Projectes de jocs i entorns interactius

CFGS.A3D.M05/0.18

Animacions 3D, jocs i entorns interactius

Generalitat de Catalunya
Departament d'Educació

ioc
institut obert
de catalunya

Aquesta col·lecció ha estat dissenyada i coordinada des de l'Institut Obert de Catalunya.

Coordinació de continguts
Paloma Bañuls Caruana

Redacció de continguts
Sara Sunyé, Rodrigo Pizarro, José Manuel Solís

Primera edició: setembre 2018

Dipòsit legal:

Introducció

Actualment, navegar i explorar continguts en aplicacions interactives, així com interactuar través d'entorns interactius i videojocs és part de la nostra vida quotidiana. Però, com es produeixen aquests productes?

La producció i desenvolupament de jocs i projectes interactius és complexa. Hi intervenen molts perfils professionals especialitzats, i s'han de tenir en compte molts aspectes que impliquen disciplines molt diverses: usabilitat, accessibilitat, ergonomia, gestió de continguts, programació, disseny gràfic... Tots aquests aspectes han d'estar molt ben lligats perquè el projecte sigui exitós i la producció es desenvolupi de manera eficaç.

El perfil professional d'aquest títol ha d'estar familiaritzat amb totes les fases de producció del projecte interactiu, tant la fase de preproducció, o disseny del projecte, com la de desenvolupament, que implica la programació i depuració del codi, així com els mètodes de difusió del projecte acabat.

Ja sigui un lloc web, una aplicació mòbil, o un entorn de realitat virtual, en tots els casos abans de començar s'ha de dissenyar i planificar el projecte que es vol portar a terme.

En aquest mòdul l'alumne aprendrà a establir els requisits i característiques del projecte, a definir la seva estructura, a planificar la seva producció, a gestionar els continguts digitals que intervenen i a controlar les versions que es generen del producte durant la seva producció.

En la primera unitat **"Disseny i planificació del projecte"**, s'explica com realitzar el document de disseny i planificació del projecte, on s'especifiquen els seus objectius, estil gràfic i estructura del projecte. Es tractarà com determinar l'arquitectura tecnològica del projecte en funció de les seves característiques que s'han definit prèviament. Per últim, es tractarà com organitzar i catalogar els continguts i diferents tipus de fonts que s'inclouran en el projecte.

En la segona unitat, **"Producció i avaluació del projecte"**, veureu eines i processos per fer el seguiment del projecte i planificar el treball al llarg del temps, aprendreu a realitzar pressupostos, així com mètodes per avaluar els resultats obtinguts.

Per treballar aquest mòdul és necessari llegir el contingut escrit, i fer els exercicis i activitats d'autoavaluació que es proposen en cada apartat. A l'aula es proposaran casos pràctics per aplicar els coneixements teòrics adquirits amb la lectura del contingut. En la primera unitat haureu de realitzar una activitat pràctica en equip, que consistirà en el disseny i planificació d'un projecte.

Resultats d'aprenentatge

En finalitzar aquest mòdul l'alumne/a:

Disseny i planificació del projecte

1. Determina els objectius, l'estil gràfic i narratiu, les especificacions i els requisits del sistema per a un projecte interactiu multimèdia, elaborant la seva documentació.
2. Determina les arquitectures tecnològiques de producció o desenvolupament i de destinació o desplegament (usuari final) dels projectes audiovisuals multimèdia interactius, relacionant les especificacions tècniques amb els requisits d'operació i de seguretat.
3. Organitza com a mòduls d'informació les diferents fonts necessàries per a la realització dels projectes audiovisuals multimèdia interactius, analitzant les necessitats tècniques, narratives i estètiques.

Producció i avaluació del projecte

1. Planifica i realitza el seguiment de projectes audiovisuals multimèdia interactius, valorant procediments d'optimització de recursos, temps i pressupostos.
2. Defineix un sistema de qualitat i avaluació del projecte audiovisual multimèdia interactiu, elaborant la documentació necessària segons la normativa internacional.

Continguts

Disseny i planificació del projecte

Unitat 1

Disseny i planificació del projecte

1. Determinació d'objectius, estil gràfic i narratiu d'un projecte
2. Determinació de l'arquitectura tecnològica del projecte
3. Organització i catalogació de continguts, fonts i mòduls d'informació

Producció i avaluació del projecte

Unitat 2

Producció i avaluació del projecte

1. Planificació del seguiment i realització del projecte
2. Avaluació del projecte

Disseny i planificació del projecte

Sara Sunyé, Rodrigo Pizarro

Projectes de jocs i entorns interactius

Índex

Introducció

Actualment, estem contínuament interactuant amb aplicacions amb interactivitat: llocs web, videojocs, apps per dispositius mòbils, aplicacions de realitat virtual... La indústria de creació d'aplicacions interactives és gegant i implica molts tipus de tasques i perfils especialitzats. En la creació d'una aplicació interactiva, és decisiva la primera fase de treball que consisteix a la definició del projecte i la seva planificació.

En aquesta primera unitat, "Disseny i planificació de projecte", del mòdul *Projectes de jocs i entorns interactius*, aprendreu a determinar i establir els requisits i característiques d'un projecte interactiu i com planificar les tasques del projecte abans de desenvolupar-lo. També aprendreu a organitzar i catalogar els continguts i a realitzar el control de les diferents versions del projecte, per tal de fer un seguiment de la realització del projecte en totes les seves fases de producció.

En el primer apartat, **"Determinació d'objectius, estil gràfic i narratiu d'un projecte"**, veureu com preparar la preproducció d'un projecte interactiu multimèdia. Aquest procés comença amb una planificació estratègica conceptual i funcional on s'especifiquen les bases del projecte: objectius, audiència, pressupost...

També veurem diferents tipus de modelització de sistemes així com les tècniques i procediments que s'utilitzen. Seguidament aprendrem què és la narrativa interactiva i l'arquitectura de la informació i com s'executa mitjançant el disseny de la interacció i la navegació. També treballarem el concepte de "matriu heurística" i analitzarem els diagrames de seqüències. Seguidament veurem què és una interfície d'usuari i les seves tipologies, així com aspectes ergonòmics, d'usabilitat, accessibilitat i les seves normatives. Finalment, estudiarem què necessitem per fer un *Game Document Design*, un prototip i per acabar l'aplicació d'unitat d'estil estètica i narrativa.

En el segon apartat, **"Arquitectures de desenvolupament i de destinació en projectes AMI"**, veureu tot un seguit de tècniques per transformar una idea en un projecte que funcioni al mercat, tenint en compte la producció, el desenvolupament i la distribució del projecte.

En concret, veureu una introducció a una de les eines més utilitzades en l'àmbit del desenvolupament: el diagrama de classes. A més a més, us introduireu en el tractament de dades amb les bases de dades relacionals. Després repassareu un seguit de criteris que cal seguir a l'hora de decidir l'arquitectura d'un projecte AMI. També podreu observar un exemple de distribució d'aplicacions a través d'una de les botigues d'aplicacions més importants del món. Finalment, veureu recomanacions respecte a la seguretat tant de la producció com del manteniment del projecte AMI.

Finalment, en el tercer apartat, **"Organització i catalogació de continguts, fonts**

i mòduls d'informació", estudiareu tot el que està relacionat amb el manteniment correcte de la informació necessària per desenvolupar un projecte AMI.

Analitzareu com establir criteris comuns per poder treballar en equip correctament, sense generar un producte incoherent. També analitzareu els sistemes tècnics que hi ha per protegir les dades i recuperar-les en cas d'errades. Finalment, veureu en profunditat una de les eines més utilitzades respecte a la gestió de versions: Git. Aquest sistema de repositoris us permetrà treballar de manera eficient amb equips grans, i és un dels sistemes més utilitzats arreu del món.

Per assolir els objectius d'aquesta unitat, heu de llegir el contingut dels materials, fer els exercicis d'autoavaluació i les activitats proposades. En cas de dubte, podeu preguntar al fòrum de l'assignatura, ja que així us podran ajudar els vostres companys o el professor.

Resultats d'aprenentatge

En finalitzar aquesta unitat, l'alumne/a:

1. Determina els objectius, l'estil gràfic i narratiu, les especificacions i els requisits del sistema per a un projecte interactiu multimèdia, i n'elabora la documentació.

- Determina els objectius (comunicatius, funcionals i formals) i fa el procés de captura de requeriments (documents de visió i guia), del projecte que es desenvoluparà, valorant la necessitat d'un tractament lineal, interactiu o mixt.
- Defineix les seqüències dinàmiques d'acció i de relació (diagrames de seqüències i col·laboració o interacció) i el comportament dinàmic d'objectes o classes (diagrames d'estats) segons el tractament del producte que es desenvoluparà.
- Determina els requisits funcionals del sistema d'informació des de la perspectiva de l'usuari i les característiques del producte audiovisual multimèdia interactiu (catàleg de productes, ensenyament assistit per ordinador, videojocs, aplicacions per a dispositius mòbils i realitat virtual, entre d'altres).
- Defineix les fases, entrades i sortides del disseny i desenvolupament del projecte segons les característiques del producte audiovisual multimèdia interactiu.
- Determina els esdeveniments causals, desplegats en el temps, i les interaccions dels agents intencionals.
- Determina els requisits ergonòmics aplicables al disseny del diàleg, els procediments d'avaluació, les proves i el mesurament de la usabilitat i de l'accessibilitat, a partir del context d'ús, les recomanacions i les condicions de disseny per a tothom.
- Realitza el model d'informació relatiu a connexions, interfícies, descripcions, activitats i requisits a partir de l'anàlisi de les necessitats d'informació.

1. Determinació d'objectius, estil gràfic i narratiu d'un projecte

Tot projecte interactiu s'ha de portar a terme seguint unes premisses molt definides i acurades per aconseguir el resultat desitjat. Abans de començar a materialitzar-lo, s'ha de tenir molt clar què es vol fer, per què, i com es vol fer.

Si parlem d'una aplicació web o d'un videojoc, el primer pas a l'hora d'afrontar el projecte és establir totes les seves característiques, com són els objectius, públic al qual es dirigeix, aspecte visual, i estructura narrativa que tindrà. Cada tipus de projecte tindrà a priori unes característiques diferents. Per exemple, si estem creant un videojoc 3D, l'objectiu principal serà l'entreteniment, el públic estarà en una determinada franja d'edat (podrien ser joves entre 15 i 25 anys) i l'estètica es basarà en gràfics 3D amb un alt grau de realisme. Al contrari, si estem realitzant una aplicació per dispositius mòbils de venda de roba femenina en línia, el públic serà un altre, l'objectiu principal serà augmentar les vendes de l'empresa de roba, i l'estètica es basarà en un disseny gràfic pla molt acurat i elegant.

