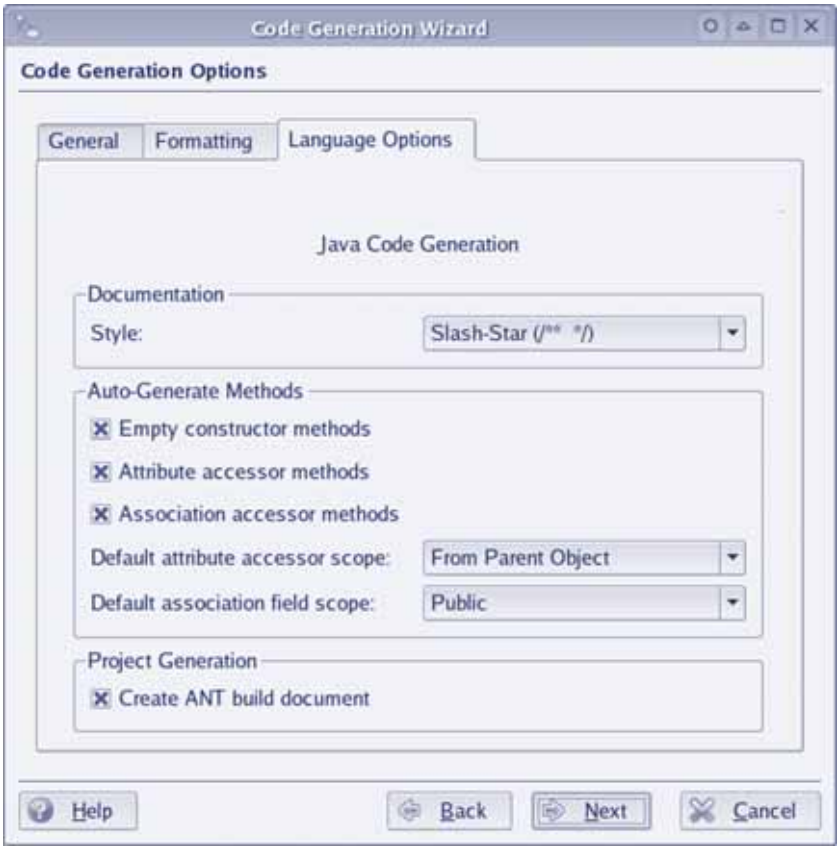
2.13.2. Umbrello

El programari de model Umbrello, http://uml.sourceforge.net o com a part del paquet kdesdk de l'entorn KDE, ha evolucionat ràpidament durant els darrers anys i actualment manté un grau alt d'activitat en el seu projecte.

A diferència de Dia, Umbrello és una aplicació totalment orientada al modelatge UML i per això disposa d'eines que faciliten molt la creació de tots els tipus de diagrames. Entre elles, destaca la barra lateral esquerra, en què apareixen tots els elements UML que hem creat en algun diagrama per a poder aprofitar-los en els diagrames nous que realitzem a partir del mateix model.

La generació de codi és molt completa i disposem d'un assistent que ens presenta multitud d'opcions. A continuació, podem veure les disponibles per a generar codi Java:

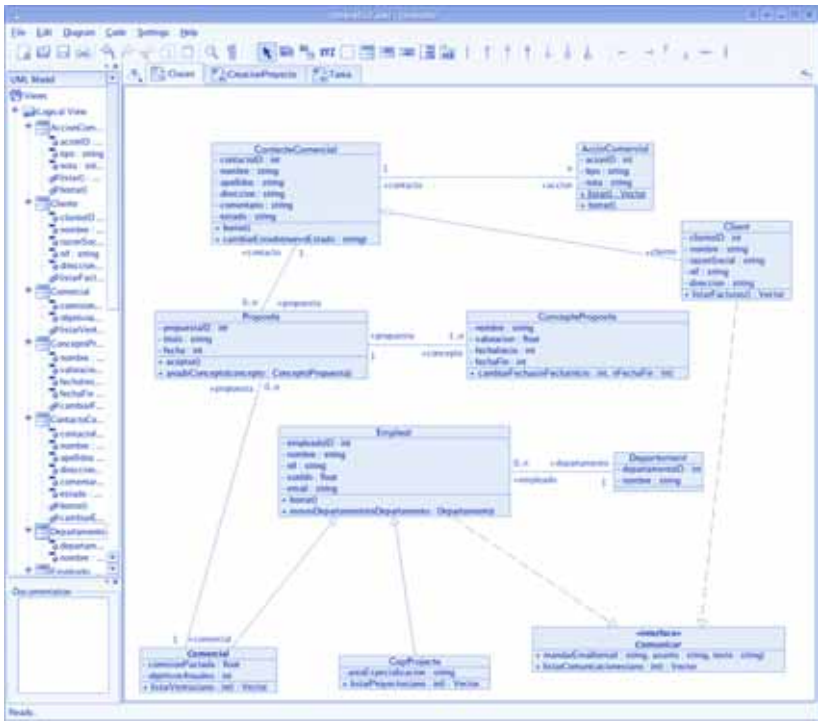
Figura 40. Quadre de diàleg amb les opcions de generació de codi Java en Umbrello



ANOTACIONS

Aquest és l'aspecte del diagrama de classes representat en Umbrello:

Figura 41. Diagrama de classes representat en Umbrello



El codi font de les classes generades és molt complet. Els comentaris estan en format JavaDoc i, encara que no aprofitem les opcions de documentació de les classes i mètodes que ofereix el mateix Umbrello, aquest ja genera molta informació.

```
import Empleat;
import Proposta;
/**
 * Class Comercial
 *
 */
public class Comercial extends Empleat {
 // Fields
 //
 private float comissioPactada;
 //
 private int objectiusAnuals;
 //
 public List propostaVector = new Vector( );
 // Methods
 // Constructors
 // Empty Constructor
 public Comercial ( ) { }
 // Accessor Methods
```

ANOTACIONS

```
/**
   * Get the value of comissioPactada
   *
   * @return the value of comissioPactada
   */
private float getComissioPactada ( ) {
   return comissioPactada;
   }
   /**
   * Set the value of comissioPactada
   *
   *
   */
 private void setComissioPactada ( float value ) {
   comissioPactada = value;
 }
 /**
   * Get the value of objectiusAnuals
   *
   * @return the value of objectiusAnuals
   */
 private int getObjectiusAnuals ( ) {
   return objectiusAnuals;
 }
 /**
   * Set the value of objectiusAnuals
   *
   *
   */
 private void setObjectiusAnuals ( int value ) {
   objectiusAnuals = value;
 }
 /**
   * Add an object of type Proposta to the List propostaVector
   *
   * @return void
   */
 public void addProposta ( Proposta value ) {
   propostaVector.add(value);
 }
 /**
   * Remove an object of type Proposta from the List propostaVector
   *
   */
 public void removeProposta ( Proposta value ) {
   propostaVector.remove(value);
 }
 /**
   * Get the list of propostaVector
   *
   * @return List of propostaVector
   */
 public List getPropostaList ( ) {
   return (List) propostaVector;
 }
 // Operations
```

ANOTACIONS

```
 /**
  *
  * @param any
  * @return Vector
  */
 public Vector llistarVendes ( int any) {

  }
}


import ContacteComercial;
import Comunicar;
/**
  * Class Client
  *
  */
 public class Client implements Comunicar {
 // Fields
 //
 private int clientID;
 //
 private String nom;
 //
 private String raoSocial;
 //
 private String nif;
 //
 private String adreca;
 //
 public ContacteComercial contacte = new ContacteComercial ( );
 // Methods
 // Constructors
 // Empty Constructor
 public Client ( ) { }
 // Accessor Methods
 /**
  * Get the value of clientID
  *
  * @return the value of clientID
  */
 private int getClientID ( ) {
  return clientID;
 }
 /**
  * Set the value of clientID
  *
  *
  */
 private void setClientID ( int value ) {
  clientID = value;
 }
 /**
  * Get the value of nom
  *
  * @return the value of nom
  */
```

ANOTACIONS

```
private String getNom ( ) {
 return nom;
}
/**
 * Set the value of nom
 *
 *
 */
private void setNom ( String value ) {
 nom = value;
}
/**
 * Get the value of raoSocial
 *
 * @return the value of raoSocial
 */
private String getRaoSocial ( ) {
 return raoSocial;
}
/**
 * Set the value of raóSocial
 *
 *
 */
private void setRaoSocial ( String value ) {
 raoSocial = value;
}
/**
 * Get the value of nif
 *
 * @return the value of nif
 */
private String getNif ( ) {
 return nif;
}
/**
 * Set the value of nif
 *
 *
 */
private void setNif ( String value ) {
 nif = value;
}
/**
 * Get the value of adreça
 *
 * @return the value of adreça
 */
private String getAdreca ( ) {
 return adreca;
}
/**
 * Set the value of adreca
 *
 *
 */
```

ANOTACIONS

```
 private void setAdreca ( String value ) {
  adreca = value;
 }
 /**
  * Get the value of contacte
  *
  * @return the value of contacte
  */
 public ContacteComercial getContacte ( ) {
  return contacte;
 }
 /**
  * Set the value of contacte
  *
  *
  */
 public void setContacte ( ContacteComercial value ) {
  contacte = value;
 }
 // Operations
 /**
  *
  * @return Vector
  */
public Vector llistarFactures ( ) {

  }
}
```

Hem marcat les opcions per generar els mètodes d'accés als atributs, i així ho ha fet, i ens proporciona de més a més comentaris que, encara que són trivials, estableixen la base perquè els completem.

Suporta adequadament totes les associacions, herència i implementació d'interfícies. A més, ha generat vectors o variables simples en les associacions segons la multiplicitat d'aquestes i mètodes per a afegir, esborrar i obtenir la llista de la classe associada.

A més, Umbrello incorpora la possibilitat d'importar les classes a partir dels seus fitxers font. Aquesta opció ens permet disposar del model de les classes juntament amb els seus atributs i mètodes, però no pinta el diagrama de classes corresponent. Per a això, simplement haurem d'arrossegar les classes des de la llista situada en la part esquerra fins al diagrama i crear les associacions que Umbrello no hagi detectat.

ANOTACIONS

En resum, podem dir que som davant d'una eina molt bona que compleix amb els objectius de qualsevol projecte de desenvolupament que cerqui un programari de modelatge amb generació de codi de qualitat. A més, la seva facilitat d'ús i la gran quantitat de llenguatges de programació suportats (es pot destacar el suport diferenciat de PHP4 i PHP5, Perl i XMLSchema) ens permet obtenir bons resultats en molt poc de temps.

2.13.3. ArgoUML

L'entorn de modelatge ArgoUML, disponible a http://argouml.tigris.org, és potser el més complet en termes de compliment amb l'estàndard. És un projecte que va començar en privat el 1995 com una eina CASE i el 1999 va evolucionar cap a un projecte de codi obert integrant el modelatge UML i les prestacions per al desenvolupament ràpid d'aplicacions.

Aquesta orientació cap al desenvolupament és notòria, no només en la seva interfície, en què el diagrama té un paper gairebé secundari, també en el suport estricte de la notació i en prestacions com les crítiques automàtiques al disseny o consells que rebem a mesura que el nostre model evoluciona.

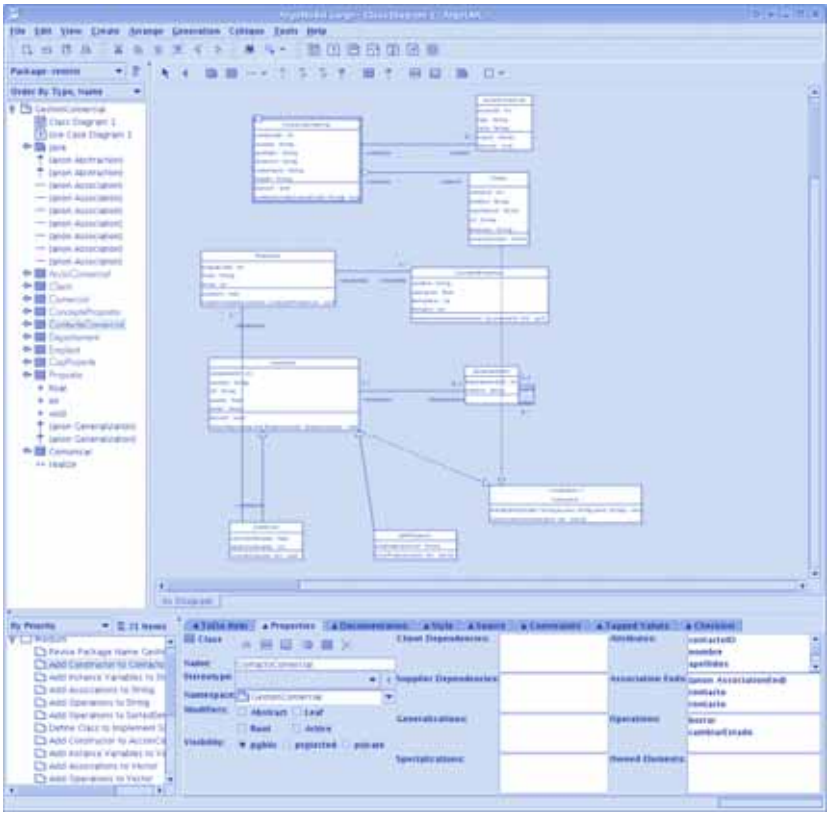
La generació de codi no ofereix tantes opcions com en els programes anteriors (només genera codi Java i la mateixa notació UML 1.3), no suporta la inclusió de fitxers de llicència o capçaleres, etc. L'enfocament d'ArgoUML és totalment cap a UML com a notació i cap al disseny d'aplicacions orientades a l'objecte.

Es pot destacar especialment la prestació de "crítiques al disseny", on per a cada classe, atribut, mètode i, en general, qualsevol element del model, ArgoUML ens presenta una llista de control que podem repassar i validar per a millorar el nostre disseny. També ens ofereix crítiques en general sobre el diagrama en aspectes com la notació (l'ús de majúscules i minúscules en els noms de classes, atributs i mètodes), l'ús de patrons de disseny, la inclusió de constructors, etc.

ANOTACIONS

Aquest és l'aspecte d'ArgoUML quan mostra el nostre diagrama de classes:

Figura 42. Diagrama de classes representat en ArgoUML



Podem veure que en la part esquerra de l'entorn disposem de l'arbre d'elements UML que inclou el nostre projecte. En la part inferior, és on es mostren les propietats de l'element seleccionat (en aquest cas, una classe), i en la part inferior esquerra tenim els consells de disseny.

Vegem aquest darrer aspecte amb una mica més de detall:

Figura 43. Detall de l'àrea de crítiques al disseny

Veiem que la informació que ofereix per a un element en concret (crear el constructor d'una classe) és molt extensa i inclou consells

ANOTACIONS

tant per a crear el constructor amb el programa com sobre la importància d'aquest i de les implicacions que té en UML.

Com hem comentat, la generació de codi no ofereix gaires opcions i es limita a crear simplement els mètodes representats; així doncs, a diferència dels programes avaluats anteriorment, no generarà codi ni per al constructor, ni per als mètodes d'accés als atributs, ni per a accedir a les variables representades per a les associacions.

```
import java.util.Vector;

public class Comercial extends Empleat {
   /* {src_lang=Java}*/

private float comissioPactada;
   /* {transient=false, volatile=false}*/

private int objectiusAnuals;
   /* {transient=false, volatile=false}*/

     /**
     *
     * @element-type Proposta
     */
   public Vector proposta;

   public void ferllistaVendes(int any) {
   }
}


import java.lang.String;
import java.util.Vector;

public class Client implements Comunicar {
   /* {src_lang=Java}*/

   private int clientID;
   /* {transient=false, volatile=false}*/

   private String nom;
   /* {transient=false, volatile=false}*/

   private String raoSocial;
   /* {transient=false, volatile=false}*/

   private String nif;
   /* {transient=false, volatile=false}*/

   private String adreca;
   /* {transient=false, volatile=false}*/

     public ContacteComercial myContacteComercial;
     public ContacteComercial contacte;
```

ANOTACIONS

```
    public Vector ferllistaFactures() {
    return null;

    }

}


import java.lang.String;
import java.util.Vector;

public class ContacteComercial {
    /* {src_lang=Java}*/

    private int contacteID;
    /* {transient=false, volatile=false}*/

    private String nom;
    /* {transient=false, volatile=false}*/

    private String cognoms;
    /* {transient=false, volatile=false}*/

    private String adreca;
    /* {transient=false, volatile=false}*/

    private String comentari;
    /* {transient=false, volatile=false}*/

    private String estat;
    /* {transient=false, volatile=false}*/

      public Client client;
      /**
      *
      * @element-type AccioComercial
      */
    public Vector accio;

    public void esborrar() {
    }

    public void canviarEstat(String nouEstat) {
    }

}
```

Veiem que ha generat codi per a les associacions i ha respectat correctament la multiplicitat, perquè ha creat un vector per a la multiplicitat i una instància per a la resta.

A causa de la qualitat i la popularitat d'aquest programari en la seva àrea d'especialització (el modelatge de disseny), han sorgit versions comercials millorades per altres fabricants de programari, entre les quals destaca PoseidonUML (http://www.gentleware.com/products/),

ANOTACIONS

que implementa molt millor una generació de codi i una integració amb entorns de desenvolupament com Eclipse.

ArgoUML també incorpora funcionalitat per a importar fitxers font (només en Java) al model, de manera que puguem aprofitar aquestes classes en els diferents diagrames que suporta.

2.14. Resum

L'anàlisi i disseny de programari té un paper crucial en qualsevol desenvolupament, però és en la programació orientada a l'objecte on les activitats relacionades amb aquesta fase d'un projecte han assolit les seves quotes més altes de sofisticació. És, a més, una àrea en què contínuament es produeixen avenços i propostes noves.

En aquest capítol, hem començat mostrant als estudiants una visió general del paradigma de programació orientada a l'objecte en la qual hem repassat des dels conceptes més bàsics fins a arribar a tots els tipus de relacions entre objectes que suporten la majoria de llenguatges de programació.

Aquesta introducció ha estat necessària per a poder centrar-nos a continuació en el llenguatge de modelatge UML i en allò que aquest pot aportar durant totes les fases del cicle de vida d'un projecte. Cada un dels diagrames s'ha estudiat detalladament a partir d'un cas pràctic.

Encara que la majoria dels diagrames d'UML ens permeten millorar la comunicació amb el client en les fases inicials, i també documentar i explorar funcionalitats i aspectes concrets de les classes del nostre sistema durant la seva anàlisi i disseny, també hem vist que UML ens pot ajudar en les fases de desenvolupament i implantació.

Hem dedicat el darrer apartat a la generació de codi mitjançant les tres mateixes eines de codi obert que hem utilitzat en el material per a generar els diagrames, amb la qual cosa hem demostrat que UML en particular i el modelatge en general és una pràctica molt bona en qualsevol projecte, no solament per la disciplina que ens imposa en

ANOTACIONS

el seu disseny, sinó també per l'estalvi de temps en el desenvolupament que pot aportar.

Per tot això, creiem que amb la lectura d'aquest capítol els estudiants haureu aconseguit els objectius plantejats al principi.

Aquest material no pretén ser la referència última d'UML per als estudiants, sinó una visió global d'aquest estàndard de modelatge, una motivació perquè exploreu les referències que us proporcionem i un incentiu perquè poseu en pràctica els coneixements i bones pràctiques que hem indicat en els pròxims projectes en què participeu.

2.15. Altres fonts de referència i informació

ArgoUML. http://argouml.tigris.org/

Birtwistle, G.M. (Graham M.) 1973. *SIMULA begin*. Philadelphia, Auerbach

Dia. http://www.gnome.org/projects/dia/

Eckel, B. (2003). *Thinking in Java* (3.ª ed.). Upper Saddle River: Prentice Hall. http://www.mindview.net/Books/TIJ/

Gamma, E.; Helm; R. y otros (1995). *Design Patterns*. Reading (Mass.): Addison Wesley.

Java Technology. http://java.sun.com

Microsoft.NET. http://www.microsoft.com/net/

Object Management Group. http://www.omg.org/

Rumbaugh, J. y otros (2004). *Object-Oriented Modeling and Design with UML* (2nd Edition). Englewood Cliff (N.J.): Prentice Hall.

Smalltalk. http://www.smalltalk.org

ANOTACIONS

Stroustrup, B. (2000). *The C++ Programming Language* (3º edition). Reading (Mass.): Addison Wesley.

The Open Source Java Directory. http://www.onjava.com/pub/q/java_os_directory

Umbrello. http://uml.sourceforge.net

UML1.5 The Current Official Version. http://www.uml.org/#UML1.5

Unified Modeling Language. http://www.uml.org/

Wikipedia. http://www.wikipedia.org/

ANOTACIONS

3. Control de qualitat i proves

3.1. Introducció

La qualitat del programari és una preocupació a la qual es dediquen molts esforços. Tanmateix, el programari gairebé mai no és perfecte. Tot projecte té com a objectiu produir el programari de la millor qualitat possible, que compleixi –i si pot ser superi– les expectatives dels seus usuaris. Hi ha literatura abundant sobre els processos de qualitat de programari i, actualment, es realitza una investigació acadèmica considerable en aquest camp dins l'enginyeria del programari.

En aquest capítol, intentarem donar una visió pràctica dels controls de qualitat i de proves en el cas particular, tant de pràctiques com d'aplicacions, del programari lliure, i veurem quins són els principis, tècniques i aplicacions més importants que s'utilitzen per a garantir la qualitat dels productes lliures.

3.2. Objectius

Els materials didàctics associats amb aquest mòdul us permetran obtenir els coneixements següents:

- Familiaritzar-vos amb la terminologia relacionada amb el control de qualitat i proves que és d'ús comú en l'enginyeria del programari.
- Conèixer les tècniques principals de comprovació manual de programari usades en l'enginyeria del programari i en entorns lliures.
- Conèixer les tècniques principals de comprovació automàtica de programari usades en enginyeria del programari i el programari lliure que s'hi utilitza.

ANOTACIONS

- Conèixer els sistemes de comprovació d'unitats de programari i entendre com funcionen.
- Conèixer els sistemes de gestió d'errors, l'anatomia d'un error, el seu cicle de vida, i familiaritzar-vos amb el sistema de gestió d'errors lliure Bugzilla.

3.3. Control de qualitat i proves

Lectura recomanada

M.A. Cusumano. *Preliminary Data from Global Software Process Survey*.

Qualsevol usuari o desenvolupador sap, per experiència pròpia, que els programes no són perfectes. Els programes tenen errors. Com millor sigui el procés d'enginyeria del programari i el procés de control de qualitat i proves que s'utilitzi, menys errors tindrà. El programari té una mitjana de 0,150 errors per cada mil línies de codi. Si tenim en compte que un producte com OpenOffice.org 1.0 té aproximadament set milions de línies de codi, l'aritmètica és senzilla.

Avui dia, qualsevol projecte de programari integra dins el seu cicle de desenvolupament processos de control de qualitat i proves. No hi ha una única pràctica acceptada unànimement, ni en el món acadèmic ni en l'empresarial, per a dur a terme aquests processos. Tampoc no hi ha cap mètode que es consideri més bo que els altres. Cada projecte de programari utilitza la combinació de mètodes que s'adapta millor a les seves necessitats.

Els projectes que no integren un control de qualitat com a part del seu cicle de desenvolupament a la llarga tenen un cost més alt. Això és degut al fet que com més avançat està el cicle de desenvolupament d'un programa, més costós resulta solucionar-ne els errors. Es requereix llavors un nombre més elevat d'hores de treball dedicades a solucionar aquests errors i, a més, les seves repercussions negatives seran més greus. Per això, sempre que iniciem el desenvolupament d'un projecte nou, hauríem d'incloure un pla de gestió de qualitat.

Per a poder dur a terme un control de qualitat, és imprescindible haver definit els requisits del sistema de programari que implementarem, ja que són necessaris per a disposar d'una especificació del

ANOTACIONS

propòsit del programari. Els processos de qualitat verifiquen que el programari compleix amb els requisits que hem definit originàriament.

3.3.1. Termes comuns

Les tècniques i sistemes de control de qualitat i proves tenen una terminologia molt específica per a referir-se a conceptes singulars d'aquest tipus de sistemes. A continuació, veurem els conceptes i termes més habituals:

- **Error** (*bug*). Error en la codificació o disseny d'un sistema que provoca que el programa no funcioni correctament o falli.

- **Abast del codi** (*code coverage*). Procés per a determinar quines parts d'un programa mai no són utilitzades o quines no són provades mai com a part del procés de proves.

- **Prova d'estrès** (*stress testing*). Conjunt de proves que tenen com a objectiu mesurar el rendiment d'un sistema amb càrregues de treball elevades. Per exemple, si una determinada aplicació web és capaç de satisfer amb uns estàndards de servei acceptables un nombre concret d'usuaris simultàniament.

- **Prova de regressió** (*regression testing*). Conjunt de proves que tenen com a objectiu comprovar que la funcionalitat de l'aplicació no ha estat danyada per canvis que hem realitzat recentment. Per exemple, si hem afegit una funcionalitat nova a un sistema o hem corregit un error, comprovarem que aquestes modificacions no han danyat la resta de funcionalitats del sistema.

- **Prova d'usabilitat** (*usability testing*). Conjunt de proves que tenen com a objectiu mesurar la usabilitat d'un sistema per part dels seus usuaris. En general, se centra a determinar si els usuaris d'un sistema són capaços d'aconseguir els seus objectius amb el sistema que és objecte de la prova.

- **Testador** (*tester*). Persona encarregada de realitzar un procés de proves, manuals o automàtiques, d'un sistema i reportar-ne els possibles errors.

ANOTACIONS

3.3.2. Principis de la comprovació de programari

Sigui quina sigui la tècnica o conjunt de tècniques que utilitzem per a assegurar la qualitat del nostre programari, hi ha un conjunt de principis que hem de tenir sempre presents. A continuació, enumerem els principals:

- **És imperatiu disposar d'uns requisits que detallin el sistema**. Els processos de qualitat es basen en el fet de verificar que el programari compleix els requisits del sistema. Sense uns requisits que descriguin el sistema de manera clara i detallada, és impossible crear un procés de qualitat amb unes garanties mínimes.

- **Els processos de qualitat han de ser integrats en les primeres fases del projecte**. Els processos de control de qualitat del sistema n'han de ser part integral des del començament. Realitzar els processos de proves quan el projecte està en una fase avançada del seu desenvolupament o a prop del final és una mala pràctica en enginyeria del programari. Per exemple, en el cicle final de desenvolupament, és molt costós corregir errors de disseny. Com més aviat detectem un error en el cicle, més econòmic serà corregir-lo.

- **Qui desenvolupa un sistema no ha de ser qui en prova la funcionalitat**. La persona o grup de persones que desenvolupen un sistema no han de ser en cap cas les mateixes que són responsables de realitzar el control de proves, de la mateixa manera que en altres disciplines, com per exemple l'escriptura, en què els escriptors no corregeixen els seus propis textos. És important recordar que sovint es produeixen errors en la interpretació de l'especificació del sistema i una persona que no ha estat involucrada en el desenvolupament del sistema pot avaluar més fàcilment si la seva interpretació ha estat correcta o no.

Cal tenir en compte tots aquests principis bàsics perquè incomplir-ne algun es tradueix en la impossibilitat de poder garantir la correcció dels sistemes de control de qualitat que apliquem.

ANOTACIONS

3.4. Tècniques manuals de comprovació de programari

Les tècniques manuals de comprovació de programari són un conjunt de mètodes usats àmpliament en els processos de control de proves. En la versió més informal, consisteixen en un grup de testadors que instal·len una aplicació i, sense cap pla predeterminat, la utilitzen. Després reporten els errors que van trobant mentre l'usen perquè siguin solucionats.

En la versió més formal d'aquest mètode, s'utilitzen guions de proves, que són petites guies d'accions que un testador ha d'efectuar i dels resultats que ha d'obtenir si el sistema funciona correctament. És habitual en molts projectes que, abans d'alliberar una versió nova del projecte, s'hagi de superar satisfactòriament un conjunt d'aquests guions de proves.

La majoria de guions de proves tenen com a objectiu assegurar que la funcionalitat més comuna del programa no ha estat afectada per les millores introduïdes des de l'última versió i que un usuari mitjà no trobarà cap error greu.

Exemple

Aquest és un exemple de guió de proves del projecte OpenOffice.org que n'il·lustra l'ús.

Àrea de la prova: Solaris - Linux - Windows
Objectiu de la prova
Verificar que OpenOffice.org obre un fitxer .jpg correctament
Requeriments:
Un fitxer .jpg és necessari per a poder efectuar aquesta prova

Accions:

Descripció:

1) Des de la barra de menús, seleccioni File-Open.

ANOTACIONS

2) La caixa de diàleg d'obertura de fitxers s'ha de mostrar.

3) Introdueixi el nom del fitxer .jpg i seleccioni el botó *Open*.

4) Tanqui el fitxer gràfic.

Resultat esperat:
OpenOffice.org ha de mostrar una finestra nova que mostri el fitxer gràfic.

Font: exemple de guió de proves extret del projecte OpenOffice.org. http://qa.openoffice.org/.

Com veiem, es tracta d'una descripció d'accions que el testador ha de seguir, juntament amb el resultat esperat per a aquestes accions.

Encara que aquest sistema avui dia s'usa àmpliament en combinació amb altres sistemes, confiar el control de qualitat únicament a un procés de proves manuals és molt arriscat i no ofereix garanties sòlides de la qualitat del producte que hem produït.

3.5. Tècniques automàtiques de comprovació de programari

3.5.1. White-box testing

El *white-box testing* (prova de caixa blanca) és un conjunt de tècniques que tenen com a objectiu validar la lògica de l'aplicació. Les proves se centren a verificar l'estructura interna del sistema sense tenir en compte els requisits d'aquest.

Hi ha diversos mètodes en aquest conjunt de tècniques, els més habituals són la inspecció del codi de l'aplicació per part d'altres desenvolupadors amb l'objectiu de trobar errors en la lògica del programa, codi que no s'utilitza (*code coverage,* en anglès) o l'estructura interna de les dades. Hi ha també aplicacions propietàries d'ús estès que permeten automatitzar tot aquest tipus de proves, com ara PureCoverge de Rational Software.

ANOTACIONS

3.5.2. Black-box testing

El *black-box testing* (prova de caixa negra) consisteix a comprovar la funcionalitat dels components d'una aplicació. Determinades dades d'entrada a una aplicació o component han de produir uns resultats determinats. Aquest tipus de prova està dirigida a comprovar els resultats d'un component de programari, no a validar com ha estat estructurat internament. S'anomena *black box* (caixa negra) perquè el procés de proves assumeix que es desconeix l'estructura interna del sistema, només es comprova que les dades de sortida produïdes per una entrada determinada són correctes.

Imaginem que tenim un sistema orientat a l'objecte en què hi ha una classe de manipulació de cadenes de text que permet concatenar cadenes, obtenir-ne la longitud i copiar fragments en altres cadenes. Podem crear una prova que consisteixi a realitzar diverses operacions amb cadenes predeterminades: concatenar-les, mesurar-ne la longitud, fragmentar-les, etc. Sabem quin és el resultat correcte d'aquestes operacions i podem comprovar la sortida d'aquesta rutina. Cada vegada que executem la prova, la classe obté com a entrades aquestes cadenes conegudes i comprova que les operacions realitzades han estat correctes.

Aquest tipus de testos és molt útil per a assegurar que a mesura que el programari evoluciona no es trenca la funcionalitat bàsica d'aquest.

3.5.3. Unitats de comprovació

Quan treballem en projectes d'una certa dimensió, necessitem tenir procediments que assegurin la funcionalitat correcta dels diferents components del programari, i molt especialment dels que formen la base del nostre sistema. La tècnica de les unitats de comprovació, *unit testing* en anglès, es basa en la comprovació sistemàtica de classes o rutines d'un programa per mitjà d'unes dades d'entrada verificant que els resultats generats són els esperats. Avui dia, les unitats de comprovació són la tècnica de caixa negra més estesa.

ANOTACIONS

Una unitat de prova ben dissenyada ha de complir els requisits següents:

- **S'ha d'executar sense atenció de l'usuari (desatesa)**. Una unitat de proves ha de poder ser executada sense cap intervenció de l'usuari: ni en la introducció de les dades ni en la comprovació dels resultats que té com a objectiu determinar si la prova s'ha executat correctament.

- **Ha de ser universal**. Una unitat de proves no pot assumir configuracions particulars o basar la comprovació de resultats en dades que poden variar d'una configuració a l'altra. Ha de ser possible executar la prova en qualsevol sistema que tingui el programari que és objecte de la prova.

- **Ha de ser atòmica**. Una unitat de prova ha de ser atòmica i ha de tenir com a objectiu comprovar la funcionalitat concreta d'un component, rutina o classe.

Dos dels entorns de comprovació més populars en entorns de programari lliure amb JUnit i NUnit. Ambdós entorns proporcionen la infraestructura necessària per a poder integrar les unitats de comprovació com a part del nostre procés de proves. JUnit està pensat per a aplicacions desenvolupades en entorn Java i NUnit per a aplicacions desenvolupades en entorn .Net. És habitual l'ús del terme xUnit per a referir-se al sistema de comprovació independentment del llenguatge o entorn que utilitzem.

Exemple

L'exemple següent és un fragment de la unitat de comprovació de la classe *StringFormat* del *namespace System.Drawing* del projecte Mono que és executat amb el sistema Nunit de comprovació d'unitats.

```
namespace MonoTests.System.Drawing
{

   [TestFixture]
   public class StringFormatTest
   {
      [Test]
      public void TestClone()
      {
```

ANOTACIONS

```
        StringFormat smf = new StringFormat();
        StringFormat smfclone = (StringFormat) smf.Clone();
        Assert.AreEqual (smf.LineAlignment, smfclone.LineAlignment);
        Assert.AreEqual (smf.FormatFlags, smfclone.FormatFlags);
        Assert.AreEqual (smf.LineAlignment, smfclone.LineAlignment);
        Assert.AreEqual (smf.Alignment, smfclone.Alignment);
        Assert.AreEqual (smf.Trimming, smfclone.Trimming);
    }
}
```

Fragment de la unitat de comprovació de la classe *StringFormat* del projecte Mono.

El mètode *TestClone* té com a objectiu comprovar la funcionalitat del mètode *Clone* de la classe *StringFormat*. Aquest mètode fa una còpia exacta de l'objecte. Les proves es basen a assegurar que les propietats de l'objecte han estat copiades correctament.

3.6. Sistemes de control d'errors

Els sistemes de control d'errors són eines col·laboratives molt dinàmiques que proveeixen el suport necessari per a consolidar una peça clau en la gestió del control de qualitat: l'emmagatzemament, seguiment i classificació dels errors d'una aplicació de manera centralitzada. L'ús d'aquests sistemes en la majoria de projectes s'estén també a la gestió de millores sol·licitades per l'usuari i idees per a versions noves i alguns usuaris els utilitzen, fins i tot, per a tenir un control de les crides d'un suport tècnic, ordres, o els utilitzen com a gestors de tasques que cal realitzar.

3.6.1. Report d'errors

Quan un usuari troba un error, una bona manera de garantir que serà bon candidat a ser corregit és escriure un informe d'error tan precís i detallat com pugui.

ANOTACIONS

Independentment de quin sigui el sistema de control d'errors que utilitzem, els consells següents reflecteixen les bones pràctiques en el report d'errors:

- **Comprovar que l'error no ha estat reportat anteriorment**. Sol ocórrer amb freqüència que errors que els usuaris troben sovint són reportats més d'una vegada. Això causa que el nombre d'errors totals pendents de corregir augmenti i que el nombre total d'errors registrats no reflecteixi la realitat. Aquests tipus d'errors es diuen *duplicats* i solen ser eliminats així que són detectats. Abans de reportar un error nou, és important fer una recerca en el sistema de control d'errors per a comprovar que no ha estat reportat anteriorment.