En aquesta unitat s'expliquen els aspectes que s'han de definir en la primera fase d'ideació del projecte, per tal d'elaborar un document amb la descripció i planificació: el tipus de producte, el públic objectiu, l'estructura narrativa i l'estructura de seqüències, el disseny de la interfície d'usuari i l'estil visual.

1.1 Productes, estratègies i mercat de productes multimèdia audiovisuals interactius

Els productes multimèdia audiovisuals interactius (PMAI) es basen en el diàleg entre usuaris i contingut, on el disseny i la realització són factors principals per captar la seva atenció, ja que l'èxit depèn de si la persona, receptor o usuari aconsegueix relacionar-se totalment amb la presentació. Aquestes particularitats han fet sorgir **noves àrees de negociació**, amb noves empreses, productes i serveis.

Primerament, podem distingir quatre usos principals d'aquest tipus de productes:

- **En la publicitat**: es tracta d'aprofitar l'eficàcia comercial de la interacció amb l'usuari per captar la seva atenció amb més facilitat i que posteriorment es pugui traduir en consum. Alguns exemples serien les xarxes socials, plataformes de màrqueting mòbil i web, *mockups*...
- **En l'art**: els videojocs actualment tenen una rendibilitat i un mercat superior al de la indústria del cine. En els últims anys han adquirit més rellevància els sistemes de realitat virtual (dels quals el més conegut és Oculus Rift i el més recent Vive, un projecte de Valve i HTC). També podem

trobar-nos amb simuladors a parcs d'atraccions o sales de jocs recreatius, televisió interactiva, museus, centres d'oci...

- **En educació**: permet organitzar de manera útil i flexible els continguts, proporcionant un clar avantatge sobre les formes d'aprenentatge tradicionals. També tenen un paper important en l'educació a distància, sobretot en algunes àrees geogràfiques. Com a exemple tenim tots aquells estudis que disposen de campus virtual, MOOCs i també sistemes basats en la gamificació.
- **Als mitjans d'informació i comunicació**: l'accés a informació esdevé més ràpida i senzilla gràcies a la interactivitat, que obre una nova guia a l'usuari a través de l'automatització de tasques i la fragmentació de continguts per tal d'obtenir així una estructura total i correcta, on es decideix a quina informació vol accedir l'usuari i per mitjà de quin itinerari, amb un exercici constant d'interacció. Alguns exemples podrien ser: llibres i revistes electròniques, enciclopèdies multimèdia, visites virtuals...

Pel que fa a **la indústria i les noves tendències**, l'aparició dels nous mitjans ha suposat l'obertura de vies potents perquè empreses i mitjans puguin arribar d'una manera més directa als consumidors potencials. Avui dia, l'àmbit de la producció multimèdia està ampliant exponencialment la seva oferta d'opcions, així com la seva complexitat tant tècnica com creativa, gràcies al desenvolupament de les aplicacions d'escriptori i a l'aparició dels telèfons intel·ligents, que usen la web i altres tecnologies per utilitzar fitxers de text, àudio, vídeo i imatge. Aquestes són algunes de les tendències més importants del sector:

- **El web 2.0**: els seus avantatges principals són la facilitat i la rapidesa, i per això les eines que generen més contingut web són les xarxes socials, els blogs, les wikis, les apps mòbils i els mitjans de comunicació de masses.
- **Els videojocs**: la imatge, el so i la interactivitat fan del videojoc un producte tan potent que gairebé engloba tots els avantatges de la producció multimèdia actual. A més a més, el sector ha incorporat com a tendència els jocs multijugador en temps real, i els videojocs a través de dispositiu mòbil, per la qual cosa molts usuaris que no tenien perfil de jugador de videojocs ara han començat a dedicar el seu temps a aquesta disciplina.
- **Projeccions interactives**: molt utilitzades en el sector cultural i educatiu per interactuar amb la imatge de manera fàcil i atractiva.
- **Ebooks i audiollibres**: ambdues eines fan molt més accessible la lectura a tothom.
- **Contingut en *streaming***: molt utilitzat per retransmetre esdeveniments on pugui interactuar l'usuari, plataformes d'emissió de contingut televisiu com per exemple sèries...
- **Realitat virtual i augmentada**: eines d'inserció i interacció de l'usuari en un món fictici.

1.1.1 Aspectes interactius com a valor afegit

Per un producte de comunicació, sigui nou o ja existent, els aspectes interactius sempre seran un valor afegit. Interactivitat i diversió estan estretament lligades. Allò interactiu produeix sensacions emocionants i sorprenents com a missatge bidireccional i que construeix en temps real.

En un projecte interactiu el component de navegació facilita els hipervincles que portaran als diferents continguts. Aquesta estructura de navegació ha de ser sempre original, usable i fàcil d'entendre, i és important no perdre aquests conceptes de vista perquè és aquesta estructura de navegació la que més interactivitat admet i no convé sobrepassar-se.

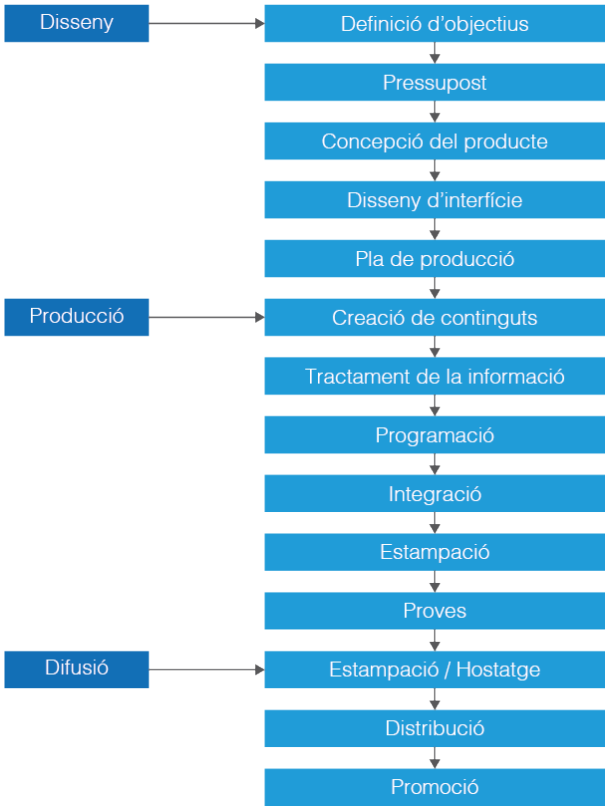
Algunes idees que ens poden ajudar a l'hora d'afegir interactivitat als projectes són:

- **Aleatorietat**: possibilitar a l'atzar intervenir en la comunicació amb l'usuari farà que aquest se senti més lliure.
- **Repetició**: ja sigui d'accions o d'imatges, donen lloc a una nova realitat.
- **Comunicació intuïtiva**: basant-nos en els principis fonamentals de la comunicació interactiva com pot ser "clicar un botó", "moure el ratolí"...
- **Exclusivitat**: qualsevol acte comunicatiu tindrà més èxit si és únic i exclusiu.
- **Canvi**: qualsevol canvi pot programar-se perquè es produeixi en una data i una hora, o sigui fruit de l'usuari o de l'atzar.
- **Imitar la realitat**: portar a la pantalla objectes reals tenint en compte que sempre ha de ser més efectiu que realista.

1.1.2 Planificació estratègica: definició d'objectius, necessitats i fases

En el procés de creació d'un producte multimèdia, es poden establir les grans etapes següents: disseny, producció i difusió. Cada una està estructurada en diferents fases, com es mostra en el gràfic adjunt (vegeu la figura 1.1)

Figura 1.1. Fases del procés de creació d'un producte multimèdia



Dins el procés de concepció de producte, s'emmarca la **creació del guió** i per tant les necessitats principals del projecte. D'aquesta manera podem saber l'equip humà necessari, el pla de treball i el material:

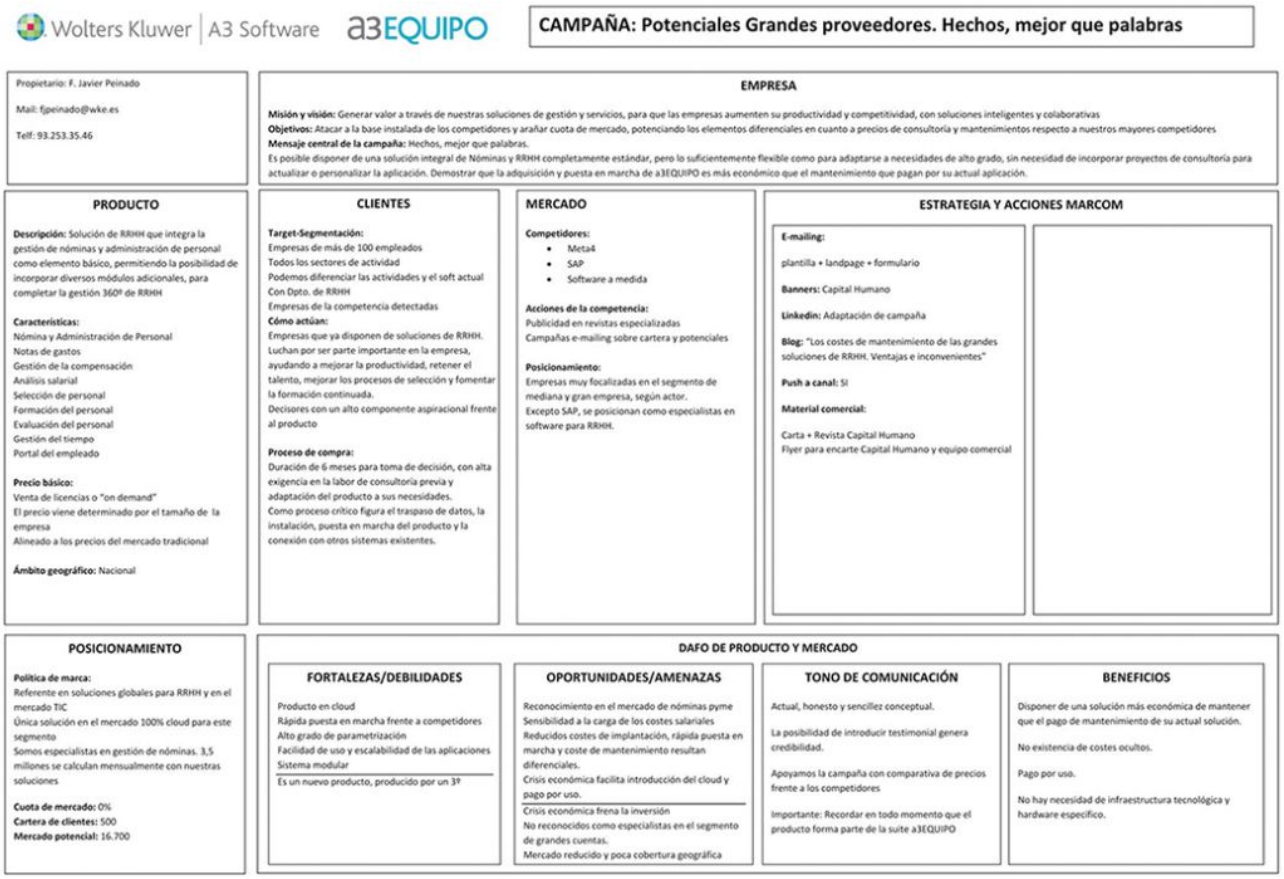
- **Equip humà**: s'ha de planificar l'equip humà que es necessitarà per a la creació del producte interactiu.
- **Pla de treball**: el conjunt del treball s'ha de dividir en mòduls o parts detallades; assenyalar el responsable i el termini de temps.
- **Equip material necessari**: ordinadors, escàner, càmera, programari... S'han de desenvolupar les aplicacions, passar a la fase de proves i tests i fer totes les iteracions que calguin per a obtenir un producte òptim. En la tercera etapa, la de difusió, s'estableix la distribució del producte i la seva promoció.

A l'**etapa de disseny** es defineixen els objectius i l'estratègia tenint en compte el pressupost per a fer l'aplicació. Una vegada definits els objectius i desenvolupada l'estratègia, ja està encaminada la tasca de generar l'aplicació multimèdia.

Per tal de **definir els objectius**, cal tenir en compte el brífing, un document habitualment creat pel client que entrega a l'empresa que desenvolupa el projecte per tal que es pugui guiar a l'hora de fer el pressupost. El brífing és un document que defineix els objectius i continguts del producte que volem desenvolupar.

Dins el bríefing trobarem les següents especificacions (vegeu figura 1.2):

FIGURA 1.2. Exemple de bríefing real



Font: Adnstudio

- **Objectius i missatges**: quin és l'objectiu del producte multimèdia que s'ha de desenvolupar, per a què li servirà a l'usuari. Format/especificacions tècniques: quin tipus de producte és (aplicació, pàgina web, videojoc...).
- **Continguts i estructura**: quins continguts inclourà i com s'estructuraran.
- ***Look&Feel***: quin disseny haurà de seguir.
- **Estudi de mercat/casos** o exemples que s'han de seguir: el client podrà suggerir altres productes multimèdia com a exemples a seguir, inspiradors per a l'empresa desenvolupadora del producte.
- **Condicions**: l'empresa marcarà condicions, com per exemple la data d'entrega del projecte.

Seguint l'esquema de la figura 1.1, la taula 1.1 mostra, més en detall, quines tasques específiques s'emmarquen dins de cada fase. Heu de tenir en compte que les etapes són **consecutives**, per tant s'han de finalitzar les tasques de la primera etapa per començar la segona etapa.

TAULA 1.1. Etapes i fases d'un procés de creació d'un producte multimèdia i tasques corresponents

Etapes del procés	Primera fase	Segona fase	Tercera fase
1a) Disseny	**Desenvolupament**: a la primera fase bàsicament definim què volem, quins recursos humans i econòmics necessitarem i com els aconseguirem. Les tasques associades són: definició d'objectius, elaboració del guió, configuració de l'equip de treball, pressupost de producció, recerca de finançament.	**Preparació**: en aquesta fase acabem de definir el guió i com treballem tant a nivell d'eines com d'organització. Les tasques associades són: concepció del producte, revisió del guió, selecció de *hardware* i *software*, configuració de l'equip de treball, establiment de processos i procediments de producció.	**Preproducció**: en aquesta fase ja treballem en el projecte sobre paper. Les tasques associades són: *storyboarding*, disseny de la interfície (models, fons, animació...) i pla de producció.
2a) Producció: en aquesta fase preparem els continguts i programem:	**Desenvolupament**, on les tasques associades són: preparació i creació dels continguts i la programació (2D, 3D, *stop-motion*, modelatge...).	**Postproducció**: en aquesta fase s'afegeixen els elements que falten i es comencen les proves. Les tasques associades són: edició de vídeos i efectes especials, música i doblatge, integració, estampació i proves.	- - -
3a) Difusió	**Entrega**: en aquesta fase es fan els últims retocs i es comença a treballar l'estratègia de difusió. Les tasques associades són: adaptació tècnica del projecte, estampació i hostatge, desenvolupament del material promocional, elaboració de la guia d'estil per a l'explotació.	**Distribució**: s'executa l'estratègia de difusió i es distribueix el producte. Les tasques associades són: tancament d'acords de distribució, assistència a mercats nacionals i internacionals, implantació del pla de màrqueting, distribució de contingut (TV, agregadors de contingut, sales de cinema, *retail*...).	- - -

1.2 Narrativa i comunicació interactiva

En les últimes dècades s'ha produït una revolució en l'àmbit de la comunicació que sorgeix de la transformació del model de desenvolupament de la societat, que ha passat de ser industrial a configurar-se com a *societat de la informació*. D'alguna manera, podem dir que hi ha hagut un canvi **de la comunicació tradicional a la comunicació interactiva**.

La revolució tecnològica ha contribuït poderosament a l'eclosió d'un nou model de comunicació, desenvolupat bàsicament a Internet, amb unes característiques diferents a les dels mitjans de comunicació de masses tradicionals. Una d'aquestes característiques és l'important desenvolupament del grau d'interacció entre emissor i receptor. La migració digital suposa un desplegament accelerat de les tecnologies del coneixement, entre les quals destaquen les tecnologies de la imatge, essencials per a la informació de la percepció i la comprensió de la realitat.

La narrativa i comunicació interactiva és aquella en què el receptor té la capacitat per a **prendre decisions** i crear el seu propi camí per arribar a l'objectiu. La capacitat del receptor per a prendre decisions dependrà en gran mesura de l'estructuració de la informació proporcionada pel projecte interactiu. Precisament, la forma d'estructurar la informació en un projecte interactiu és mitjançant l'**arquitectura de la informació**. Altres aspectes que cal tenir en compte són la diferència entre la narrativa lineal i interactiva, l'estètica informacional de l'espai, l'anàlisi de situacions, així com les matrius heurístiques i alguns aspectes comunicatius.

1.2.1 Arquitectura de la informació, disseny de la interacció i la navegació

L'arquitectura de la informació (AI) se centra a organitzar, estructurar i etiquetar continguts de manera efectiva i sostenible. L'arquitecte de la informació s'encarrega d'**organitzar els continguts** perquè siguin fàcilment accessibles i llegibles. La complexitat del treball d'un arquitecte de la informació està relacionada directament amb el volum de continguts que s'han d'incloure en una aplicació. Així, l'arquitectura de la informació es pot definir segons aquests quatre aspectes:

- El disseny estructural dels entorns d'informació compartida.
- La combinació dels sistemes d'organització, etiquetatge, cerca i navegació dels llocs web i les intranets.
- L'art i la ciència de donar forma a productes i experiències relacionats amb la informació a fi de fomentar la facilitat d'ús i de localització d'elements (*findability*).
- Una disciplina emergent i la seva corresponent comunitat de pràctica, l'objectiu de la qual és aplicar al món digital els principis del disseny i l'arquitectura.

El seu principal objectiu és facilitar al màxim els processos de comprensió i assimilació de la informació, així com les tasques que executen els usuaris en un espai d'informació definit.

> "S'utilitza el terme *informació* per distingir l'arquitectura de la informació de la gestió de les dades i del coneixement. Les dades són fets i xifres. Les bases de dades relacionals són sistemes molt estructurats que donen respostes concretes a preguntes concretes. El coneixement és el que les persones tenim en la ment. Els gestors del coneixement desenvolupen eines, processos i incentius per animar la gent a compartir aquest material. La informació es troba en un punt mitjà confús. En els sistemes d'informació, sovint no hi ha una única resposta «correcta» per a una determinada pregunta. Hem de manejar informació de tot tipus: llocs web, documents, programes informàtics, imatges... A això cal afegir-hi les metadades, termes que s'usen per a descriure i representar objectes que formen part del contingut, com documents, persones, processos i organitzacions."
>
> P. Morville i L. Rosenfeld (2006). *Arquitectura de la informació per al World Wide Web*.

Pel que fa al **disseny de la interacció**, cal tenir clar que la interactivitat és un recurs propi de les màquines informàtiques. Podem entendre el concepte d'interactivitat com la relació entre l'usuari (persona) i la màquina, de manera que cada un respon als estímuls de l'altre. Per tant, seria com la capacitat del receptor per a controlar un missatge no lineal fins al grau establert per l'emissor, dins dels límits del mitjà de comunicació asincrònic. Una interfície intuïtiva entre l'usuari i la màquina és aquella que requereix poc entrenament per part de l'usuari i ofereix un entorn amigable.

Hi ha tres **graus d'interactivitat**:

1. El primer grau és el més simple: premem un botó i passa alguna cosa. Tots els programes interactius inclouen aquest grau d'interactivitat.

2. El segon grau d'interactivitat es refereix al fet que els efectes de les accions de l'usuari tenen conseqüències en el temps. Aquest segon grau està lligat als conceptes d'història i d'intel·ligència.

3. El tercer grau és el grau més sofisticat d'interactivitat. El trobem quan les accions de l'usuari transcendeixen l'ús del producte. Així passa quan l'aplicació obliga l'usuari a fer accions que van més enllà de les comunes en un producte multimèdia: agafar un atles per a resoldre un enigma, connectar-se a Internet per participar en un fòrum o per recollir una informació...

Pel que fa al **disseny de la navegació**, cal tenir en compte que la navegabilitat és la facilitat amb què un usuari pot desplaçar-se per totes les pàgines que formen un lloc web. Per arribar a aquest objectiu, un lloc web ha de proporcionar un conjunt de recursos i estratègies de navegació dissenyats per aconseguir un resultat òptim en la localització de la informació i en l'orientació per a l'usuari.

La **interactivitat** és el diàleg que s'estableix entre una persona i una màquina en què cadascú responen als estímuls de l'altre, mentre que el **disseny de la navegació** s'ocupa de pensar i dissenyar com es mourà l'usuari per l'aplicació, com accedirà a la informació.

Segons la Guia d'usabilitat del Departament de Salut i Serveis Humans dels Estats Units (2006), és important tenir en compte les **pautes d'usabilitat** següents aplicades al disseny de la navegació:

1. **Ubicació de les opcions**: s'ha comprovat que la navegació s'agilita quan el menú principal s'ubica a la zona esquerra de la interfície. També és important agrupar o col·locar junts els menús secundaris i terciaris. Es poden col·locar sols (per exemple, com a menú horitzontal si el menú principal és vertical) o ben agrupats amb el menú principal. Això no implica que no es puguin col·locar opcions a la zona dreta de la pantalla, però en el cas del menú principal no és la ubicació preferida pels usuaris.