- **Donar un bon títol a l'informe d'error**. El títol de l'informe d'error ha de ser molt descriptiu, ja que això ajudarà els desenvolupadors a poder valorar l'error quan en facin llistes i també ajudarà els altres usuaris a trobar errors ja reportats i evitar així possibles duplicats.

  Per exemple, un títol del tipus "El programa es penja" seria un exemple d'un mal títol per a descriure un error perquè és molt poc descriptiu. Tanmateix, "El programa es penja en la previsualització d'imatges" és molt més precís i descriptiu.

- **Descriure detalladament i pas a pas com reproduir l'error**. És important que l'error sigui determinista i que puguem donar una guia pas a pas de com reproduir-lo perquè la persona que l'ha de corregir ho pugui fer fàcilment. Sempre hem de tenir present que la claredat en la descripció de l'error facilita el treball de tots els involucrats en el procés.

- **Donar la informació màxima sobre la nostra configuració i les seves particularitats**. La majoria de sistemes de report d'error disposen de camps que permeten especificar la nostra configuració (tipus d'ordinador, sistema operatiu, etc.). Hem de donar la informació màxima sobre les particularitats de la nostra configuració, ja que aquestes dues dades sovint són imprescindibles per a solucionar un error.

ANOTACIONS

- **Reportar un únic error per informe**. Cada informe d'error que efectuem ha de contenir la descripció d'un únic error. Això facilita que l'error pugui ser tractat com una unitat per l'equip de desenvolupament. Molt sovint, mòduls diferents d'un programa estan sota la responsabilitat de desenvolupadors diferents.

Seguir tots aquests consells millora la claredat dels nostres informes d'error i ens ajuda a assolir l'objectiu final de qualsevol report d'error, que és aconseguir que aquest pugui ser solucionat amb la màxima rapidesa i amb el cost més baix possible.

3.6.2. Anatomia d'un informe d'error

Sigui quin sigui el sistema de control d'errors que usem, tots els sistemes detallen l'error amb una sèrie de camps que permeten definir-lo i classificar-lo de manera precisa. A continuació, descrivim els camps principals que formen part d'un informe d'error:

- **Identificador**. Codi únic que identifica l'informe d'error de manera inequívoca i que assigna el sistema automàticament. Normalment, els identificadors solen ser numèrics, la qual cosa permet que ens referim a l'error pel seu número, com per exemple l'error 3210.

- **Títol**. Frase de cinquanta caràcters aproximadament que descriu l'error de manera sintètica i li dóna un títol.

- **Informador**. Persona que envia l'informe d'error. Normalment, un usuari registrat en el sistema de control d'errors.

- **Data d'entrada**. Data en què l'error va ser registrat en el sistema de control d'errors.

- **Estat**. Situació en què es troba l'error. En l'apartat de cicle de vida d'un error, veurem els diferents estats pels quals pot passar un error.

ANOTACIONS

- **Gravetat**. Indica la gravetat de l'error dins el projecte. Hi ha diferents categories definides que inclouen des d'un error trivial fins a un error que pot ser considerat greu.

- **Paraules clau**. Conjunt de paraules clau que podem usar per a definir l'error. Per exemple, podem usar *fallada* per a indicar que és un error que té com a conseqüència que el programa es pengi. Això és molt útil perquè després permet als desenvolupadors fer llistes o recerques per mitjà d'aquestes paraules clau.

- **Entorn**. Normalment hi ha un o més camps que permeten definir l'entorn i configuració de l'usuari en què es produeix l'error. Alguns sistemes, com Bugzilla, tenen camps que permeten especificar la informació separadament, com per exemple: sistema operatiu, arquitectura del sistema, etc.

- **Descripció**. Una descripció de l'error i com reproduir-lo. Alguns sistemes permeten que usuaris diferents puguin ampliar la informació de l'error a mesura que el cicle de vida de l'error avança amb detalls relacionats amb la seva possible solució, la seva implicació en altres errors o, simplement, per a concloure que l'error ha estat solucionat.

Aquests són els camps que podem considerar els mínims comuns en tots els sistemes de control d'errors. Cada sistema afegeix camps addicionals propis per a ampliar aquesta informació bàsica o per a permetre realitzar funcions avançades pròpies.

3.6.3. Cicle de vida d'un error

El cicle de vida d'un error comença quan és reportat i acaba quan l'error se soluciona. Entre aquests extrems del cicle de vida, hi ha una sèrie d'estats pels quals passa l'error.

Encara que no hi ha un estàndard definit entre els diferents sistemes de control d'errors, utilitzarem la terminologia que utilitza el

ANOTACIONS

sistema Bugzilla, ja que és la més estesa dins el món del programari lliure.

Aquests són els estats principals en què es pot trobar un error durant el seu cicle de vida amb Bugzilla:

- **Sense confirmar**. Es produeix quan l'informe d'error ha estat introduït en el sistema. És l'estat inicial pel qual passa qualsevol error.

- **Nou**. L'error estava en estat "Sense Confirmar", se n'ha comprovat la validesa i ha passat a l'estat "Nou", que representa que ha estat acceptat com a error formalment.

- **Assignat**. L'error ha estat assignat a un desenvolupador perquè el solucioni. Ara hem d'esperar que el desenvolupador tingui temps per a poder avaluar l'error i solucionar-lo.

- **Corregit**. L'error ha estat corregit i hauria d'estar solucionat.

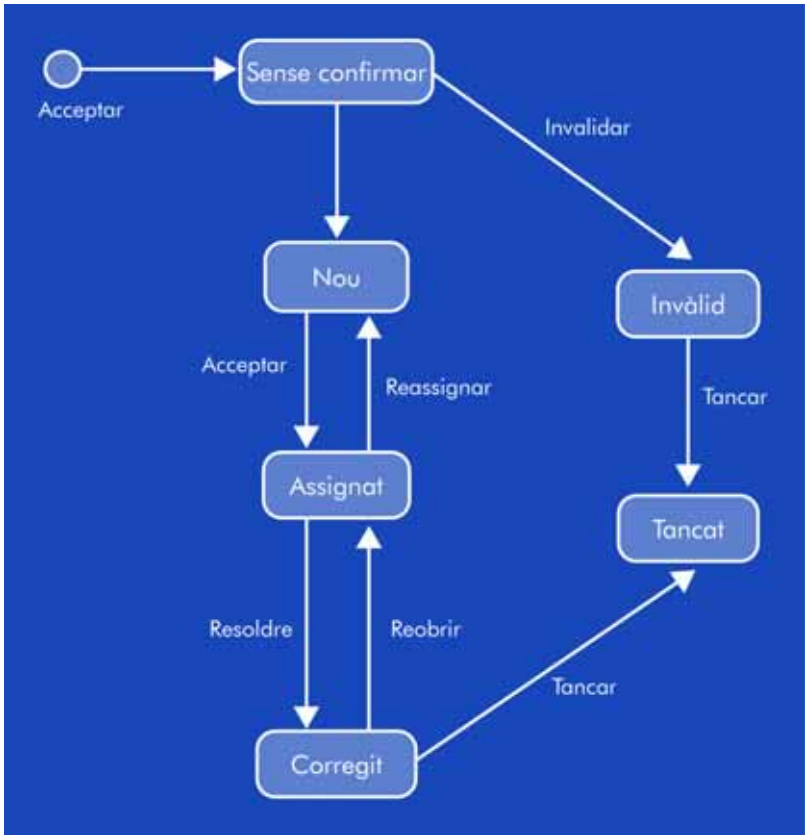
- **Tancat**. L'error ha estat corregit i s'ha confirmat que la solució és correcta. Aquest és el final de cicle de vida de l'error.

- **Invàlid**. L'error no era un error vàlid o simplement no era un error. Per exemple, quan es descriu una funcionalitat que és correcta.

A continuació, veurem un esquema que il·lustra els estats principals. Bugzilla i altres sistemes disposen d'estats addicionals en què es pot trobar un error durant el seu cicle de vida.

ANOTACIONS

Figura 1. Estats principals en què es pot trobar un error durant el seu cicle de vida



Podem observar que quan un error entra en el sistema queda classificat com a "Sense Confirmar". Una vegada que és revisat, es pot descartar com a error i es pot marcar com a "Invàlid" per a després "Tancar" l'error. Si l'error és acceptat com a "Nou", el pas següent és assignar-lo a un desenvolupador, que és un usuari en el sistema de control d'errors. Una vegada assignat, l'error és corregit, i quan s'ha verificat, es tanca.

3.6.4. Bugzilla

Lectura recomanada

http://www.bugzilla.org/

Un aspecte central en qualsevol projecte de programari és gestionar i seguir els errors. Quan Netscape va alliberar el codi de Mozilla el 1998, va trobar la necessitat que tingués una aplicació de gestió d'errors per mitjà del web que permetés la interacció entre usuaris i desenvolupadors. Van decidir adaptar l'aplicació que usaven internament en Netscape a les necessitats d'un projecte obert i així va néixer Bugzilla. Inicialment, va ser el sistema de ges-

tió i seguiment d'errors del projecte Mozilla, però amb el temps ha estat adoptat per molts projectes lliures, incloent-hi KDE i GNOME, entre d'altres. Bugzilla permet als usuaris enviar errors i facilita classificar-los, assignar-los a un desenvolupador perquè els resolgui i fer tot el seguiment de les incidències relacionades.

Entre les característiques que proveeix Bugzilla, destaquen les següents:

- **Interfície web**. Una interfície web completa que permet que qualsevol persona amb un simple navegador ens pugui enviar un error o administrar el sistema.

- **Entorn col·laboratiu**. Cada report d'error es converteix en un fil de discussió en què usuaris, provadors i desenvolupadors poden discutir sobre les implicacions d'un error, el seu possible origen o com corregir-lo.

- **Notificació per correu**. Bugzilla permet notificar als usuaris diversos esdeveniments per correu, com per exemple informar els usuaris involucrats en un error concret de canvis en aquest.

- **Sistema de recerques**. El sistema de recerques és molt complet: ens permet buscar errors pel seu tipus, per qui va informar d'ells, per la prioritat que tenen o per la plataforma en què han aparegut.

- **Informes**. Té integrat un sistema complet que permet generar informes sobre l'estat dels errors del projecte.

- **Votacions**. Bugzilla permet votar els errors de manera que es puguin establir prioritats per a resoldre'ls basant-se en els vots que han rebut dels usuaris.

- **Assegurança**. Preveu el registre d'usuaris, nivells de seguretat diferents i una arquitectura dissenyada per a preservar la privadesa dels usuaris.

ANOTACIONS

Figura 2. Captura de pantalla del sistema de control d'errors Bugzilla

3.6.5. Gnats

Gnats (*GNU problem report management system*) és l'eina de gestió d'errors per a entorns Unix desenvolupada per la Free Software Foundation al començament dels anys noranta. És més antiga que Bugzilla. Gnats és usat per projectes com FreeBSD o Apache i, lògicament, pels projectes impulsats per la mateixa Free Software Foundation.

El nucli de Gnats és un conjunt d'eines d'indicador d'ordres que exposen tota la funcionalitat del sistema. Té tots els avantatges i la flexibilitat de les eines GNU, però el seu ús pot resultar difícil als qui no estiguin familiaritzats amb aquest tipus d'eines. Hi ha també una interfície web que va ser desenvolupada posteriorment i és el processador d'accés (*front-end*) més usat avui dia.

ANOTACIONS

Figura 3. Captura de pantalla del sistema de control d'errors Gnat

3.7. Conclusions

Durant aquest capítol, hem constatat la necessitat d'integrar els processos de control de qualitat i proves dins el cicle de desenvolupament d'un programa des de l'inici d'aquest. Hem vist els principis que regeixen la comprovació del programari, com s'escriu un bon informe d'error i una unitat de comprovació.

També hem assolit una visió pràctica dels sistemes de control de qualitat i proves en el cas particular, tant de pràctiques com d'aplicacions, del programari lliure, i hem vist quins són els principis, tècniques i aplicacions més importants que s'utilitzen per a garantir la qualitat dels productes lliures.

ANOTACIONS

3.8. Altres fonts de referència i informació

Lewis, W. (2000). *Software Testing and Continuous Quality Improvement*. CRC Press LLC.

Meyers, G. (2004). *The Art of Software Testing*. John Wiley & Sons.

ANOTACIONS

4. Construcció de programari en entorn GNU

4.1. Introducció

El sistema operatiu GNU/Linux ha significat una autèntica revolució en el món del programari lliure. Com a conseqüència de la seva expansió, el desenvolupament de programari en entorns GNU s'ha multiplicat per deu l'últim any (2004). Gran quantitat de companyies que desenvolupaven els seus productes en entorn propietari els migren a plataformes lliures, la qual cosa demana un gran nombre de programadors en un entorn molt diferent de l'habitual en el programari propietari.

4.2. Objectius

Amb aquest capítol, volem que els programadors assoliu els objectius següents:

Introduir-vos en l'entorn de desenvolupament GNU guiant-vos per a instal·lar els components necessaris per a començar a desenvolupar, per a crear els fitxers de configuració i per a reconèixer els fitxers generats després d'una compilació, per a mantenir aquests fitxers, per a documentar el codi i els seus canvis, i també per a usar les biblioteques i les utilitats que hi estan relacionades.

4.3. Instal·lar tot el que és necessari. Autotools

Les autotools són un conjunt d'utilitats per a automatitzar la compilació de programes en diferents entorns. Permeten al desenvolupador especificar unes regles generals per a la compilació i que la resta de la configuració sigui detectada en la màquina destinació de manera que es simplifiqui la tasca de portabilitat entre sistemes i en general la creació de fitxers `Makefile` d'un projecte.

ANOTACIONS

En realitat, es componen de dos paquets:

http://directory.fsf.org/GNU/autoconf.html
http://directory.fsf.org/GNU/automake.html

Ambdós estan interrelacionats entre ells i necessiten un intèrpret de Perl i de macros m4.

L'estructura de funcionament és la següent:

1) `automake` tradueix un fitxer de configuració `Makefile.am` i genera un `Makefile.in`.

2) `automake` processa un `configure.in` (o `configure.ac`) i genera `./configure`.

3) En la màquina de destinació, s'executa `./configure`, que genera els `Makefile` i opcionalment altres fitxers, possiblement en tots els directoris que calgui.

4.3.1. Fitxer de configuració d'usuari (entrada)

Un projecte de programari constarà d'una sèrie de fitxers amb codi font i una sèrie de passos que cal fer per a convertir-los en programa executable.

Aquests passos, amb autotools, es divideixen en dues parts principals:

1) D'una banda, un fitxer `Makefile.am` a partir del qual es crearà el `Makefile` final en què es basarà la compilació del codi font.

2) D'una altra banda, un fitxer `configure.in` o `configure.ac` que servirà de plantilla per a crear un fitxer de seqüència d'intèrpret d'ordres amb els passos necessaris per a detectar i establir la configuració necessària perquè el programa sigui compilat i instal·lat sense problemes. També serà l'encarregat de crear els `Makefile` necessaris seguint el `Makefile.am` inicial.

ANOTACIONS

Tanmateix, fins i tot aquests passos es poden automatitzar amb la utilitat autoscan, que generarà un `configure.ac` preliminar, amb el nom de `configure.scan`, basant-se en els fitxers de codi font que trobi en el directori (i subdirectoris) en què s'executi.

Vegem aquest funcionament amb un programa d'exemple extremadament simple:

nul.c

```
void main() {}
```

Creem un `Makefile.am` senzill:

Makefile.am

```
bin_PROGRAMS = nul
nul_SOURCES = nul.c
```

En aquest exemple, només crearem un binari `nul` a partir del fitxer de codi font `nul.c`.

Simplement executant `autoscan` obtindrem un fitxer `configure.ac` base:

configure.scan

```
AC_PREREQ(2.59)
AC_INIT(FULL-PACKAGE-NAME, VERSION, BUG-REPORT-ADDRESS)
AC_CONFIG_SRCDIR([nul.c])
AC_CONFIG_HEADER([config.h])

# Checks for programs.
AC_PROG_CC

AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

És recomanable editar aquest fitxer per a adequar-lo al projecte que desenvolupem:

configure.ac

```
AC_INIT(nul, 0.1)
AM_INIT_AUTOMAKE(nul, 0.1)
AC_CONFIG_SRCDIR([nul.c])

# Checks for programs.
AC_PROG_CC

AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

ANOTACIONS

El significat d'aquest fitxer seria:

- Inicialitzar `automake` (amb el nom i versió del programa que cal compilar).
- Inicialitzar `automake` (una altra vegada amb nom i versió).
- Establir el directori origen com el de `nul.c`.
- Buscar un compilador de llenguatge C.
- Establir fitxers de configuració `Makefile`.
- Generar els fitxers de sortida.

4.3.2. Fitxers generats (sortida)

Abans de poder generar els fitxers de sortida definitius, haurem de preconfigurar les macros que usaran les autotools, i ja podrem executar `autoconf`:

```
$ aclocal
$ autoconf
$ ls -l
-rw-r--r--  1 user user  37129 nov 17 18:03 aclocal.m4
drwxr-xr-x  2 user user   4096 nov 17 18:03 autom4te.cache/
-rwxr-xr-x  1 user user 103923 nov 17 18:03 configure*
-rw-r--r--  1 user user    145 nov 17 18:03 configure.ac
-rw-r--r--  1 user user     39 nov 17 18:03 Makefile.am
-rw-r--r--  1 user user     15 nov 17 18:03 nul.c
```

Podria semblar que ja està creat `./configure`; tanmateix, encara hem de preparar les macros d'`automake` perquè funcioni:

```
$ automake --add-missing
automake: configure.ac: installing `./install-sh'
automake: configure.ac: installing `./mkinstalldirs'
automake: configure.ac: installing `./missing'
automake: configure.ac: installing `./config.guess'
automake: configure.ac: installing `./config.sub'
automake: Makefile.am: installing `./INSTALL'
automake: Makefile.am: required file `./NEWS' not found
automake: Makefile.am: required file `./README' not found
automake: Makefile.am: installing `./COPYING'
automake: Makefile.am: required file `./AUTHORS' not found
automake: Makefile.am: required file `./ChangeLog' not found
```

ANOTACIONS

Com es pot observar, `automake` ha instal·lat una sèrie de seqüències necessàries per a executar `./configure` en conjunció amb `automake`, i a més ha creat dos fitxers `INSTALL` i `COPYING` i informa que en falten quatre més.

Vegem què són cada un d'aquests fitxers:

- `INSTALL`: aquest fitxer descriu el procés d'instal·lació. `automake` crea per defecte un fitxer amb una guia d'instal·lació bàsica, però serà recomanable editar-lo per a adequar-lo a les necessitats de cada projecte en concret.
- `COPYING`: especifica la llicència amb què es permet o restringeix fer còpies del programari i del codi font del projecte.
- `NEWS`: una llista de les modificacions ("novetats") del projecte. Sol anar ordenat amb les últimes entrades al principi i de manera decreixent segons les versions.
- `README`: aquest fitxer és històricament el primer que mira l'usuari. Sol incloure una descripció curta i potser recomanacions sobre l'ús i altres detalls relatius al paquet. També advertències que puguin evitar problemes a l'usuari.
- `AUTHORS`: recull una llista de les persones que han intervingut en el desenvolupament del projecte. Se solen incloure les adreces de correu i possiblement altres formes de contacte, tant per a aspectes legals de propietat intel·lectual com per a consultes, suggeriments i informes d'errors, i també com a reconeixement merescut de cada un.
- `ChangeLog`: un dels fitxers més importants que permetrà conèixer els canvis, les actualitzacions i l'estat en general del projecte de programari (us oferim informació addicional en una secció posterior).

Continuant amb l'exemple simple, per ara emplenarem aquests fitxers amb dades simples:

```
$ echo '2004-11-17: Projecte creat' > NEWS
$ echo 'No fa res. Sintaxis: nul' > README
$ echo 'David Aycart <david@aycart.com>' > AUTHORS
$ cat <<END > ChangeLog
2004-11-17  David Aycart  <david@aycart.com>
       * nul.c: creat
END
```

ANOTACIONS

Ara ja podrem executar `automake` sense que doni error:

```
$ automake --add-missing
$ ls
aclocal.m4      config.guess@   configure.ac  Makefile.am   NEWS
AUTHORS         config.log      COPYING@      Makefile.in   nul.c
autom4te.cache/ config.sub@     INSTALL@      missing@      README
ChangeLog       configure*      install-sh@   mkinstalldirs@
```

En aquest punt, ja es podria comprimir aquesta estructura de directori i distribuir-la com a paquet `.tar`, o afegir els fitxers de control necessaris i crear un paquet de codi font `rpm`, `deb`, o qualsevol altre.

Per a compilar-lo a partir d'aquest punt només cal que executem `./configure` i després `make all`:

```
$ ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking whether make sets $(MAKE)... yes
checking for working aclocal-1.4... found
checking for working autoconf... found
checking for working automake-1.4... found
checking for working autoheader... found
checking for working makeinfo... found
checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ANSI C... none needed
configure: creating ./config.status
config.status: creating Makefile

$ make all
gcc -DPACKAGE_NAME=\"nul\" -DPACKAGE_TARNAME=\"nul\" -DPACKAGE_VERSION=\"0.1\"-DPACKAGE_STRING=\"nul\
0.1\" -DPACKAGE_BUGREPORT=\"\" -DPACKAGE=\"nul\" -DVERSION=\"0.1\"  -I. -I.      -g -O2 -c nul.c
nul.c: In function `main':
nul.c:1: warning: return type of 'main' is not `int'
gcc  -g -O2  -o nul  nul.o

$ ls -l nul
-rwxr-xr-x  1 user user 7893 nov 17 17:03 nul
```

4.3.3. Com es mantenen els fitxers d'entrada

Si s'edita algun dels fitxers d'entrada de les autotools –és a dir, `configure.ac` o `Makefile.am`–, caldrà actualitzar els fitxers generats per `autoconf` i `automake`.

Una manera és crear una seqüència que repeteixi els passos que hem fet anteriorment per crear els fitxers. En aquest cas, les autotools ja porten un programa que fa exactament això, anomenat `autoreconf`. Exactament, el que fa és executar una seqüència estàndard de les utilitats d'autotools que normalment és suficient per a actualitzar els fitxers.

De fet, en projectes simples com el de l'exemple anterior (`nul.c`), seria possible executar `autoreconf` directament després de crear `configure.ac` i `Makefile.am`. En aquest cas, afegint els fitxers de documentació de la falta dels quals ens informaria `autoreconf` (en executar `automake --add-missing`) i executant-lo novament.

4.3.4. Empaquetar fitxers de sortida

Una pràctica comuna en els paquets de codi font distribuïts al costat de les distribucions més grans és incloure tots els fitxers generats de tal manera que n'hi hagi prou d'executar `./configure` i `make all` o `make install` perquè el programa sigui totalment funcional en el sistema de destinació.

Això té dues implicacions, una de bona i una altra de no tan bona.

D'una banda, a un usuari novell –o no tan novell–, li facilita molt la tasca de passar pedaços i poder gaudir del programa corregit en el mínim de temps possible sense necessitar més coneixements sobre el sistema.

D'altra banda, si aquests fitxers font s'intentessin compilar en un altre sistema podria ser que el fitxer `./configure` no fos completament compatible amb les autotools instal·lades, per la qual cosa caldria generar-lo des de zero.

ANOTACIONS

Opinem que s'haurien de distribuir ambdós tipus de fitxers, tant els originals `configure.ac` i `Makefile.am` com els seus derivats, fins a un conjunt executable directament en una plataforma preferida, especificant clarament quina és aquesta plataforma (per exemple, una distribució i versió determinada), encara que potser també seria interessant incloure una llista dels fitxers originals abans d'executar les autotools o, fins i tot, una seqüència que permetés treure els fitxers generats per a satisfer totes les necessitats possibles.

4.3.5. ChangeLogs i documentació

Un aspecte que no s'ha de descuidar és documentar correctament els projectes de programari. Encara que això és important fins i tot en un entorn de programació "artístic" en què una sola persona desenvolupa i manté tot el codi, aquest aspecte es torna crucial en els desenvolupaments GNU, on el mateix codi serà analitzat i millorat normalment per diferents desenvolupadors amb poc o, fins i tot, cap contacte previ entre ells, sia per motius geogràfics, temporals o d'altres.

A causa de la rellevància de la documentació i de la facilitat amb què es pot oblidar incloure-la, `automake` en comprova la presència abans de donar per vàlid el conjunt de fitxers.

El fitxer de documentació més rellevant, de referència tant per a usuaris novells o avançats com per a administradors, és el `ChangeLog`. I per això el format té una estructura molt definida i acceptada comunament.

El fitxer `ChangeLog` té una sèrie d'"entrades", en ordre de dates o versions decreixents (el més nou primer), cada una de les quals comença definint amb una línia la data, persona i correu electrònic en què es realitzen els canvis seguida d'una llista de canvis separats per línies en blanc, normalment amb el marge lleugerament desplaçat i precedits d'asterisc (*).

Cada canvi sol incloure una referència del mòdul, fitxer o funcionalitat canviada o actualitzada, i una descripció curta del canvi en si mateix,

ANOTACIONS

opcionalment seguida d'un codi identificatiu (per exemple, si es tracta d'una correcció d'un error de programació, el número de l'error).

Els canvis que es facin sobre elements relacionats es poden incloure dins d'un únic canvi o es poden detallar en línies diferents, tanmateix, en el segon cas, poden no anar separades per una línia en blanc.

Lectura recomanada

Podeu trobar una descripció més detallada en l'adreça següent:

http://www.gnu.org/prep/standards/html_node/Change-Logs.html#Change-Logs

De moment, vegem uns exemples.

Per començar, un fragment del fitxer `./kdb/ChangeLog` del `kernel`:

```
2004-08-14 Keith Owens  <kaos@sgi.com>
       * kdb v4.4-2.6.8-common-1.
2004-08-12 Keith Owens  <kaos@sgi.com>
       * kdb v4.4-2.6.8-rc4-common-1.
2004-08-05 Keith Owens  <kaos@sgi.com>
       * Mark kdb_initcall as __attribute_used__ for newer gcc.
       * kdb v4.4-2.6.8-rc3-common-2.
```

O del fitxer `ChangeLog` de `gcc`:

```
2004-07-01  Release Manager
       * GCC 3.4.1 released.
2004-06-28  Neil Booth  <neil@duron.akihabara.co.uk>
        PR preprocessor/16192
        PR preprocessor/15913
        PR preprocessor/15572
        * cppexp.c (_cpp_parse_expr): Handle remaining cases where an
        expression is missing.
        * cppinit.c (post_options): Traditional cpp doesn't do // comments.
        * doc/cpp.texi: Don't document what we do for ill-formed expressions.
        * doc/cppopts.texi: Clarify processing of command-line defines.

2004-06-28  Richard Sandiford  <rsandifo@redhat.com>

        PR target/16176
        * config/mips/mips.c (mips_expand_unaligned_load): Use a temporary
        register for the destination of the lwl or ldl.
```

Altres registres de canvis segueixen un format lògic semblant, encara que no directament compatible, com per exemple el fitxer `CHANGES` d'Apache 2.0.52:

ANOTACIONS

```
Changes with Apache 2.0.52
*) Use HTML 2.0 <hr> for error pages. PR 30732 [André Malo]

*) Fix the global mutex crash when the global mutex is never allocated
   due to disabled/empty caches. [Jess Holle <jessh ptc.com>]

*) Fix a segfault in the LDAP cache when it is configured switched
   off. [Jess Holle <jessh ptc.com>]

*) SECURITY: CAN-2004-0811 (cve.mitre.org)
   Fix merging of the Satisfy directive, which was applied to
   the surrounding context and could allow access despite configured
   authentication.  PR 31315.  [Rici Lake <rici ricilake.net>]
```

4.3.6. Creació de configure.in

Un dubte que es pot plantejar quan es defineix el fitxer `configure.in` és què posar a dins. En concret, quines revisions s'haurien de realitzar perquè el programa funcionés correctament, sense excedir-se en revisions que puguin fallar en el sistema de destinació i sense que influeixin en la compilació o, simplement, en quin ordre realitzar-les.

4.3.7. Què significa portabilitat?

Un programa idealment portable és aquell que un usuari pot fer funcionar amb poc esforç en qualsevol plataforma i sistema possible. De manera ideal, això comportaria que només hagués d'executar una ordre per a tenir el programa instal·lat i en funcionament.

Aquest seria un cas ideal, que seria fàcil de resoldre si tots els sistemes i plataformes fossin idèntics entre ells. Tanmateix, en la realitat això no es compleix mai i, fins i tot, no seria desitjable que es complís. Els sistemes difereixen entre ells, les plataformes difereixen, fins i tot dins una mateixa plataforma i sistema difereixen els programes, les utilitats i les versions instal·lades de cada un.

Per tant, per a ser portable, un programa ha de ser capaç d'adaptar-se a l'entorn en què s'executi.

A fi que aquestes adaptacions no es facin impossibles a causa de l'existència de variables infinites, hi ha establerta una sèrie d'estàn-

ANOTACIONS

dards, convencions i costums que facilitaran al programa tant identificar l'entorn en què s'executarà com adaptar-s'hi.

En el cas concret de les autotools, se sol parlar de sistemes de "tipus Unix", com ara Linux, BSD, SunOS, Solaris, etc., a més de les diferents variants de cada un. Fins i tot, es pot pensar en compatibilitat amb sistemes Windows, MacOS o d'altres.

4.3.8. Introducció al sh portable

L'eina més important per a instal·lar un programa és el programa de línia d'ordres en si mateix. Aquest serà l'encarregat de servir de base a la resta d'ordres executades durant el procés d'instal·lació.

Per a evitar problemes de compatibilitat entre diferents implementacions de sh, especialment entre aquelles mancades d'algunes funcionalitats esteses, les autotools intenten generar un fitxer `./configure` amb el conjunt de funcionalitats de sh més compatible possible. Això pot portar que el codi resultant no sigui fàcilment llegible per a algú acostumat a les implementacions més difoses.

D'altra banda, la programació amb `sh` se serveix en gran manera d'altres programes "de suport", que també tenen diferents implementacions i no sempre es comporten d'una manera exactament igual que l'esperada.

A causa d'això, els programes GNU solen incloure una secció en la seva documentació (man) en què es detallen les opcions que suporten compatibles amb POSIX, l'estàndard de compatibilitat entre sistemes operatius. Seguint aquestes opcions suportades i evitant les altres, és molt probable que una seqüència sigui altament portable.

Lectura recomanada

L'estàndard POSIX, el podeu consultar en l'adreça següent:

http://www.unix.org/version3/ieee_std.html

Especialment interessant és la secció "Shell Command Language".

4.3.9. Revisions necessàries

Gràcies a autoscan, hem pogut veure una configuració bàsica compatible amb el projecte. Tanmateix, moltes vegades no n'hi ha prou amb aquesta configuració bàsica i cada projecte pot tenir en compte les seves peculiaritats a l'hora de fer les revisions.

ANOTACIONS

Cal tenir en ment que el fitxer configure.am és interpretat com a fitxer de seqüència amb "macros intercalades" que seran expandides per `autoconf` (usant m4). Per tant, es pot incloure en aquest qualsevol fragment de codi de seqüència d'interpretació d'ordres per a fer comprovacions addicionals o condicionar la realització de comprovacions en diferents aspectes.

La seqüència principal recomanada sol ser la següent:

1) **Inicialització**. Aquí s'inclou el codi d'inici de les diferents comprovacions, com pot ser `AC_INIT` o `AM_INIT_AUTOMAKE`.

2) **Configuració**. S'estableixen diferents opcions com per exemple `AC_CONFIG_SRCDIR` o `AC_CONFIG_HEADER`.

3) **Programes**. Aquí es comprova la presència de programes necessaris per a compilar o executar el mateix procés de configuració. En cas que n'hi hagi diversos, també se sol elegir el més adequat.

4) **Biblioteques**. Es comprova que hi ha les biblioteques necessàries per a compilar el programa. Això és recomanable fer-ho abans dels passos següents ja que en alguns es comprova la presència de funcionalitats quan s'intenta compilar programes d'exemple que les utilitzen.

5) **Encapçalaments**. Això es refereix als encapçalaments de biblioteques i mòduls instal·lats en el sistema per als quals també siguin instal·lats els encapçalaments de compilació, necessaris per a compilar el programa.

6) **Typedefs i estructures**. Es comprova que hi hagi typedefs i estructures en els encapçalaments detectats en el pas anterior.

7) **Funcions**. Aquesta comprovació és realment la més important, ja que detectarà la presència de funcions utilitzades en el programa segons els encapçalaments i biblioteques presents i comprovarà la possibilitat d'usar-les aplicant els typedefs i estructures trobats.

ANOTACIONS

8) **Sortida**. Aquest és el pas final que crearà els fitxers de sortida a partir de les opcions detectades.

Lectura recomanada

Podeu trobar una referència completa de les opcions de configuració en l'adreça següent:

http://www.gnu.org/software/autoconf/autoconf.html (Online Manual)

4.4. Introducció a GNU Automake

Automake és la utilitat encarregada de crear els fitxers `Makefile.in` partint d'una definició simple del que cal fer i complint els estàndards de creació de fitxers `Makefile` de GNU.

D'una banda, això facilita crear els `Makefile` perquè no s'han d'especificar en cada un tots els passos que s'han de fer.

D'altra banda, en crear els fitxers `Makefile` amb un format uniforme, serà més fàcil processar-los amb utilitats estàndard compatibles amb aquest format.

Alhora, atesa la gran popularitat d'aquests fitxers `Makefile`, necessaris per a la compilació de pràcticament tots els programes, una gran quantitat d'usuaris és capaç d'aportar informació sobre qualsevol tipus d'incompatibilitat, error, o optimització que s'hagi de corregir o incorporar a `automake`.

Lectura recomanada

Podeu trobar una referència completa sobre `automake` en l'adreça següent:

http://directory.fsf.org/GNU/automake.html (manual d'ususari)

4.4.1. Principis generals d'automake

Com ja hem esmentat, `automake` s'encarrega de convertir `Makefile.am` en `Makefile.in` que posteriorment serà usat per `./configure` com una plantilla per als fitxers `Makefile` finals.

Per tant, els fitxers `Makefile.am` es poden entendre com a fitxers `Makefile` normals amb macros.

Un aspecte diferent respecte als `Makefile` són els comentaris interns d'automake, que seran descartats quan es creï el `Makefile.in` i que començaran amb '##' en comptes de '#':

ANOTACIONS

```
## Aquest és un fitxer Makefile.am d'exemple (aquesta línia no apareixerà)
```

Això a banda, automake suporta línies condicionals i expansions d'include que es processaran per a generar el fitxer `Makefile.in`.

Totes les instruccions i possibles macros no interpretades directament per automake són passades tal qual a `Makefile.in`.

4.4.2. Introducció a les primàries

Les variables primàries s'utilitzen indirectament per a derivar noms d'objectes a partir d'elles.

Per exemple, `PROGRAMS` és un nom primari, i `bin_PROGRAMS` és un nom derivat d'aquell:

```
bin_PROGRAMS = nul
```

En aquest cas, s'usa el prefix especial `bin` que indicaria la destinació del programa en "`bindir`". Algunes de les destinacions possibles són les següents:

- `bindir`: on han d'anar els fitxers binaris.
- `sbindir`: per als fitxers binaris de sistemes.
- `libexecdir`: programes i dimonis executables per altres programes.
- `pkglibdir`: biblioteques no compartides en el sistema.
- `noinst`: fitxers que no s'instal·laran.
- `check`: només seran compilats amb "make check" per a comprovacions.
- `man`: pàgines de manual.