2. **Menús desplegables**: els menús desplegables permeten accedir a les subopcions d'un menú principal sense consumir espai de pantalla. No obstant això, presenten l'inconvenient que les subopcions apareixen inicialment ocultes, de manera que és possible que l'usuari hagi de desplegar tots els menús per trobar l'opció que està buscant. Per tant, és molt important organitzar adequadament la distribució de les subopcions i comprovar que tant l'ordre en el qual s'han distribuït com les seves etiquetes són entesos adequadament per part dels usuaris. D'altra banda, els menús desplegables en cascada presenten l'inconvenient que poden resultar difícils d'utilitzar, ja que el moviment del ratolí requereix molta precisió. S'haurien d'evitar sempre que sigui possible, especialment en el cas d'usuaris amb discapacitats motrius.

3. **Menús estàtics**: l'ordre de les opcions d'un menú no s'ha de modificar mai, ja que pot desorientar greument l'usuari. Fins i tot en el cas dels menús adaptables, en els quals es col·loquen en primer lloc les opcions més utilitzades pels usuaris (per exemple, Microsoft Office), s'ha observat que els usuaris prefereixen menús estàtics.

4. **Utilització dels mapes**: en el cas de les aplicacions o llocs web amb un volum de continguts important, és recomanable incloure un mapa del lloc (que mostra l'estructura jeràrquica dels continguts) o un índex (que presenta la llista de continguts). Aquest mapa serà més útil com més s'aproximi a l'estructura mental de l'usuari.

5. **Indicació de la situació de l'usuari**: l'usuari ha de poder conèixer sempre tres factors –per a la qual cosa es poden utilitzar diversos recursos com rutes de navegació, la diferència visual de l'opció activa del menú o els títols de les pàgines–:

    - En quin punt de l'aplicació es troba.
    - D'on procedeix.
    - Cap a on es pot dirigir.

6. **Rutes de navegació**: les rutes de navegació són un element que permet a l'usuari orientar-se dins del lloc web i se solen col·locar a la zona superior de l'àrea de continguts de la pàgina. N'hi ha de diferents tipus:

    - Localitzador: informa del lloc en el qual està situada la pàgina respecte a l'estructura de continguts de l'aplicació; és independent de la navegació efectuada prèviament per l'usuari.
    - Dinàmica: informa del camí seguit per l'usuari fins a arribar al punt en el qual es troba.
    - Descriptiva: mostra a l'usuari en quina categoria se situa el contingut que es mostra. Permet que l'usuari pugui accedir a altres continguts similars o relacionats. Encara que la ruta de navegació és un element gaire útil per a situar l'usuari i s'està estenent cada vegada més, cal tenir en compte que pot resultar poc comprensible per a usuaris amb poca experiència. Per tant, no hauria de substituir mai les opcions de navegació habituals.

7. **Diferència visual** de l'opció activa del menú: quan l'usuari accedeix a un apartat de l'aplicació, cal indicar clarament quina és l'opció del menú corresponent a l'apartat en què es troba. Per fer-ho es poden utilitzar recursos gràfics com el canvi de color o un element que destaqui visualment l'opció activa. És important que el recurs gràfic triat s'apliqui de manera consistent al llarg de tota l'aplicació.

8. **Títol de pàgina**, clar i visible: l'usuari ha de poder distingir clarament quin és el títol de la pàgina, que ha de ser coherent amb l'opció que es mostra activa en el menú i no s'ha de confondre visualment amb el text de contingut.

9. Proporcionar **opcions de navegació**: una opció de menú o un enllaç no s'ha de dirigir mai a una pàgina que no contingui opcions de navegació, ja que

l'usuari ha de poder navegar lliurement per totes les pàgines o pantalles de l'aplicació. En cas que un enllaç obri una nova finestra, aquesta ha de contenir un botó per a tancar la finestra en actiu i tornar a la finestra original.

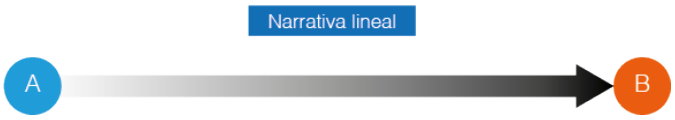
10. **Diferenciar i agrupar** elements de navegació: les opcions i enllaços s'han de diferenciar clarament de la resta d'elements no actius d'una interfície. A més, les opcions relacionades s'han d'agrupar visualment, de manera que les que tinguin una relació més estreta siguin les que estiguin més properes entre elles. D'altra banda, els menús verticals es llegeixen més àgilment que els horitzontals.

11. En **pàgines extenses**, incloure una llista de continguts seleccionable: quan el contingut d'una pàgina és molt extens, pot resultar difícil per a l'usuari localitzar el contingut que està buscant. Per a facilitar aquesta tasca, es recomana col·locar al principi de la pàgina una llista o índex clicable de les seccions o apartats de la pàgina, de manera que l'usuari pugui veure si conté el que està buscant i accedir-hi directament.

1.2.2 Narrativa lineal i interactiva: estructura seqüencial determinada i modular

Pel que fa a la seqüencialitat, podem referir-nos a aquelles narracions en les quals els esdeveniments passen consecutivament en un ordre d'un rere l'altre. Com en una novel·la o en una pel·lícula, els esdeveniments d'una història se succeeixen en un ordre inalterable segons l'ha disposat el seu autor. És el que es coneix com a **narrativa lineal** (vegeu la figura 1.3)

Figura 1.3. En una narrativa lineal el camí a recórrer és únic



Per la pròpia essència de la narració, una història sempre és un encadenament d'esdeveniments que succeeixen un darrere d'un altre. En una narració seqüencial la història és lineal, i així és percebuda per l'espectador o lector, per molts salts temporals que presenti l'argument.

> Quan és l'usuari qui decideix en quin ordre accedirà a la informació i aparentment no hi ha un ordre establert per l'autor, parlem de **narració no lineal**.

Quan creem un producte multimèdia, una de les primeres decisions que hem de prendre és com presentarem la informació, de manera lineal o no lineal, i quines són les implicacions per a l'usuari d'un tipus de presentació o una altra. En un producte multimèdia tenim totes dues opcions i, fins i tot, opcions mixtes:

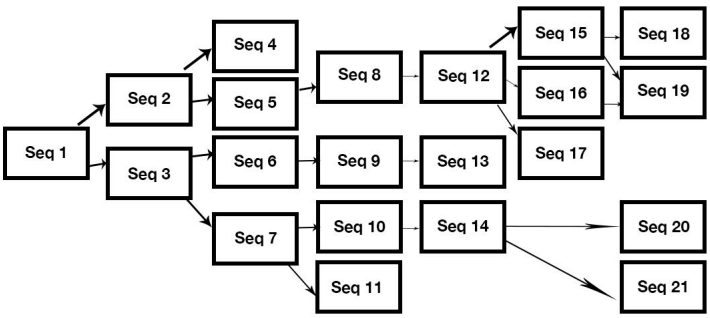
- **Presentar els continguts de manera lineal**: podem presentar de manera ordenada els continguts. L'estructura seqüencial de la informació s'usa quan l'assimilació d'una unitat de contingut és un requisit indispensable per a la presentació de la següent. En un llibre hem de llegir tots els capítols tal com els ha establert l'autor per a tenir tota la informació que necessitem per a entendre la història.
- **Presentar els continguts de manera no lineal**: l'usuari té l'opció d'accedir on vulgui; pot accedir des de qualsevol part del contingut a qualsevol altra quan decideixi. L'exemple més característic d'aquest tipus de presentació és la navegació per internet, en què circulem mitjançant enllaços entre els diferents nodes de contingut. A internet l'usuari ha de participar activament i es converteix en lector-autor. Ell és qui decideix crear la seqüència narrativa a la seva manera. Per tant, l'avantatge que obté és la llibertat d'acció.
- **Presentar els continguts de manera mixta**: lineal i no lineal. Fins i tot en els projectes aparentment aleatoris, és necessari que l'autor pensi el recorregut necessari que ha de fer l'usuari per a l'assimilació correcta de la informació. S'aconsella estudiar la combinació de l'aleatorietat i la seqüencialitat per a poder aconseguir els resultats de transmissió de continguts. Una manera de presentació o una altra té uns resultats diferents segons el gènere del producte.

D'altra banda, tenim la narrativa interactiva. Si narrar és explicar esdeveniments units per una cadena de causa-efecte a través del temps i l'espai, en la **narració aleatòria** aquests esdeveniments es multipliquen, donat que els camins narratius es ramifiquen constantment. La diversitat de maneres de multiplicar i diversificar la història està directament relacionada amb les possibilitats d'intervenció que l'autor hagi donat a l'usuari sobre els diferents elements que constitueixen la narració.

A la **narrativa interactiva** l'autor és qui es planteja quines són les opcions que transmet a l'usuari, i té un gran ventall de possibilitats al seu abast.

La interactivitat produeix tota una sèrie de possibilitats, ja que els elements de la narrativa lineal persisteixen en la narració multilineal, però ara l'usuari assumeix alguns dels papers que abans estaven exclusivament a mans de l'autor (vegeu la figura 1.4).

FIGURA 1.4. Exemple de seqüència en una narració interactiva



Hi ha diferents **maneres de crear ficcions interactives**, com per exemple:

- Donar la possibilitat que sigui l'usuari mateix qui vagi descobrint la trama. L'usuari es mou lliurement per un món virtual i va recollint dades per a completar així la trama prèviament pensada per l'autor.
- Donar la possibilitat a l'usuari d'intervenir sobre un o diversos personatges. Per exemple, els personatges porten elements o objectes que l'usuari ha d'escollir per a enfrontar-se a certes parts del joc.
- Donar a l'usuari la possibilitat de veure una història des dels diferents punts de vista, amb tot el que això implica. A vegades simplement pot ser mostrar la mateixa història des de diferents plans narratius.
- Donar la possibilitat que sigui l'usuari mateix el copartícip en la creació de la trama. Mitjançant les diferents accions de l'usuari sobre la història, els personatges, l'espai i el temps, es van creant diferents trames.
- Donar la possibilitat que l'usuari s'impliqui aportant la seva experiència pròpia i exposant-la al costat de l'experiència d'altres usuaris. Un exemple podria ser el fòrum de discussió. Alguns d'aquests fòrums són verdaders generadors de contingut.

1.2.3 Estètica informacional de l'espai i accions

L'**espai** és una de les dimensions que es tenen en compte en la narració, igual que el concepte de **temps**. En els productes multimèdia, aquest concepte està molt lligat a com es mostra, és a dir, a qüestions de llenguatge audiovisual i tècniques (tipus de pla, moviments de càmera...). No es pot elaborar un guió sense descriure on es desenvolupa l'acció i quina relació té aquest "espai" amb els personatges. El **guionista**, al principi de cada escena i seqüència, ha de descriure el lloc on es desenvolupa l'acció, si és interior o exterior, així com també descriure els objectes que formaran l'ambient per a contextualitzar l'escena.

En el guió hem de definir les **característiques de l'espai**: què és, com és, on és, per a què serveix l'espai i quina relació té amb els personatges. Segons Eugene Vale, els llocs en els quals transcorre l'acció es poden caracteritzar segons el següent:

- La seva naturalesa: despatx, hospital, fàbrica...
- La seva funció: poblat, desert, misteri...
- El seu propòsit: fabricar objectes si és una fàbrica, curar malalts si és un hospital.
- La seva situació: en una ciutat, en un país, en un desert.
- La seva relació amb un o diversos personatges: el seu amagatall, un símbol de poder. A aquestes funcions, Chion n'hi afegeix una altra: l'ambient que es desprèn, que pot ser anguniós, màgic o opressor, entre d'altres.

E. Vale (1993). *Técnicas del guión para cine y televisión*.

El cinema té moltes maneres de **mostrar els espais i ambients** en una pel·lícula, i de la mateixa manera es mostra a qualsevol producte multimèdia interactiu; per exemple:

- Ensenyant el decorat.
- Dient-ho al diàleg o en la seqüència anterior utilitzant el mecanisme de l'anticipació.
- Donant-ho a conèixer mitjançant objectes determinats o l'attrezzo (decorat).
- Per les accions que faci el personatge.
- Mostrant-ho des del principi o ensenyant-ho progressivament per ocultar alguna part a l'espectador.

Així, **com pensa, actua i sent el personatge?** A les novel·les és fàcil expressar aquest aspecte del personatge, però en el cinema, exceptuant algunes obres amb narrador, la imatge ens obliga a expressar el pensament del personatge amb les característiques pròpies d'aquesta eina.

En el cinema, hi ha tres maneres d'expressar el pensament: la **veu interior** (veu en off), el **diàleg** i l'**acció** (actituds, expressió corporal i comunicació no verbal). Molts guionistes professionals diuen que un guió ha d'estar escrit sempre en clau d'acció. La imatge audiovisual és, sobretot, acció. El diàleg és un element secundari. Aquesta característica és el que més separa el cinema de la literatura i moltes vegades també del teatre. Per tant, per saber com **exposar l'acció**, ens fixarem en com ho fa el cinema, ja que és la millor manera d'entendre com poden succeir els esdeveniments en l'espai-temps.

Un dels aspectes més importants amb relació al temps és l'**ordre com es presenten els esdeveniments** en el cinema, ja que poden passar simultàniament o successivament. En canvi, en l'argument, l'ordre dels fets pot passar de les dues maneres possibles. Aquesta característica produeix quatre combinacions possibles i cada una es pot exemplificar en pantalla mitjançant diverses opcions de caràcter estètic; aquests en serien uns exemples:

1. Esdeveniments simultanis en presentació simultània:
    - Composició en profunditat.

- Divisió de pantalla.
- So o veu en off.

2. Esdeveniments successius en presentació simultània:
    - Personatges que veuen una pel·lícula o un programa que descriu esdeveniments previs a la història (l'acte de veure i els esdeveniments passats es representen simultàniament en l'argument).
    - Divisió de pantalla.
    - So encavalcat: el so de l'escena següent comença sobre les últimes imatges de la seqüència que s'està desenvolupant.
3. Esdeveniments simultanis en presentació successiva:
    - Muntatge en paral·lel.
4. Esdeveniments successius en presentació successiva:
    - Successió en la història; successió en l'argument.

La **durada** i l'**ordre** en el qual es presenten els esdeveniments influeixen en l'espectador o, en el cas d'un projecte interactiu, en l'experiència de l'usuari.

1.2.4 L'avaluació heurística

L'**avaluació heurística** consisteix en verificar la qualitat d'una sèrie de paràmetres establerts prèviament. La porten a terme avaluadors experts en usabilitat que actuen imitant les reaccions que tindria un usuari mitjà a l'hora d'interactuar amb el sistema que s'està avaluant.

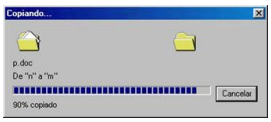
Dit d'una altra manera, consisteix en aplicar normes conversacionals en la interacció entre una persona i un sistema: l'objectiu és intentar crear un "pont comunicatiu" en el qual tant la persona com el sistema s'entenguin i treballin junts per tal d'arribar a un objectiu. Aquestes **regles empíriques generals** són utilitzades dins d'una planificació com a punt de partida per a la creació d'una llista de comprovació d'ítems que posteriorment utilitzarà l'avaluador expert dins la posada en marxa de l'avaluació.

D'aquesta manera, aquestes regles generals són adequades a cada cas concret d'avaluació per reflectir en els ítems que s'han d'avaluar la naturalesa i el tipus d'interfície per avaluar i el context d'ús. Aquesta avaluació pot servir per a diferents **propòsits**:

- Guiar els dissenyadors durant el procés de disseny.
- Ajudar els avaluadors a identificar problemes en les interfícies d'usuari, comprovant que es respectin les regles d'usabilitat.

- Explicar problemes d'usabilitat observats.
- Donar pautes sobre per què els usuaris fan determinades errades.

L'any 1986, Ben Schneiderman presenta les seves *Vuit regles d'or per al disseny d'interfícies*. En l'actualitat, aquestes vuit regles d'or són enteses com a principis heurístics generals que poden guiar tant el disseny com l'avaluació de la usabilitat d'un sistema interactiu:



Una cosa tan senzilla com indicar el temps d'espera suposa un "feedback" i una cortesia molt agraïda per part de l'usuari.

1. **Esforçar-se per la consistència**: les seqüències constants d'accions s'ha de repetir en situacions similars; la mateixa terminologia s'ha d'utilitzar en avisos, menús i pantalles de l'ajuda; i els comandaments constants s'han d'utilitzar en totes les parts de la mateixa manera.
2. **Crear dreceres per als usuaris freqüents**: quan la freqüència d'ús augmenta, els usuaris agraeixen reduir el nombre d'interaccions. Les abreviatures, els comandaments ocults i les macros són molt útils per a un usuari expert.
3. **Oferir *feedback***: per cada acció de l'usuari hi ha d'haver una certa regeneració del sistema. Per a les accions freqüents i de menor importància, la resposta pot ser modesta, mentre que per a les accions no freqüents i importants, la resposta ha de ser més substancial.
4. **Dissenyar el diàleg per mostrar feina pendent**: les seqüències d'accions s'han d'organitzar en grups amb un inici, un punt mig i un final. La regeneració informativa en la terminació d'un grup d'accions dona als usuaris la satisfacció de la realització, i una indicació clara per a preparar-se per al següent grup d'accions.
5. **Oferir una gestió senzilla de les errades**: tant com sigui possible, el sistema ha d'estar dissenyat perquè l'usuari no pugui causar una errada greu. En el cas que hi hagi una errada, el sistema ha de poder detectar-la i oferir mecanismes per a poder recuperar-se i solucionar l'errada.
6. **Permetre una fàcil recuperació d'accions**: aquesta característica ofereix tranquil·litat a l'usuari, perquè sap que les errades poden ser desfetes; anima així a l'exploració d'opcions desconegudes. La recuperació pot ser una sola acció, una entrada de dades o un grup complet d'accions.
7. **Suportar el control per part de l'usuari**: els usuaris experimentats desitgen tenir el control total del sistema i que el sistema respongui a les seves accions. El disseny del sistema ha de respondre a les accions de l'usuari, i que l'usuari sigui l'indicador de les accions, no només qui respongui a les accions del sistema.
8. **Reduir la càrrega de memòria recent de l'usuari**: la limitació humana del tractament de la informació en memòria a curt termini requereix que el que es mostri en pantalla sigui simple.

Simplificar la interfície

Reduir la càrrega de memòries és especialment important si els usuaris focals són persones grans, ja que generalment la capacitat per a memoritzar elements durant un termini relativament llarg de temps disminueix amb l'edat.

Pel que fa al seu **desenvolupament**, l'avaluació heurística és un **mètode d'inspecció** i la realitza un grup d'experts en usabilitat que examina la interfície i determina el grau de compliment dels principis d'usabilitat (o heurístics).

Durant la sessió, l'avaluador recorre la interfície diverses vegades, examina els elements d'interacció i els compara amb els principis d'usabilitat (heurístiques). Si l'avaluador considera que hi falta alguna cosa pot afegir principis d'usabilitat addicionals o altres qüestions que siguin rellevants per a la interfície que s'està avaluant.

Cada avaluador decideix per si mateix com vol avaluar la interfície, tot i que es recomana que examini tota l'aplicació almenys dues vegades (la primera per a obtenir-ne una visió de conjunt i la segona per a entrar-hi detalladament).

Com que els avaluadors no estan utilitzant la interfície per obtenir un resultat real, l'avaluació heurística es pot dur a terme en les primeres fases de desenvolupament, fins i tot en el disseny sobre paper.

El **resultat de l'avaluació heurística ha de ser una llista detallada de cadascun dels problemes d'usabilitat** que ha trobat en l'aplicació, amb referències als aspectes que no s'han respectat en cada cas i fins i tot pot incloure les opinions de l'avaluador.

Després de l'avaluació, es pot fer una posada en comú dels resultats, on participin tots els avaluadors i els membres de l'equip de disseny. La reunió resulta molt útil per a proposar solucions als problemes, i també per a comentar els aspectes positius del disseny i possibles millores.

Finalment, volem destacar les següents **recomanacions**:

- L'avaluació heurística l'hauria de fer **més d'un expert**, entre tres i cinc es considera la millor opció, ja que només un o massa dificultaria la trobada de tots els problemes d'usabilitat existents en la interfície.
- Cada avaluador examina la interfície **individualment** i no es poden comunicar entre si fins que no hagin exposat la seva inspecció. és important mantenir aquesta distància inicial perquè els resultats de cada avaluador siguin independents i no estiguin influïts pels altres.
- Cada avaluador pot escriure **un informe o bé es poden gravar** els seus comentaris per tal d'obtenir els resultat de l'avaluació de la interfície.
- Els observadors, sense donar opinions personals, poden ajudar els inspectors a utilitzar la interfície en cas que hi hagi problemes, i encarregar-se d'anotar els comentaris dels avaluadors de manera que aquests no hagin de redactar informes.