Per a indicar el codi font que formarà la destinació `nul` especificada abans, usarem la primària `SOURCES`:

```
nul_SOURCES = nul.c
```

A banda d'aquestes primàries principals, també hi ha disponibles:

- `HEADERS`: permet fer una llista dels encapçalaments `.h` dels fitxers.
- `SCRIPTS`: fa una llista de les seqüències executables. Seran copiades sense interpretar, encara que amb atributs d'execució.
- `DATA`: fitxers que seran copiats directament. Solen ser continguts fixos utilitzats pel programa, com ara fonts de dades o gràfics.
- MANS: es refereix a les pàgines de manual. Se solen usar com a `man_MANS` o `man1_MANS`, `man2_MANS`, etc. per a permetre instal·lar en seccions concretes del manual.

4.4.3. Programes i biblioteques

Com ja hem mencionat, els programes s'indiquen amb la primària `PROGRAMS`.

En el cas de les biblioteques, s'aplica la primària `LIBRARIES`:

```
lib_LIBRARIES = libtest.a
libtest_a_SOURCES = ltest.c ltest.h
```

Per a biblioteques libtool (vegeu més endavant), hi ha la primària `LTLIBRARIES`:

```
lib_LIBRARIES = libtest.la
libtest_la_SOURCES = ltest.c ltest.h
```

Com es pot apreciar, el punt de "`libtest.a`" es converteix en un guió baix per a donar lloc al prefix d'objecte "`libtest_a`".

ANOTACIONS

4.4.4. Directoris múltiples

Un projecte mínimament gran i ben organitzat sol estar dividit en diferents subdirectoris.

Per a processar estructures de directori respecte de l'actual, automake accepta l'opció `SUBDIRS`:

```
SUBDIRS = src libsrc man
```

L'única restricció és que els directoris han de ser descendents de l'actual, per la qual cosa si es vol processar també un segon nivell, com per exemple `src/main/`, s'ha d'incloure aquest subdirectori main al fitxer `Makefile.am` de `src`:

src/Makefile.am

```
SUBDIRS = . main
```

4.4.5. Com es prova

Una de les característiques d'`automake` i dels `Makefile` generats és la capacitat de realitzar tests de funcionament del codi compilat.

Sota la destinació `check` de `Makefile`, invocada amb "make check", s'executaran una sèrie de programes especificats en la variable `TESTS` de `Makefile.am`. En aquesta variable, es poden especificar tant objectes font com objectes derivats.

Cada programa testat ha de tornar un valor zero per a indicar test passat, o diferent de zero per a indicar test fallit. Els testos que no tinguin validesa en la plataforma o entorn de compilació poden tornar el valor 77 perquè no siguin tinguts en compte en el còmput final.

Igualment, és possible especificar una llista de tests que s'espera que fallin amb la variable `XFAIL_TESTS`, que seran considerats com a vàlids en el cas invers que els no especificats. Aquests tests també s'han d'incloure en la variable `TESTS` per a ser executats.

ANOTACIONS

Per a especificar una destinació comuna a tots els programes de test, es pot usar la destinació check amb la primària `PROGRAMS`:

```
check_PROGRAMS = test1 test2
test1_SOURCES = test1.c
test2_SOURCES = test2.c
```

Així mateix, es poden executar els tests de DejaGnu amb:

```
AUTOMAKE_OPTIONS = dejagn
```

4.5. La biblioteca GNU Libtool

Libtool és una eina que permet abstreure el procés de compilació de biblioteques del format concret que aquestes adoptin en la plataforma en què siguin compilades, incloent-hi el control de versions i les dependències entre biblioteques.

Això és especialment important en el cas de biblioteques que puguin ser compartides en el sistema de destinació, encara que també és un afegit còmode per al desenvolupament normal de biblioteques, independentment de la plataforma.

Per a aconseguir-ho, les biblioteques i els executables es compilen en un format executable modificat, però s'expressen amb l'extensió `.la` en comptes de `.a`.

A més, `libtool` ha estat pensada de manera que estigui integrada amb les utilitats `autoconf` i `automake`, encara que no sigui obligatori que aquestes estiguin instal·lades per a poder usar `libtool`.

Lectura recomanada

Podeu trobar més informació sobre `libtool` en l'adreça següent:

http://www.gnu.org/software/libtool/manual.html

4.5.1. Codi independent

Per a una biblioteca interna d'un programa, no cal que el seu codi sigui independent de posició (***position independent code*** = PIC). Tanmateix, quan es creen biblioteques compartides entre diferents programes,

ANOTACIONS

cal que l'adreça en què es carrega la biblioteca sigui alterable, de manera que les adreces internes hauran de ser relatives al punt d'inici, o alterades en el moment de carregar la biblioteca.

Per a crear codi independent, cal passar una sèrie d'opcions al compilador. Aquesta tasca es pot ocultar usant libtool, que s'encarregarà de tot el procés de compilació.

Normalment, el codi independent es generarà com a objectes `.lo`, a diferència del codi normal `.o`.

4.5.2. Biblioteques compartides

Per defecte, `libtool` crearà sempre biblioteques compartides, excepte en les plataformes en què això no sigui possible o si se li requereix explícitament construir biblioteques estàtiques.

Vegem això amb uns quants exemples.

Per a això, crearem una biblioteca bàsica d'exemple:

suma.c

```
int suma (int a, int b) {
 return a+b;
}
```

Una compilació normal sense utilitzar `libtool` consistiria en el següent:

```
$ gcc -c suma.c
$ ls
suma.c  suma.o
$ ar cru libsuma.a suma.o
$ ranlib libsuma.a
$ ls
libsuma.a  suma.c  suma.o
```

En aquest cas, la biblioteca creada és estàtica.

Per a crear una biblioteca compartida, podríem recórrer a `libtool`:

```
$ libtool --mode=compile gcc -c suma.c
mkdir .libs
  gcc -c suma.c  -fPIC -DPIC -o .libs/suma.o
  gcc -c suma.c -o suma.o >/dev/null 2>&1
$ libtool --mode=link gcc -o libsuma.la suma.lo -rpath /usr/local/lib/
rm -fr  .libs/libsuma.a .libs/libsuma.la
gcc -shared  .libs/suma.o   -Wl,-soname -Wl,libsuma.so.0 -o .libs/libsuma.so.0.0.0
(cd .libs && rm -f libsuma.so.0 && ln -s libsuma.so.0.0.0 libsuma.so.0)
(cd .libs && rm -f libsuma.so && ln -s libsuma.so.0.0.0 libsuma.so)
ar cru .libs/libsuma.a .libs/suma.o
ranlib .libs/libsuma.a
creating libsuma.la
(cd .libs && rm -f libsuma.la && ln -s ../libsuma.la libsuma.la)
$ ls
libsuma.la  suma.c  suma.lo  suma.o
```

Amb l'opció `-rpath` especifiquem l'adreça en què estarà instal·lada la biblioteca.

4.5.3. Biblioteques estàtiques

Per a aprofitar la capacitat d'interfície estàndard de creació de biblioteques, `libtool` també permet crear biblioteques estàtiques sense haver de cridar directament el compilador, especificant l'opció `-static` en la fase d'enllaçament:

```
$ libtool --mode=link gcc -static -o libsuma.la suma.lo
rm -fr  .libs/libsuma.a .libs/libsuma.la
ar cru .libs/libsuma.a  suma.o
ranlib .libs/libsuma.a
creating libsuma.la
(cd .libs && rm -f libsuma.la && ln -s ../libsuma.la libsuma.la)
```

D'aquesta manera, es crearà una biblioteca que contindrà tots els objectes referenciats per aquesta.

Per a provar que aquesta biblioteca funciona correctament, l'haurem d'enllaçar a un executable.

ANOTACIONS

Per exemple, en aquest cas, usarem aquest executable mínim:

main.c

```
#include <stdio.h>

int suma (int a, int b);

int main (int argc, char *argv[]) {
 printf ("%i", suma (1, 2) );
 exit (0);
}
```

Ara compilem aquest programa incloent-hi la biblioteca anterior:

```
$ libtool --mode=link gcc -o suma main.c libsuma.la
gcc -o .libs/suma main.c  ./.libs/libsuma.so -Wl,--rpath -Wl,/usr/local/lib/
creating suma
$ ./suma
3
```

Veiem que el programa funciona perfectament cridant la funció de la biblioteca.

4.5.4. Enllaçar una biblioteca. Dependències entre biblioteques

És molt comú enllaçar una biblioteca amb d'altres de ja existents per a afegir o agrupar funcionalitat en una única lliberia, o per a maximitzar les capacitats de portabilitat d'un programa per no dependre de l'existència o no de biblioteques en el sistema de destinació.

`libtool` usarà el sistema suportat en la destinació per a la interdependència de biblioteques, de manera transparent per a l'usuari.

En cas de no trobar un mitjà natiu per a resoldre-les, `libtool` ofereix una emulació pròpia.

ANOTACIONS

4.5.5. Usar biblioteques de conveniència

Les biblioteques de conveniència són biblioteques enllaçades parcialment que es poden usar com a agrupacions intermèdies d'objectes encara que no siguin ni executables ni biblioteques carregables en si mateixes. Aquestes biblioteques es defineixen simplement sense especificar ni `-static` ni `-rpath` en el moment de cridar `libtool` com a `--mode=compile`, i són usades només per a ser enllaçades dins altres biblioteques o fitxers executables.

4.5.6. Instal·lar biblioteques i executables

La instal·lació de biblioteques per mitjà de `libtool` en realitat no representa massa problema.

```
$ libtool --mode=install cp libsuma.la /usr/local/lib/libsuma.la
cp libsuma.la /usr/local/lib/libsuma.la
cp .libs/libsuma.a /usr/local/lib/libsuma.a
ranlib /usr/local/lib/libsuma.a
```

Per a acabar la instal·lació, usarem `--mode=finish`:

```
libtool --mode=finish /usr/local/lib
PATH="$PATH:/sbin" ldconfig -n /usr/local/lib
----------------------------------------------------------------------
Libraries have been installed in:
   /usr/local/lib
If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the `-LLIBDIR'
flag during linking and do at least one of the following:
   - add LIBDIR to the `LD_LIBRARY_PATH' environment variable
     during execution
   - add LIBDIR to the `LD_RUN_PATH' environment variable
     during linking
   - use the `-Wl,--rpath -Wl,LIBDIR' linker flag
  - have your system administrator add LIBDIR to `/etc/ld.so.conf'

See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
----------------------------------------------------------------------
```

Amb això, ja podrem usar la nostra biblioteca compartida.

ANOTACIONS

4.5.7. Desinstal·lació

Podem fer la desinstal·lació per mitjà de `--mode=uninstall` amb l'ordre `rm`:

```
$ libtool --mode=uninstall rm -f /usr/local/lib/libsuma.la
rm -f /usr/local/lib/libsuma.la /usr/local/lib/libsuma.a
rm -f /usr/local/lib/libsuma.la /usr/local/lib/libsuma.so.0.0.0 /usr/local/lib/libsuma.so.0 /usr/
local/lib/libsuma.so /usr/local/lib/libsuma.a
```

D'aquesta manera, s'assegura que la biblioteca quedarà desinstal·lada independentment de la plataforma, inclòs qualsevol enllaç a aquesta que pugui haver calgut crear en la instal·lació.

4.5.8. GNU Libtool, configure.in i Makefile.am

Per a utilitzar `libtool` amb `autoconf` i `automake`, només cal definir-ne l'ús en `configure.ac`.

La macro que ens ofereix `autoconf` per a dur a terme això és l'`AC_PROG_LIBTOOL`. Així mateix, no caldran comprovacions addicionals relacionades amb la compilació de biblioteques, ja que de tot això se n'encarregarà `libtool`:

```
AC_PROG_LIBTOOL
```

Després d'això, només haurem d'executar `autoconf` i `automake` com de costum.

4.5.9. Integració amb configure.in, opcions extres i macros per a Libtool

La compilació amb `libtool` afegeix una sèrie d'opcions noves a `./configure`, entre les quals podem esmentar les següents:

`--enable-shared`

`--disable-shared`: permeten activar o desactivar la compilació de biblioteques compartides.

ANOTACIONS

`--enable-static`

`--disable-static`: igualment, per a biblioteques estàtiques.

`--with-pic`: força a construir biblioteques amb codi independent (PIC).

Aquestes opcions tenen les seves opcions corresponents de `configure.ac`:

```
AC_DISABLE_SHARED
```

per a desactivar la compilació de biblioteques compartides, i

```
AC_DISABLE_STATIC
```

per a desactivar la compilació de les estàtiques.

4.5.10. Integració amb Makefile.am, creació de biblioteques amb Automake i enllaçament contra biblioteques Libtool

Automake suporta la compilació i l'enllaç de biblioteques per mitjà de la destinació `lib` i la primària `LIBRARIES`:

```
lib_LIBRARIES = libsuma.a
libsuma_a_SOURCES = suma.c
```

La construcció de biblioteques amb l'ús de libtools utilitza la primària `LTLIBRARIES`:

```
lib_LTLIBRARIES = libsuma.la
libsuma_la_SOURCES = suma.c
```

L'enllaçament contra biblioteques `libtool` s'obté per mitjà de la primària `LDADD`:

```
bin_PROGRAMS = suma
suma_SOURCES = main.c
suma_LDADD = libsuma.c
```

ANOTACIONS

4.5.11. Utilització de libtoolize

L'eina que ens ajudarà a configurar el nostre projecte per a usar `libtool`, i a afegir i comprovar que el suport per a `libtool` és present és `libtoolize`.

El procés complet de creació del nostre projecte amb `libtool` pot ser el següent.

configure.ac

```
AC_INIT(suma, 0.1)
AM_INIT_AUTOMAKE(suma, 0.1)
AC_CONFIG_SRCDIR([suma.c])

# Checks for programs.
AC_PROG_CC
AC_PROG_LIBTOOL

AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

Makefile.am

```
bin_PROGRAMS = suma
suma_SOURCES = main.c
suma_LDADD = libsuma.la

lib_LTLIBRARIES = libsuma.la
libsuma_la_SOURCES = suma.c
```

Una vegada fetes les modificacions corresponents per a afegir suport de `libtool` i establir els fitxers font i la destinació que cal usar en `configure.ac` i `Makefile.am`, comprovem que el projecte compila correctament:

```
$ ls
AUTHORS  ChangeLog  configure.ac  main.c  Makefile.am  NEWS  nul.c  README  suma
$ libtoolize
You should add the contents of `/usr/share/aclocal/libtool.m4' to `aclocal.m4'.
$ aclocal
$ automake --add-missing
automake: configure.ac: installing `./install-sh'
automake: configure.ac: installing `./mkinstalldirs'
automake: configure.ac: installing `./missing'
automake: Makefile.am: installing `./INSTALL'
automake: Makefile.am: installing `./COPYING'
```

ANOTACIONS

```
$ autoconf
$ ./configure
[...]
$ make all
[...]
creating suma
$ ls *suma*
libsuma.la suma*  suma.c  suma.lo  suma.o
$ ./suma
3
```

Com podem observar, tots els elements s'han creat correctament i el programa final ha estat compilat.

Nota

En alguns casos, no caldrà executar `libtoolize` directament, sinó que serà cridat per `automake --add-missing` (en aquest exemple, hauria funcionat correctament).

4.5.12. Com es fan versions de biblioteques

La creació de biblioteques, especialment de les compartides per a ser instal·lades en el sistema, requereix una sèrie de consideracions addicionals relatives a la versió de les biblioteques distribuïdes.

A l'hora de carregar biblioteques, serà escollida aquella que compleixi una sèrie de condicions relatives a la compatibilitat amb el programa que la necessiti, definides per mitjà del número de versió. Aquest número de versió, encara que pot ser diferent en cada plataforma, és abstret per `libtool` amb un format unificat vist a partir de les interfícies exportades per la biblioteca. D'aquesta manera, es permet que un programa amb interfície compatible pugui carregar una biblioteca posterior a la seva distribució, que pot tenir errors corregits o funcionalitat afegida sense perdre compatibilitat cap enrere.

Entendrem per *interfície*, a efectes de versió, els tipus i noms de variables globals, les funcions globals (tipus d'arguments, valor de retorn, noms de funcions), entrada/sortida/error estàndard i formats de fitxers, sòcols, canonades, i altres formes de comunicació entre processos i els seus formats.

Quant a libtool, s'entén que les biblioteques exporten "conjunts d'interfícies" numerats seqüencialment. Quan s'intenti carregar, serà escollida aquella biblioteca que suporti tots els números d'interfície requerits pel programa.

ANOTACIONS

Els números d'interfície s'estableixen per mitjà de l'opció `--version-info`, especificada en el mode de `libtool --mode=link` o en la primària `LDFLAGS` de `Makefile.am`, amb el format següent:

```
actual : revisió : antiguitat
```

- `actual`: estableix el número de la interfície més moderna suportada.
- `revisió`: permet distribuir biblioteques actualitzades amb la mateixa interfície.
- `antiguitat`: nombre d'interfícies suportades anteriorment.

El total d'interfícies suportades estarà definit per l'interval:

des d'`actual-antiguitat` fins a `actual`

Així, una biblioteca definida com a 1:1:0 suportarà només la interfície número 1, revisió 1.

Mentrestant, la biblioteca definida com a 2:1:1 suportarà tant la interfície 1 com la 2, de la qual suportarà la revisió 1, per la qual cosa podrà ser carregada en comptes de la 1:1:0.

La biblioteca amb versió 2:2:1 igualment podrà ser carregada pels programes que necessitin la 1 o la 2:1:1.

Tanmateix, una biblioteca 2:3:0 haurà trencat la compatibilitat amb els programes per a la 1, i només serà compatible amb els que acceptin la 2, encara que no amb un programa que necessiti la 2:4.

4.6. Conclusions

La preparació d'un entorn de desenvolupament GNU és molt diferent (en la forma, les utilitats i aplicacions que cal instal·lar, i en el fons) de la dels entorns propietari i visuals usats en sistemes operatius Windows.

ANOTACIONS

Aquestes diferències es fan paleses per l'ús d'eines poc comunes en altres sistemes, no només funcionalment sinó també visualment.

En aquest capítol, hem definit les tasques, hem establert les bases i hem reconegut el programari necessari per a configurar el nostre entorn de desenvolupament GNU, de manera que puguem avançar en l'ús d'IDE de desenvolupament i eines visuals en capítols posteriors.

ANOTACIONS

5. Control de versions

5.1. Introducció

Els sistemes de control de versions funcionen com la columna vertebral; permeten a grups de persones treballar conjuntament en el desenvolupament de projectes, freqüentment per mitjà d'Internet. Són sistemes que assenyalen les diferents versions per a identificar-les posteriorment, faciliten el treball en paral·lel de grups d'usuaris, indiquen l'evolució dels diferents mòduls del projecte, i disposen d'un control detallat dels canvis que s'han realitzat; funcions que són indispensables durant la vida del projecte. Aquests sistemes no solament tenen aplicació en el desenvolupament del programari, sinó que a més són utilitzats àmpliament a l'hora de crear documentació, llocs web i, en general, qualsevol projecte col·laboratiu que requereixi treballar amb equips de persones de manera concurrent.

En el món del programari propietari, els sistemes de control de versions han estat usats tradicionalment per a gestionar equips de desenvolupament en entorns corporatius tancats. Amb l'explosió d'Internet i l'avenç del programari lliure, es van dissenyar sistemes oberts que permetessin treballar simultàniament a milers de persones distribuïdes en diferents regions del món en un mateix projecte per mitjà d'Internet.

CVS (*concurrent versions system*) és el programa més utilitzat en el món del programari lliure per a controlar versions de programari. Basat en el model client-servidor, hi ha versions del programa per a multitud de plataformes. La seva solidesa i la seva eficàcia provada, tant en grups petits d'usuaris com en grups grans, l'han convertit en l'eina que utilitzen projectes de programari lliure d'èxit reconegut com Mozilla, OpenOffice.org, KDE o GNOME, per esmentar-ne només uns quants. Veurem també Subversion, un nou sistema de control de versions amb prestacions superiors a CVS.

ANOTACIONS

Durant aquest capítol, aprendrem com utilitzar CVS i Subversion per a facilitar el treball entre un grup d'usuaris que treballen en un mateix projecte, i veurem la utilitat de disposar d'un sistema de control de versions.

5.2. Objectius

Els materials didàctics associats amb aquest capítol us permetran adquirir els coneixements següents:

- Familiaritzar-vos amb la part client de CVS i Subversion per a poder seguir l'evolució d'un projecte de programari en què participin diversos desenvolupadors.
- Com s'envien canvis i correccions per mitjà de CVS o per correu electrònic usant les ordres `diff` i `patch`.
- Com s'instal·len i administren un servidor CVS propi en què es publiquen els projectes i compartir-los amb els altres usuaris.
- El funcionament dels sistemes de versions de fitxers, per etiqueta, dates o branques.

5.3. Sistemes de control de versions

Les funcions principals dels sistemes de control de versions són proporcionar un històric dels canvis dels fitxers d'un projecte i permetre recuperar una versió determinada del fitxer en qualsevol moment. Imaginem que tenim la versió 1.00 d'un fitxer d'instruccions d'instal·lació d'un programa. A aquest fitxer de text afegim diversos paràgrafs que indiquen els passos detallats que cal seguir per a instal·lar el nostre programa en versions de UNIX BSD, i aquesta nova versió amb instruccions per a BSD es converteix en la versió 1.01 del document. A mesura que el projecte avança, aquest fitxer continua evolucionant. Amb un sistema de control de versions, en qualsevol moment podem obtenir la ver-

ANOTACIONS

sió 1.00 o 1.01 sense necessitat de mantenir còpies separades dels fitxers.

Els sistemes de control de versions es basen en el fet de mantenir tots els fitxers del projecte en un lloc centralitzat, normalment un únic servidor. També hi ha, però, sistemes distribuïts en què els desenvolupadors es connecten i descarreguen una còpia en local del projecte. Amb aquesta, envien periòdicament al servidor els canvis que realitzen i actualitzen el seu directori de treball que, al seu torn, altres usuaris han anat modificant.

Els sistemes de control de versions de codi estan integrats en el procés de desenvolupament de programari de moltes empreses. Qualsevol empresa que tingui més d'un programador treballant en un mateix projecte acostuma a tenir un sistema d'aquest tipus, i a mesura que creix el nombre de persones que s'involucren en un projecte, més indispensable es fa un sistema de control de versions.

5.3.1. Alguns termes comuns

Els sistemes de control de versions tenen una terminologia molt específica per a referir-se a conceptes singulars d'aquest tipus de sistemes. A continuació, veurem els conceptes i termes més habituals del paquet de programari CVS i que són extensibles a la majoria de sistemes:

Accés anònim. Tipus d'accés a un servidor amb el qual tenim dret a veure i copiar els fitxers del dipòsit, però no a modificar-los.

Dipòsit. Lloc centralitzat on s'emmagatzema la còpia de tots els fitxers del projecte que compartim amb tots els usuaris i que habitualment resideix en un únic servidor.

Directori de treball. Versió del projecte que prové del dipòsit i que un usuari té en el seu disc dur local i sobre la qual treballa.

Commit. Acció amb què enviem els canvis que hem realitzat en el nostre directori de treball al directori central del servidor. També s'utilitza per a afegir de manera definitiva nous fitxers en el dipòsit.

ANOTACIONS

Branca. A partir d'un punt concret, es crea una bifurcació del projecte que pot evolucionar separadament.

Etiqueta. Nom simbòlic que assignem a un fitxer o grup de fitxers del projecte per a indicar que la versió té un significat especial en el nostre procés de desenvolupament. Amb aquest nom, ens podrem referir als fitxers més endavant.

Tarball. Còpia completa d'un projecte en un punt determinat realitzada amb l'ordre tar de Unix. És molt freqüent fer una còpia de tots els fitxers d'un projecte en aquest format perquè és molt més ràpid de descarregar per Internet que en CVS o Subversion.

5.3.2. Característiques dels sistemes de control de versions

Tots els sistemes de control de versions moderns permeten un conjunt de funcions que es consideren indispensables per a conèixer d'una forma exacta l'estat d'un projecte. A continuació, enumerem les funcions principals:

- **Control d'usuaris**. Establir quins usuaris tenen accés als fitxers del projecte i quin tipus d'accés tenen assignat.

- **Mantenir un control detallat de cada fitxer**. Disposar d'un control dels canvis que s'han efectuat en el fitxer, a quina data, amb quin motiu, qui els va realitzar i qui va mantenir les diferents versions.

- **Barrejar dues versions diferents del mateix fitxer**. En cas que dos usuaris modifiquin un fitxer (sistemes de no bloqueig de fitxers), proporcionar eines per a gestionar aquest tipus de conflictes.

- **Bifurcació de projectes**. A partir d'un punt concret, crear dues branques del projecte que poden evolucionar separadament. És habitual utilitzar aquesta tècnica quan hem alliberat una versió d'un producte i necessitem que continuï evolucionant.

ANOTACIONS

5.3.3. Principals sistemes de control de versions

Hi ha una bona oferta de sistemes de control de versions disponibles com a programari propietari i lliure, encara que durant aquest capítol ens centrarem en els sistemes CVS i Subversion, que són solucions lliures. Esmentarem breument els sistemes principals de control de versions que hi ha en el mercat:

- **CVS** (*concurrent versions system*)

És el sistema més utilitzat en el món del programari lliure per a controlar versions. Basat en el model client-servidor, el mateix sistema és programari lliure i hi ha versions del programari per a multitud de plataformes. La seva solidesa i la seva eficàcia provada, tant en grups petits d'usuaris com en grups grans, l'han convertit en l'eina que utilitzen projectes de programari lliure com Mozilla, OpenOffice.org, KDE o GNOME, per esmentar-ne només uns quants.

Nota

http://www.cvshome.org

- **RCS** (*revision control system*)

El sistema CVS està basat en realitat en un altre d'anterior, RCS (*revision control system*), que va ser desenvolupat per Walter Tichy a la Universitat de Purdue al començament de 1980. El sistema RCS és extremadament simple i fàcil d'instal·lar, però no pot cobrir les necessitats de projectes o equips mitjans. Entre les limitacions més notables de RCS, destaca que només pot treballar en un únic directori, mentre que la majoria de projectes tenen múltiples directoris. RCS utilitza un sistema de bloqueig de fitxers que impedeix a diversos desenvolupadors treballar sobre el mateix fitxer.

Nota

http://www.gnu.org/software/rcs/rcs.html

- **BitKeeper**

BitKeeper és un producte propietari desenvolupat per l'empresa BitMover. És probablement el producte més sofisticat de la seva categoria. Entre les característiques que el diferencien de la resta de productes, destaquen la possibilitat de treballar amb dipòsits distribuïts i

Nota

http://www.bitkeeper.com

ANOTACIONS

un sistema molt avançant per a integrar versions diferents d'un mateix fitxer.

- **Microsoft Visual Source Safe**

Nota

http://msdn.microsoft.com/ssafe

És un dels productes més utilitzats per a desenvolupar aplicacions Windows. Principalment perquè s'integra en l'entorn de treball de Visual Studio i amb la resta d'eines de desenvolupament de Microsoft. Té funcions de comparació visual de fitxers realment avançades, en el seu mode de funcionament bàsic utilitza bloqueig de fitxers.

5.3.4. Sistemes de compartició

Un punt crític per als sistemes de control és gestionar un fitxer que és utilitzat per diversos usuaris alhora. El mecanisme que utilitzin per a resoldre aquest tipus de situació tindrà implicacions en la manera amb què els usuaris poden treballar simultàniament amb el fitxer i amb què es gestionen els canvis que es realitzen. Hi ha dos sistemes de compartició:

- **Per bloqueig de fitxer**

És el sistema més primitiu i el que utilitzen RCS i les versions antigues de Visual Source Safe de Microsoft. Es tracta de bloquejar el fitxer quan un usuari està fent canvis, de manera que els altres usuaris no poden fer modificacions fins que no acabi aquell usuari.

És habitual que un usuari comenci a treballar en un fitxer i necessiti unes quantes hores per a afegir el nou codi o modificar el que hi havia; per aquesta raó, aquests sistemes no són adequats, ja que un mateix usuari pot bloquejar un fitxer durant un període llarg de temps durant el qual cap altre usuari no pot fer canvis.

- **Per gestió de canvis**

Aquest sistema permet que els usuaris o grups d'usuaris modifiquin alhora un fitxer. Quan han acabat els canvis, fan una operació com-

ANOTACIONS

mit que comprova les diferències entre el fitxer en el servidor i el fitxer millorat pel desenvolupador i envia els canvis. Aquest és el mètode que utilitza CVS.

El mètode és una millora substancial sobre el sistema de bloqueig de fitxers ja que permet a més d'un usuari treballar sobre el mateix fitxer, però introdueix un problema nou, la gestió de conflictes. Imaginem que tenim dues persones que treballen sobre el mateix fitxer i sobre la mateixa zona del fitxer. Quan el primer usuari enviï els seus canvis, aquests es desaran en el servidor i, poc després, quan el segon usuari enviï els seus, trobarà que la còpia que tenia en local ha canviat i que els canvis no es poden integrar automàticament. Disposem de sistemes sofisticats per a gestionar de la millor manera possible aquests tipus de situacions, incloent-hi eines visuals. CVS utilitza un sistema rudimentari, en què bàsicament nosaltres mateixos haurem de reimplementar els canvis.

5.4. Primers passos amb CVS

5.4.1. Instal·lar CVS

Mitjançant el lloc web www.cvshome.org, es poden obtenir les diferents versions client i servidor del programa CVS. La majoria de distribucions GNU/Linux inclouen paquets preparats amb les eines CVS. De fet, en la majoria estan instal·lats per defecte.

Si som usuaris de Debian i no tenim instal·lat CVS, la seva instal·lació, com la de la resta de programari d'aquesta distribució, és molt senzilla:

```
$ apt-get install cvs
```

Per a usuaris de la distribució Red Hat, el millor és recórrer als CD-ROM originals i fer la instal·lació des de l'entorn gràfic o amb l'ordre de línia d'ordres següent:

```
$ rpm -ivh cvs
```

ANOTACIONS

En aquest temari, ens referirem a les versions GNU/Linux de les eines CVS exclusivament. Si necessitem treballar amb Microsoft Windows hi ha almenys dos clients:

- **Cygwin**

Cygwin és una emulació completa d'un entorn tipus Unix per a Windows. Disposem de la majoria d'eines que hi ha en GNU/Linux, incloent-hi una versió adaptada de CVS. De fet, aquest és l'entorn en què es desenvolupen alguns projectes lliures, com Abiword en la seva versió Windows.

Nota

http://www.cygwin.com/
http://www.abiword.com

L'avantatge de Cygwin és que treballem amb la versió adaptada de l'eina original de CVS per a Unix en entorn Windows, de tal manera que el seu comportament és idèntic al de la versió GNU/Linux descrita en aquest mòdul.

- **WinCVS**

Nota

http://www.wincvs.org

WinCVS és un programa lliure amb entorn gràfic per a plataforma Windows que permet treballar amb dipòsits CVS. És altament intuïtiu i no cal conèixer les ordres CVS ja que des de l'entorn gràfic es poden realitzar totes les operacions necessàries.

5.4.2. Obtenir un directori de treball d'un projecte ja existent

Una bona manera de tenir un primer contacte amb CVS és utilitzar un servidor d'un projecte de programari lliure i obtenir una còpia dels fitxers del projecte.

Nota

En la pàgina http://mozilla.org/cvs.html, hi ha informació detallada sobre com s'accedeix a aquest servidor mitjançant CVS.

El lloc web www.mozilla.org acull el projecte Mozilla. Aquest projecte té com a objectiu produir les tecnologies necessàries per a construir un conjunt d'eines a fi de treballar a Internet. Els components principals de Mozilla són un navegador web, un sistema de correu electrònic i un editor de pàgines web, encara que el nombre d'eines que s'introdueix creix constantment.

Primer definirem la variable d'entorn CVSROOT, que indica el servidor, el protocol d'accés, i l'usuari que utilitzarem per a accedir al dipòsit CVS. En el cas del projecte Mozilla, definirem la variable d'entorn amb el valor següent:

```
$ CVSROOT=:pserver:anonymous@cvs-mirror.mozilla.org:/cvsroot
$ export CVSROOT
```

Si parem atenció a la línia, observarem que el nostre nom d'usuari és *anonymous* i ja forma part de la cadena CVSROOT, d'aquesta manera només hem d'introduir la contrasenya per a identificar-nos amb accés anònim. També és possible usar el paràmetre -d de l'ordre CVS per a especificar el camí del servidor CVS.

Ara podem escriure l'ordre següent:

```
$ cvs login
```

CVS contesta:

```
Logging in to :pserver:anonymous@cvs-mirror.mozilla.org:2401/cvsroot
CVS password:
```

Aquesta ordre ens permet autentificar-nos en el servidor. La contrasenya de l'usuari *anonymous* per al projecte Mozilla és *anonymous*. En alguns servidors, per a accedir anònimament la contrasenya és simplement <retorn>, és a dir, sense contrasenya. En qualsevol cas, depèn de com hagi configurat l'administrador el sistema de dipòsit.

A continuació, podem utilitzar l'ordre `checkout` per a obtenir el codi del projecte Mozilla. Aquesta ordre és la que utilitzem la primera vegada que baixem un projecte d'un dipòsit, i admet com a paràmetre el nom del mòdul. Per al projecte Mozilla, escriurem:

```
$ cvs checkout mozilla
```

ANOTACIONS

CVS mostra els noms dels fitxers que es van baixant:

```
cvs server: Updating mozilla
U mozilla/.cvsignore
U mozilla/LEGAL
U mozilla/LICENSE
U mozilla/Makefile.in
U mozilla/README.txt
U mozilla/aclocal.m4
...
```

El nom dels mòduls, en aquest cas Mozilla, és arbitrari i és definit per l'administrador del dipòsit. De fet, els noms dels mòduls són els noms dels directoris arrel del projecte.

Si examinem l'estructura de directoris creada, observarem que hi ha un directori CVS per a cada subdirectori del projecte. Aquests directoris contenen tres fitxers:

- **Entries**. Conté els fitxers d'aquest directori que estan reflectits en el dipòsit.
- **Repository**. Conté la ruta relativa al mòdul al qual pertany aquest fitxer.
- **Root**. Conté la ruta principal al servidor CVS.

De cap manera, no s'han d'esborrar, i en cas d'editar-se, s'ha de fer amb molta cura.

Nota

Podem practicar amb altres projectes lliures. Trobarem instruccions detallades de com es baixa el projecte GNOME amb CVS en el lloc web www.gnome.org.

5.4.3. Sincronitzar-se amb el dipòsit

Una vegada que disposem d'una còpia en local dels fitxers, ja tenim una còpia del projecte amb la qual podem començar a treballar lo-

ANOTACIONS

calment. Tanmateix, a mesura que passen els dies, altres usuaris van efectuant canvis en el directori del projecte i, en conseqüència, el nostre directori de treball queda desactualitzat perquè aquests canvis nous no s'han aplicat. És habitual que ens sincronitzem amb el dipòsit regularment per a mantenir el nostre directori de treball tan actualitzat com sigui possible quant al projecte. Aquesta operació es realitza amb l'ordre `update` de la manera següent:

```
$ cvs update -d -P
```

L'ordre `update` accepta diversos paràmetres, però els paràmetres -d i -P són molt habituals: el primer construeix els directoris necessaris i el segon elimina del nostre directori de treball els directoris que han quedat buits.

L'ordre `update`, quan informa dels fitxers actualitzats, afegeix una lletra al principi del nom de fitxer. Aquestes lletres tenen el significat següent:

- **C**: indica que hi ha un conflicte en el fitxer. Hi ha hagut una actualització en el dipòsit i en el directori de treball i el conflicte s'ha de solucionar manualment.

- **U**: el directori de treball que teníem era antic i s'ha actualitzat amb la versió més recent del dipòsit.

- **M**: el directori de treball del fitxer té canvis però s'ha aconseguit una actualització automàtica amb la versió més recent del dipòsit.

- **A**: el fitxer ha estat afegit però encara no hem efectuat l'ordre commit.

- **R**: el fitxer ha estat esborrat però encara no hem efectuat l'ordre commit.

Durant aquest procés de sincronització, només els fitxers que han estat modificats són actualitzats. És important entendre que l'ordre update no envia en cap cas les modificacions que hàgim realitzat en el nostre directori de treball dels fitxers al dipòsit (pujar canvis

ANOTACIONS

al dipòsit), sinó que simplement actualitza el nostre directori de treball amb els canvis del dipòsit (baixar els canvis i actualitzar el directori de treball).

5.4.4. Canvis en el dipòsit

Durant el procés normal de treball en un projecte, és habitual modificar fitxers a mesura que introduïm millores. Una vegada que hem acabat els canvis, aquests s'han d'enviar al directori perquè tots els usuaris puguin accedir a les millores que hem efectuat.

Abans d'enviar els canvis, resulta interessant veure exactament quins canvis hem fet. Per a això, tenim l'ordre `diff`, que segueix la sintaxi següent:

```
$cvs diff -u nombre_archivo
```

Aquesta ordre ens mostra les diferències del nostre directori de treball respecte al dipòsit, d'aquesta manera podem veure quins canvis hem fet exactament. Fins ara, hem vist com determinar les diferències. Ara veurem com enviar aquestes diferències al directori: l'ordre `commit` és l'encarregada d'enviar els canvis.

Per exemple, modifiquem el fitxer README.TXT del projecte Mozilla que hem baixat prèviament i que es troba en el directori principal. Tot seguit, introduïm:

```
$cvs commit
```

Després de processar tots els fitxers del projecte a la recerca de fitxers amb modificacions, mostrarà una llista dels canvis realitzats; per defecte, aquesta llista es mostra en l'editor *vi*.

Podem definir la variable d'entorn CVSEDITOR per a canviar per defecte l'editor que volem usar. Com a resposta a l'ordre `commit`, CVS mostrarà:

ANOTACIONS

```
CVS: ----------------------------------------------------------------------
CVS: Enter Log. Lines beginning with `CVS:' are removed automatically
CVS:
CVS: Committing in .
CVS:
CVS: Modified Files:
CVS: README.txt
CVS: ----------------------------------------------------------------------
```

Si volem continuar, escriurem: `!q`, que és l'ordre de l'editor *vi* per a sortir sense desar els canvis del registre que se'ns acaba de mostrar. A continuació, ens preguntarà si volem enviar els canvis al servidor. CVS ens respondrà:

```
cvs [server aborted]: "commit" requires write access to the repository
```

Cosa que és normal en aquest cas, ja que com a usuaris anònims del projecte Mozilla no tenim dret de modificar els fitxers. L'ordre `commit` té el paràmetre `-m` (missatge), que és molt utilitzada per a indicar comentaris i, en alguns projectes, és gairebé d'ús obligat. Per exemple:

```
cvs commit -m "Resoldre el problema amb l'script d'instal·lació'
```

Els canvis dels fitxers modificats seran enviats al dipòsit i seran registrats amb el comentari indicat. Els comentaris resulten fonamentals durant el projecte per a entendre per què nosaltres o altres persones hem introduït canvis en el projecte.

5.4.5. Publicar canvis amb diff i patch

No és estrany que en un projecte del qual no som col·laboradors habituals trobem un error en el programa, fins i tot és probable que tinguem la solució i la vulguem enviar als responsables del projecte. Com és lògic, no tindrem accés d'escriptura al dipòsit de CVS del projecte, de manera que no podrem publicar la solució en el directori directament. Tanmateix, sí que podem publicar la nostra solució al

ANOTACIONS

problema en una llista de correu o enviar-la per correu electrònic al responsable del projecte. Per a aquesta i altres situacions, hi ha la parella d'ordres `diff` i `patch`.

Imaginem que hem modificat un fitxer per a corregir un error. Per a crear un fitxer amb les diferències que hem introduït, farem:

```
$diff -c nom_primer_fitxer nom_segon_fitxer > fix.patch
```

Per defecte, l'ordre diff envia la sortida a la pantalla, per això redirigirem la sortida al fitxer fix.patch. Aquest fitxer contindrà els canvis.

La persona que rebi el fitxer fix.patch podrà recuperar les modificacions que hem fet utilitzant l'ordre patch, de la manera següent:

```
$patch < fitxer_per_a_aplicar_pedacos < pedacos
```

Com hem pogut observar, aquesta és una manera molt senzilla i eficient d'enviar correccions als altres usuaris i és una pràctica molt comuna en projectes de programari lliure.

5.5. Crear i administrar dipòsits

Crear un dipòsit és una tasca relativament senzilla. Primer hem d'escollir en quin directori del sistema de fitxers ubicarem el dipòsit. Un bon lloc pot ser /var/lib/cvs. És important tenir accés d'escriptura al directori perquè, quan executem CVS, pugui crear els fitxers necessaris.

Una vegada que tenim els permisos, creem el directori en què s'ubicarà el dipòsit:

```
$mkdir cvs
```

Tindrem el directori /var/lib/cvs. Com veurem més endavant, CVS utilitza la gestió d'usuaris i permisos del sistema operatiu. Les ordres

ANOTACIONS

CVS són executats amb l'identificador de l'usuari que executa l'ordre. És molt important que ens assegurem que tots els usuaris tindran accés al dipòsit. Un mètode senzill és assignar al directori el grup users perquè tots els usuaris que pertanyen a aquest grup puguin accedir al dipòsit:

```
$chgrp users /var/lib/cvs
```

Després, hem d'assignar a aquest grup tots els usuaris que volem que tinguin accés al dipòsit.

Ara ja ho tenim tot preparat perquè CVS inicialitzi el dipòsit i creï els fitxers necessaris per a gestionar-los:

```
$ cvs -d /var/lib/cvs init
```

L'ordre `init` crearà el dipòsit en el directori central i inicialitzarà els fitxers necessaris. El paràmetre `-d` és necessari, ja que indica el directori en què s'ubicarà el dipòsit. L'ordre no retorna cap resultat, però podrem observar com ha creat un directori a `/var/lib/cvs/CVSROOT` que conté tots els fitxers que necessita CVS per a gestionar el dipòsit.

Per a poder compartir el dipòsit amb els altres usuaris, ens interessa executar CVS en mode servidor. CVS utilitza el port TCP 2401. Per a instal·lar aquest servei, seguirem els passos següents:

1) Editem el fitxer de configuració de CVS, anomenat *cvs.conf*. Aquest fitxer es troba en el directori `/var/lib/cvs`. Hem d'afegir la línia següent que indicarà on és el dipòsit:

```
CVS_REPOS="/var/lib/cvs"
```

2) Necessitem activar el servei. Per a això, haurem de crear un fitxer anomenat cvspserver en el directori `/etc/xinetd.d`. Aquest fitxer indica la descripció del servei.

ANOTACIONS

Nota

És important comprovar quin CVS hi ha realment instal·lat a /usr/bin/cvs.

```
service cvspserver
{
port = 2401
disable = no
socket_type = stream
protocol = tcp
wait = no
user = root
server = /usr/local/bin/cvs
server_args = -f --allow-roof = /var/lib/cvs pserver
}
```

Una vegada activat el dimoni i creat el dipòsit, ens podem connectar com hem fet fins ara. Algunes distribucions com Debian usen el dimoni *inetd* i la forma de configuració difereix de la comentada.

```
$ CVSROOT=:pserver:anonymous@NOM_DEL_NOSTREPC:/var/lib/cv
$ export CVSROOT
$ cvs login
```

En la línia `CVSROOT`, hem de substituir `NOM_DEL_NOSTREPC` pel nom de la nostra màquina GNU/Linux.

Per a identificar-nos, qualsevol nom d'usuari –amb els privilegis adequats i amb la contrasenya corresponent– que tinguem definit en el sistema en què hem instal·lat el servei serà vàlid per a accedir al dipòsit.

5.5.1. Tipus de connexió al dipòsit

Hem vist la connexió al dipòsit mitjançant el mètode `pserver`. Aquest sistema és molt bàsic i es basa a enviar les contrasenyes sense xifrar, de manera molt semblant a com ho fan els protocols associats amb les aplicacions Telnet o FTP. Aquest nivell de seguretat no és suficient per a projectes en què garantir la seguretat del dipòsit sigui un factor important. També hem de recordar que els usuaris de CVS són vàlids en la mateixa màquina, per la qual cosa un problema de seguretat amb CVS s'estén a tota la màquina.

CVS proporciona la possibilitat de treballar amb mètodes externs d'autentificació per a millorar la seguretat. El sistema més estès és

ANOTACIONS

SSH (*secure shell*), que permet connexions xifrades: la contrasenya viatja xifrada per la Xarxa, i a més permet assegurar-nos que l'hoste al que ens connectem és el correcte. Els dipòsits de SourceGear, per exemple, només accepten connexions amb un mètode segur, en concret SSH.

Nota

www.sourcegear.net

Per a activar la connexió mitjançant SSH, cal definir la variable d'entorn CVS_SSH de la manera següent:

```
$ CVS_SSH=ssh
$ export CVS_SSH
```

El mètode de connexió, com hem vist, es defineix en la variable CVSROOT. Utilitzar SSH requereix instal·lar correctament el programari OpenSSH.

5.5.2. Importar projectes ja existents

Una vegada que el dipòsit s'ha inicialitzat, podem crear l'estructura de directoris i un projecte nou, o bé importar un projecte ja existent. En qualsevol cas, és important que decidim prèviament l'estructura de directoris que volem que tingui el nostre projecte, ja que moure fitxers amb CVS no és gaire còmode.

L'ordre `import` de CVS ens facilita importar projectes ja existents al dipòsit. Amb `import`, podem importar un fitxer o conjunt de fitxers com a part d'un projecte. Per exemple, importem tot el directori /home/jordi

```
$cvs import directori vendortag releasetag
```

En el nostre cas:

```
$cvs import codi vendortag releasetag
```

En què CVS respon:

```
I codi/simul.c
I codi/comun.c
I codi/corto/etapas.c
```

ANOTACIONS

Depenent del nombre i nom de fitxers que contingui el directori.

Els paràmetres `vendortag` i `releasetag` s'utilitzen per a controlar versions; en el nostre cas, utilitzem dos noms arbitraris.

5.5.3. Afegir fitxers o directoris

Per a afegir fitxers a un repertori CVS, disposem de l'ordre `add`, la qual permet tant afegir un fitxer o grup de fitxers com afegir un directori complet. Per a afegir un fitxer, escriurem:

```
$ cvs add nom_de_fitxer
```

CVS respon:

```
cvs server: scheduling file `nom_de_fitxer' for addition
cvs server: use 'cvs commit' to add this file permanently
```

Els fitxers no s'afegiran de manera definitiva al dipòsit fins que no realitzem una ordre `commit` per a enviar tots els canvis al servidor, és a dir, fins que executem:

```
$ cvs commit
```

que enviarà els canvis al servidor i els farà efectius.

5.5.4. Els fitxers binaris

La majoria d'eines relacionades amb CVS que hem vist funcionen sobre fitxers de text, ja que de fet CVS va ser dissenyat originalment per a això. Tanmateix, en un projecte qualsevol és habitual trobar fitxers que no són pròpiament de text, aquells que anomenem *binaris*. Entre els fitxers binaris més habituals, tenim els fitxers

ANOTACIONS

gràfics, de so, o versions d'un programari compilades que mantenim en el CVS.

El problema principal resideix en el fet que CVS utilitza dues característiques –l'expansió de paraules clau i la conversió de tecles de retorn– que, aplicades a fitxers binaris, en modificaran el contingut i els faran inservibles.

Si afegim un fitxer binari, és important desactivar aquestes funcions per a aquest fitxer. Per a afegir fitxers binaris, hem de fer:

```
$cvs add -kb nom_del_fitxer
```

El paràmetre `-kb` desactiva les característiques que poden fer malbé un fitxer binari. Com en el cas dels fitxers de textos, els canvis no s'enviaran fins que no executem l'ordre `commit`.

5.5.5. Eliminar fitxers i directoris

Eliminar fitxers del dipòsit és una operació que hem d'efectuar amb precaució. Per a poder eliminar un fitxer, primer l'hem d'eliminar del nostre directori de treball –és a dir, del nostre disc– i després del dipòsit. La seqüència d'instruccions és la següent:

```
$ rm nom_de_fitxer
$ cvs remove nom_de_fitxer
```

Com en el cas d'afegir fitxers, l'operació no es completa fins que no executem l'ordre CVS `commit`.

El mètode vist fins ara serveix per a eliminar fitxers, però no per a eliminar directoris. De fet, CVS no permet eliminar directoris a causa de la manera en què organitza internament els dipòsits. L'únic que podem fer és eliminar tots els fitxers del directori, deixar-lo buit, i utilitzar sempre el paràmetre `-P` de l'ordre `update` quan ens sincronitzem amb el dipòsit, perquè no es creïn en el nostre directori de treball els directoris buits.

ANOTACIONS

5.5.6. Moure fitxers

CVS no disposa de cap ordre específica per a poder moure fitxers d'un lloc del dipòsit a un altre. Tanmateix, podem emular aquesta funcionalitat combinant diverses operacions de CVS.

Per a moure un fitxer, el copiarem del seu lloc d'origen en el nostre directori de treball al lloc de destinació, és a dir, el lloc on volem que hi sigui en el futur. Una vegada realitzada aquesta operació sobre el dipòsit, eliminarem el fitxer del lloc d'origen i afegirem el fitxer nou.

La seqüència d'ordres és la següent:

```
$mv dirorigen/origen dirdestinacio/destinacio
$cvs add dirdestinacio/destinacio
$cvs remove dirorigen/origen
```

Els canvis no es faran efectius fins que no s'executi l'ordre `commit`. És important destacar que amb aquesta operació perdrem l'històric d'etiquetes i comentaris realitzats en el dipòsit, ja que efectivament incloem un fitxer nou.

5.6. Treballar amb versions, etiquetes i branques

5.6.1. Etiquetes i revisions

Desenvolupar un projecte és un procés viu en què constantment es produeixen canvis com a part del procés continu de millora. En alguns punts del projecte, interessa tenir la capacitat de poder indicar que en aquell moment precís els fitxers que hi ha al dipòsit formen una versió determinada. Aquest procés es diu *tag* (etiqueta). S'utilitza molt per a marcar una versió exacta que alliberem per a després poder referir-nos-hi de manera senzilla.

Podem establir una *etiqueta* en el dipòsit fàcilment:

```
$ cvs tag Versio100
```

ANOTACIONS

D'aquesta manera, etiquetarem tots els fitxers del projecte amb l'etiqueta "Versio100". A mesura que el projecte evoluciona, pot ser necessari recuperar una versió que havíem marcat anteriorment. Per a això, utilitzarem l'ordre `update` que ja coneixem:

```
$ cvs update -r Versio100
```

D'aquesta manera, obtindrem una còpia nova del projecte tal com era en el moment de marcar-lo com a "Versio100". Tanmateix, això representa un petit inconvenient, ja que el directori de treball retrocedirà completament a l'estat en què es trobava quan fem l'etiqueta, probablement fa uns mesos. Acostuma a ser molt més pràctic descarregar una versió determinada en un directori a banda; ho podem fer de la manera següent:

```
$ cvs checkout -r Versio100
```

També és possible realitzar les mateixes operacions basades en la data del fitxer. Resulta més intuïtiu treballar amb etiquetes, ja que són noms que poden tenir un significat i són més fàcils d'associar que les dates.

5.6.2. Crear branques

Crear branques és molt útil quan treballem en paral·lel amb diverses versions del projecte. Per exemple, si alliberem una versió 1.0 d'un producte. En aquell moment, creem dues branques del projecte, una en què hi ha la versió 1.0 i en què podem continuar corregint els possibles errors que apareguin i crear més endavant una versió 1.01. Alhora, podem tenir una altra branca en paral·lel en què el projecte ja ha evolucionat a la versió 1.1 i encara es troba en fase de desenvolupament. Les branques permeten treballar amb diverses versions del mateix projecte de manera concurrent.

Podem crear una branca amb l'ordre següent:

```
cvs tag -b nom_de_la_branca fitxer
```

ANOTACIONS

A partir d'aquell moment, podem continuar operant amb la branca principal de CVS i quan ens vulguem referir a la branca creada anteriorment, ho podrem fer usant el `nom_de_la_branca` escollit.

5.7. Bonsai: gestió de CVS amb el web

Nota

A http://www.mozilla.org/bonsai.html podeu trobar la pàgina principal del projecte Bonsai, amb el seu codi font i instruccions detallades per a instal·lar-lo.

Bonsai és una eina que va ser creada en l'entorn del projecte de programari lliure Mozilla. Aquesta eina permet gestionar amb el web un dipòsit CVS. La seva utilitat resideix en el fet que afegeix una interfície visual i és extremadament útil per a fer seguiments de projectes en què hi ha canvis constants en el dipòsit.

En aquest capítol, ens limitarem a veure'n la funcionalitat des de la perspectiva de l'usuari.

Entre les característiques que ofereix Bonsai, destaquen:

- Possibilitat de veure els darrers canvis en el dipòsit, qui els ha efectuat, a quina hora, el comentari del canvi, etc.
- Veure quins canvis s'han efectuat per un usuari determinat, en una branca del projecte, o en un període de temps.
- Veure un fitxer determinat amb anotacions de qui va ser la darrera persona que va canviar cada línia del fitxer.
- Comparació visual dels diferents canvis que s'han produït en el dipòsit.

Bonsai només permet fer el seguiment del projecte, però no permet fer-hi canvis. Mitjançant la pàgina http://bonsai.mozilla.org/, podem efectuar totes les operacions descrites sobre el dipòsit del projecte lliure Mozilla.

Taula d'ordres de CVS

La taula que ve a continuació conté una descripció de les ordres principals de CVS.

ANOTACIONS

Taula 1. Descripció de les ordres principals de CVS

Ordre	Descripció
add	Afegeix un fitxer nou o un conjunt de fitxers d'un directori al dipòsit.
admin	Interfície d'administració.
annotate	Mostra el fitxer indicant el número de revisió de cada línia del fitxer i quin usuari va ser l'últim que la va modificar.
checkout	Obté una còpia dels fitxers del dipòsit i crea el nostre directori de treball.
commit	Acció amb què enviem els canvis que hem realitzat en el nostre directori de treball al dipòsit central del servidor.
diff	Mostra les diferències d'un fitxer o grup de fitxers entre el nostre directori de treball al dipòsit central del servidor.
edit	Edita els fitxers sota observació.
editors	Mostra qui edita fitxers sota observació.
export	Exporta fitxers de CVS, semblant a *checkout*.
history	Mostra l'històric d'accés d'usuaris al dipòsit CVS.
import	Importa fitxers al dipòsit CVS.
init	Crea i inicialitza un dipòsit nou.
log	Mostra l'historial dels fitxers.
login	Autentifica l'usuari. Amb aquesta ordre, introduïm la contrasenya del nostre usuari i ens valida en el servidor.
logout	Desconnecta l'usuari.
pserver	Servidor en mode contrasenya.
rannotate	Mostra el fitxer indicant qui ha modificat cada línia del fitxer.
rdiff	Crea un pedaç amb la diferència entre dues versions d'un fitxer.
release	Indica que el mòdul ja no s'usa.
remove	Elimina un objecte del dipòsit.
rlog	Mostra l'històric per a un mòdul del dipòsit.
rtag	Afegeix una etiqueta.
server	Mode servidor.
status	Mostra la informació d'estat dels fitxers.
tag	Afegeix una etiqueta als fitxers del dipòsit.
unedit	Desfà una modificació realitzada per una ordre d'edició.
update	Actualitza el directori local de treball amb la còpia en el servidor.
version	Mostra la versió actual del programa CVS.
watch	Estableix els fitxers observats.
watchers	Mostra els usuaris que estan observant fitxers.

5.7.1. Subversion

Subversion és un programari lliure de control de versions que va néixer l'any 2001 amb l'objectiu de resoldre les mancances principals de CVS. Molts desenvolupadors veuen Subversion com el sistema de control de versions que substituirà CVS.

Entre les millores principals de Subversion respecte de CVS, destaquen:

- Possibilitat de moure fitxers. Una de les grans limitacions de CVS ha estat la manca d'una ordre per a poder moure fitxers entre diferents parts del dipòsit. Subversion finalment corregeix aquesta mancança.

- *Commits* atòmics. Fa possible enviar un conjunt de canvis al dipòsit en bloc, amb la qual cosa s'eviten els problemes que s'esdevenen en CVS quan enviem una gran quantitat de canvis d'un mateix treball al dipòsit i només una part d'aquests són acceptats per CVS.

- Metadades. Subversion permet desar per cada fitxer o directori un conjunt de claus aparellades amb el seu valor que ens permeten emmagatzemar metadades i després recuperar-les.

- Versions per directoris. CVS només ofereix control de versions per fitxer, amb Subversion podem tenir també control per directori.

- Suport per a diferents transports de xarxa. Subversion ha estat dissenyat perquè sigui molt senzill afegir nous transports de xarxa, com ara connexions segures mitjançant SSH o mitjançant WebDAV.

- La comparació de fitxers es realitza amb un algorisme binari que és molt més efectiu.

5.7.2. Instal·lar Subversion

Per mitjà del lloc web http://subversion.tigris.org/, es poden obtenir les diferents versions client i servidor del programa Subversion, i també el seu codi font. La majoria de distribucions GNU/

ANOTACIONS

Linux inclouen paquets preparats amb les eines Subversion. Podem descarregar d'aquest lloc web la versió específica de la nostra distribució GNU/Linux o bé baixar el codi font, compilar-lo i instal·lar-lo nosaltres mateixos com faríem amb qualsevol altre paquet de programari.

Subversion instal·la diverses eines, però les principals són:

- **svn**. El client de Subversion. Amb aquest, realitzarem les operacions principals com ara obtenir una còpia de treball, sincronitzar la nostra còpia amb el servidor o enviar els canvis realitzats.

- **svnadmin**. L'eina per a crear i administrar dipòsits. Aquesta és l'eina principal d'administració de Subversion.

- **svnserve**. Servidor lleuger que funciona com a dimoni, usualment en el port 3690, que permet a clients svn connectar-se amb connexions TCP/IP mitjançant els protocols svn:// i svn+ssh://.

- **svnlook**. Eina d'administració de dipòsits que ens permet examinar l'històric del dipòsit.

Hi ha també alguns clients gràfics per a diferents plataformes:

- **TortoiseSVN**

És un client gràfic de Subversion per a Microsoft Windows implementat com una extensió de l'intèrpret d'ordres de Windows.

Nota

http://tortoisesvn.tigris.org/download.html

- **Rapid SVN**

Rapid SVN és un client SVN multiplataforma que funciona sobre GNU/Linux, Mac i Windows i que proporciona un entorn senzill per a treballar amb Subversion.

Nota

http://rapidsvn.tigris.org/

ANOTACIONS

5.7.3. Obtenir un directori de treball d'un projecte ja existent

Per a poder treballar amb un projecte que utilitza Subversion, necessitem començar obtenint un directori de treball dels fitxers del projecte. Igual que amb CVS, usem l'ordre `checkout` per a aquest propòsit. Ho podem fer de la manera següent:

```
$ svn checkout diposit
```

On *diposit* és el dipòsit i mòdul del qual obtindrem una còpia. Subversion utilitza URI per a especificar les rutes dels dipòsits, cosa que permet utilitzar una manera estàndard per a descriure localitzacions a les quals podem accedir amb diferents transports, com ara file:/// (fitxer local), http:// (via web) o https:// (via web amb connexió segura).

Nota

http://www.mono-project.com

Per exemple, per a obtenir una còpia del dipòsit Subversion del projecte Mono, usarem:

```
$ svn checkout svn+ssh://user@mono-cvs.ximian.com/source/trunk/mono
```

On `svn+ssh` és el tipus de transport, `user` l'usuari que utilitzarem, i la resta, la ruta a la màquina en la xarxa que conté el dipòsit.

5.7.4. Crear dipòsits

Una vegada instal·lat Subversion, hem de crear un dipòsit en què s'emmagatzemaran els nostres fitxers. Per a això, primer ens situarem en el directori en què volem crear el dipòsit i després usarem l'eina d'administració `svnadmin` per a crear-lo amb les ordres següents:

```
$mkdir diposit
$svnadmin create diposit
```

ANOTACIONS

Nota

Per a la resta dels exemples assumim que el directori dipòsit s'ha creat en el subdirectori /home/jordi/subversion, per la qual cosa el directori complet en el dipòsit és /home/jordi/subversion/diposit.

La segona ordre crea tots els fitxers necessaris per a controlar el dipòsit en el directori anomenat *dipòsit*, que és on resideix el dipòsit que acabem de crear.

A continuació, el més usual és importar al nostre dipòsit un projecte que ja tinguem.

L'ordre `import` ens facilita aquesta tasca. Amb `import`, podem importar un fitxer o conjunt de fitxers com a part d'un projecte. Per exemple, importem tot el directori /home/jordi al dipòsit que hem creat. Per a això, escrivim:

```
$svn import /home/jordi/ file:///home/jordi/subversion/dipòsit
```

L'ordre mostrarà tots els fitxers que s'han importat al dipòsit.

Ara, hem d'especificar els usuaris que tindran permís per a modificar el dipòsit. Per a fer això, la manera més senzilla és crear un grup d'usuaris i assignar tots els usuaris que vulguem que tinguin accés de modificació al dipòsit a l'empara d'aquest grup. Per exemple, creem el grup *usuarissvn* escrivint:

```
$groupadd usuarissvn
```

I a continuació canviem el directori del dipòsit de grup:

```
$chgrp -R usuarissvn diposit
```

Amb aquesta ordre, el directori dipòsit passa a pertànyer al grup d'usuaris *usuarissvn*. A continuació, podem editar el fitxer /etc/groups per a afegir els usuaris que volem que tinguin accés en escriptura al dipòsit al grup *usuarissvn*.

ANOTACIONS

Una vegada creat i configurat el dipòsit i els seus usuaris, podem obtenir un directori de treball local escrivint:

```
svn checkout svn://usuari@adreca/home/jordi/subversion/diposit
```

On *usuari* representa l'usuari que utilitzem per a autentificar-nos en el dipòsit, i *adreca*, l'adreça de la màquina en què ens connectem, per exemple, una adreça IP. Per a poder connectar-nos-hi, és important iniciar el servidor Subversion escrivint:

```
svnverve -d
```

5.7.5. Ordres bàsiques amb Subversion

A continuació, veurem algunes de les ordres bàsiques de Subversion que necessitem en un projecte. Pràcticament totes les ordres de Subversion tenen el mateix nom i sintaxi que les ordres corresponents en CVS. Això és així per a minimitzar la corba d'aprenentatge dels usuaris que migren del sistema CVS a Subversion. No entrarem en detall en aquelles ordres que són de comportament idèntic.

Per a actualitzar el nostre directori de treball, igual que amb CVS usarem l'ordre `up` (abreviació d'*update*):

```
$svn up
```

Durant aquest procés de sincronització, només els fitxers que han estat modificats són actualitzats. Aquesta ordre no envia en cap cas les modificacions que hàgim realitzat en el nostre directori de treball, sinó que simplement actualitza el nostre directori de treball amb els canvis del dipòsit.

Per a enviar al dipòsit els canvis que hàgim realitzat usarem l'ordre `commit`:

```
$svs commit
```

Aquesta ordre envia els canvis dels fitxers modificats i els registra amb el comentari indicat. Els comentaris resulten fonamentals durant el projecte per a entendre per què nosaltres o altres persones hem introduït canvis en el projecte.

Per a veure les diferències del nostre directori de treball amb el dipòsit, usarem l'ordre `diff`. Escrivim:

```
$svn diff
```

Aquesta ordre mostrarà la llista detallada de canvis que hem introduït en el nostre directori de treball.

La taula que ve a continuació conté una descripció de les ordres principals de Subversion (ordre `svn`).

Taula d'ordres de Subversion

Taula 2. Descripció de les ordres principals de Subversion (ordre svn)

Ordre	Descripció
add	Afegeix un fitxer nou o un conjunt de fitxers d'un directori al dipòsit.
blame (praise, annotate, ann)	Mostra la darrera revisió del fitxer indicant el número de revisió de cada línia del fitxer i quin usuari va ser l'últim que la va modificar.
cat	Examina les diferències entre diferents versions d'un fitxer.
checkout (co)	Aquesta operació és la que realitzem quan volem obtenir una còpia dels fitxers del dipòsit en la nostra màquina.
cleanup	Neteja el nostre directori de treball de possibles operacions no finalitzades o fitxers bloquejats.
commit	Acció amb què enviem els canvis que hem realitzat en el nostre directori de treball al dipòsit central del servidor.
copy (cp)	Copia un fitxer en el nostre directori de treball o dipòsit mantenint-ne l'històric.
delete (del, remove, rm)	Elimina fitxers i directoris permanentment del dipòsit.
diff (di)	Exporta una còpia del dipòsit.
export	Mostra l'ajuda del programa.
help (?, h)	Importa fitxers al dipòsit.
import	Mostra informació sobre el nostre directori de treball del dipòsit.
info	Mostra els fitxers que hi ha al dipòsit.

Ordre	Descripció
list (ls)	Mostren el fitxer de registre per a un fitxer o conjunt de fitxers.
log	Aplica les diferències entre dues versions d'un fitxer al nostre directori de treball.
merge	Exporta una còpia del dipòsit.
mkdir	Crea un directori en el dipòsit.
move (mv, rename, ren)	Mou o canvia de nom un fitxer en el dipòsit.
propdel (pdel, pd)	Elimina una propietat i el seu valor d'un fitxer, conjunt de fitxers o directori.
propedit (pedit, pe)	Edita una propietat usant un editor extern.
propget (pget, pg)	Mostra el valor d'una propietat en un fitxer, conjunt de fitxers o directori.
proplist (plist, pl)	Llista totes les propietats en un fitxer, conjunt de fitxers o directori.
propset (pset, ps)	Estableix el valor d'una propietat en un fitxer, conjunt de fitxers o directori.
resolved	Elimina l'estat de conflicte en fitxers i directoris.
revert	Desfà els canvis introduïts en un directori de treball d'un fitxer.
status (stat, st)	Mostra l'estat de fitxers i directoris.

5.8. Conclusió

Els sistemes de control de versions són la columna vertebral, perquè propicien que grups de persones treballin conjuntament en el desenvolupament de projectes. Aquests sistemes permeten a milers d'usuaris en localitzacions geogràfiques diferents treballar col·laborativament en el desenvolupament de programari lliure.

Hem après a usar tant la part client com el servidor de CVS (*concurrent versions system*), que és el sistema de control de versions utilitzat actualment en el món del programari lliure. Hem vist també Subversion, un producte de control de versions relativament jove, que va néixer l'any 2001 i que té com a objectiu resoldre les mancances principals de CVS, i que ja han començat a adoptar molts projectes en substitució de CVS.

Amb els coneixements adquirits d'ambdós productes, som perfectament capaços de poder obtenir una còpia de treball de qualsevol projecte lliure, enviar millores o instal·lar el nostre propi servei de control de versions.

ANOTACIONS

5.9. Altres fonts de referència i informació

Control de versions amb Subversion (versió lliure).
<http://svnbook.red-bean.com/>

Notes d'ús de CVS. <http://www.cs.columbia.edu/~hgs/cvs/>

Open Source Development with CVS (3a. ed.) (versió lliure).
<http://cvsbook.red-bean.com/>

Source Control.
<http://software.ericsink.com/scm/source_control.html>

Targeta de referència amb les ordres i opcions de CVS.
<http://refcards.com/refcards/cvs/index.html>

ANOTACIONS

6. Gestió de programari

6.1. Introducció

L'avenç del sistema operatiu GNU/Linux, en aquests moments l'exponent màxim del programari lliure, ha fet evolucionar la manera amb què el programari és distribuït en aquest tipus d'entorns. El motiu principal ha estat la necessitat d'empaquetar de manera homogènia i consistent el programari juntament amb la base del sistema operatiu.

La gran varietat de distribucions de GNU/Linux dirigides a diferents fins i usuaris també està fent que es desenvolupin sistemes nous de distribució de programari que, encara que amb una base comuna, difereixen en la manera d'organitzar i relacionar el programari entre ells i amb els seus components interns.

6.2. Objectius

Amb la lectura d'aquest capítol, els lectors heu de ser capaços d'assolir els objectius següents:

- Analitzar els sistemes de distribució i compressió tradicionals, de manera individual i en conjunció, per a posteriorment estudiar els dos sistemes de distribució principals, RPM i *deb*, entrant no només en el seu maneig, sinó també en la seva creació.

- Conèixer els principals processadors d'accés (*front-end*) o programaris gràfics de gestió de paquets per a apropar-vos més a la realitat existent.

Els lectors, per tant, haureu d'acabar el capítol coneixent i manejant tots els sistemes exposats, tant des del punt de vista funcional com de creació.

ANOTACIONS

6.3. L'empaquetador universal: tar

Un dels problemes principals a l'hora de distribuir, copiar o emmagatzemar fitxers és el de mantenir intactes totes les metadades associades, com ara la data de creació, els permisos d'execució o l'usuari i grup a què pertanyen. En la distribució, també és recomanable que cada element de programari sigui presentat com un fitxer únic identificable fàcilment.

Per a totes aquestes necessitats, hi ha la utilitat `tar`.

Encara que originalment va ser pensada per a realitzar còpies de suport en casset (*Tape ARchiver* o arxivador en cinta), el seu ús en entorns Linux ha estat motivat principalment per la seva facilitat de maneig i versatilitat en la tasca concreta de realitzar còpies idèntiques, tant de fitxers com d'estructures de directori completes.

La sintaxi bàsica de la utilitat `tar` és la següent:

```
tar mode [opcions] [-f fitxer_tar] [fitxers_origen]
```

La utilitat `tar` pot operar tant sobre fitxers especificats explícitament com sobre l'entrada i sortida estàndard del programa (per exemple, amb readreces).

Contingut complementari

Utilitat tar

c: Crear

x: eXtreure

t: lisTar

Les opcions es componen d'una ordre principal (mode operació) que defineix si s'està creant (`c`) el contingut del fitxer empaquetat, s'està extraient (`x`) o se n'està fent una llista (`t`), etc.

Amb l'opció `f nom_fitxer` especifiquem que s'operarà sobre un fitxer `tar`, i no sobre l'entrada/sortida estàndard.

Nota

Per motius de seguretat, els exemples no es fan com a superusuari (***root user***), , llevat que es digui el contrari.

Afegint `v` (*verbose* = 'detallat') obtenim un informe de les operacions que realitza `tar`, la qual cosa pot ser especialment útil per a detectar errors, o simplement per a mantenir la calma quan es manegen grans quantitats de dades i sembla que `tar` "no fa res".

Vegem un exemple del seu ús.

ANOTACIONS

Començarem creant un directori temporal:

```
$ mkdir /tmp/test
$ cd /tmp/test
```

Per a omplir-lo ràpidament de fitxers, podem fer:

```
$ cp /dev/null /tmp/test/fitxer1
$ cp -r /tmp/test/ /tmp/test/directori
$ chmod 400 /tmp/test/fitxer1
$ ls -l /tmp/test
total 0
drwxr-xr-x 2 user user 60 Nov. 17 18:03 directori/
-r-------- 1 user user 0 nov. 17 18:03 fitxer1
```

Ara suposem que volem fer una còpia de seguretat del directori `test`; podríem fer el següent:

```
$ cd /tmp
$ tar cvf /tmp/test.tar test
test/
test/directori/
test/directori/fitxer1
test/fitxer1
$ ls -l /tmp/test.tar
-rw-r--r-- 1 user user 10240 nov. 17 18:03 /tmp/test.tar
```

Si ara esborréssim el directori `test`:

```
$ rm -rf /tmp/test
$ ls test
ls: test: No such file or directory
```

El podríem restaurar completament des del fitxer `test.tar`:

```
$ cd /tmp
$ tar xvf /tmp/test.tar
test/
test/directori/
test/directorio/fitxer1
test/fitxer1
$ ls -l /tmp/test
total 0
drwxr-xr-x 2 user user 60 Nov. 17 18:03 directori/
-r-------- 1 user user 0 nov. 17 18:03 fitxer1
```

ANOTACIONS

Es pot apreciar que el fitxer `/tmp/test/fitxer1` s'ha restaurat amb els mateixos permisos, usuari i data que tenia quan va ser emmagatzemat amb `tar`.

També podem consultar en qualsevol moment quin és el contingut d'un fitxer `tar`:

```
$ tar tvf test.tar
drwxr-xr-x user/user      0 2004-11-17 18:03:49 test/
drwxr-xr-x user/user      0 2004-11-17 18:03:49 test/directori/
-rw-r--r-- user/user      0  2004-11-17  18:03:49  test/directori/
fitxer1
-r-------- user/user      0 2004-11-17 18:03:48 test/fitxer1
```

6.3.1. Comprimir: gzip

La majoria dels fitxers emmagatzemats en un ordinador ho solen estar de tal manera que es poden llegir del disc i utilitzar al més aviat possible. Això implica que aparegui informació emmagatzemada en blocs de mida fixa (independentment de si són plens o no), informació d'un conjunt de caràcters reduït (per exemple, text llegible directament) i altres formes d'emmagatzemament en què no es té massa en compte l'espai que ocupen.

Tanmateix, tota informació pot ser codificada per a reduir l'espai que ocupa sense perdre part d'aquesta. Es tracta de fitxers de text, executables o de qualsevol altre tipus, ja que en aquests sempre apareixen seqüències de bytes que es repeteixen i que són susceptibles de ser comprimides ("simplificades").

Una eina molt útil per a comprimir i descomprimir formats diferents basats en la codificació Lempel-Ziv és la utilitat `gzip/gunzip`.

Basada en un document de 1977 per J. Ziv i A. Lempel sobre un algorisme universal de compressió seqüencial, aquesta codificació permet comprimir i descomprimir els fitxers directament mentre es llegeix byte rere byte.

Igual que la majoria de les utilitats del sistema, permet comprimir i descomprimir tant des d'entrada estàndard com des de fitxer, registrant el resultat en un altre fitxer o en la sortida estàndard.

ANOTACIONS

La sintaxi bàsica és molt simple (en ordre: comprimir, descomprimir):

```
gzip nom_fitxer
gunzip nom_fitxer.gz
```

Si no s'especifiquen més opcions, `gzip` comprimeix el fitxer `nom_fitxer` i el reemplaça amb el resultat desat afegint l'extensió `.gz` com a `nom_fitxer.gz`.

Inversament, `gunzip` descomprimeix el fitxer `nom_fitxer.gz` i el reemplaça amb el resultat desat extraient l'extensió `.gz` i el deixa com a `nom_fitxer`.

A causa d'aquest funcionament i la capacitat de compressió seqüencial, és el complement ideal per a `tar`.

En els exemples anteriors, només s'havien manejat fitxers buits, i tot i així es pot observar que el fitxer `.tar` ocupa més de 10.000 bytes. Això és perquè internament la informació s'emmagatzema en blocs de com a mínim 512 bytes (inclòs un bloc per fitxer amb les metadades) i es desa en blocs de 10.240 bytes.

Normalment, això no suposa un problema amb fitxers de mida considerable; tanmateix, és un factor que cal tenir en compte si es vol emmagatzemar un gran nombre de fitxers petits.

En tot cas, vegem com podem usar `gzip/gunzip` tant separadament com juntament amb `tar`.

Suposem que ara, una vegada creat `/tmp/test.tar`, volem fer que ocupi menys espai. Només cal executar:

```
$ ls -l /tmp/test.tar
-rw-r--r-- 1 user user 10240 nov. 17 18:03 /tmp/test.tar
$ gzip /tmp/test.tar
```

I obtindrem un fitxer comprimit:

```
$ ls -l /tmp/test.tar.gz
-rw-r--r-- 1 user user 191 nov. 17 18:03 /tmp/test.tar.gz
```

ANOTACIONS

Observeu que en comprimir el fitxer NO s'ha canviat ni l'usuari, ni els permisos, ni la data d'aquest. En canvi, sí que s'ha esborrat la còpia original:

```
$ ls -l /tmp/test.tar
ls: /tmp/test.tar: No such file or directory
```

Vegem si el podem descomprimir:

```
$ gunzip /tmp/test.tar.gz
$ ls -l /tmp/test.tar
-rw-r--r-- 1 user user 10240 nov. 17 18:03 /tmp/test.tar
$ ls -l /tmp/test.tar.gz
ls: /tmp/test.tar.gz: No such file or directory
```

El fitxer `.tar.gz` ha estat esborrat després de crear-se el `.tar` descomprimit.

Ara suposem que volem crear el fitxer `.tar.gz` però sense el pas intermedi de crear el `.tar` i ocupar tant d'espai en disc (igual a: les dades originals + metadades + el fitxer `.tar.gz` abans de ser esborrat el `.tar`).

Per a aquests casos, tenim diferents opcions.

D'una banda, es poden usar readreces aprofitant que tant `tar` com `gzip` poden treballar amb la sortida i entrada estàndard:

```
$ cd /tmp
$ tar cv test | gzip > /tmp/test.tar.gz
test/
test/fitxer1
test/directori/
test/directori/fitxer1
$ ls -l /tmp/test.tar.gz
-rw-r--r-- 1 user user 184 nov. 17 18:03 /tmp/test.tar.gz
```

O podem aprofitar directament una de les opcions de `tar`, la `z` (zip):

```
$ cd /tmp
$ tar czvf /tmp/test.tar.gz test
```

ANOTACIONS

```
test/
test/fitxer1
test/directori/
test/directori/fitxer1
$ ls -l /tmp/test.tar.gz
-rw-r--r-- 1 user user 184 nov. 17 18:03 /tmp/test.tar.gz
```

6.3.2. Usar tar, gzip, i uns disquets

Una situació que es pot presentar quan intentem desar una sèrie de fitxers en un mitjà com un disquet és que hi hagi algun fitxer més gran que la capacitat del disc. Tant en aquest cas com per a optimitzar l'espai en els disquets, el més còmode és poder partir el conjunt de les dades que cal desar en fragments de la mida de cada disquet.

Segons això, s'ha establert l'opció `M` (Multivolum) de `tar`.

Suposem que tenim tres fitxers aleatoris d'1 MB cada un:

```
$ dd bs=1024 count=1024 < /dev/urandom > /tmp/test/fitxer1
$ dd bs=1024 count=1024 < /dev/urandom > /tmp/test/fitxer2
$ dd bs=1024 count=1024 < /dev/urandom > /tmp/test/fitxer3
-rw-r--r-- 1 root root 1048576 nov. 17 18:03 /tmp/fitxer1
-rw-r--r-- 1 root root 1048576 nov. 17 18:03 /tmp/fitxer2
-rw-r--r-- 1 root root 1048576 nov. 17 18:03 /tmp/fitxer3
```

Si els intentéssim desar en disquets d'1,4 MB (1.440 kB = 147.4560 bytes), caldria escollir entre desar cada fitxer separadament o comprimir els fitxers separadament i esperar que dos o més qualssevol ocupessin menys d'1,4 MB:

```
$ gzip /tmp/test/fitxer1
$ gzip /tmp/test/fitxer2
$ gzip /tmp/test/fitxer3
$ ls -l /tmp/test/fitxer*.gz
-rw-r--r-- 1 root root 1048763 nov. 17 18:03 /tmp/fitxer1.gz
-rw-r--r-- 1 root root 1048763 nov. 17 18:03 /tmp/fitxer2.gz
-rw-r--r-- 1 root root 1048763 nov. 17 18:03 /tmp/fitxer3.g
```

ANOTACIONS

En aquest cas, els fitxers no es comprimeixen perquè es van crear contenint només seqüències de bytes aleatòries, difícilment comprimibles.

Per a solucionar aquest problema, podem recórrer a l'opció `M` (multivolum) de `tar` comprimint directament en el disquet:

```
$ tar cMf /tmp/floppy/test.tar /tmp/test/fitxer*
Prepari el volum #2 per a 'test.tar' i premi intro:
Prepari el volum #3 per a 'test.tar' i premi intro:
```

Amb aquesta opció, `tar` intentarà desar tantes dades com pugui en cada disc, posteriorment demanarà que es canviï per un buit, i tornarà a repetir el procés fins que totes les dades estiguin desades.

Si en comptes de desar directament en el disc el que volem és preparar primer els fitxers, podem recórrer a l'opció `L` (longitud), que ens permetrà especificar la "longitud del mitjà" (o disc) en múltiples de 1.024 bytes.

Així mateix, podem posar més d'un fitxer `tar` de sortida amb repetides opcions `f fitxer (ff fitxer fitxer, fff fitxer fitxer fitxer...)`.

```
$ tar cMfff /tmp/test1.tar /tmp/test2.tar /tmp/test3.tar \
> /tmp/test/fitxer* -L 1440
$ ls -l /tmp/test?.tar
-rw-r--r-- 1 root root 1474560 nov. 17 18:03 test1.tar
-rw-r--r-- 1 root root 1474560 nov. 17 18:03 test2.tar
-rw-r--r-- 1 root root 204800 nov. 17 18:03 test3.tar
```

L'única limitació de `tar` és que, a causa del seu ús de `gzip` per a comprimir el resultat del mateix tar, no permet fraccionar ni comprimir els fitxers alhora (això és vàlid per a GNU tar 1.14).

6.4. RPM

Amb nom autoreferent, RPM Package Manager (gestor de paquets RPM), és un format per a distribuir i gestionar paquets de programari entesos en el sentit de "mòduls funcionals indivisibles". La distribució

ANOTACIONS

que el va desenvolupar i implementar en primer lloc va ser Red Hat Linux, a la qual van seguir Suse i Mandrake entre d'altres.

L'avantatge d'utilitzar RPM en comptes de fitxers `.tar` (o `.tar.gz`) rau en les opcions addicionals que RPM permet especificar, a la persona que crea el paquet, per a ser executades automàticament quan l'instal·la, desinstal·la, etc. D'aquesta manera, simplifica al màxim la tasca de l'usuari final i, si el paquet s'ha creat correctament, manté una llista dels paquets instal·lats i les relacions ells per a evitar que apareguin conflictes entre els fitxers instal·lats o pendents d'instal·lar.

6.4.1. Treballar amb RPM

Les fases del sistema RPM es poden dividir en tres parts:

1) El creador del paquet `.rpm`
2) Una base de dades de paquets instal·lats en el sistema de l'usuari
3) El mateix programa RPM

En aquest cas, ens ocuparem del darrer punt (el programa RPM) i de la seva relació amb la base de dades (segon punt) quan s'instal·len i gestionen els diferents paquets `.rpm`.

Els sistemes de paquets han fet possible que es creïn distribucions GNU/Linux, ja que és una manera senzilla de distribuir-los i desar una base de dades que els doni consistència.

Lectura complementària

http://rpmfind.net
http://freshmeat.net
ftp://ftp.rediris.es

Es poden trobar en els discos d'instal·lació; a Internet, dins de servidors ftp amb còpies de les distribucions; poden ser facilitats pels autors dels programes, etc.

Se solen expressar amb el format:

```
paquet – versió del programari – versió-rpm . arquitectura .rpm
```

Per exemple, aquests són paquets `.rpm`:

```
gcc-3.4.3-3.i386.rpm
apache-1.3.31-7mdk.i586.rpm
bash-3.0-22.x86_64.rpm
perl-5.00503-12.alpha.rpm
```

ANOTACIONS

Els paquets `.rpm` són versions executables (compilades) dels programes. Com que els programes sota llicències lliures també distribueixen el codi font, també hi ha les versions en codi font dels mateixos paquets:

```
gcc-3.4.3-3.src.rpm
apache-1.3.31-7mdk.src.rpm
bash-3.0-22.src.rpm
perl-5.00503-12.src.rpm
```

Com es pot observar, en aquest cas, només s'ha substituït el nom de l'arquitectura per a la qual està destinat el paquet pel text `src`. Això és perquè un paquet de codi font sol estar preparat per a compilar-se en qualsevol arquitectura. Així, encara que el paquet compilat per a `x86_64` tindria problemes per a executar-se en un entorn PowerPC (`ppc`), el paquet de codi font pot ser compilat per a donar lloc a ambdós `x86_64` i `ppc`, que posteriorment es podran distribuir i instal·lar directament en aquestes arquitectures.

També es poden expressar amb la cadena `noarch`, que significa estrictament "independent d'arquitectura", o `nosrc`, que designaria un paquet amb només les pautes per a construir l'executable a partir del codi font però sense incloure'l.

Entre altres coses, això permet que es desenvolupi programari en una arquitectura i s'usi en una altra només compilant el paquet `src`; o que es distribueixi només en format `src` i que cada usuari el pugui compilar a l'hora d'instal·lar, sigui per a optimitzar-lo per al seu sistema o a fi d'assegurar-se que la versió executable correspon realment al codi font obtingut del programa.

6.4.2. Instal·lació

La instal·lació d'un paquet en versió executable no representa cap dificultat amb l'opció `-i` d'`rpm`:

```
$ rpm -ivh nc-1.10-18mdk.i586.rpm
Preparing... ########################## [100%]
    1:nc      ########################## [100%]
```

Contingut complementari

Utilitat rpm
i: instal·lar
u: actualitzar (update)
q: consultar (query)
e: eliminar
V: verifica

ANOTACIONS

L'opció `v` (minúscula) fa que es mostrin els passos que es van fent. L'opció `h` mostra les barres de progrés (#).

Com hem dit abans, RPM permet assegurar-nos que no hi ha conflictes entre paquets. Una de les maneres d'aconseguir-ho és especificant en cada paquet quins altres faran falta perquè funcioni correctament. Això és el que s'anomena "dependències" (els paquets "depenen" els uns dels altres).

Juntament amb la base de dades que fa la llista dels paquets instal·lats, això fa que quan s'instal·la un paquet sigui possible detectar la falta d'alguna dependència (un altre paquet) necessària perquè funcioni correctament:

```
$ rpm -ivh gaim-1.0.3-1mdk.i586.rpm
error: Failed dependencies:
       libgaim-remote = 1:1.0.3-1mdk is needed by gaim-1.0.3-1mdk
```

Les dependències poden ser d'una versió concreta o d'una comparació booleana (per exemple: >=, que significaria "més gran o igual que la demanada").

Per a resoldre-les, caldrà baixar el paquet que falta i instal·lar-lo abans o al costat del paquet que el necessita:

```
$ rpm -ivh gaim-1.0.3-1mdk.i586.rpm \
>  libgaim-remote0-1.0.3-1mdk.i586.rpm
Preparing...                ########################### [100%]
    1:libgaim-remote0        ########################### [ 50%]
    2:gaim                   ########################### [100%]
```

En aquest cas, RPM s'encarrega de l'ordre correcte en què ha de ser instal·lat cada un dels paquets.

Si només es vol comprovar si és possible o no instal·lar un paquet determinat però sense instal·lar-lo realment, es pot usar l'opció `--test`:

```
$ rpm -ivh --test gaim-1.0.3-1mdk.i586.rpm \
>  libgaim-remote0-1.0.3-1mdk.i586.rpm
Preparing...                ########################### [100%]
$ echo $?
  0
```

ANOTACIONS

El valor de retorn d'aquesta opció es pot usar per a automatitzar les decisions d'instal·lació:

```
$ rpm -ivh --test gaim-1.0.3-1mdk.i586.rpm
error: Failed dependencies:
  libgaim-remote = 1:1.0.3-1mdk is needed by gaim-1.0.3-1mdk
  libgaim-remote.so.0 is needed by gaim-1.0.3-1mdk
$ echo $?
  1
```

Si fallen les dependències, també es pot triar que s'ignori aquest error i instal·lar el paquet de tota manera amb l'opció `--nodeps`, encara que probablement després no funcionarà:

```
$ rpm -ivh --nodeps gaim-1.0.3-1mdk.i586.rpm
Preparing...                ########################### [100%]
  2:gaim                    ########################### [100%]
```

En cas que dos paquets entrin en conflicte:

```
$ rpm -ivh --nodeps gaim-1.0.2-1mdk.i586.rpm
Preparing...                ########################### [100%]
package gaim-1.0.3-1mdk (which is newer than gaim-1.0.2-1mdk) is already installed
file /usr/bin/gaim from install of gaim-1.0.2-1mdk conflicts with file from package gaim-1.0.3-1mdk
file /usr/bin/gaim-remote from install of gaim-1.0.2-1mdk conflicts with file from package gaim-1.0.3-1mdk
file /usr/lib/gaim/autorecon.so from install of gaim-1.0.2-1mdk conflicts with file from package gaim-1.0.3-1mdk
file /usr/lib/gaim/docklet.so from install of gaim-1.0.2-1mdk conflicts with file from package gaim-1.0.3-1mdk
...
```

Nota

Advertim que aquesta manera d'instal·lar ignora les recomanacions d'RPM i, per tant, de l'autor del programa o de l'encarregat de crear i testar el paquet. Això pot conduir a un sistema inestable i diferents conflictes entre programes, que normalment no seran tinguts en compte pels desenvolupadors.

Se'n pot buscar algun de compatible amb l'instal·lat (o actualitzar aquest), o forçar que s'instal·li el nou paquet independentment dels avisos. En aquest cas, és probable que el paquet no funcioni correctament, que sobreescrigui parts d'un altre paquet, etc. Hi ha diferents opcions que permeten saltar-se diferents parts (`--replacepkgs`, `--replacefiles`, `--oldpackage`) de la comprovació, o saltar-se-les totes amb `--force`:

```
$ rpm -ivh --nodeps --force gaim-1.0.3-1mdk.i586.rpm
Preparing...                ########################### [100%]
    2:gaim                  ########################### [100%]
```

6.4.3. Actualització

Per a actualitzar un paquet, el més obvi seria desinstal·lar la versió antiga i instal·lar en el seu lloc la nova. Tanmateix, això pot trencar algunes dependències de paquets que són imprescindibles per a aquell que intentem actualitzar.

Això es pot evitar amb l'opció `-U` (*update* = actualitzar) d'`rpm`.

El seu ús és igual que el de la `-i` per a instal·lar, amb la diferència que si alguna de les dependències que entren en conflicte amb la instal·lació és del mateix paquet però amb versió anterior, aquest serà eliminat i reemplaçat pel nou automàticament. Primer instal·lem una versió anterior:

```
$ rpm -ivh gaim-1.0.2-1mdk.i586.rpm \
> libgaim-remote0-1.0.2-1mdk.i586.rpm
Preparing...                ########################### [100%]
    1:libgaim-remote0        ########################### [ 50%]
    2:gaim                   ########################### [100%]
$ rpm -q gaim libgaim-remote0
gaim-1.0.2-1mdk
libgaim-remote0-1.0.2-1mdk
```

I posteriorment, actualitzem a una versió superior:

```
$ rpm -Uvh gaim-1.0.3-1mdk.i586.rpm \
> libgaim-remote0-1.0.3-1mdk.i586.rpm
Preparing...                ########################### [100%]
    1:libgaim-remote0        ########################### [ 50%]
    2:gaim                   ########################### [100%]
$ rpm -q gaim libgaim-remote0
gaim-1.0.3-1mdk
libgaim-remote0-1.0.3-1mdk
```

Es pot apreciar que `rpm` detecta que la versió nova és una actualització de l'anterior i que no presenta missatge d'error per al conflicte entre paquets i fitxers instal·lats.

Durant una actualització, es mantenen els fitxers de configuració dels paquets llevat que els nous siguin molt diferents i que amb els antics

ANOTACIONS

possiblement no funcionessin els paquets. En aquest cas, `rpm` faria una còpia de seguretat (afegint l'extensió `.rpmsave` al nom del fitxer) i instal·laria en el seu lloc la versió nova del paquet. Altres vegades, quan el fitxer antic és compatible amb la versió nova del programa però hi ha un fitxer nou amb altres opcions, és el nou el que se salva a banda (afegint l'extensió `.rpmnew`).

6.4.4. Consulta

La base de dades de paquets instal·lats i tota la informació relativa al paquet es pot consultar amb l'opció `-q` (*query* = consultar) d'`rpm`:

```
$ rpm -q gaim
gaim-1.0.3-1mdk
```

Per a veure una llista de tots els paquets instal·lats, podem usar l'opció a (*all* = tots):

```
$ rpm -qa
libltdl3-1.5.6-4mdk
termcap-11.0.1-8mdk
libofx-0.6.6-2mdk
...
```

Com que aquesta llista sol ser molt llarga, per a veure els paquets instal·lats és més còmode usar:

```
$ rpm -qa | less
```

Per a consultar les dades d'un paquet `-i` (informació):

```
$ rpm -qi gaim
Name        : gaim                         Relocations: (not relocatable)
Version     : 1.0.3                             Vendor: Mandrakesoft
Release     : 1mdk                       Build Date: vie 12 nov 2004 23:29:22 CE
Install Date: sáb 20 nov 2004 08:28:52 CET      Build Host: n4.mandrakesoft.com
Group       : Networking/Instant messaging   Source RPM: gaim-1.0.3-1mdk.src.rpm
Size        : 8514053                           License: GPL
Signature   : DSA/SHA1, vie 12 nov 2004 23:40:27 CET, Key ID dd684d7a26752624
```

ANOTACIONS

```
Packager    : Pascal Terjan <pterjan@mandrake.org>
URL         : http://gaim.sourceforge.net/
Summary     : A GTK+ based multiprotocol instant messaging client
Description :
Gaim allows you to talk to anyone using a variety of messaging
protocols, including AIM (Oscar and TOC), ICQ, IRC, Yahoo!,
MSN Messenger, Jabber, Gadu-Gadu, Napster, and Zephyr.  These
protocols are implemented using a modular, easy to use design.
To use a protocol, just load the plugin for it.

Gaim supports many common features of other clients, as well as many
unique features, such as perl scripting and C plugins.
Gaim is NOT affiliated with or endorsed by AOL.
```

En què es poden veure diferents dades relatives al paquet.

Si és d'un paquet no instal·lat en el sistema, i que per tant no és en la base de dades, tenint el fitxer `.rpm` hi podem fer referència amb `-p` (fitxer):

```
$ rpm -qp gaim-1.0.3-1mdk.i586.rpm
gaim-1.0.3-1mdk
```

La llista de fitxers continguts en el paquet es pot consultar amb `-l`. Afegint `-v`, es mostren també les dades de cada fitxer:

```
$ rpm -qlv gaim
-rwxr-xr-x    1 root    root           853644 nov 12 23:29 /usr/bin/gaim
-rwxr-xr-x    1 root    root             7608 nov 12 23:29 /usr/bin/gaim-remote
drwxr-xr-x    2 root    root                0 nov 12 23:29 /usr/lib/gaim
-rwxr-xr-x    1 root    root             8676 nov 12 23:29 /usr/lib/gaim/autorecon.so
...
```

També es pot sol·licitar només un tipus determinat de fitxers, com ara documentació `-d`:

```
$ rpm -qld coreutils
/usr/share/doc/coreutils-5.2.1/README
```

O només els de configuració:

```
$ rpm -qlc coreutils
/etc/DIR_COLORS
/etc/pam.d/su
```

ANOTACIONS

D'aquesta manera, es poden automatitzar tasques relacionades amb la documentació (per exemple, creació d'índexs) o la configuració (per exemple, còpia de seguretat) dels paquets.

6.4.5. Desinstal·lació

La desinstal·lació d'un paquet es duu a terme amb l'opció `-i`.

Igual que en la instal·lació, si s'intenta treure un paquet de què depenen uns altres, `rpm` avisarà amb un missatge d'error:

```
$ rpm -ev libgaim-remote0
error: Failed dependencies:
        libgaim-remote = 1:1.0.3-1mdk is needed by (installed) gaim-1.0.3-1mdk
        libgaim-remote.so.0 is needed by (installed) gaim-1.0.3-1mdk
```

I, igual que abans, n'hi ha prou d'especificar alhora tots els paquets que es desinstal·laran:

```
$ rpm -ev libgaim-remote0 gaim
```

6.4.6. Verificació

Una altra de les funcions interessants d'RPM és la possibilitat de verificar l'estat de cada fitxer instal·lat per a detectar possibles modificacions.

En cada paquet, es desa la informació sobre la mida, suma de comprovació, propietaris, etc. de cada fitxer. A l'hora d'instal·lar-lo, aquestes dades també són emmagatzemades en la base de dades d'RPM, de manera que després poden ser utilitzades a fi de comparar-les amb les mateixes dades obtingudes dels fitxers presents en el sistema i verificar que aquests no han estat modificats.

La sintaxi bàsica se centra en l'opció `--verify` (`-V`) d'`rpm`:

```
rpm -V coreutils
S.5....T c /etc/pam.d/su
```

Contingut complementari

Tipus de verificacions
S: mida (size) del fitxer
M: mode (permisos)
5: suma MD5
D: dispositiu (més gran / més petit)
L: destinació de l'enllaç
U: usuari (propietari)
G: grup (propietari)
T: data (time) del fitxer

ANOTACIONS

Per cada fitxer diferent del seu corresponent del paquet original, es mostra una línia amb la llista dels canvis detectats i el tipus de fitxer: configuració, documentació, *ghost* (no inclòs però creat), llicència, *readme* (llegeixi'm). No es mostren, tanmateix, diferències per a fitxers no instal·lats originalment amb el paquet.

Una altra manera de verificar consisteix a validar la signatura OpenPGP (GPG del paquet mateix per mitjà de l'opció `-checksig`):

```
# rpm --checksig X11R6-contrib-6.7.0-4.2.101mdk.i586.rpm
X11R6-contrib-6.7.0-4.2.101mdk.i586.rpm:
    Header SHA1 digest: OK (caa530fe83f24cbdeefbfbfb0db53f21cd860e07)
    MD5 digest: OK (023c21cf6ef0f6f0841ffa0b9010ac43)
    V3 DSA signature: OK, key ID 22458a98
```

Per a això, la part pública de la clau GPG amb què estigui signat el paquet ha d'estar inclosa en l'anell de claus de l'usuari que executi RPM, o en l'especificat per la macro `%_gpg_path`.

6.4.7. Crear paquets i gestionar pedaços

La creació de paquets se centra en la utilitat `rpmbuild` i un fitxer de `.spec` que especifiqui les metadades i passos que cal fer per a compilar, instal·lar i desinstal·lar cada paquet.

Potser la manera més simple és crear un paquet binari a partir d'un altre amb el codi font:

```
$ rpmbuild --rebuild -v gaim-1.0.1-0.src.rpm
Installing gaim-1.0.1-0.src.rpm
Executing(%prep): [...]
+ tar -xvvf - [...]
Executing(%build): [...]
+ ./configure
checking for [...]
+ make [...]
Executing(%doc): [...]
Wrote: /usr/src/RPM/RPMS/i586/gaim-1.0.1-0.i586.rpm
Wrote: /usr/src/RPM/RPMS/i586/gaim-devel-1.0.1-0.i586.rpm
Executing(%clean): [...]
```

ANOTACIONS

Aquí veiem els passos que fa `rpmbuild` per a crear el paquet final:

1) Secció `%prep` del fitxer `.spec`. En aquesta secció, se sol descomprimir el codi font i es passen tots els pedaços que poden ser necessaris.

2) Secció `%build` del fitxer `.spec`. Aquí és on es compila el paquet. En cas de tenir seqüències de configuració, aquestes s'executen.

3) Secció `%install` del fitxer `.spec`. S'instal·la el programa compilat en un directori temporal.

4) Secció `%doc` del fitxer `.spec`. S'instal·la la documentació corresponent.

5) S'empaqueta la instal·lació temporal en un fitxer `.rpm`.

6) Es crea un paquet addicional amb els encapçalaments (`.h`) dels fitxers de codi font per a permetre desenvolupar i compilar paquets basats en aquest.

7) Secció `%clean` del fitxer `.spec`. Per a acabar, s'eliminen tots els fitxers temporals (inclòs el codi font instal·lat).

No és obligatori efectuar aquests passos tots seguits d'una vegada. Per a això, rpmbuild ofereix les opcions `-bp`, `-bc` i `-bi` que corresponen a deturar l'execució després de les fases `%prep`, `%build` i `%install` respectivament.

D'aquesta manera, es pot continuar la instal·lació després de realitzar les modificacions oportunes en cada fase. Per exemple, es pot executar només `%prep`, passar algun pedaç addicional al codi font i seguir la instal·lació des d'allà.

Per a crear paquets, rpmbuild ofereix les opcions `-bs`, `-bb` i `-ba`, amb què es poden crear paquets de codi font, binaris o d'ambdós tipus respectivament.

ANOTACIONS

Com a pas addicional, igual que abans hem pogut comprovar la signatura dels paquets, ara la podem crear per mitjà de l'opció `--addsign`:

```
$ rpm --addsign -v X11R6-contrib-6.7.0-4.2.101mdk.i586.rpm
Enter pass phrase:
Pass phrase is good.
X11R6-contrib-6.7.0-4.2.101mdk.i586.rpm:
```

La configuració de la clau que s'haurà de fer servir es pot especificar a `/etc/rpm/macros` o a `~/.rpmmacros` amb la sintaxi:

```
%_signature gpg
%_gpg_path /etc/rpm/.gpg
%_gpg_name Nombre Usuario <nombre@dominio.com>
%_gpgbin /usr/bin/gpg
```

Agafant primer la clau de l'anell de l'usuari en curs i posteriorment de l'especificat a `%_gpg_path`.

Se'n pot comprovar la validesa una altra vegada amb `--checksig`:

```
$ rpm --checksig -v X11R6-contrib-6.7.0-4.2.101mdk.i586.rpm
X11R6-contrib-6.7.0-4.2.101mdk.i586.rpm:
    Header V3 DSA signature: NOKEY, key ID 0f219629
    Header SHA1 digest: OK (caa530fe83f24cbdeefbfbfb0db53f21cd860e07)
    MD5 digest: OK (023c21cf6ef0f6f0841ffa0b9010ac43)
    V3 DSA signature: NOKEY, key ID 0f219629
```

On s'observa que la signatura nova ha substituït l'anterior. Això és, un paquet només pot estar signat amb una única signatura alhora.

6.4.8. Per a finalitzar RPM

La configuració d'`rpm` es defineix en el fitxer `/etc/rpm/macros` i en el `~/.rpmmacros` de cada usuari com a modificacions de les macros per defecte.

Per a consultar les macros i altres configuracions efectives en cada moment, es pot usar l'opció `--showrc`:

ANOTACIONS

```
$ rpm --showrc
ARCHITECTURE AND OS:
build arch            : i586
compatible build archs: athlon i686 i586 i486 i386 noarch
build os              : Linux
compatible build os's : Linux
install arch          : athlon
install os            : Linux
compatible archs      : athlon i686 i586 i486 i386 noarch
compatible os's       : Linux

RPMRC VALUES:
macrofiles          : /usr/lib/rpm/macros:/usr/lib/rpm/athlon-linux/macros:/etc/rpm/macros.specspo:/etc/
rpm/macros.prelin
k:/etc/rpm/macros.solve:/etc/rpm/macros.up2date:/etc/rpm/macros:/etc/rpm/athlon-linux/macros:/etc/
rpm/macros.jpackage:~/.rp
mmacros
optflags              : -O2 -fomit-frame-pointer -pipe -march=athlon %{debugcflags}

Features supported by rpmlib:
    rpmlib(VersionedDependencies) = 3.0.3-1
        PreReq:, Provides:, and Obsoletes: dependencies support versions.
[...]
========================
-14: GNUconfigure(MCs:)
  CFLAGS="${CFLAGS:-%optflags}" ; export CFLAGS;
  LDFLAGS="${LDFLAGS:-%{-s:-s}}"  ; export LDFLAGS;
  %{-C:_mydir="`pwd`"; %{-M: %{__mkdir} -p %{-C*};} cd %{-C*}}
  dirs="`find ${_mydir} -name configure.in -print`"; export dirs;
    for coin in `echo ${dirs}`
do
  dr=`dirname ${coin}`;
if test -f ${dr}/NO-AUTO-GEN; then
 :
else
[...]
======================== active 321 empty ()
```

ANOTACIONS

Per a modificar qualsevol d'aquestes macros i opcions de configuració, n'hi ha prou d'afegir-les a un dels fitxers de configuració vigents durant l'execució d'`rpm`.

En rares ocasions (per exemple, després d'un ús "creatiu" d'`rpm`) és possible que la base de dades contingui inconsistències en relació amb les dades dels paquets instal·lats. Un símptoma pot ser, per

exemple, que quan intentem fer la llista dels paquets instal·lats amb `rpm -qa`, RPM en mostri alguns però, arribat a un punt, no pugui continuar.

Per a solucionar-ho, sol haver-n'hi prou de recrear els índexs de la base de dades a partir de la llista de paquets instal·lats amb l'opció `--rebuilddb`:

```
$ rpm --rebuilddb
```

Si no n'hi ha prou amb això, o directament es vol netejar la base de dades d'`rpm`, es pot recórrer a l'opció `--initdb`:

```
$ rpm --initdb
```

Una altra manera d'obtenir dades dels paquets instal·lats és per mitjà de l'opció `--queryformat`, que permet especificar les dades concretes que ens interessen de cada paquet:

```
$ rpm -q --queryformat "%{NAME} %{OS}\n%{LICENSE}\n" gaim
gaim linux
GPL
```

Una llista de tots els camps que es poden usar en la cadena de format s'obté amb l'opció `--querytags`:

```
$ rpm --querytags
ARCH
ARCHIVESIZE
[...]
VERSION
XPM
```

6.5. DEB

El format de paquets utilitzat per Debian i distribucions derivades és el `.deb`. A banda de diferenciar-se dels `.rpm` pel format del paquet, també se'n diferencia per certes opcions de dependències i pel fet

ANOTACIONS

que està pensat directament per a ser utilitzat per gestors de paquets com APT. Aquesta última diferència va perdent importància a mesura que es milloren els gestors de paquets per a RPM.

6.5.1. Apt, Dpkg, Dselect, Aptitud, Console Apt, etc.

Els paquets són per si mateixos els .deb. Encara que el format pot variar entre distribucions –consulteu deb(5)–, són arxius que contenen un fitxer .tar.gz amb el contingut i unes metadades de control.

La gestió d'aquests paquets es realitza per mitjà del programari dpkg, que manté una base de dades de metadades dels paquets instal·lats i s'encarrega d'instal·lar-los i desinstal·lar-los.

Mentrestant, la gestió automatitzada de dependències es realitza per mitjà de la utilitat APT, que permet autodetectar, baixar i instal·lar automàticament tots els elements necessaris perquè un programa funcioni correctament.

Tant dpkg com APT tenen els seus corresponents processadors d'accés (*front-end*) anomenats dselect i aptitude respectivament. Encara que directament dpkg i APT ofereixen més flexibilitat i un nombre d'opcions més elevat, el maneig de llistes de paquets es fa més fàcil perquè es pot navegar per aquestes llistes i operar sobre cada paquet en concret.

Hi ha altres programes capaços de realitzar aquestes tasques, fins i tot gràfics, com Synaptic, encara que no són oficials del projecte Debian.

6.5.2. Apt, ordres bàsiques

apt es compon d'una sèrie d'utilitats encarregades de gestionar, d'una banda, dependències, i de l'altra, dipòsits i llistes de paquets.

Aquestes utilitats són:

apt-get: encarregat de baixar/instal·lar els paquets.

ANOTACIONS

`apt-cache`: ofereix un accés a les llistes de paquets.

`apt-cdrom`: permet afegir dipòsits de paquets a CD o DVD.

`apt-ftparchive`: permet afegir dipòsits accessibles per FTP.

A banda d'aquestes utilitats, la configuració de dipòsits s'emmagatzema a:

```
/etc/apt/sources.list
```

descrita en l'entrada de manual `source.list(5)`.

Abans que res, s'ha d'obtenir una llista de les darreres versions dels paquets disponibles en els diferents dipòsits que tinguem configurats. Aquests dipòsits són conjunts de paquets emmagatzemats en llocs accessibles per xarxa, l'estructura dels quals és recognoscible pel programari instal·lador.