1.2.5 Interactivitat funcional i intencional

La interactivitat pot ser funcional (*aprenent-màquina*) i intencional (*aprenent-autor*). Una màquina pot tenir un grau feble d'interactivitat funcional i, al mateix temps, permetre una bona interacció cognitiva. Una màquina amb alt grau d'interactivitat funcional com el computador pot no afavorir la interacció cognitiva, si el programa no ha estat dissenyat amb una interactivitat intencional real. Aquest és el cas d'alguns jocs d'ordinador. En aquest tema encara hi ha molt per fer, necessitem tenir models d'interacció tan rics com les imatges de què disposem.

La interactivitat treu profit de la intel·ligència del receptor invocant una decisió contínua. En mitjans menys interactius, com el televisor actual, la intel·ligència es troba en l'emissor; el receptor només pren el que rep. Així, quan un aprenent mira el televisor la seva interactivitat té un nivell molt elemental: encendre'l, canviar de canal, apagar-lo. Quan un aprenent interactua amb un ordinador, la seva interactivitat pot arribar a establir un diàleg, un alt grau d'interactivitat, una conversa, és a dir, la resposta per mitjà del llenguatge al llenguatge.

> En la interactivitat hi ha la possibilitat de canviar la **distribució de les intel·ligències**, movent-les del transmissor al receptor.

Un agent és un sistema informàtic, situat en algun entorn, que **percep l'entorn** (entrades sensibles del seu entorn) i a partir d'aquestes percepcions determina (mitjançant tècniques de resolució de problemes) i **executa accions** (de manera autònoma i flexible) que li permeten aconseguir els seus objectius i que poden canviar l'entorn. Un agent intencional és un procés de llarga vida (permanent), amb independència i autonomia, i intel·ligència.

1.3 La interfície d'usuari (UI)

La interfície d'usuari o UI (*user interface*) és el medi amb què l'usuari pot comunicar-se amb una màquina, un equip o una computadora, i comprèn tots els punts de contacte entre l'usuari i l'equip; normalment solen ser fàcils d'entendre i fàcils d'accionar. Altres aspectes que cal tenir en compte són:

El **sistema operatiu** (SO) és el conjunt dels diferents programes que controlen el funcionament d'un ordinador. Les seves funcions, entre d'altres, consisteixen a gestionar les transferències d'informació internes, procurar la comunicació de l'ordinador amb els operadors, controlar l'execució dels programes amb la detecció dels errors, encadenar automàticament les feines, optimitzar els recursos (memòria, unitat aritmètica...), carregar i descarregar automàticament els programes en funció de l'espai de memòria i dels diferents perifèrics (vegeu la figura 1.5).

FIGURA 1.5. El SO fa d'enllaç entre les aplicacions i el maquinari

Per treballar amb un sistema, els usuaris han de ser capaços de controlar-lo i avaluar-ne l'estat. Per exemple, en conduir un automòbil, el conductor fa servir el volant per controlar la direcció del vehicle, i els pedals de l'accelerador, del fre i del canvi de marxes per controlar la velocitat del vehicle. És per a aquesta finalitat que existeixen aquesta mena de dispositius.

Es poden definir tres **tipus d'interfície d'usuari**:

- Una interfície de *hardware*, al nivell dels dispositius utilitzats per ingressar, processar i entregar les dades: teclat, ratolí i pantalla visualitzadora.
- Una interfície de *software*, destinada a entregar informació sobre els processos i eines de control, mitjançant el que l'usuari observa habitualment a la pantalla.
- Una interfície de *software-hardware*, que estableix un pont entre la màquina i les persones, permet a la màquina entendre la instrucció i a l'home entendre el codi binari traduït a informació llegible.

I les seves **funcions principals** són:

- Manipulació de fitxers i directoris
- Eines de desenvolupament d'aplicacions
- Comunicació amb altres sistemes
- Informació d'estat
- Configuració de la mateixa interfície i entorn
- Intercanvi de dades entre aplicacions
- Control d'accés
- Sistema d'ajuda interactiu

Alhora, hi ha diverses **tipologies d'interfícies d'usuari**; per tant, també les podem classificar segons la forma d'interactuar de l'usuari. Així, atenent a com l'usuari pot interactuar amb una interfície, les interfícies poden ser:

- Interfície de línia de comandaments (Command-Line Interface, CLI): interfícies alfanumèriques (intèrprets de comandaments) que només presenten text.

- Interfícies gràfiques d'usuari (Graphic User Interface, GUI): permeten comunicar-se amb la computadora de manera ràpida i intuïtiva representant gràficament els elements de control i mesura.
- Interfície natural d'usuari o NUI (*Natural User Interface*): poden ser tàctils, representant gràficament un "panell de control" en una pantalla sensible al tacte que permet interactuar amb el dit de forma similar a quan s'acciona un control físic; poden funcionar mitjançant el reconeixement de la parla, com per exemple Siri; o mitjançant moviments corporals, com és el cas de Kinect.

D'altra banda, les interfícies d'usuari han passat per **diverses generacions**. En els orígens de la tecnologia informàtica, els ordinadors eren grans màquines que processaven comandaments organitzats en grups o lots de treball (*batch*). Els usuaris són bàsicament els mateixos programadors.

La dècada dels seixanta va ser crucial en el desenvolupament de la IPO (*input-process-output* o 'interacció persona-ordinador') per l'augment del nombre de persones que comencen a tenir accés a ordinadors. Entre les personalitats més destacades d'aquesta dècada esmentem Ivan Sutherland, l'aportació més significativa del qual va ser l'***Sketchpad***, un programari que va desenvolupar per a la seva tesi doctoral al MIT (Massachusetts Institute of Technology) i que va canviar la manera com interactuen les persones amb els ordinadors.

Als anys setanta, el fet que va tenir més impacte va ser la creació, per part de la companyia Xerox, del centre de recerca i desenvolupament **Xerox PARC** (*Palo Alto Research Center*). Va ser el primer ordinador que va incorporar una interfície gràfica d'usuari i va utilitzar la metàfora de l'escriptori.

A partir d'aquí, ja entrem en l'etapa de l'**ordinador personal**. L'avenç de la informàtica i la disminució de costos de producció obren un mercat incipient per als ordinadors personals de consum. Als anys noranta neix el World Wide Web, el canvi més important d'aquest període, entès com l'aplicació estrella d'internet. El mercat comença a prendre vertadera consciència de la importància de la usabilitat en el desenvolupament de productes interactius i, com a conseqüència d'això, es produeix una tangible professionalització de la IPO sota l'emblema de l'enginyeria de la usabilitat.

Xerox Alto va ser la primera computadora que va incorporar una interfície gràfica d'usuari, i va obrir el camí al que serien els ordinadors personals.

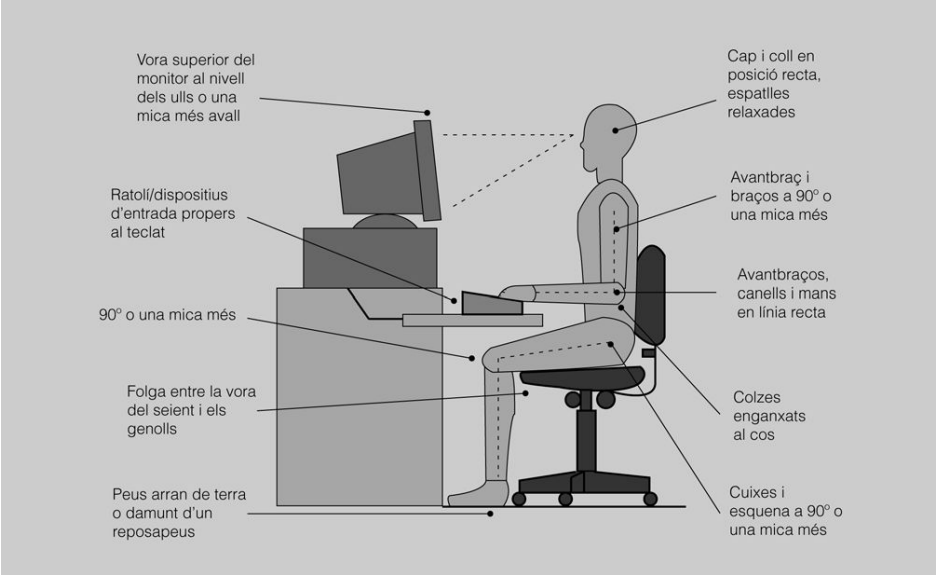
En l'última dècada, s'han posat de manifest clarament les limitacions dels models cognitius arrelats a la disciplina. La recerca en IPO sempre ha centrat l'estudi en el comportament racional de l'usuari, i ha deixat de banda el seu **comportament emocional** (estats afectius, estats d'humor i sentiments), i també la importància de l'estètica en aquest comportament. Darrerament s'ha començat a reconèixer que aquests aspectes emocionals tenen un paper fonamental en la interacció de l'usuari, ja que tenen un gran impacte en la motivació d'ús, la valoració del producte, els processos cognitius, la capacitat d'atenció i memorització i el rendiment.

1.3.1 Aspectes ergonòmics, psicològics i cognitius

Els aspectes ergonòmics a les interfícies d'usuari estudien la relació interactiva entre les persones, les màquines i els entorns de treball, amb una atenció especial als factors humans psicològics, socials i físics que condicionen aquesta interacció.

La relació entre l'ergonomia i la IPO (la interacció persona-ordinador) és molt estreta, ja que la IPO es nodreix principalment del coneixement que genera la comunitat professional i científica de l'ergonomia (estudis antropomètrics, ergonomia cognitiva...), i adapta aquest coneixement a les característiques de les tecnologies emergents (ordinadors...). Actualment no ens ha d'estranyar que hi hagi molts contextos en què ergonomia i IPO s'utilitzen com a sinònims (vegeu la figura 1.6).

FIGURA 1.6. Ergonomia entre l'ordinador i la persona



Els aspectes psicològics a les interfícies d'usuari estableixen coneixements i teories sobre el comportament de les persones i la manera com processen la informació, així com metodologies i eines per avaluar el grau de satisfacció de les persones amb el disseny de la interfície. La psicologia cognitiva és la psicologia que s'encarrega de l'**estudi de la cognició**, és a dir, dels processos mentals implicats en el coneixement. Té com a objecte d'estudi els mecanismes bàsics i profunds pels quals s'elabora el coneixement; des de la percepció, la memòria i l'aprenentatge, fins a la formació de conceptes i raonaments lògics. Per *cognitiu* entenem l'acte de coneixement, en les seves accions per a emmagatzemar, recuperar, reconèixer, comprendre, organitzar i utilitzar la informació rebuda a través dels sentits.

De tots els vessants de la psicologia, la **psicologia cognitiva** és sens dubte la que ha tingut més presència i impacte en el desenvolupament de la IPO, en concret el model conegut com a processament humà de la informació.