```
# apt-get update
Get:1 http://non-us.debian.org stable/non-US/main Packages [44.5kB]
Get:2 http://http.us.debian.org stable/main Packages [1773kB]
[...]
Get:17 http://security.debian.org stable/updates/non-free Packages [1221B]
Get:18 http://security.debian.org stable/updates/non-free Release [114B]
Fetched 2151kB in 31s (67.8kB/s)
Reading Package Lists... Done
Building Dependency Tree... Done
```

Una vegada actualitzada la base de dades, podem començar a instal·lar algun paquet que ens interessi amb `apt-get install`:

```
# apt-get install aptitude
Reading Package Lists... Done
Building Dependency Tree... Done
```

... després d'analitzar la base de dades de dependències, és possible que calguin paquets addicionals perquè el programari que ens interessa funcioni correctament:

```
The following extra packages will be installed:
  libsigc++0
```

ANOTACIONS

```
The following NEW packages will be installed:
  aptitude libsigc++0
0 packages upgraded, 2 newly installed, 0 to remove and 19  not upgraded.
Need to get 939kB of archives. After unpacking 3310kB will be used.
Do you want to continue? [Y/n]
```

... si no tenim res en contra de la detecció realitzada per APT, el següent pas és baixar els paquets des del dipòsit més adequat:

```
Get:1 http://http.us.debian.org stable/main libsigc++0 1.0.4-3 [28.0kB]
Get:2 http://http.us.debian.org stable/main aptitude 0.2.11.1-4 [910kB]
Fetched 939kB in 14s (64.5kB/s)
```

... i posteriorment instal·lar-los:

```
Selecting previously deselected package libsigc++0.
(Reading database ... 5275 files and directories currently installed.)
Unpacking libsigc++0 (from .../libsigc++0_1.0.4-3_i386.deb) ...
Selecting previously deselected package aptitude.
Unpacking aptitude (from .../aptitude_0.2.11.1-4_i386.deb) ...
Setting up libsigc++0 (1.0.4-3) ...
Setting up aptitude (0.2.11.1-4) ...
```

A més d'instal·lar-los, també podem actualitzar tots els paquets a les últimes versions disponibles en els dipòsits amb `apt-get upgrade`:

```
# apt-get upgrade
Reading Package Lists... Done
Building Dependency Tree... Done
18 packages upgraded, 0 newly installed, 0 to remove and 0  not upgraded.
Need to get 6778kB of archives. After unpacking 419kB will be used.
Do you want to continue? [Y/n]
```

... si acceptem la proposta d'actualització d'`apt-get`, passarà a baixar els paquets necessaris:

```
Get:1 http://security.debian.org stable/updates/main gzip 1.3.2-3woody3 [62.1kB]
Get:2 http://http.us.debian.org stable/main bsdutils 1:2.11n-7 [39.5kB]
[...]
```

ANOTACIONS

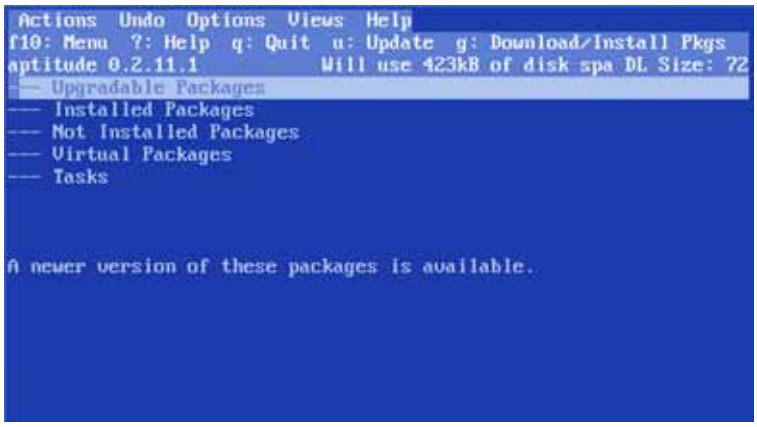
... i a instal·lar-los:

```
(Reading database ... 6842 files and directories currently installed.)
Preparing to replace bsdutils 1:2.11n-4 (using .../bsdutils_1%3a2.11n-7_i386.deb) ..
Unpacking replacement bsdutils ...
Setting up bsdutils (2.11n-7) ...
[...]
```

D'aquesta manera, podem mantenir sempre actualitzats els programes en la darrera versió disponible en els dipòsits escollits.

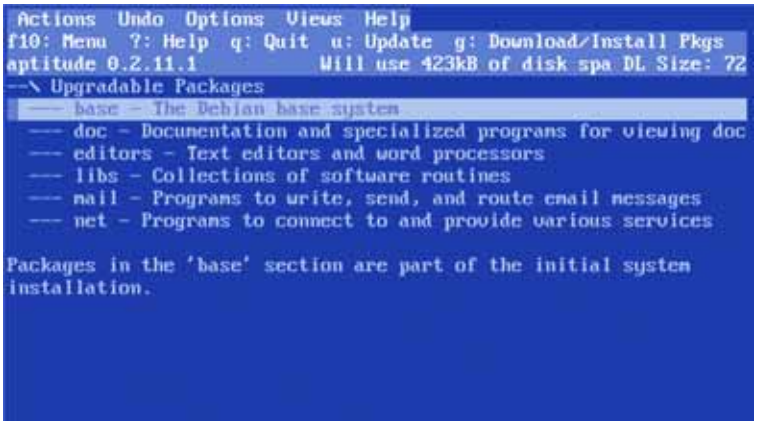
Una manera més còmoda de gestionar els paquets és amb el processador d'accés (*front-end*) oficial d'APT anomenat Aptitude:

Figura 1



Aquí trobarem una llista dels paquets instal·lats i instal·lables dels quals en podrem escollir qualsevol per a instal·lar-ne de nous i actualitzar o desinstal·lar els ja presents.

Figura 2

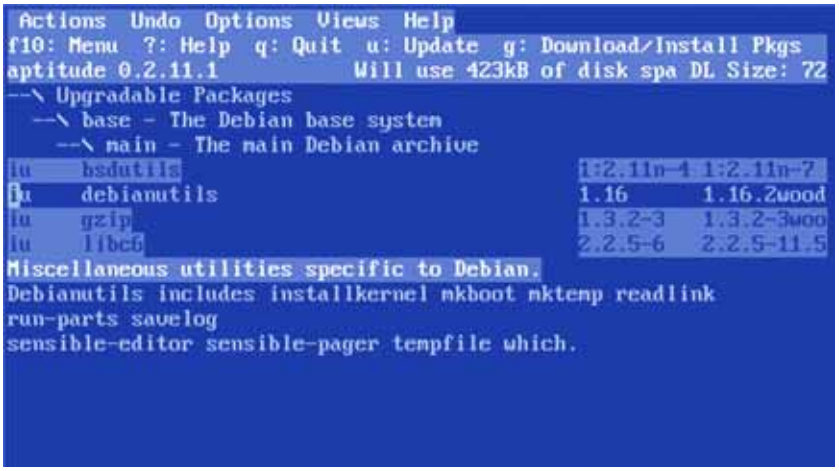


ANOTACIONS

Els paquets apareixen ordenats per temes per a facilitar la localització d'aquell que ens interessi.

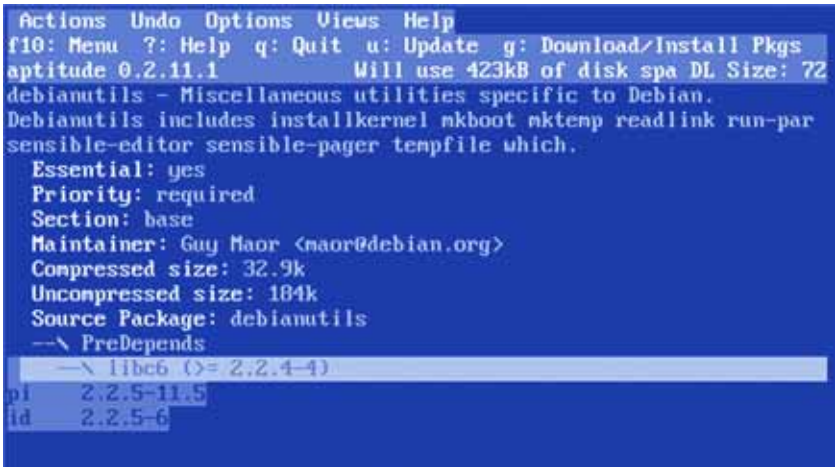
Dins la llista de paquets, quan ens situem sobre qualsevol d'ells, se'ns mostra la informació pertinent en la meitat inferior de la pantalla:

Figura 3



Quan seleccionem un paquet, apareix una descripció i una sèrie de metadades relacionades, incloses les dependències i suggeriments per a instal·lar-lo.

Figura 4



De la mateixa manera, es fa una llista dels paquets virtuals –és a dir, paquets "de funcionalitat"– agrupant diferents paquets equivalents

ANOTACIONS

entre ells que ofereixen una funcionalitat determinada al sistema (servidor DNS, servidor HTTP, etc.):

Figura 5

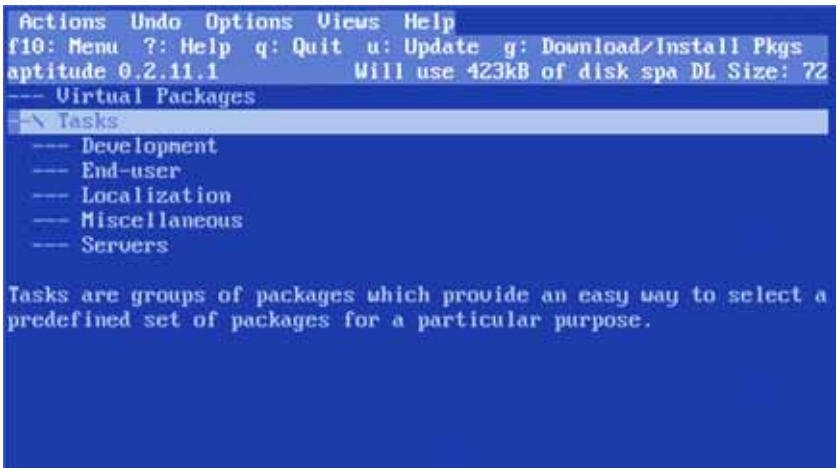
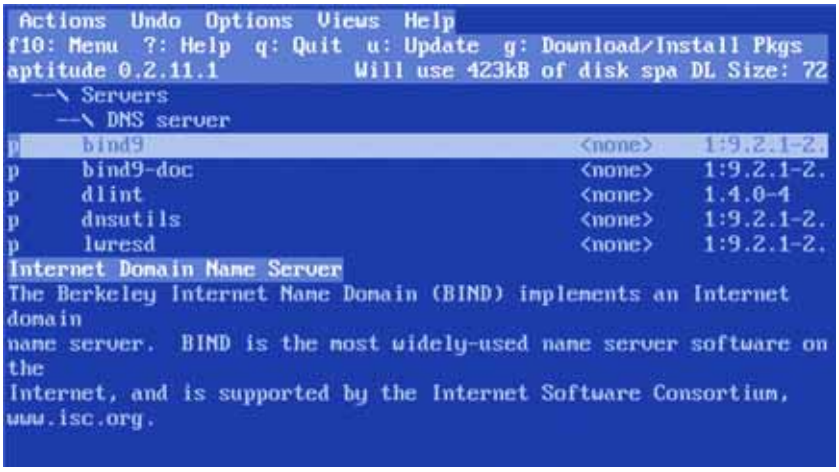


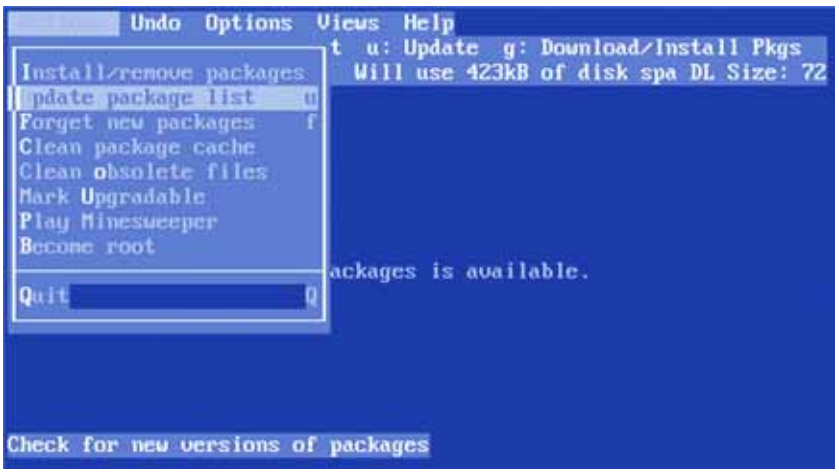
Figura 6



Per a facilitar les tasques de manteniment, Aptitude presenta un menú (accessible amb F10) des del qual es poden realitzar diferents tasques d'administració, inclòs algun entreteniment alternatiu com mirar la barra de progrés en instal·lacions llargues.

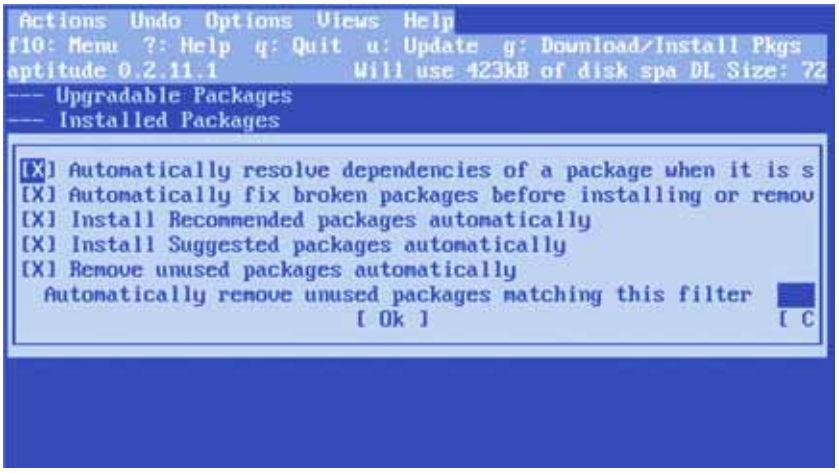
ANOTACIONS

Figura 7



També des d'aquest menú es poden configurar diversos aspectes del comportament d'APT:

Figura 8



6.5.3. Dpkg

El programa encarregat de gestionar la base de dades i d'instal·lar paquets és Dpkg.

Per a la funcionalitat d'instal·lació, normalment es cridarà des d'Apt, per la comoditat que ofereix la detecció automàtica de dependències.

Tanmateix, Dpkg ofereix altres funcionalitats addicionals relacionades amb la base de dades de paquets instal·lats.

ANOTACIONS

En la base de dades, un paquet es pot trobar en un dels estats següents:

`not-installed`: sense instal·lar en el sistema.

`half-installed`: mig instal·lat (la instal·lació possiblement s'ha interromput).

`unpacked`: copiat en el sistema però sense configurar.

`half-configured`: mig configurat (s'ha començat a configurar però no ha acabat).

`installed`: instal·lat correctament.

`config-files`: desinstal·lat però sense esborrar els fitxers de configuració.

Normalment tots els paquets són gestionats per Dpkg, excepte els marcats com a *hold*.

Per a veure una llista de tots els paquets instal·lats juntament amb els seus estats i versions, podem utilitzar l'opció `-l` de `dpkg`:

```
$ dpkg -l
Desired=Unknown/Install/Remove/Purge/Hold
| Estado=No/Instalado/Config-files/Unpacked/Failed-config/Half-installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err: mayúsc.=malo)
||/ Nombre         Versión        Descripción
+++-==============-==============-============================================
ii  adduser        3.47           Add and remove users and groups
ii  alien          8.05-3         install non-native packages with dpkg
ii  apt            0.5.4          Advanced front-end for dpkg
ii  apt-utils      0.5.4          APT utility programs
ii  aptitude       0.2.11.1-4     curses-based apt frontend
[...]
```

La primera columna indica l'estat en què es voldria tenir el paquet, mentre que la segona indica l'estat actual en el sistema. En la tercera columna, s'indiquen els errors d'instal·lació, que és buida quan no hi ha errors.

ANOTACIONS

Encara que és recomanable fer-ho des d'`apt`, també es poden instal·lar paquets separadament des de `dpkg` amb l'opció `-i`:

```
$ dpkg -i /var/cache/apt/archives/alien_8.05_all.deb
Seleccionant el paquet alien prèviament no seleccionat.
(Llegint la base de dades...
7225 fitxers i directoris instal·lats actualment.)
Desempaquetant alien (de .../archives/alien_8.05_all.deb)...
Configurant alien (8.05)...
```

I desinstal·lar-los amb l'opció `-r`:

```
$ dpkg -r alien
(Llegint la base de dades...
7258 fitxers i directoris instal·lats actualment.)
Desinstal·lant alien...
```

Un pas intermedi de la instal·lació que consisteix a copiar només els fitxers en el sistema es pot aconseguir amb l'opció `--unpack`:

```
$ dpkg --unpack /var/cache/apt/archives/alien_8.05_all.deb
Seleccionant el paquet alien prèviament no seleccionat.
(Llegint la base de dades...
7225 fitxers i directoris instal·lats actualment.)
Desempaquetant alien (de .../archives/alien_8.05_all.deb)...
```

Per a configurar-lo més tard amb l'opció `--configure`:

```
$ dpkg --configure alien
Configurant alien (8.05)...
```

Els guions de configuració dels paquets tenen una altra aplicació interessant. A banda de permetre la configuració interactiva a l'hora d'instal·lar cada paquet, això afegeix possibilitats interessants de configuració perquè es pot "reconfigurar" un paquet ja instal·lat:

```
$ dpkg-reconfigure locals
```

ANOTACIONS

Per exemple, amb l'ordre anterior podem canviar la configuració d'idioma instal·lada en el sistema sense haver-nos de preocupar de conèixer els fitxers exactes de configuració ni la seva sintaxi.

6.5.4. Crear paquets deb i gestionar pedaços

Una part de dpkg és la gestió de paquets `.deb`, que es realitza per mitjà de `dpkg-deb` (es pot anomenar directament `dpkg`).

Per a fer una llista dels fitxers que conté un paquet, hi ha l'opció `-c` (`--contents`):

```
$ dpkg -c alien_8.05-2_all.deb
drwxr-xr-x root/root         0 2004-11-17 18:03:37 ./
drwxr-xr-x root/root         0 2004-11-17 18:03:33 ./usr/
drwxr-xr-x root/root         0 2004-11-17 18:03:33 ./usr/bin/
-rwxr-xr-x root/root     13482 2002-04-01 18:37:26 ./usr/bin/alien
drwxr-xr-x root/root         0 2004-11-17 18:03:34 ./usr/share/
drwxr-xr-x root/root         0 2004-11-17 18:03:33 ./usr/share/alien/
drwxr-xr-x root/root         0 2004-11-17 18:03:33 ./usr/share/alien/patches/
[...]
```

Una informació general sobre el paquet s'obté amb `-I` (`--info`):

```
$ dpkg -I alien_8.05-2_all.deb | less
 paquet debian nou, versió 2.0.
 mida 112716 bytes: fitxer de control= 1511 bytes.
518 bytes,    19 líneas     control
    1989 bytes,    27 línies     md5sums
     249 bytes,     9 línies  *  postinst             #!/bin/sh
     192 bytes,     7 línies  *  prerm                #!/bin/sh
Package: alien
 Version: 8.05-2
 Section: alien
 Priority: extra
 Architecture: all
 Installed-Size: 251
 Maintainer: root < root@ >
 Description: install non-native packages with dpkg
  Alien allows you to convert LSB, Red Hat, Stampede and Slackware
  Packages
  into Debian packages, which can be installed with dpkg.
[...]
```

ANOTACIONS

Per a extreure únicament les metadades, hi ha l'opció `-f` (`--field`):

```
$ dpkg -f alien_8.05-2_all.deb | less
Package: alien
Version: 8.05-2
Section: alien
Priority: extra
Architecture: all
Installed-Size: 251
Maintainer: root <root@>
Description: install non-native packages with dpkg
  Alien allows you to convert LSB, Red Hat, Stampede and Slackware
  Packages
  into Debian packages, which can be installed with dpkg.
```

Si necessitem extreure els fitxers del paquet en un directori que no sigui el d'instal·lació, podem usar l'opció `-x` (`--extract`). Si volem veure cada fitxer a mesura que és extret (*verbose*), usarem `-X` (`--vextract`):

```
$ mkdir /tmp/alien1
$ dpkg -x alien_8.05-2_all.deb /tmp/alien1/
./
./usr/
./usr/bin/
./usr/bin/alien
./usr/share/
./usr/share/alien/
./usr/share/alien/patches/
./usr/share/alien/patches/motif-devel_2.1.10-7.diff.gz
./usr/share/alien/patches/motif_2.1.10-7.diff.gz
[...]
```

Per a extreure únicament les dades de control del paquet sense necessitat d'extreure tots els fitxers, podem usar `-e` (`--control`):

```
$ mkdir /tmp/alien1/DEBIAN
$ dpkg -e alien_8.05-2_all.deb /tmp/alien1/DEBIAN
$ ls -l /tmp/alien1/DEBIAN
total 5
-rw-r--r--    1 root     root          518 nov 17 18:03 control
-rw-r--r--    1 root     root         1989 nov 17 18:03 md5sums
-rwxr-xr-x    1 root     root          249 nov 17 18:03 postinst
-rwxr-xr-x    1 root     root          192 nov 17 18:03 prerm
```

ANOTACIONS

Així mateix, amb `-b` (`--build`) podem construir un paquet `.deb` a partir d'un directori que contingui les dades i un directori addicional DEBIAN amb les metadades i seqüències de control.

Amb aquesta opció, es pot reconstruir un paquet prèviament extret amb `-x` i `-e`:

```
$ dpkg -b /tmp/alien1
dpkg-deb: construint el paquet `alien` a `/tmp/alien1.deb`.
$ ls -l /tmp/alien1.deb
-rw-r--r--    1 root     root       112722 nov 17 18:03 /tmp/alien1.deb
```

Per a crear un paquet `.deb`, l'únic que cal és copiar en un directori els fitxers que es vulguin incloure en una estructura de directoris relativa al directori arrel (com si fos un *chroot*) i afegir un directori DEBIAN en què s'inclourà un fitxer de control amb les dades relatives al paquet amb les línies següents:

`Package`: nom del paquet.

`Version`: número de versió.

`Section`: secció temàtica on anirà inclòs (per exemple, a dselect).

`Architecture`: plataforma sobre la qual funcionarà el paquet.

`Depends`: llista de paquets de què depèn. Cada paquet pot anar acompanyat d'un número de versió entre parèntesis precedit de "<<" (més petit que), "<=" (més petit o igual), "=" (exactament la versió especificada), ">=" (més gran o igual), ">>" (només més gran que).

`Enhances`: especifica que la funcionalitat de determinat paquet o paquets serà millorada si s'instal·la aquest (per exemple, si és un paquet de documentació).

`Pre-Depends`: especifica paquets que han d'estar ja instal·lats obligatòriament abans d'instal·lar aquest paquet (no permet instal·lacions de "més d'un paquet" en ordre no definit).

ANOTACIONS

`Recommends`: especifica paquets relacionats que comunament són instal·lats amb aquest paquet.

`Suggests`: paquets que tenen alguna relació amb aquest, però que no tenen per què ser instal·lats perquè funcioni amb normalitat.

`Installed-Size`: defineix l'espai que ocuparà el paquet una vegada instal·lat.

`Maintainer`: nom i adreça de correu de l'encarregat del paquet en format `nom <correu@domini>`.

`Conflicts`: paquets que no poden estar instal·lats alhora que aquest perquè funcioni correctament.

`Replaces`: aquest paquet substitueix determinats paquets (que no podran estar instal·lats alhora).

`Description`: un text descriptiu del paquet.

A més d'aquest fitxer de control, s'han d'especificar dues seqüències per a ser executades en el moment de la instal·lació (`postinst`) i de la desinstal·lació (`prerm`).

6.5.5. Alien (convertir paquets deb, RPM i tgz)

Per a pal·liar el problema de distribució que es presenta quan hi ha diferents formats de paquets entre les diferents distribucions, es pot recórrer al programa Alien que permetrà convertir paquets ja creats entre els formats `rpm`, `deb`, `tgz` i alguns més.

6.5.6. Ordres bàsiques Alien

La forma bàsica d'ús és amb les opcions `--to-deb` i `--to-rpm`:

```
$ alien --to-rpm alien_8.05_all.deb
alien-8.05-2.noarch.rpm generated
```

ANOTACIONS

Igualment, si volem passar de `deb` a `rpm`:

```
$ alien --to-deb alien-8.05-2.noarch.rpm
alien_8.05-3_all.deb generated
```

També podem instal·lar directament el paquet convertit (per exemple, instal·lar un paquet `.rpm` en un sistema basat en `.deb`):

```
$ alien --to-deb -i alien-8.05-2.noarch.rpm
(Llegint la base de dades...
7257 fitxers i directoris instal·lats actualment.)
Preparant per a reemplaçar alien 8.05 (usant alien_8.05-
_all.deb)...
Desempaquetant el reemplaçament d'alien...
Configurant alien (8.05-3)...
```

Per a distribuir paquets convertits d'aquesta manera, cal tenir en compte que Alien, per defecte, augmenta en un el número més petit de versió (en l'exemple anterior, de 8.05-2 a 8.05-3). Això es pot evitar especificant l'opció `-k`:

```
$ alien --to-deb -k alien-8.05-2.noarch.rpm
alien_8.05-2_all.deb generated
```

Si el que ens interessa és poder compilar el paquet en qüestió en la plataforma desitjada però podent canviar abans alguna cosa en aquest, tenim l'opció `-g` que extraurà el contingut en un directori temporal:

```
$ alien -g alien-8.05-2.noarch.rpm
Directories alien-8.05 and alien-8.05.orig preparet.
```

En aquest cas, com que Alien exporta per defecte al format `.deb`, s'hauran creat dos directoris des dels quals es podrà crear un paquet `.deb` i un paquet `.rpm` respectivament.

6.6. Conclusions

El contenidor `tar` desa estructures completes de fitxers i directoris i juntament amb el compressor `gzip` és capaç de distribuir programa-

ANOTACIONS

ri de manera homogènia però inconsistent, és a dir, la seva instal·lació és independent d'altres paquets de programari i, fins i tot, d'una instal·lació idèntica anterior.

RPM i deb prenen el millor dels anteriors ja que desen una consistència de dades i versions necessària per a distribuir el programari per Internet i les instal·lacions desateses.

L'evolució d'aquests sistemes consistents i la seva compatibilitat determinarà el sistema o sistemes de distribució de programari del futur.

El més desitjable: un processador d'accés (*front-end*) independent del sistema de distribució que compatibilitzi versions i unifiqui bases de dades, de manera que qualsevol d'elles es pugui gestionar de manera transparent. La comunitat hi està posada.

ANOTACIONS

7. Sistemes de creació de documentació

7.1. Introducció

Aquest capítol comprèn els diferents aspectes relacionats amb els sistemes de documentació més acceptats dins el món del programari lliure i de font oberta. La documentació, en l'àmbit de programari, exerceix un paper importantíssim com a complement indispensable dels programes informàtics. Es plantegen les diferents motivacions que porten a tenir sistemes de documentació lliure i es presenten els diferents tipus de llicències amb què la documentació és distribuïda. També s'analitza cada un dels sistemes de documentació en forma particular que li permeten crear documents bàsics.

En particular, es descriu des de la simple documentació d'una pàgina de manual, passant per documentació de desenvolupament i documentació de manuals breus, fins a la possibilitat d'escriure llibres complets; tot amb caràcter professional i útil per a obtenir cooperació, com s'espera en el món del programari lliure.

7.2. Objectius

Una vegada estudiat aquest capítol, els lectors podreu fer el següent:

- Crear documents bàsics en cada un dels sistemes plantejats.
- Conèixer cada un dels sistemes de documentació, la qual cosa us permetrà poder decidir quin sistema utilitzar per a documentar el vostre programari, d'acord amb les necessitats del cas.
- Crear pàgines de manual.
- Crear documents curts (com es fa, manuals, etc.).
- Crear llibres de documentació (manuals, tractats, etc.).

ANOTACIONS

7.3. Documentació lliure: estàndards i automatització

Un sistema o programari documentat pobrament manca de valor encara que hagi funcionat bé en alguna ocasió. En el cas de programes petits i poc importants que només s'utilitzen durant un període de temps curt, uns quants comentaris en el codi podrien ser suficients. No obstant això, la majoria dels programes l'única documentació dels quals és el codi no tenen acceptació i és impossible mantenir-los. Dedicar una mica d'esforç a la documentació, fins i tot dins els límits d'un projecte petit, constitueix una molt bona pràctica.

Aprendre a documentar programari és una tasca complicada i exigeix un criteri d'enginyeria madur. Documentar escaridament és un error habitual, però l'altre extrem pot resultar igualment perjudicial: si s'escriuen documentacions extenses, aquestes aclapararan el lector i constituiran una càrrega a l'hora de mantenir-les. És essencial documentar només els assumptes correctes. La documentació no serveix d'ajuda per a ningú si la seva extensió desanima la gent a l'hora de llegir-la.

Aquest apartat proveeix els lectors de les directrius sobre com documentar, principalment orientat a sistemes de programari lliure, és a dir, documentar amb eines lliures i en formats lliures. Així mateix, us proporcionarà una estructura esquemàtica de tipus de documents, de manera que la utilització d'aquests exemples quedarà al vostre criteri.

7.3.1. La documentació del programari

Una part fonamental del programari és la documentació que l'acompanya. La documentació és una peça tan important d'un programa que en ocasions aquest serà inútil sense aquella.

La documentació del programari es divideix en diferents tipus segons el que documenta, i generalment hi ha:

- Documentació de desenvolupament:
  - Anàlisis, requisits, especificacions
  - Diagrames

ANOTACIONS

– Comentaris en codis
– Documentació de proves, etc.

• Documentació de programa:
– Ajuda en línia
– Pàgina de manual (*man*)

• Documentació d'usuari:
– Manual d'ús
– Llibres i programes d'aprenentatge
– Guies d'ensenyament o autoaprenentatge

Òbviament, aquesta és una llista reduïda de les possibilitats de documentació entorn d'un programari determinat i la documentació que s'utilitzarà dependrà de l'abast del programari en qüestió.

Una altra característica de la documentació del programari és que ha d'acompanyar el desenvolupament o evolució del programari que documenta. És a dir, de la mateixa manera que el programari avança i es desenvolupa, la documentació ha d'avançar i desenvolupar-se conjuntament, de manera que la darrera versió de la documentació reflecteixi les característiques i l'estat de la darrera versió del programari.

7.3.2. El problema de documentar programari lliure

Si la documentació ha d'evolucionar al costat del programari, quan parlem de programari lliure ens trobem davant el problema que estem documentant un programari potencialment realitzat per moltes persones, que pot ser modificat i millorat lliurement. És obvi, llavors, que la documentació del programari lliure ha de poder ser modificada i millorada lliurement per a acompanyar l'evolució del programari lliure. De fet, es considera que un programari lliure sense documentació lliure no és un programari lliure, ja que el desenvolupament lliure del programari es pot veure perjudicat per la falta d'un accés lliure a la seva documentació.

Penseu, per exemple, que heu agafat un programari lliure i n'heu fet algunes millores; si la documentació no és lliure, llavors no en podrà reflectir la millora en el mateix document, la qual cosa significarà un problema important per a desenvolupar o almenys per a difondre i aplicar-ne la modificació.

ANOTACIONS

7.3.3. Llicències lliures per a la documentació

Contingut complementari

El programari lliure és aquell que permet a l'usuari copiar-lo, millorar-lo i distribuir-lo lliurement.

El programari privatiu és aquell que manca d'alguna o de la majoria de les llibertats que es reben amb el programari lliure.

Tot programari, tant si és lliure com privatiu, està acompanyat per un contracte de llicència.

La documentació que acompanya el programari és, igual que aquest, una producció de l'intel·lecte humà, per la qual cosa s'hi apliquen les lleis de drets d'autor o de *copyright*.

Per aquest motiu, per a poder copiar, modificar o distribuir la documentació, cal tenir el permís de l'autor d'aquesta documentació o tenir un document que atorgui aquest permís. Aquest permís es materialitza en una llicència per a la documentació.

Nota

http://www.gnu.org/copyleft/fdl.html
http://creative-commons.org/text/

La llicència per a la documentació de programari lliure generalment és la mateixa que la del programari al qual acompanya, encara que hi ha alguns documents publicats amb llicències específiques per a documentació com la *free documentation licence* (FDL) de la Free Software Foundation, encara que últimament molta documentació és lliurada amb la llicència coneguda com a Creative Commons.

En tots els casos en què modifiqueu el treball intel·lectual aliè, haureu de conèixer si teniu permís per a fer-ho, si està especificat en algun lloc, i si teniu una llicència, l'haureu de llegir.

7.3.4. Formats lliures i propietari

Tan important com tenir les llibertats per a la documentació, és documentar en formats lliures, de manera que, una vegada que el seu document arribi als lectors, aquests puguin accedir-hi, modificar-lo i tornar a distribuir-lo amb totes les llibertats, sense dependre de restriccions d'accés o modificació imposades al programari. Això s'aconsegueix documentant en formats lliures.

Imaginem per un moment que algú documenta el seu programari amb una eina que no es pot copiar lliurement: en què cada còpia requereix una llicència. Podria fer arribar els seus documents a diferents persones, però aquestes estarien obligades a acceptar un contracte de llicenciament per a poder llegir els documents.

ANOTACIONS

D'altra banda, considereu que desa els seus documents en un format (format binari, format de fitxer de dades) que només coneix i maneja una empresa i que hi accedeix i el modifica mitjançant el programari d'aquesta mateixa empresa. Però als pocs anys, aquesta empresa decideix modificar els seus formats de fitxers, o l'empresa tanca el projecte, o l'empresa abandona el mercat. La seva necessitat d'accedir a la informació desada allà continuarà existint i potser no podrà accedir a les seves pròpies dades o textos.

Empresarialment o estatal, manejar fitxers de deu anys d'antiguitat (comptabilitat, declaracions jurades, etc.) és normal i de vegades obligatori. Possiblement, en deu anys, ni tan sols el suport físic (disquets, CD-ROM, DVD, cintes) no serà el mateix, molt menys el programari per a accedir a les dades, per la qual cosa, només si s'han desat les dades en formats estàndard que són coneguts per tots i accessibles per a molts sistemes, els podrem visualitzar.

Els formats lliures més usats (txt, RTF, HTML, TeX, XML, etc.) estan avalats per organitzacions que estableixen els estàndards per a cada un d'ells.

7.3.5. Eines de control i administració de versions

Com indicàvem anteriorment, la documentació ha d'evolucionar en concordança amb el programari que documenta. És a dir, a cada versió nova del programari li correspondrà una versió nova de la documentació. Almenys, la documentació haurà d'indicar a quines versions del programari es poden aplicar les diferents opcions documentades.

El programari lliure es crea i desenvolupa generalment per grups de treball guiats per un líder de projecte. La documentació també es realitza mitjançant un procés semblant. Per a això, s'utilitzen diferents eines que permeten controlar i versionar la documentació de manera cooperativa i automàtica; la més utilitzada és el sistema CVS (*concurrent versions system*), que és en un sistema en què cada autor pot pujar els seus textos o modificacions, i el mateix sistema controla automàticament els canvis entre versions. El sistema CVS no és un sistema específic per

ANOTACIONS

a documentació, també és utilitzat per a codi font o qualsevol sistema de textos.

Amb l'extensió d'Internet, en els darrers temps s'han desenvolupat alguns sistemes de documentació cooperativa en línia que permeten un treball eficaç en grups d'autors que treballen simultàniament. En particular, abordarem més endavant el sistema WikiWikiWeb, encara que s'estan estudiant altres sistemes per a aquesta mateixa finalitat.

7.4. Crear pàgines de manual

Les aplicacions, utilitats i ordres tenen usualment les seves pàgines de manual corresponents (denominades *man pages*), que mostren les opcions disponibles i els valors de fitxers o executables. Les pàgines *man* estan estructurades de manera que els usuaris poden buscar fàcilment la informació pertinent, la qual cosa és molt important quan es treballa amb ordres amb què mai no s'havia treballat abans.

Es pot accedir a les pàgines man per mitjà de l'intèrpret d'ordres escrivint l'ordre `man` i el nom de l'executable.

7.4.1. Seccions de les pàgines de manual

Cada pàgina de manual està dividida en seccions que varien d'acord amb la complexitat de l'ordre o aplicació que documenten, la següent és una llista de les seccions estàndard:

NAME (nom): mostra el nom de l'executable i una explicació breu de la funció que realitza.

SYNOPSIS (sinopsi): mostra l'ús comú de l'executable, com ara quines opcions són declarades i quin tipus d'entrades (fitxers, valors, arguments) suporta l'executable.

DESCRIPTION (descripció): mostra les opcions i valors associats amb un fitxer o executable.

ANOTACIONS

SEE ALSO (vegeu també): mostra altres termes, fitxers o programes relacionats amb l'executable de la pàgina man consultada.

Les altres seccions ja depenen directament de l'ordre documentada en la pàgina man, com per exemple:

OPTIONS (opcions): mostra una descripció de cada un dels paràmetres que pot rebre l'executable.

EXAMPLES (exemples): mostra una llista d'invocacions de l'executable més usuals.

FILES (fitxers): llista de fitxers involucrats en l'execució del programa, generalment fitxers de configuració.

AUTHORS (autors): autors de l'ordre o programa i les seves adreces electròniques.

BUGS (errors): mostra una llista d'errors coneguts en l'executable o el programa.

HISTORY (història): mostra una llista de les versions del document man i de les seves modificacions.

Òbviament, l'autor del man té flexibilitat per a agregar les seccions que consideri oportunes amb l'objectiu de tenir degudament documentat la seva ordre o programa; de la mateixa manera, no totes les pàgines de manual tenen totes les seccions anteriors.

7.4.2. Camí de recerca de pàgines man

Quan és invocat l'ordre `man` per a mostrar la pàgina de manual d'una ordre determinada, cal conèixer on hi ha disponibles els fitxers que contenen els textos que cal desplegar: això s'aconsegueix mitjançant la utilització de MANPATH.

MANPATH permet determinar la ruta per a cercar les pàgines de manual després d'invocar l'ordre `man`. Generalment, una distribució de

ANOTACIONS

GNU/Linux ja té una configuració apropiada de MANPATH i no cal modificar-la.

Hi ha dues maneres de configurar MANPATH. Una és configurant la variable d'entorn $MANPATH, que es realitza generalment en el fitxer `/etc/etc/profile` del sistema amb la línia `MANPATH=/usr/man:/usr/share/man:/usr/X11R6/man`. I l'altra és amb el fitxer `/etc/manpath.conf`.

La sintaxi per a definir MANPATH és semblant a la definició del PATH per a cercar executables en el sistema: la ruta, delimitada per dos punts (:). MANPATH ha de contenir la ruta en la base de les jerarquies de capítols (vegeu la secció següent) de les pàgines `man`.

7.4.3. Jerarquia de capítols de les pàgines man

El conjunt de les pàgines `man` es classifica en capítols que estan destinats a optimitzar les recerques i a permetre que els sistemes només tinguin instal·lats aquells capítols que necessiten, depenent de la seva finalitat.

Sota el directori definit en MANPATH, hi ha una jerarquia de directoris definits com a *manX*, on *X* correspon al capítol de la pàgina man. Els capítols són els següents:

1) Ordres de l'usuari
2) Crides al sistema (*system calls*)
3) Biblioteques i funcions del llenguatge C
4) Dispositius
5) Formats de fitxers
6) Jocs
7) Diversos
8) Ordres d'administració del sistema

ANOTACIONS

7.4.4. Generar pàgines man usant eines estàndard

Les pàgines man són escrites en un llenguatge de marques, generalment conegut com a `fitxer nroff`; de fet, hi ha altres processadors per a aquests fitxers de marques (com ara `troff` o `groff`), però tots suporten una sintaxi semblant. `nroff` és un llenguatge de marques més primitiu que SGML o HTML. D'altra banda, és un llenguatge de macros, per la qual cosa podem realitzar tasques complexes amb ell.

Per a escriure la seva pròpia pàgina `man`, primerament ha de tenir un fitxer font `nroff`, per exemple:

```
.TH "elmeuprograma" 1
.SH NAME
elmeuprograma \- El meu primer programa
.SH SYNOPSIS
.B elmeuprograma
elmeuprograma [-laFh]
.SH DESCRIPTION
El meu primer programa per a provar pàgines man.
.SH OPTIONS
.TP
.B \-l
Paràmetre l d'elmeuprograma.
.TP
.B \-a
Paràmetre a d'elmeuprograma.
.TP
.B altres paràmetres . . .
.SH "SEE ALSO"
elmeualtreprograma(1)
.SH BUGS
No hi ha errors reportats.
```

La macro `.TH` al costat del nom del programa té un 1. Això significa que el programa que s'està documentant s'arxivarà en el capítol 1 (`man1`) de les jerarquies de pàgines `man`.

Una vegada que es disposa del fitxer font `nroff`, es necessita que el programa `nroff` l'interpreti per a produir el fitxer de pàgina `man`, amb la línia següent:

```
nroff -man elmeuprograma.nroff > elmeuprograma.1
```

ANOTACIONS

El resultat de l'execució d'aquest paràmetre és dirigit a un fitxer que es crida com l'ordre i que té l'extensió del número del capítol en què serà arxivat. Es pot veure el resultat amb l'ordre:

```
less elmeuprograma.1
```

Perquè la pàgina *man* quedi a disposició per a llegir amb l'ordre `man`, s'ha de col·locar en el directori corresponent del capítol.

7.4.5. Generar pàgines de manual usant perlpod

Nota

http://www.perl.org/about.html

Una altra manera de fer pàgines de manual és utilitzant el llenguatge d'interpretació de guions *perl* amb una utilitat anomenada `pod2man` (a més de pod2html, pod2tex i pod2text), que permet crear pàgines de manual fàcilment (POD vol dir *Plain Old Documentation*). Per exemple, per a crear la pàgina d'un programa dict2docbook, es pot escriure el següent en un fitxer elmeuprograma.pod:

```
=head1 NOM
elmeuprograma - El meu primer programa
=head1 SINOPSI
S<elmeuprograma [-laFh] >
=head1 DESCRIPCIO
El meu primer programa per a provar pàgines man.
=head1 AUTOR
Juan Pueblo E<lt>jpueblo@mail.orgE<gt >.
=cut
```

Després s'executa

```
pod2man -lax elmeuprograma.pod > elmeuprograma.1
```

per a crear la pàgina de manual. L'opció `-lax` s'usa perquè s'accepten noms diferents dels habituals per a les seccions, ja que en aquest cas estan traduïts al català. La pàgina creada es pot llegir amb:

```
man -l elmeuprograma.1
```

La sintaxi completa del POD està explicada en `man perlpod`, i és molt simple; també heu de consultar la pàgina de manual de `pod2man`. L'únic paquet necessari és *perl*.

ANOTACIONS

Si el programa que es vol documentar està escrit en *perl*, el codi POD pot ser dins del mateix programa i pot servir tant per a documentar el codi font com per a crear la documentació addicional per a *man*.

7.5. TeX i LaTeX

TeX és un programa de Donald E. Knuth que està orientat a compondre i imprimir textos i fórmules matemàtiques.

LaTeX és un paquet de macros que permet a l'autor d'un text compondre i imprimir un document amb qualitat, ocupant patrons definits. Originalment, LaTeX va ser escrit per Leslie Lamport i utilitza TeX com a element de composició. LaTeX és una eina potent de processament de textos científics que encara no ha estat substituïda pels editors de text moderns en el món de les editorials científiques i acadèmiques. Hi ha qui afirma que LaTeX és, probablement, el millor processador de textos del món, però té l'inconvenient que no es treballa de manera WYSIWYG (*what you see is what you get*) la qual cosa dificulta en gran manera el seu ús per part de persones acostumades a editors de text.

La gran problemàtica que presenten els editors de textos WYSIWYG és que l'autor (usuari) ha d'estar observant contínuament el format dels textos (si el títol va centrat, la numeració de les vinyetes, el format dels paràgrafs, etc.), mentre que amb LaTeX l'autor es pot dedicar exclusivament al text (contingut) sense necessitat de distreure l'atenció en la presentació. Aquesta separació entre contingut i presentació en ambients de producció (com un diari) és natural i hi ha rols diferents per a cada un: l'autor i el dissenyador. Els editors de text moderns obliguen l'autor a ocupar-se de les tasques del dissenyador.

Amb LaTeX l'autor només s'haurà d'ocupar del text, del contingut, i deixarà que el programa s'encarregui de la presentació. Així, un mateix text serà presentat de manera diferent si és per a imprimir en format A4 o carta, o si la seva destinació és un PDF o una pàgina web.

ANOTACIONS

LaTeX no solament s'ocupa de les tasques de presentació final, sinó també de la numeració dels capítols, els índexs temàtics, els enllaços encreuats, els enllaços web, l'organització de la bibliografia esmentada, la inclusió d'altres fitxers, la numeració de pàgines, les notes al peu, etc.

LaTeX va néixer a UNIX, amb la qual cosa està disponible en plataformes amb aquest sistema operatiu, incloent-hi ordinadors personals amb GNU/Linux. Com ocorre amb molts programes de distribució lliure, ha estat importat amb èxit a moltes altres plataformes, incloent-hi DOS/Windows. Aquest és un altre punt a favor de LaTeX: la possibilitat de treballar-hi virtualment en totes les plataformes de maquinari i programari que hi ha.

7.5.1. Instal·lació

No donarem massa detalls sobre la instal·lació de LaTeX, ja que afortunadament forma part dels paquets de les distribucions de GNU/Linux més usades actualment i, per tant, la instal·lació és completament automàtica. Destaquem, tanmateix, que cal tractar d'instal·lar-se tots els paquets de LaTeX (excepte alguns d'evidents, com els suports d'idiomes orientals i altres rareses), fent èmfasi en les eines de visualització i conversió de fitxers processats *.dvi* (*dvips*, *xdvi*, etc.), la qual cosa ens permetrà imprimir o veure el resultat per pantalla.

7.5.2. Utilització

L'ús de LaTeX és semblant al d'un compilador, per això és denominat *processador de textos* (a diferència dels *editors de text*). Els passos que cal seguir en un document són els següents:

1) S'escriu el text *codi font* amb la sintaxi (o marques) de LaTeX amb qualsevol editor de textos ASCII (se'n recomana algun que destaqui les marques amb colors).

2) Es compila el fitxer escrit, escrivint l'ordre `LaTex fitxer.tex`.

ANOTACIONS

3) Es visualitza el resultat en format gràfic amb el visor de *dvi* amb l'ordre `xdvi fitxer.dvi`.

4) Es reedita el fitxer per a corregir o continuar el treball, i finalment s'imprimeix amb l'ordre `dvips fitxer.dvi.` o es transforma a formats PDF, HTML o RTF.

7.5.3. Fitxer font LaTeX

El següent és un exemple d'un document bàsic en LaTeX:

```
\documentclass[a4paper,11pt]{article}
\userpackage[activeacute,spanish]{babel}
\author{Juan Pueblo}
\title{El meu primer document LaTeX}
\begin{document}
\maketitle
\tableofcontents
\section{Introducció}
La primer part del document i un paràgraf.
```

Un altre paràgraf de la introducció:

```
\section{Última part}
```

L'última part i aquí s'acaba:

```
\end{document}
```

7.5.4. Classes de documents

La instal·lació estàndard de LaTeX ja porta un grup important de documents que es poden utilitzar per a aconseguir resultats diferents:

- **Article** per a articles de revistes especialitzades, ponències, treballs de pràctiques de formació, treballs de seminaris, informes petits, sol·licituds, dictàmens, descripcions de programes, invitacions i molts altres.
- **Report** per a informes més grans que tenen més d'un capítol, projectes de final de carrera, tesis doctorals, llibres petits, dissertacions, guions o semblants.

ANOTACIONS

- **Book** per a llibres.
- **Slide** per a transparències. Aquesta classe de documents produirà pàgines amb tipus de caràcters grans.

7.5.5. Extensions o paquets

Un dels avantatges de LaTeX és que pot ser estès per a incloure situacions en què el sistema bàsic no soluciona el problema, com ara gràfics, textos amb colors, codis font a partir de fitxers, etc. Per a això, s'utilitzen els paquets mitjançant l'ordre:

```
\usepackage[opcions]{paquet}
```

En l'exemple bàsic previ, s'inclou el paquet *Babel*, que permet expandir els caràcters per a incloure els de l'idioma espanyol.

7.5.6. Edició WYSWYG amb LaTeX

Nota

http://www.lyx.org

Si bé LaTeX està orientat a treballar com s'indica anteriorment, hi ha un editor WYSWYG anomenat *LyX* que fa un processament en temps d'escriptura del document LaTeX. No obstant això, a causa que el resultat d'un text processat amb LaTeX és fix, d'acord amb la classe del document, el *LyX* no pot ser usat amb la mateixa flexibilitat que un editor WYSWYG; encara que és molt fàcil acostumar-se a escriure amb *LyX*. Lyx també s'ha importat per funcionar en múltiples sistemes operatius.

Alguns editors WYSWYG tenen la capacitat de desar els documents en format de font LaTeX; també hi ha eines que converteixen documents d'altres formats a `.tex`. No obstant això, per a l'usuari mitjanament entrenat en LaTeX, serà més fàcil editar amb qualsevol editor i processar amb LaTeX els seus textos.

7.6. SGML

SGML és la sigla de *standard generalized markup language* i és un sistema per a organitzar i etiquetar elements en un document.

ANOTACIONS

SGML va ser desenvolupat com un estàndard internacional per l'International Organization for Standards (ISO) el 1986. SGML en si mateix no especifica cap format en particular, ja que és la norma que determina les regles per a incloure etiquetes de marcatge en els documents. I aquestes etiquetes poden ser interpretades per elements de lectura, presentació, recuperació d'informació, etc., de diferents maneres.

SGML és utilitzat en documents grans que són objecte de revisions freqüents i requereixen ser impresos en formats diferents. Com que SGML és un sistema complex i ampli, no s'utilitza massivament. No obstant això, l'estàndard HTML per al World Wide Web és una implementació d'SGML; és a dir, l'HTML és un conjunt d'etiquetes que segueixen l'estàndard SGML quan són incloses en un document.

7.6.1. Documentació en format HTML

Són molts els editors que permeten desar documents en format HTML, amb la qual cosa els deixen preparats per a ser publicats en un servidor web. No obstant això, quan parlem de documentació de programari lliure, hem de pensar en documents que han de ser modificats per diferents persones i freqüentment, per la qual cosa es requereixen eines més complexes que un editor de textos.

Eina web de documentació cooperativa: el WikiWikiWeb

Davant la problemàtica d'editar documentació per diferents autors, de mantenir versions de cada document i que aquests documents estiguin en línia immediatament, es va crear una eina anomenada *WikiWikiWeb*.

El terme *WikiWiki* (*wiki* significa 'ràpid' en la llengua hawaiana) s'utilitza en molts llocs web a Internet per a denominar una col·lecció de pàgines web d'hipertext, cada una de les quals pot ser visitada i editada per qualsevol persona. Una versió web d'un *wiki* també es diu *WikiWikiWeb*. Es tracta d'un simple joc de paraules, ja que les inicials són *WWW*, com en el World Wide Web.

ANOTACIONS

La forma abreujada *wiki* expressa l'aplicació d'informàtica col·laborativa que permet crear col·lectivament documents web usant un esquema simple d'etiquetes i marques, sense que sigui necessari revisar el contingut abans de ser acceptat per a ser publicat en el lloc web a Internet.

Atesa la gran rapidesa amb què s'actualitzen els continguts, la paraula *wiki* adopta tot el seu sentit. El document d'hipertext resultant, denominat també *wiki* o *WikiWikiWeb*, el produeix típicament una comunitat d'usuaris. Molts d'aquests llocs són immediatament identificables pel seu ús particular de paraules en majúscules, o text capitalitzat; ús que consisteix a posar en majúscules les inicials de les paraules d'una frase i eliminar els espais entre elles, com per exemple en AquestEsUnExemple. Això converteix automàticament la frase en un enllaç. Aquest wiki, en els seus orígens, es comportava així, però actualment es respecten els espais i només cal tancar el títol de l'enllaç entre dos claudàtors.

L'objectiu d'un *wiki* és democratitzar la creació i el manteniment de les pàgines eliminant la "síndrome d'un únic administrador web".

El gran potencial del *wiki* resideix en el fet que no cal aprendre a utilitzar etiquetes HTML complicades per a escriure de manera senzilla documents i establir enllaços des del lloc web.

Nota

http://www.c2.com/cgi/wiki? WelcomeVisitors

El primer WikiWikiWeb va ser creat per Ward Cunningham, que va inventar i va donar nom al concepte *wiki* i va produir la primera implementació d'un servidor WikiWiki. En paraules del mateix Ward, un *wiki* és "la base de dades en línia més simple que possiblement pot funcionar" ("*the simplest online database that could possibly work*").

En principi, un *wiki* s'usa per a qualsevol cosa que els seus usuaris vulguin i s'adopta per a documentar molt de programari lliure cooperativament. El format es presta a la col·laboració, col·laboració que involucra qualsevol persona.

Nota

http://es.wikipedia.org

Un exemple és Wikipedia; un *wiki* que té com a missió específica ser una enciclopèdia lliure i que pot actualitzar tothom qui vulgui.

ANOTACIONS

7.6.2. Documentació en format DocBook

DocBook és un dialecte d'SGML (com ho és HTML) especialment orientat a l'escriptura de documentació tècnica. En particular, és una aplicació de l'estàndard SGML/XML, que inclou una DTD pròpia i que s'utilitza de manera més destacada en l'àrea de la documentació tècnica, especialment per a documentar tot tipus de materials i programes informàtics. Hi ha un Comitè Tècnic de DocBook en OASIS (originalment SGML Open) que manté i actualitza aquest estàndard. DocBook va començar inicialment com una DTD d'SGML, però a partir de la versió 4 hi ha un equivalent per a l'XML.

Nota

http://www.oasis-open.org/

DocBook és molt utilitzat en alguns contextos, entre els quals destaquen Linux Documentation Project (Projecte de documentació Linux), les referències de les API de GNOME i GTK+, i també la documentació del nucli Linux. Les pàgines man de l'entorn operatiu Solaris es generen també a partir de documents que utilitzen les DTD de DocBook.

Norman Walsh i l'equip de desenvolupament del DocBook Open Repository mantenen un conjunt de fulls d'estil DSSSL i XSL per a generar versions PDF i HTML de documents DocBook (i també per a desenvolupar altres formats, incloent-hi pàgines de referència man i d'ajuda en HTML). Walsh és també l'autor principal del llibre *DocBook: The Definitive Guide*, la documentació oficial de DocBook. Aquest llibre es pot obtenir amb llicència GFDL i la seva versió impresa està editada per O'Reilly & Associates.

Instal·lació i consideracions preliminars

Per a poder utilitzar DocBook i poder treure un millor profit de l'escriptura, es recomana tenir instal·lades les utilitats següents:

DocBook i utilitats: aquests paquets són requerits perquè tot funcioni degudament; generalment, les distribucions de GNU/Linux ja porten paquets preparats per a instal·lar.

Emacs: si bé els documents poden ser editats amb qualsevol editor de textos, Emacs és l'únic editor orientat al context (funcionalitat molt útil a l'hora d'editar DocBook).

ANOTACIONS

Mode PSGML d'Emacs: suport per a treballar amb la DTD de DocBook a l'hora d'editar en Emacs.

Fitxer font DocBook

El següent és un exemple d'un document bàsic DocBook:

```
<!doctype book PUBLIC "-//OASIS//DTD DocBook V4.1//EN">
 <Book lang=es>
   <Bookinfo>
     <title>El meu primer document DocBook</title>
     <subtitle>Un subtítol</subtitle>
   </Bookinfo>
   <toc></toc>
   <!-- Podem posar un comentari o directament començar a escriure
        el document -->
   <Chapter>
     <title>Introducció</title>
     <para>La primera part del document i un paràgraf.</para>
     <para>Un altre paràgraf de la introducció</para>
   </Chapter>
   <Chapter>
     <title>Última part</title>
     <para>L'última part i aquí s'acaba.</para>
   </Chapter>
</Book>
```

La utilització per a editar aquest document d'una eina orientada al context (Emacs + PSGML) ens permetrà tenir una sintaxi controlada, és a dir, no ens deixarà escriure <title> si encara no hem obert un <chapter>, ja que l'editor sap quines etiquetes hi ha disponibles dins el context en què escrivim.

Utilització

Una vegada que tenim el nostre document escrit en DocBook, l'hauríem de revisar per a conèixer si no hem comès cap error de sintaxi (tancar una etiqueta; text en una etiqueta, etc.), per a això hi ha l'ordre següent:

```
nsgmls -s elmeudocument.sgml
```

Ara haurem de decidir quina sortida volem per al nostre document i ho podem fer utilitzant els processadors de textos *db2html*, *db2ps*, *db2pdf*, *db2rtf*, de la manera següent:

```
db2pdf elmeudocument.sgml
```

ANOTACIONS

7.7. Doxygen. Documentació de codi font

Doxygen és un sistema de documentació per a codis font de programes escrits en una varietat important de llenguatges, entre els quals hi ha: C++, C, Java, Objective-C, IDL (Corba), PHP, C# i D.

Doxygen pot generar documentació per a ser publicada a Internet o per a ser processada amb LaTeX. També pot generar sortides en formats RTF, PostScript, PDF i pàgines de manual de UNIX.

La idea al darrere de Doxygen és documentar en el mateix codi font a mesura que s'escriu, de manera que Doxygen extreu la documentació del mateix codi font.

Doxygen va ser creat amb GNU/Linux i Mac OS X, però ha estat importat per la majoria dels sistemes UNIX i també per Windows.

7.7.1. Utilització

Doxygen utilitza un fitxer de configuració per a determinar com ha de processar la documentació. Cada projecte haurà de tenir el seu propi fitxer de configuració que, entre altres coses, inclourà quin codi font ha de ser analitzat, quins directoris, etc.

Hi ha una manera simple de crear una plantilla del fitxer de configuració amb l'ordre:

```
doxygen -g fitxer-config
```

Les versions de Doxygen més noves porten una utilitat denominada *doxywizard*, que permet editar el fitxer de configuració gràficament mitjançant diàlegs.

Per a un projecte petit constituït per una font en C o C++, no cal fer modificacions, ja que Doxygen buscarà els fitxers de fonts en el directori actual.

Per a generar la documentació, n'hi ha prou d'executar l'ordre:

```
doxygen fitxer-config
```

ANOTACIONS

7.7.2. Documentar en els codis font

La documentació dins dels codis font ha de ser realitzada dins de blocs de text especials que puguin ser reconeguts per Doxygen i al seu torn no interfereixin en el compilador del llenguatge.

Dins de cada bloc de documentació, hi ha dos tipus de descripcions, que ajuntant-les creen la documentació:

- descripció abreujada
- descripció detallada

Per a marcar un bloc com una descripció detallada, es pot utilitzar l'estil JavaDoc, que és un format de bloc de comentari tipus C, iniciant amb un asterisc, com en aquest exemple:

```
/**
* ... text...
*/
```

També es pot utilitzar un format de comentari tipus Qt, que és un format de bloc de comentari tipus C, iniciat amb un símbol d'exclamació, com per exemple:

```
/*!
* ... text...
*/
```

En ambdós casos, l'asterisc intermedi és opcional, és a dir, que es pot acceptar:

```
/*!
... text...
*/
```

Una tercera manera de marcar un bloc és amb la notació de comentari de C++, que s'individualitza amb una barra addicional:

```
///
/// ... text...
///
```

ANOTACIONS

o també

```
//!
//! ... text...
//!
```

Per a individualitzar la descripció abreujada, també hi ha diverses possibilitats de marcatge.

Una manera és iniciar la línia amb l'ordre

```
\brief
```

en algun dels formats de blocs detallats anteriorment, seguits d'una línia en blanc:

```
/*! \brief Descripció abreujada.
 * Continuació de la descripció abreujada.
 *
 * ... text...
 */
```

Si el fitxer de configuració es configura amb `JAVADOC_AUTOBRIEF` en `YES`, és possible utilitzar l'estil de blocs de JavaDoc que automàticament inicia la descripció abreujada fins al pròxim punt seguit d'un espai o una línia nova; com en aquest exemple:

```
/* * Descripció abreujada que acaba en un punt. Descripció
* detallada que comença després.
*/
```

Un tercer mètode per a diferenciar els tipus de descripcions és utilitzar un comentari en format C++ en una sola línia.

```
/// Descripció abreujada.
/* * Descripció detallada */
```

o també deixar una línia en blanc entre una descripció i l'altra:

```
//! Descripció abreujada.