1.3.2 Signes visuals i interactius

Una interfície gràfica proveeix interacció visual quan l'usuari, per comunicar-se amb el sistema, pot expressar-se seleccionant, arrossegant, movent objectes presents a la pantalla... Això generalment s'aconsegueix de la mà dels elements simbòlics o icònics que permeten, de manera associativa, donar-li indicis del que pot fer; és a dir, confereixen cert control a través de la manipulació directa.

Una **interfície visual** ha d'utilitzar gràfics, colors, moviments, animacions i so per transmetre informació a l'usuari del sistema.

Aquí entra el concepte del **disseny icònic**, que es diferencia del disseny visual per la qualitat i la semàntica expressada a través dels recursos visuals que s'utilitzen en aquesta superfície d'interacció. L'objectiu del disseny icònic és proveir una imatge al sistema interactiu que concordi fidelment amb la representació mental que l'usuari tingui. Però, **què entenem per icona?** En aquest context una icona és molt més que una petita imatge en forma de botó, element interactiu i barra d'eines. La definició més acurada és la que la descriu com una imatge, una figura o un símbol que representa un concepte subjacent.

El que diferencia la icona de la imatge tradicional és que té una semàntica o part lògica associada que ha d'estar perfectament homologada a la part física. Així, podem dir que el disseny visual camina de paral·lelament al disseny icònic en el context de les tecnologies traduïdes per la seva interpretació a través de la interfície. De manera anàloga al disseny visual, el disseny icònic pot estar present en qualsevol part de la interfície; és a dir, pot ser aplicat en:

- **La visualització icònica**: en aquest cas, s'utilitza la icona com a mitjà de representació per modelar o estructurar la metàfora que permet el domini de l'aplicació i els conceptes que es porten a terme a nivell d'interfície. Es poden visualitzar icònicament tots els components i entitats del sistema, les seves funcionalitats, dades d'entrada, de sortida, els seus estats possibles...
- **La interacció icònica**: el disseny icònic aplicat a la IPO (interacció persona-ordinador) dona lloc a un "diàleg" acompanyat per la selecció d'icones, per una manipulació directa que ha de ser significativa, a través d'una retroalimentació semàntica, i això es pot aconseguir per mitjà de gestos metafòrics, per una animació representativa, per menús icònics mitjançant els quals els usuaris porten a terme les seves tasques d'una manera natural i simple.

1.3.3 Consistència de la interfície gràfica d'usuari (GUI)

La interfície gràfica d'usuari (GUI) és aquella que permet interactuar amb els dispositius electrònics a través d'icones gràfiques i d'indicadors visuals, com ara la **notació secundària**, en comptes de les interfícies d'usuari basades en text, les etiquetes d'ordre o la navegació de text.

Notació secundària vs. Notació formal

Un exemple de notació secundària és el ressaltat en colors del text d'un codi de programació. Els colors no ofereixen cap significat sinó que només diferencien les parts del codi. Al contrari, la notació formal serien els elements del codi que tenen un significat: els caràcters i construccions sintàctiques utilitzades en un llenguatge de programació.

El disseny de la GUI implica diverses disciplines, com ara el disseny d'interacció, el disseny gràfic i l'arquitectura de la informació. L'objectiu primordial és aconseguir que la GUI sigui consistent. La **consistència de la GUI** dependrà de la claredat per indicar les seves funcions a l'usuari i per mostrar el seu contingut, del model conceptual que tingui, i de les opcions de retroacció que ofereix (*feedback*).

GUI vs. CLI

Les interfícies gràfiques d'usuari (GUI) es van introduir com a reacció a la corba d'aprenentatge empinada percebuda de les interfícies de línia d'ordres (CLI), que requereixen comandaments per escriure en un teclat d'ordinador.

Les accions d'una GUI se solen fer mitjançant la manipulació directa dels elements gràfics. Més enllà de les computadores, les interfícies gràfiques d'usuari s'utilitzen en molts dispositius amb superfícies tàctils, com ara, els telèfons intel·ligents i les tauletes, peró també en altres dispositius com els reproductors de MP3, reproductors portàtils, petits controls domèstics, d'oficina i industrials, i dispositius de jocs.

Dissenyar la composició visual i el comportament temporal d'una GUI és una part important de la programació d'aplicacions de programari en l'àmbit de la interacció entre humans i ordinadors. El seu objectiu és **millorar l'eficiència i la facilitat d'ús** del disseny lògic subjacent d'un programa emmagatzemat, que és una disciplina de disseny anomenada ***usabilitat***. Els mètodes de disseny centrat en l'usuari s'utilitzen per garantir que el llenguatge visual introduït en el disseny estigui ben adaptat a les tasques.

Les característiques d'interfície gràfica visibles d'una aplicació es denominen, de vegades, **chrome o GUI** (pronunciat [gooey]). Normalment, els usuaris interactuen amb la informació mitjançant la **manipulació de *widgets* visuals** que permeten les interaccions adequades al tipus de dades que tenen. Els *widgets* d'una interfície ben dissenyada se seleccionen per donar suport a les accions necessàries per assolir els objectius dels usuaris.

Un *model-view-controller* permet una estructura flexible en la qual la interfície és independent i està indirectament relacionada amb les funcions de l'aplicació, de manera que la GUI es pot **personalitzar fàcilment**. Això permet als usuaris seleccionar o dissenyar una altra pell a voluntat, i facilita el treball del dissenyador per canviar la interfície a mesura que evolucionen les necessitats dels usuaris. Un bon disseny d'interfície d'usuari es refereix més als usuaris i menys a l'arquitectura del sistema.

Grans *widgets*, com ara finestres, solen proporcionar un marc o un contenidor per al contingut principal de presentació, com ara una pàgina web, un missatge de correu electrònic o un dibuix. Els més petits solen actuar com a eina d'entrada de l'usuari.

Es pot dissenyar una GUI per als requisits d'un **mercat vertical**, a mode d'interfícies d'usuari gràfiques específiques de l'aplicació. Alguns exemples inclouen caixers automàtics (ATM), pantalles tàctils de punt de venda (POS) a restaurants, xecs d'autoservei que s'utilitzen en una botiga minorista, autoconsulta i *check-in* aeri, quioscos d'informació en un espai públic, com ara una estació de tren o un museu, i monitora o controla pantalles en una aplicació industrial incrustada que utilitza un sistema operatiu en temps real (RTOS).

Un mercat vertical és un grup de consumidors molt reduït, i que pertany a un col·lectiu molt específic. Un exemple seria la comunitat professional mèdica.

Una **interfície gràfica d'usuari (GUI)** utilitza una combinació de tecnologies i dispositius per proporcionar una plataforma amb la qual els usuaris poden interactuar, per a les tasques de recollida i producció d'informació.

Hi ha una sèrie d'elements que conformen un llenguatge visual que han evolucionat per representar la informació emmagatzemada a les computadores. Això facilita a les persones amb pocs coneixements informàtics treballar i utilitzar programari d'ordinador. La combinació més comuna d'aquests elements a les GUI és el paradigma de **finestres, icones, menús i punter** (WIMP), especialment en ordinadors personals.

L'estil d'interacció WIMP utilitza un dispositiu d'entrada virtual per representar la posició d'un dispositiu apuntador, sovint un ratolí, i presenta informació organitzada a les finestres i representada amb icones. Les ordres disponibles es compilen entre els menús, i es fan accions fent gestos amb el dispositiu indicador. Un gestor de finestres facilita les interaccions entre finestres, aplicacions i el sistema de finestres. El **sistema de finestres** controla els dispositius de maquinari, com ara els dispositius de senyalització, el maquinari gràfic i el posicionament del punter.

En ordinadors personals tots aquests elements es modelen a través d'una metàfora d'escriptori per produir una simulació anomenada **entorn d'escriptori**, en què la pantalla representa un escriptori on es poden col·locar documents i carpetes de documents. Els administradors de finestres i un altre programari es combinen per simular l'entorn d'escriptori amb diversos graus de realisme.

1.3.4 El 'look and feel' i les necessitats d'acomodació

En el disseny del programari, ***look and feel*** (en català, 'aspecte i sensació') és un terme utilitzat en relació amb una interfície gràfica d'usuari i inclou aspectes del seu disseny, com ara colors, formes, disseny i tipus (l'aspecte); així com el comportament d'elements dinàmics com botons, quadres i menús (la sensació). El

terme també pot fer referència a aspectes d'una interfície d'usuari no gràfica (com ara una interfície de línia d'ordres), així com a aspectes d'una API, principalment a parts d'una API que no estan relacionades amb les seves propietats funcionals. El terme s'utilitza en referència tant al programari com als llocs web.

Per tant, el *look and feel* fa referència a les **necessitats d'acomodació** d'aspectes gràfics i/o formals a la funció. Si posem com a exemple un lloc web, en els seus termes més bàsics, el *look and feel* seria l'aspecte del lloc per a l'usuari i el que senti quan hi estigui interactuant. Dit d'una altra manera:

- El *look* del teu lloc web es defineix pels següents components:
  - Paleta de color.
  - Imatges.
  - Disseny.
  - Opcions de font.
  - Estil general.
- El *feel* és determinat per les següents característiques:
  - El moviment i la resposta dels components dinàmics com menús desplegables, botons, formes i galeries.
  - Efectes de so.
  - La rapidesa amb què les pàgines i les imatges es carreguen.

L'aspecte i la sensació de les interfícies d'usuari del sistema operatiu serveixen per a dos propòsits generals: en primer lloc, **proporcionar *branding***, ajudant a identificar un conjunt de productes d'una empresa; en segon lloc, **augmentar la facilitat d'ús**, ja que els usuaris es familiaritzaran amb el funcionament d'un producte (aspecte, lectura...) i poden traduir la seva experiència a altres productes amb la mateixa aparença.

Per què és important l'aparença d'un projecte multimèdia interactiu? Doncs perquè l'aspecte i la sensació general del teu projecte són importants, ja que mostren una actitud als teus clients abans que comencin a utilitzar-lo; es poden descriure com **"la personalitat"**. La personalitat del teu projecte ha de coincidir amb l'actitud del teu negoci i els objectius.

Així, com podeu utilitzar el *look and feel* per **millorar el vostre projecte**? Doncs bé, l'aspecte i la sensació es poden descriure fent servir adjectius com ho faries per descriure un amic o soci de negocis. Mitjançant l'ús d'adjectius precisos, es pot ajudar l'equip de disseny a triar les seves opcions de disseny i composició abans que presentin la seva feina.

Quan es treballa amb un equip de disseny, s'ha de dedicar temps a definir clarament els objectius de negoci i els adjectius clau en relació amb l'aparença del lloc web per assegurar-se que tothom està a la mateixa pàgina abans de començar amb la feina.

1.3.5 Disseny centrat en l'usuari (DCU)

Per explicar el disseny del diàleg entre les persones (usuaris) i els sistemes d'informació ens basarem en la perspectiva de **disseny centrat en l'usuari** (DCU). En termes generals, el disseny centrat en l'usuari és una filosofia i procés de disseny en el qual les necessitats, els desigs i les limitacions de l'usuari final d'una interfície, o document, prenen una atenció i rellevància considerables en cada nivell del procés de disseny.