//! Descripció detallada
//! en més d'una línia
```

ANOTACIONS

En aquest últim cas, haurà de tenir `JAVADOC_AUTOBRIEF` explícitament indicat en `NO` perquè siguin detectades pròpiament les descripcions.

Això mostra com Doxygen és flexible per a adaptar-se als formats de comentaris de cada un dels desenvolupadors de programari.

7.8. Conclusions

La documentació en programari lliure adquireix una rellevància molt més gran que en sistemes de llicenciament privatius, ja que en programari lliure es busca la cooperació en el desenvolupament; per això, tenir una bona documentació ajudarà els desenvolupadors i usuaris a conèixer millor i més ràpidament l'eina de programari, amb la qual cosa quedaran habilitats per a cooperar en el projecte, sigui informant d'errors, proposant millores, aportant codi o contestant preguntes a altres usuaris i desenvolupadors.

Els sistemes de documentació presentats aquí permeten tenir una idea de les característiques generals de cada un d'ells i són una guia breu per a poder construir un document bàsic utilitzant les eines descrites; dependrà ara dels estudiants aprofundir el sistema o sistemes que considereu més apropiats per a obtenir-ne un coneixement complet.

7.9. Altres fonts de referència i informació

Referencias de Tex y Latex

Michael Doob. *A Gentle Introduction to TeX.*

George Gratzer. *First Steps in LaTeX.*

SGML

SGML for Dummies (SGML per a principiants)

Developing SGML DTDs: From text to Model to Markup

DocBook

DocBook: the definitive guide

Doxygen

http://www.stack.nl/~dimitri/doxygen/manual.html

ANOTACIONS

8. Comunitats virtuals i recursos existents

8.1. Introducció

Des del seu començament, el desenvolupament del programari lliure ha estat molt relacionat amb el desenvolupament d'Internet.

Al principi, els desenvolupadors es comunicaven amb llistes de correu en les xarxes precursores d'Internet i en la mesura que aquestes es desenvolupaven, també ho feia la seva manera de comunicar-se.

Amb el creixement actual de la xarxa i els llenguatges nous de desenvolupament web, hem arribat a un punt en què la quantitat de serveis oferts en portals per als desenvolupadors i enginyers de programari és enorme, i no només inclou eines que ajuden aquests en el seu desenvolupament diari, sinó que també posen en contacte patrocinadors de projectes amb desenvolupadors, de manera que el programari lliure pot créixer i expandir-se.

D'altra banda, els desenvolupadors troben llocs en què poden fer publicitat d'ells mateixos, amb la qual cosa aconsegueixen una notorietat molt valuosa per al seu desenvolupament personal i laboral.

8.2. Objectius

Després de concloure el capítol, els lectors sereu capaços del següent:

- Valorar les opcions en les quals podeu fer publicitat dels vostres projectes de programari o dels que són gestionats per a tercers.
- Conèixer el funcionament dels portals més comuns com ara Freashmeat i Sourceforge, i aprendre el sistema de creació

ANOTACIONS

d'un compte personal, de projecte i la gestió d'aquest i dels seus subscriptors.

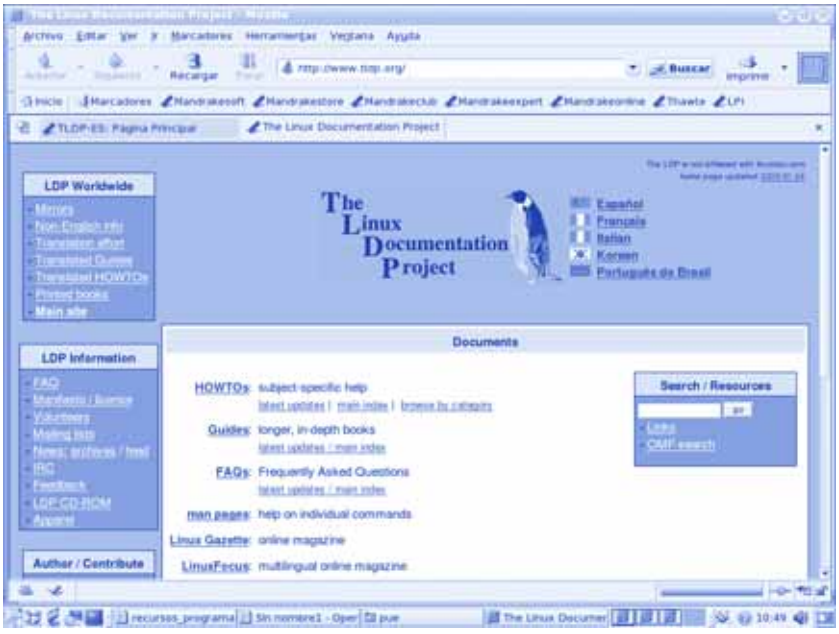
8.3. Recursos documentals per a l'enginyer de programari lliure

En aquest punt, mirarem de mostrar els recursos documentals més coneguts amb els quals els enginyers de programari lliure disposareu no només de manuals, sinó de cursos, guies, normativa, tipus de llicència i, fins i tot en alguns casos, de projectes complets dels quals prendre exemple per a elaborar el vostre.

8.3.1. http://www.tldp.org

La sigla *TLDP* significa *the Linux documentation proyect*. Aquesta pàgina té versió en diferents idiomes:

Figura 1



Des d'aquí, podem accedir a molta documentació lliure:

1) Guies
2) Com es fa
3) PMF

4) Pàgines de manual Linux
5) Manuals
6) Cursos

A més, tindrem accés a diversos serveis com el de publicacions, el de desenvolupament de projectes, enllaços amb altres projectes i pàgines en general.

8.3.2. http://www.freestandards.org

Aquesta organització (Free Standards Group) independent té com a intenció accelerar l'ús del programari lliure i de codi obert mitjançant el desenvolupament i la promoció d'estàndards o regles.

Per a això, crea eines d'ajuda per a desenvolupadors de programari lliure. A més, s'encarrega de testar i certificar mitjançant programes que el programari desenvolupat segueix els estàndards que hi ha establerts.

Una altra funció d'aquest grup és ajudar a migrar cap al sistema operatiu Linux aquells que estiguin acostumats a manejar altres entorns.

Figura 2

En la pàgina, tindrem informació mitjançant notícies i premsa sobre tot el desenvolupament de programari lliure, sobre empreses relaci-

ANOTACIONS

onades amb el camp i sobre els seus projectes nous dins l'àrea de Linux.

Nota

http://www.linuxbase.org

Es donen certificacions a venedors de programari a través d'LSB (Linux Standard Base). Des de la pàgina, hi ha un enllaç a LSB amb el qual es poden rebre les certificacions descrites abans. LSB és un grup de treball de *freestandards*, però no l'únic.

A més, hi ha els següents:

- Openi18n. http://www.openi18n.org/
- LANANA. http://www.lanana.org/
- Openprinting. http://www.openprinting.org/
- Accesibility. http://accessibility.freestandards.org
- DWARF. http://dwarf.freestandards.org/
- Open Cluster. http://www.opencf.org/home.html

Cada un dels grups de treball s'encarrega d'una àrea diferent dins l'entorn del programari lliure i per tant de Linux.

8.3.3. http://www.unix.org

Aquest producte, i la pàgina, pertanyen a una empresa denominada Open Group, que està especialitzada en el desenvolupament d'estàndards globals i integració d'informació per a una interacció global. La tecnologia que ven pretén ser neutral i, per tant, segueix els paradigmes que es persegueixen dins el programari lliure i, en conseqüència, del sistema operatiu Linux.

UNIX és el sistema operatiu predecessor de Linux. Moltes de les característiques de Linux s'hereten de UNIX (sistema robust, estable, multiusuari, multitasca, multiplataforma, etc.). Per això, aquesta pàgina proporciona informació molt interessant dins l'entorn del programari lliure: des de certificacions estàndard fins a consultes i notícies relacionades amb el sistema operatiu. A més, té fòrums amb l'objectiu clar de reunir professionals per a posar en comú coneixements relacionats amb el sistema i el programari que s'hi executa.

ANOTACIONS

Figura 3

8.3.4. http://www.opensource.org

És una organització sense ànim de lucre l'objectiu de la qual és gestionar i promocionar la definició de codi obert per al bé de la comunitat.

La llicència codi obert implica una sèrie de característiques concretes:

1) Redistribució lliure.
2) Accés al codi font.
3) Permís per a realitzar treballs derivats.
4) Garantia de la integritat del codi font de l'autor.
5) No-discriminació de persones o grups.
6) No-discriminació de cap possible camp de l'aplicació.
7) Llibertat per a distribuir la llicència.
8) La llicència no ha de ser específica d'un producte.
9) La llicència no ha de posar restriccions d'ús sobre cap altre programari.

ANOTACIONS

Figura 4

La pàgina ofereix els paràmetres, marca les pautes de la definició i defineix estàndards d'*open source* i el que comporta.

8.3.5. http://standards.ieee.org/reading/ieee/std_public/description/posix/

Aquesta pàgina es dedica per complet a definir estàndards sota la normativa IEEE (*Institute of Electrical and Electronic Engineering*). Concretament d'estàndard POSIX.

POSIX són les inicials de *Portable Operating System Interface* en *uniX*. És un estàndard que pretén aconseguir la portabilitat del programari en el nivell del codi font. Dit d'una altra manera: un programa escrit per a un SO que sigui compatible amb POSIX s'ha de poder compilar i executar sobre qualsevol altre POSIX, encara que sigui d'un altre fabricant diferent. L'estàndard POSIX defineix la interfície que el SO ha d'oferir a les aplicacions: el joc de crides al sistema. POSIX és desenvolupat per IEEE i és estandarditzat per ANSI (American National Standards Institute) i ISO (International Standards Organization). Evidentment, POSIX està basat en UNIX.

ANOTACIONS

Figura 5

8.3.6. http://www.faqs.org

PMF o Preguntes Més Freqüents (FAQ o *Frequently Asked Questions*) pretén respondre totes aquelles preguntes relacionades amb el món de la informàtica, el programari, els sistemes operatius i també d'altres camps no relacionats directament amb la tecnologia.

S'hi pot accedir per categories, per autor, per número de referència, etc.

En cada PMF tindrem els RFC (*Request For Comments*) corresponents, és a dir, les PDC (Peticions Per Comentaris) o respostes a les preguntes freqüents. La base de dades és molt extensa, per la qual cosa moltes vegades cal fer recerques per algun dels camps que defineixen les preguntes i les respostes.

ANOTACIONS

Figura 6

8.3.7. http://www.dwheeler.com/oss_fs_refs.html

Pàgina que ofereix enllaços a pàgines relacionades amb el programari *Open Source Software* i el *Free Software* (OSS/FS).

Figura 7

Com podem veure, hi ha molts recursos documentals. L'enginyer d'PL ha de saber, almenys, on trobar documentació sobre les aplicacions principals que integren els projectes de programari lliure, per a la qual cosa el més còmode és dirigir-se al *Linux documention project* i, més concretament, a la seva zona traduïda a l'espanyol per a conèixer aquelles pàgines que el poden ajudar a resoldre problemes de legislació o assimilació d'estàndards, i també les web de propòsit general i de preguntes més corrents.

8.4. Comunitat

Amb la presentació d'aquestes pàgines, intentem donar únicament una referència a la gran quantitat de fòrums sobre diferents temes relacionats amb el programari lliure. Les webs que detallem són, en el cas de les relacionades amb preguntes i respostes, recopilacions de llistes de distribució de gran quantitat de temes. D'altra banda, en mostrem una en què el mateix desenvolupador es fa publicitat i es pot relacionar amb altres desenvolupadors ampliant la seva xarxa de relacions.

8.4.1. http://www.mail-archive.com

Aquesta pàgina proporciona un servei d'accés a fitxers pertanyents a llistes de correus públiques. A més, permet afegir llistes noves per tal d'incloure els seus fitxers per a recerques futures.

És un bon lloc per a buscar informació de tot tipus, inclosa la relacionada amb el programari lliure. La comunitat Linux disposa d'una gran quantitat de llistes de correu, en què es resolen dubtes i es posen en comú els desenvolupaments i coneixements dels usuaris que hi pertanyen.

ANOTACIONS

Figura 8

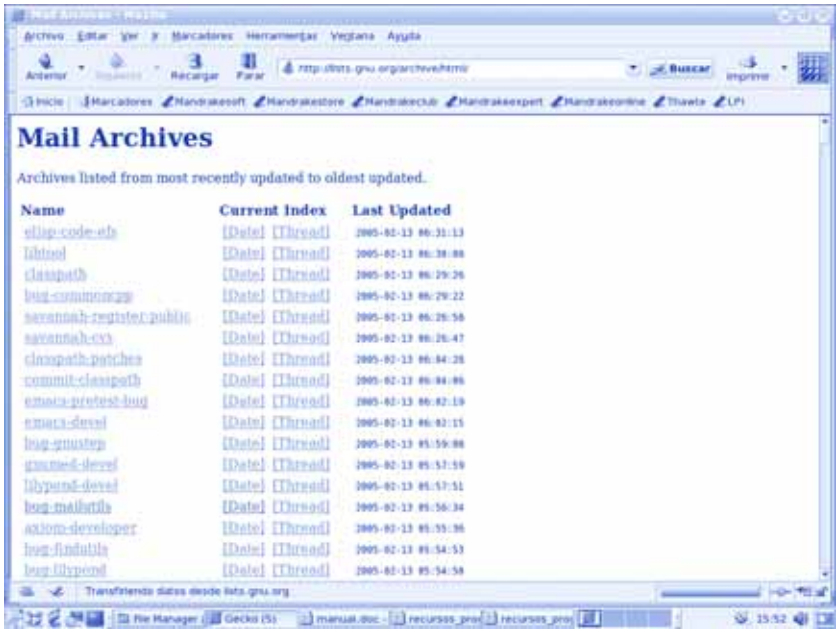
8.4.2. http://mail.gnu.org/archive/html/

La pàgina proporciona totes les llistes de correu que pertanyen a GNU. GNU (GNU isn't UNIX) és un projecte de FSF (*Free Software Fundation*). El seu objectiu és desenvolupar un sistema operatiu basat en UNIX però amb la idea del programari lliure de fons. GNU és l'estàndard que segueixen les noves distribucions Linux per a permetre una integració màxima dins els sistemes operatius que hi ha amb l'epígraf *free software*.

La informació a què accedim des d'aquesta pàgina ens ajudarà a entendre millor la filosofia GNU i a més ens servirà de suport en el cas d'intentar desenvolupar programari lliure.

ANOTACIONS

Figura 9



8.4.3. http://www.advogato.org/

Advogato, segons el seu *mission statement*, és un lloc per a la comunitat de desenvolupadors *open source*. El lloc, amb un disseny molt simple, ofereix la possibilitat de crear un compte a qualsevol persona i disposa d'un sistema molt interessant de certificació d'intrausuaris denominat *trust-metric*, que permet que els usuaris del lloc "certifiquin" els altres usuaris. Bàsicament una evolució *web-style* del vell tema de la validació d'usuaris en els BBS.

Nota

http://www.advogato.org/trust-metric.html

L'opció més interessant del lloc consisteix a crear un "diari" públic que els altres poden llegir i comentar, en què els desenvolupadors poden compartir el seu dia a dia. És interessant llegir com treballen els altres o què pensen, i serveix, eventualment, com una base del que un ha anat fent durant el temps.

Altres opcions són crear "projectes", establir relacions entre projectes i usuaris, certificar usuaris i enviar articles (es requereix un cert nivell de certificació).

El disseny simple i sobri i la serietat general del lloc el fan més que recomanable. El programari amb què està fet el lloc està disponible, és un mòdul en llenguatge C per a Apache, una cosa realment curiosa.

ANOTACIONS

Figura 10

8.5. Allotjar projectes de programari lliure (Sourceforge)

Amb aquest apartat, intentem mostrar per mitjà de Sourceforge –l'espai d'allotjament i gestió de projectes més important del moment– la manera més habitual de crear un compte de projecte, a més de les opcions i facilitats que ens ofereix aquest lloc.

L'operativa de creació de projecte i les utilitats que ens dóna són semblants. L'única barrera pot ser l'idioma, per a la qual presentem l'alternativa en castellà, *software-libre.org*.

8.5.1. Sourceforge.net

Sourcefoge.net és la forja per excel·lència. Aquí es creen i s'allotgen més del 80% dels projectes de programari lliure que hi ha.

Encara que com el seu nom indica és una incubadora de projectes, aquests gairebé mai no desapareixen de Sourceforge en assolir la majoria d'edat, sinó que continuen conservant les pàgines inicials del projecte, encara que tinguin el seu propi domini i pàgina inicial en el futur.

ANOTACIONS

El seu problema més gran, encara que per a qualsevol programador que es preï no ho hauria de ser, és l'idioma. El seu idioma principal és l'anglès, encara que puguem allotjar un projecte en espanyol o un altre, les eines, utilitats i navegació del portal són en anglès.

En aquest capítol, repassarem la totalitat de Sourceforge. Crearem un usuari i un projecte i una vegada creat, farem un tomb per les utilitats que Sourceforge posa a disposició del programador.

8.5.2. Crear un compte d'usuari

Crear un compte d'usuari de Sourceforge és senzill, en primera instància només ens demana una contrasenya i una adreça electrònica.

En un segon pas, hem d'introduir un nom d'usuari per a l'accés al portal i un nom publicitari o nom amb què apareixerà el nostre usuari i els projectes que allotgem sota aquest nom.

A més, se'ns pregunta l'idioma del projecte i si volem rebre correu de les llistes d'actualització i de la resta de comunitats.

Després de realitzar el registre complet, rebem en la nostra bústia un missatge de confirmació que ens indica que si volem acabar el nostre registre hem de prémer l'enllaç que apareix amb l'epígraf "[...] *please access the following URL:*".

Al mateix temps que rebem el missatge de confirmació, acaba el procés de registre amb una pàgina en què se'ns detallen les dades del nostre usuari: nom del compte d'usuari; àlies per al compte de creació, per exemple, usuari@domini.com; compte de Sourceforge, que és un àlies al nostre compte real; la pàgina en què podem modificar les nostres dades d'usuari i en què podem trobar informació, documentació i respostes a les nostres preguntes més corrents.

Una vegada creat el compte, hi podem accedir amb el nostre nom d'usuari i la nostra contrasenya. Aquesta última la vam introduir en el primer pas de creació del compte. A més, des d'aquest mateix

ANOTACIONS

compte podem sol·licitar un contrasenya nova en cas d'oblit o crear un compte nou.

En accedir, s'obre la nostra pàgina d'usuari, en què apareix tota la informació d'aquest, si tenim o no projectes en curs o pendents d'activació, si s'han fet donatius al nostre usuari i si tenim alguna pregunta pendent amb el centre de suport, entre d'altres.

8.5.3. Alta d'un nou projecte

La primera vegada que creem un compte, no se'ns demana que introduïm un nom real per a poder activar-lo, però aquest procés sí que és necessari en el cas de voler donar d'alta un projecte nou.

D'altra banda, depenent del tipus d'usuari que hem creat i del tipus de llicència que utilitzem, pot ser interessant que habilitem els donatius; per a fer-ho, hem de disposar d'un compte en PayPal. Aquest sistema permet transferir fons a un compte sense necessitat d'utilitzar targeta de crèdit.

Qualsevol que consideri útil el nostre projecte podrà donar la quantitat de diners que cregui oportuna i així ajudar-nos a mantenir-lo.

Quan creem un projecte nou, que no és necessari que sigui lliure, se'ns indica, per si no ho sabem, quins avantatges té l'PL i la creació d'aquest tipus de projectes davant el programari propietari.

El primer pas és indicar el tipus de projecte que volem publicar, en el nostre cas, documentació. D'altra banda, se'ns pregunta si hem entès el significat de la definició d'*open source*.

ANOTACIONS

Figura 11. Esquema del procés de registre d'un projecte

Abans de continuar, se'ns indiquen els passos que seguirem i, en primer lloc, el temps que durarà aquest procés d'alta de projecte.

Acceptem els termes i condicions per a allotjar un projecte en Sourceforge i incloem les dades de la llicència. Si el nostre projecte es desenvolupa sota una llicència que no apareix en la llista, per exemple, GFDL, escrivim una descripció breu d'aquesta en el requadre inferior.

El pas següent és escriure una descripció pública i una altra per a registrar-la en la base de dades de Sourceforge.

Finalment se'ns demana que repassem totes les dades introduïdes per si hi ha cap error.

Després d'aquest repàs, rebem un correu que confirma la creació del projecte i el pas a aprovació, és a dir, el projecte serà revisat per per-

ANOTACIONS

sonal de Sourceforge i serà allotjat definitivament. Aquest procés sol durar dos o tres dies hàbils.

8.5.4. Utilitats que Sourceforge posa a disposició dels usuaris d'un projecte

Una vegada acabat el procés de creació d'un projecte, es posa a disposició de l'administrador del projecte una sèrie d'eines que permetran un treball col·laboratiu correcte dels diferents partícips:

En la seva barra d'eines, ens podem fer una idea del que se'ns ofereix.

Figura 12. Barra d'eines de projecte

En aquest cas, és per al projecte Gaim, que proporciona el desenvolupament d'un client de missatgeria instantània compatible amb gairebé tots els servidors disponibles.

Com veiem, se'ns mostra un sumari del projecte amb la seva descripció, àrees públiques disponibles i últimes notícies.

Figura 13. Descripció del projecte

A dalt, podem veure la descripció del projecte. I en les parts següents, les àrees públiques disponibles.

ANOTACIONS

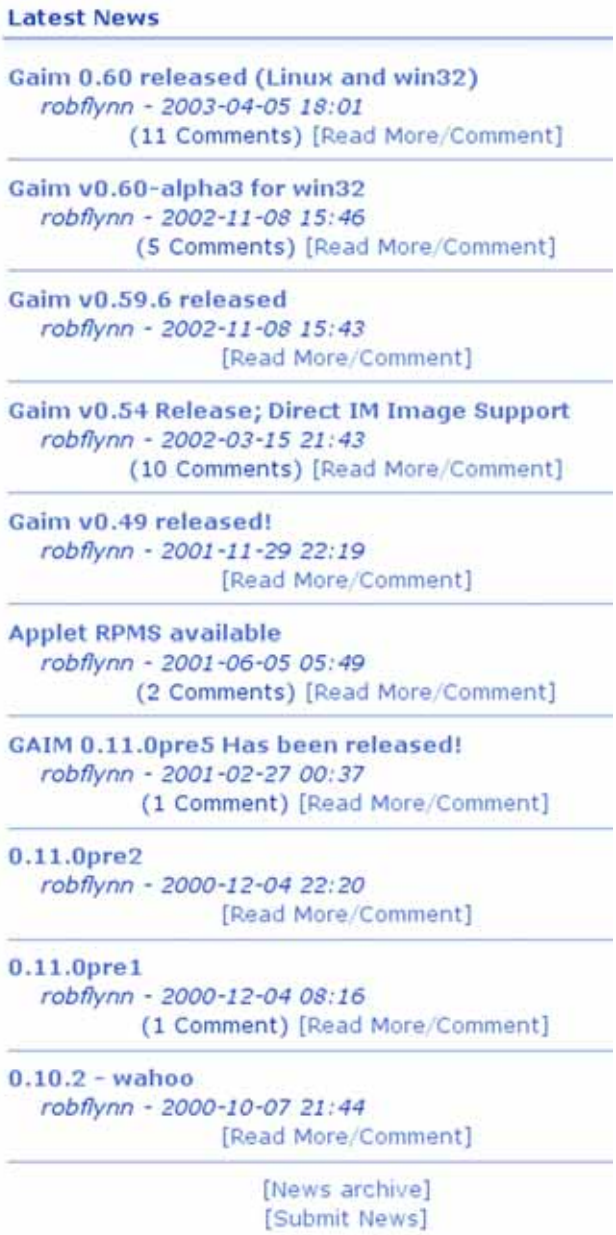
Figura 14. Àrees públiques disponibles

I també les últimes notícies:

Apareixen les últimes versions i addicions al projecte en forma de llista ordenada per data.

ANOTACIONS

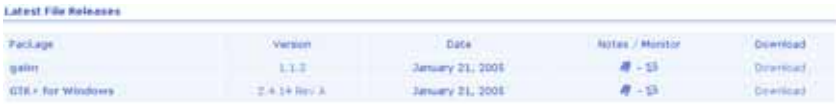
Figura 15. Últimes notícies



Zona de baixades d'últimes versions:

Tant de versions finals com d'utilitats o programari addicional necessari.

Figura 16. Zona de baixades



ANOTACIONS

Zona de fòrums

Figura 17. Fòrums

Seguiment d'àrees de treball, i zona d'errors i pedaços. Tots inclosos com a opcions visibles d'un sistema complet de control de versions.

Per tant, en Sourceforge no només incloem el nostre projecte, sinó que també se'ns dóna l'espai necessari per a allotjar-lo i multitud d'eines que ajuden els programadors a col·laborar: comunicant-se en els fòrums, enviant els seus treballs amb el CVS, a més de petites utilitats com la readreça web per al nostre projecte o la readreça del correu electrònic del nostre compte de Sourceforge al nostre compte personal.

Tot això només amb la recomanació de desenvolupar el nostre projecte com un projecte obert en què es permeti que altres programadors col·laborin i amb una infinitat de llicències obertes.

Des de fa relativament poc, s'ha afegit a Sourceforge l'opció *power*, en què per un preu assequible s'ofereixen a l'usuari certs serveis no disponibles en la gratuïta com ara:

- Accés a utilitats de cerca avançada per als projectes de Sourceforge
- Suport tècnic prioritari
- Seguiment de projectes
- Baixades directes de programari sense passar per servidors de rèplica

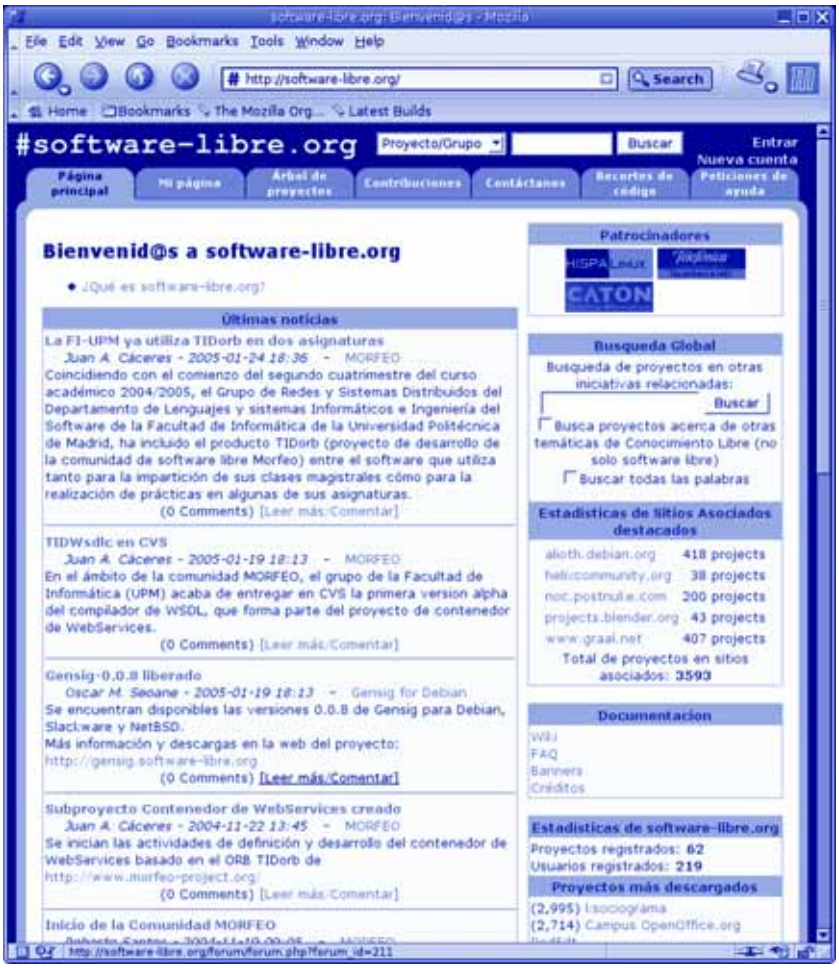
ANOTACIONS

8.5.5. Software-libre.org

L'homònim espanyol de Sourceforge és *software-libre.org*. Aquest portal utilitza el mateix programari que Sourceforge i acull projectes gratuïtament però en espanyol.

És una opció molt vàlida per a tots aquells desenvolupadors hispans que vulguin fer públics projectes en la seva zona d'influència o que no coneguin l'anglès.

Figura 18. Principal de software-libre.org



La seva llista de projectes és molt més reduïda que la de Sourceforge i també és molt més jove. Va néixer el 2003, per la qual cosa el seu desenvolupament ha estat molt més petit.

Figura 19. Projectes disponibles

El procés de registre d'usuaris i projectes és molt semblant al de Sourceforge.

Aquesta pot ser una opció bona, encara que molt localitzada, per a allotjar els nostres projectes. De moment, el seu ús és limitat i gairebé es localitza a Espanya, però el sistema creix contínuament. Esperem que en poc de temps pugui ser la referència hispana com a forja de projectes lliures.

8.5.6. Fer públics projectes de programari lliure i obtenir notorietat

Cada dia és més necessari –tant per a trobar col·laboradors com patrocinadors– fer públics els nostres projectes amb les eines adequades. Gairebé cap eina no compleix totes les expectatives, però sí que podem aconseguir l'objectiu desitjat usant-ne unes quantes.

La més usada de totes per a publicar projectes de programari i relacionar-se amb altres desenvolupadors és *Freshmeat.net*.

Ens introduirem en aquesta eina i més endavant en mostrarem una altra de suport que completarà el nostre objectiu.

ANOTACIONS

8.5.7. Freshmeat.net

Figura 20. Pàgina d'inici

Com veiem en la seva pàgina principal, amb Freshmeat aconseguirem publicar el nostre projecte de programari fent que aparegui en la llista de la seva pàgina principal, de manera que tant desenvolupadors interessats a participar-hi com usuaris interessats a usar-lo hi puguin accedir sense necessitat de buscar-lo.

Proporciona un control de versions que ens permet anar afegint revisions successives i comprovar-ne l'historial al mateix temps que ens posa en contacte amb altres desenvolupadors de projectes semblants amb què podem cooperar.

Crearem un compte d'usuari i seguirem el procés de creació d'un projecte fictici:

- Crear un compte d'usuari és senzill. Només se'ns sol·licita un nom d'usuari, un nom de projecte que serà visible en la llista principal de Freshmeat, una adreça de correu i una contrasenya. Com a dada opcional, podem introduir el lloc web del projecte.

ANOTACIONS

- Una vegada creat l'usuari, rebem un missatge de confirmació i activació. Com en el cas de Sourceforge, hem de prémer l'enllaç per a activar el nostre compte.

Una vegada creat l'usuari des de la seva pàgina principal, podem fer les tasques necessàries per a administrar el nostre compte i les seves preferències personals:

Figura 21. Administració de preferències de projecte i personals

I també per a crear un projecte de visibilitat nou. Totes aquestes opcions, les podem realitzar des de dos menús:

- El principal

Figura 22. Menú principal

Des del qual podem afegir un projecte nou o accedir a informació general del portal.

ANOTACIONS

- I el d'usuari

Figura 23. Menú d'usuari

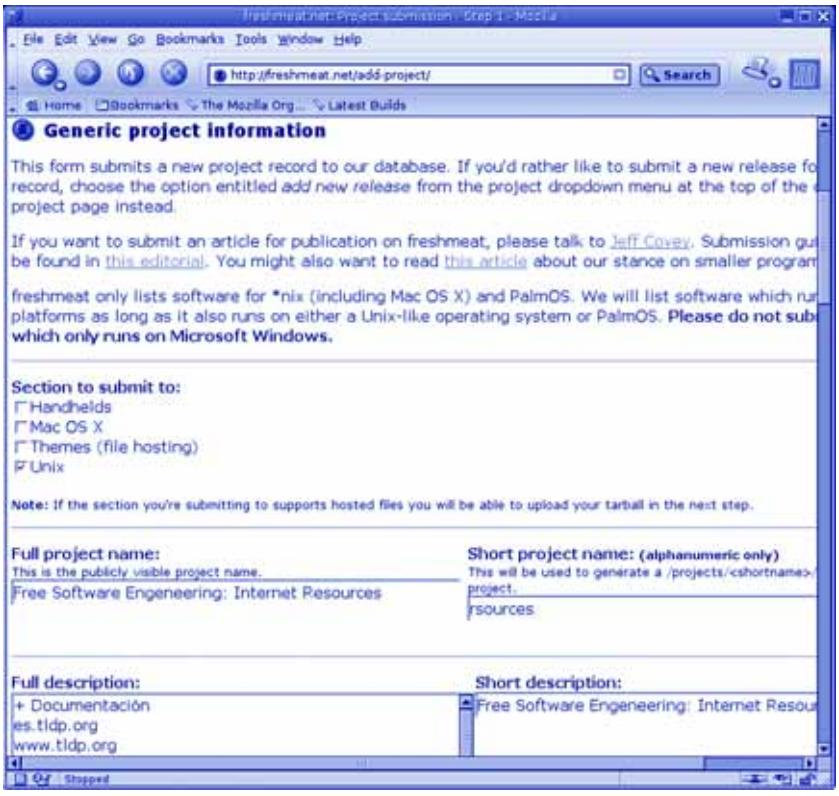
home | filters | friends | inbox | password | preferences | stored searches | subscriptions | logout

Des d'on podem afegir filtres per a veure només aquells projectes que ens interessin i accedir als missatges que ens puguin haver enviat altres usuaris, a les preferències del nostre usuari, a les estadístiques, etc.

L'opció d'amics és realment interessant, ja que des d'aquesta mantenim la nostra llista de contactes de Freshmeat, una de les coses més importants a l'hora de fer públic el nostre projecte i col·laborar amb la resta de la comunitat.

Com s'afegeix un projecte nou? Solament hem de prémer l'enllaç *submit* del menú principal i ho podrem fer:

Figura 24. Crear un projecte nou: informació general



Com veiem és senzill, només hem d'escollir el sistema operatiu del projecte, el nom complet, el nom curt, la descripció completa i la descripció curta d'aquest.

ANOTACIONS

I seguidament hem d'afegir el projecte a una categoria i repassar les dades que hem introduït tal com mostra la imatge següent.

Figura 25. Dades del nostre projecte

Amb aquests tres passos simples i esperant la confirmació de l'administrador del lloc, que sol trigar uns dos dies, tindrem el nostre projecte creat i llest per a enviar comunicacions i avisos de noves versions o notícies.

Descripció funcional de Freashmeat

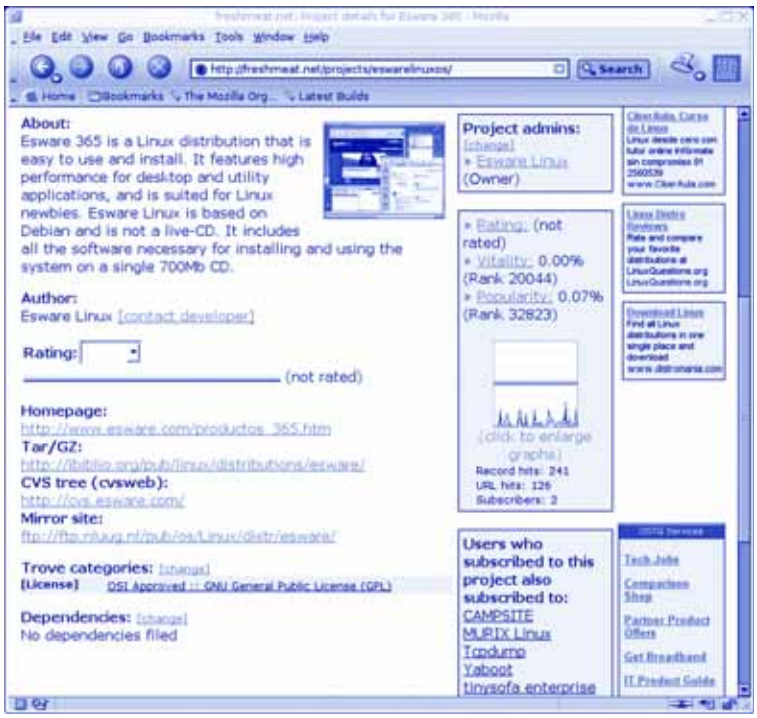
Des del menú d'usuari i amb un projecte donat d'alta, podem accedir a l'administració d'aquest mitjançant el menú de projecte:

Figura 26. Menú de projecte

I en la pàgina principal del projecte podem veure els subscriptors del projecte, les estadístiques de visita, una descripció i una imatge de previsualització de l'eina, i també pàgines d'interès sobre aquesta:

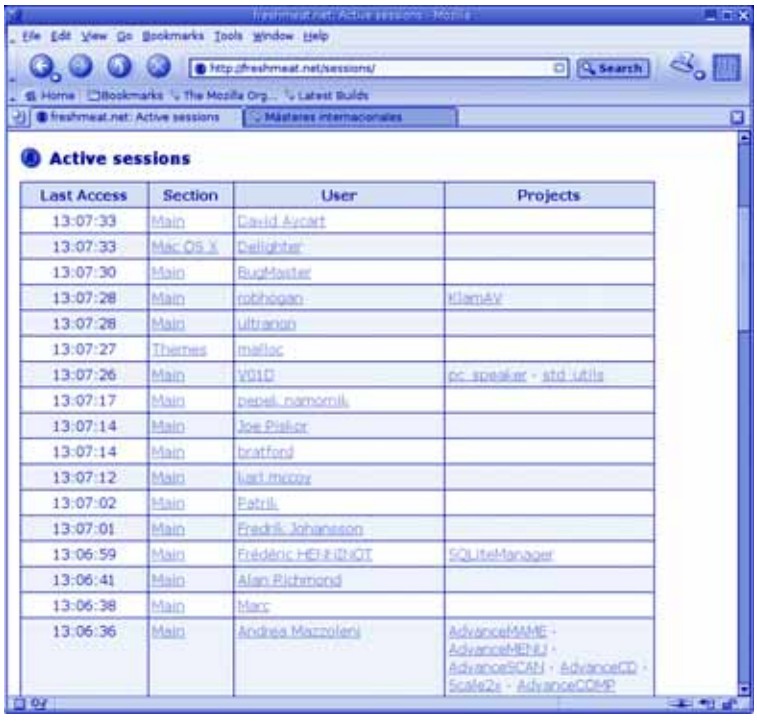
ANOTACIONS

Figura 27. Pàgina principal del projecte



No obstant això, la millor utilitat oferta per Freshmeat és la seva llista de desenvolupadors des de la qual es pot localitzar la pràctica totalitat dels mantenidors dels projectes de programari lliure més coneguts.

Figura 28. Líders de projecte connectats



8.6. Com podem obtenir notorietat per als nostres projectes?

La notorietat per als nostres projectes i per a nosaltres mateixos s'obté per regla general amb el treball d'anys, però també amb ajuda d'índexs temàtics i cercadors.

En donarem tres com a orientació, i intentarem, a més, que siguin referències hispanes, cosa una mica difícil.

- **dmoz.com**. Dmoz és un índex temàtic que intenta indexar pàgines adjuntes a determinats epígrafs, de manera que la cerca resulta molt més natural i centrada.

  En Dmoz, participen gran nombre d'editors de contingut que es dediquen a mantenir pàgines de confiança dins d'epígrafs de confiança, és a dir, si nosaltres sabem de l'existència d'una pàgina, de la qual podem ser els administradors, ens podem fer editors d'aquesta en una secció determinada, per exemple, desenvolupament de programari, programari lliure, etc., de manera que mantenint-la en l'índex, en fem publicitat.

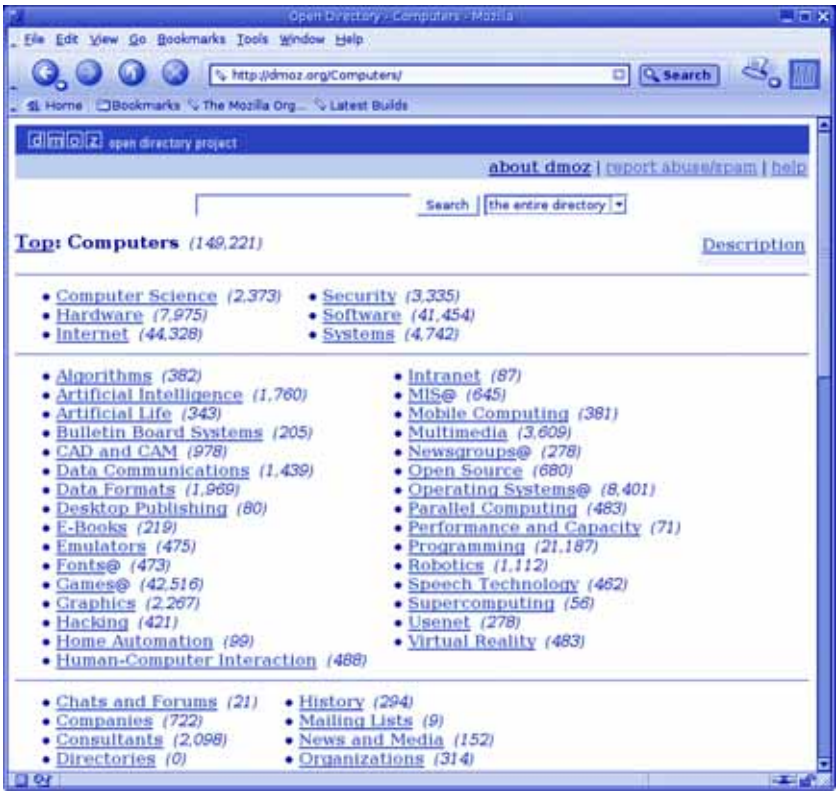
Figura 29. Principal de Dmoz.com

ANOTACIONS

- Contingut d'un directori

Dins del directori *computers*, hi ha infinitat de subdirectoris entre els quals trobem el d'*open source*. Podem mantenir el nostre projecte en el seu interior.

Figura 30. Pàgina de recursos sobre informàtica



- **linux.bankhacker.com**. Una pàgina espanyola molt consultada per a projectes és la del grup de desenvolupadors de Bankhacker. S'hi poden suggerir productes de programari per a incloure'ls en el directori de programari lliure i Linux.

ANOTACIONS

Figura 31. Principal de Bankhacker.com

Podem suggerir una aplicació en concret emplenant el formulari que apareix després de prémer l'enllaç Suggerir.

8.7. Conclusions

La conclusió final d'aquest apartat és clara. Disposem de multitud de recursos útils per als enginyers del programari lliure. No n'hi ha un de sol per a allotjar projectes, però sí un de més important que els altres. El mateix ocorre amb els recursos per a fer publicitat dels nostres projectes i de nosaltres mateixos com a desenvolupadors, n'hi ha un de més important, com és Freshmeat, però no és l'únic, sinó que hi ha d'altres que cobreixen deficiències d'aquest.

El que és clar és que cada dia surten nous recursos útils i nous projectes recomanables que no sempre es fan públics en els mateixos llocs i que no sempre arribem a conèixer.

Es necessita una centralització a la qual s'està arribant mitjançant la sindicació de continguts i notícies, però encara queden barreres per superar i una de les més paleses és la de l'idioma.

ANOTACIONS

Ara com ara, els desenvolupadors han de conèixer l'anglès per a poder donar a conèixer els seus projectes. Esperem que en un futur pròxim els recursos principals també estiguin traduïts per a apropar-los als desenvolupadors no angloparlants.

ANOTACIONS

Appendix A. GNU Free Documentation License

A.1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of *copyleft* which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

ANOTACIONS

A.2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The *Document* below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as *you* You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A *Modified Version* of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A *Secondary Section* is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The *Invariant Sections* are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The *Cover Texts* are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

ANOTACIONS

A *Transparent* copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not *Transparent* is called *Opaque*.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The *Title Page* means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, *Title Page* means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section *Entitled XYZ* means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as *Acknowledgements*, *Dedications*, *Endorsements*, or *History*. To *Preserve the Title* of such a section when you modify the Document means that it remains a section *Entitled XYZ* according to this definition.

ANOTACIONS

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

A.3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.4. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

ANOTACIONS

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the

ANOTACIONS

Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

- C. State on the Title Page the name of the publisher of the Modified Version, as the publisher.

- D. Preserve all the copyright notices of the Document.

- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

- H. Include an unaltered copy of this License.

- I. Preserve the section Entitled *History*. Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled *History* in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the *History* section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- K. For any section Entitled *Acknowledgements* or *Dedications* Preserve the Title of the section, and preserve in the section all the

ANOTACIONS

substance and tone of each of the contributor acknowledgements and/or dedications given therein.

- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

- M. Delete any section Entitled *Endorsements* Such a section may not be included in the Modified Version.

- N. Do not retitle any existing section to be Entitled *Endorsements* or to conflict in title with any Invariant Section.

- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled *Endorsements*, provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

ANOTACIONS

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled *History* in the various original documents, forming one section Entitled *History*; likewise combine any sections Entitled *Acknowledgements*, and any sections Entitled *Dedications*. You must delete all sections Entitled *Endorsements*.

A.7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this

ANOTACIONS

License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an *aggregate* if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

A.9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a

ANOTACIONS

translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled *Acknowledgements*, *Dedications*, or *History*, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

A.10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

A.11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License *or any later version* applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version

ANOTACIONS

number of this License, you may choose any version ever published (not as a draft) by the Free Software Foun2dation.

A.12. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled *GNU Free Documentation License*.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the *with...Texts* line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

ANOTACIONS


UOC
www.uoc.edu