Aquest tipus de disseny entre les persones i els sistemes d'informació entén **el nostre enfocament**, en què són les persones normals i corrents, i no els tecnòlegs, les que utilitzaran les aplicacions, les que n'estaran satisfetes o no i les que en definitiva en determinaran l'èxit en la seva implementació. és a dir, en lloc de comencÌ§ar pensant en l'aplicació i provar posteriorment amb alguns usuaris un prototip que hàgim elaborat, hauríem de comencÌ§ar pensant en les persones que la utilitzaran:

- Qui són els usuaris?
- Quines són les seves tasques i metes?
- Quin nivell d'experiència tenen els usuaris?
- Quines funcions es necessiten?
- Quina informació necessiten els usuaris i de quina manera?
- Com s'espera que funcioni?
- Quins són els casos més adversos?
- Es faran diverses tasques alhora?

Com podem obtenir les respostes adequades a aquestes preguntes? El nostre objectiu no és recollir informació sobre el que diu l'usuari sinó sobre **el que fa**, encara que la majoria de les vegades el que ens expliqui (aplicant adequadament les tècniques de test, entrevista i reunió de grup) serà informació molt rellevant sobre les seves pràctiques quotidianes i les seves expectatives.

Si volem que una aplicació sigui eficacÌ§ i satisfactòria, i que posseeixi una interfície d'usuari realment intuiÌˆtiva i fàcil d'aprendre, l'usuari ha d'estar en el centre d'atenció des del principi.

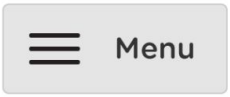
Recomanacions i normatives aplicables: els 10 principis de la usabilitat

El **disseny UX** (*user experience*) és el disseny en què es basen la majoria dels dissenyadors per fer els seus projectes. El disseny centrat en l'usuari (DCU) no és altra cosa que dissenyar pensant en millorar la usabilitat, l'accessibilitat i la satisfacció que s'obté quan s'interactua amb una interfície.

En aquest sentit, Jackob Nielsen és un guru de la usabilitat reconegut a tot el món. L'any 1995 va formular 10 principis bàsics d'usabilitat, o també anomenats "principis heurístics" perquè es basen en regles generals i no directrius específiques d'usabilitat. Per arribar a aquests deu principis d'usabilitat i accessibilitat va aplicar el **mètode heurístic**; en aquest ordre:

1. Identificar el problema.
2. Definir el pla per solucionar el problema.
3. Dur a terme el pla establert.
4. Analitzar-ne el resultat.

Els principis d'usabilitat de Jakob Nielsen són la base perquè qualsevol producte interactiu sigui ***user-friendly***. O, el que és el mateix, és un tipus de disseny centrat en l'usuari per aconseguir que en millori l'experiència. El decàleg sobre usabilitat de Jackob Nielsen està centrat en el disseny web, però és perfectament adaptable a altres projectes interactius. Els deu punts són els següents:



Icona de menú desplegable en un web versió mòbil.

1. **Visibilitat de l'estat del sistema**: aquest principi ens indica que l'usuari sempre ha d'estar informat del que està passant i oferir-li una resposta en el menor temps possible. Exemples d'aquest principi:
   - Les barres de càrrega de les imatges o dels processos de descàrrega d'arxius.
   - Els *Breadcrumbs* o molles de pa que ens mostren on ens trobem.
   - Els indicadors dels processos de compra que solen indicar-nos en quina fase ens trobem.
2. **Relació entre el sistema i el món real**: hem de connectar amb l'usuari. El sistema ha de "parlar" el llenguatge de l'usuari amb paraules o frases que li siguin familiars i que pugui reconèixer amb facilitat, usar imatges i icones clares. L'exemple més clar és l'ús de la paperera com a símbol d'esborrar. Aquesta icona sabem per a què serveix o almenys ho intuïm només de veure-la.
3. **Control i llibertat de l'usuari**: de vegades, un usuari s'equivoca. És normal, forma part de la naturalesa humana, equivocar-se. Hem de donar a l'usuari la possibilitat d'esmenar l'error i no sentir-se frustrat per no poder fer alguna cosa. No heu afegit mai un producte en un carret de la compra i després us n'heu penedit? El botó d'esborrar l'article de la llista de la compra és un clar exemple d'aquest principi d'usabilitat per a pàgines web o botigues en línia. Un altre exemple podria ser el botó de desfer.
4. **Consistència i estàndards**: un altre punt que hem de tenir en compte és que cal seguir els convenis establerts per a certes icones. Amb l'increment dels dispositius mòbils han aparegut nous gestos i icones que ja hem assumit com a normals. Per exemple:

- Les línies horitzontals que indiquen el menú desplegable. Si no utilitzéssim aquesta icona estaríem empitjorant la usabilitat. De fet, l'usuari pot arribar a no entendre una icona nova i buscar la que ja coneix.
- Els botons verds els associem a acceptar una cosa i els vermells a cancel·lar.
- El menú de navegació normalment és a l'esquerra.

5. **Prevenció d'errors**: diu el refrany "val més curar-se amb salut"; hem de prevenir, en tot el possible, qualsevol error que pugui cometre l'usuari. I, si en comet un, hem de posar al seu abast totes les opcions possibles per poder-lo corregir. En són exemples:

   - L'opció d'autocompletar de Google és un bon exemple d'aquest principi d'usabilitat web. El cercador et dona l'opció de completar el text, i amb això t'ajuda, d'una banda, a fitar la teva cerca i, de l'altra, s'assegura que escrius el text correctament.
   - La confirmació d'adreça de correu electrònic o de la contrasenya amb doble camp en els formularis.
   - La comprovació de camps de formularis en temps real.

6. **Reconèixer abans que recordar**: sempre és millor reconèixer que obligar l'usuari a memoritzar accions o objectes perquè pugui complir el seu objectiu: ajudem l'usuari a no memoritzar. Com ho podem fer? L'exemple més fàcil d'entendre és amb un editor de textos. Pensa-hi: quan vas a seleccionar una font de les moltes que has instal·lat, què és més fàcil: recordar-ne el nom i com era o veure com és si se'ns en mostra una previsualització?

7. **Flexibilitat i eficiència d'ús**: hem de tenir un producte preparat per a tot tipus d'usuari, des dels més novells fins als més experimentats. Si aconseguim que qualsevol persona pugui utilitzar-lo aconseguim flexibilitat. I si tenim opcions per als més experimentats obtenim eficiència. Un exemple en seria el cercador de Google: si no tens molta experiència en el seu ús, simplement poses el que vols buscar i llestos; però si ets més experimentat i vols aconseguir cerques més específiques pots usar operadors dins del cercador. No cal conèixer els operadors de Google per poder complir el teu objectiu. Però, si els coneixes, pots trigar la meitat de temps a aconseguir la teva meta.

8. **Disseny estètic i minimalista**: els productes interactius multimèdia no han de contenir informació innecessària; distreu l'usuari i pot arribar a molestar. La millor manera de recordar aquest principi bàsic d'usabilitat és amb l'acrònim KISS, *keep it simple stupid*.

9. **Ajudar els usuaris a reconèixer, diagnosticar i corregir els errors**: hem d'intentar que tots els errors que puguin passar al teu lloc estiguin expressats en un llenguatge que entengui tothom, no només els especialistes. Per exemple, la majoria coneixem què és un error 404, però hi ha gent que no sap el que és. Per això l'hem de canviar perquè en lloc de l'error 404 aparegui

una cosa una mica més amigable com: "Ho sento, pàgina no trobada" i donar a l'usuari una possible sortida afegint pàgines relacionades o un cercador intern perquè pugui buscar i no surti del lloc. D'aquesta manera tan senzilla, estem indicant a l'usuari què passa i què ha de fer per sortir.

10. **Ajuda i documentació**: amb aquests principis s'intenta sempre que l'usuari no hagi de fer servir documents d'ajuda per poder navegar, utilitzar una aplicació o qualsevol altre producte interactiu. Així i tot, sempre hem de donar a l'usuari la possibilitat de tenir un petit manual de funcionament. Aquesta ajuda ha de ser fàcil de localitzar, definir els passos clarament i no ser molt extensa. Hi ha nombrosos exemples d'aquest principi general d'usabilitat web:

    - FAQs, *frequently asked questions*, preguntes freqüents
    - La icona de la interrogació a prop d'algunes opcions
    - Minitours quan s'obri una aplicació, on es mostri el que és essencial

La **usabilitat** és la facilitat amb què les persones interactuen amb una eina amb la finalitat d'aconseguir un objectiu concret. En definitiva, l'objectiu d'aquest principi és donar un seguit d'orientacions i recomanacions per aconseguir crear **un disseny per a tothom**.

1.3.6 Tècniques i procediments per a l'estudi de l'usuari

A continuació, farem una revisió de les metodologies de recollida sobre els requisits de l'usuari; és a dir, per a dur a terme estudis de les persones que faran servir el nostre producte. Pararem atenció, amb especial èmfasi, a dos tipus de metodologies qualitatives: l'entrevista i el grup d'estudi, ja que ens permeten aproximar-nos al que l'usuari realment fa i a les interaccions que efectua a l'hora de dur a terme les seves tasques. També explicarem com cal recollir la informació dels primers resultats per tal d'esbrinar les necessitats a les quals l'usuari esperarà donar resposta mitjancÌ§ant el producte o servei interactiu.

Així, una vegada hem identificat i revisat els objectius del producte o servei, caldrà esbrinar quina metodologia serà la més adient per analitzar el grup d'usuaris definit per aquest producte; n'hi ha de dos tipus: quantitatives i qualitatives.

Amb una **metodologia quantitativa** recollim dades sobre l'usuari que es puguin quantificar; per exemple, de tipus demogràfic i social, que ens ajudin a dibuixar quin és el perfil de l'usuari, com ara l'edat, el sexe, la localització... Per fer-ho, podem dissenyar qüestionaris que, analitzats mitjancÌ§ant paquets estadístics, ens facilitaran determinades agrupacions de dades que es correspondran amb determinats perfils d'usuari.

Però, tot i que aquesta informació és molt útil, encara és massa superficial per a donar-nos informació qualitativa d'importància sobre els requisits i les

necessitats dels usuaris. Una bona manera seria comencÌ§ar per aquestes anàlisis quantitatives i, un cop finalitzades, utilitzar una aproximació qualitativa que ens permeti aprofundir en els seus interessos, motivacions i expectatives quant al producte o servei que tractem en aquest moment. Com més informació tinguem sobre l'usuari més acurades seran les properes avaluacions.

Així, amb les **metodologies qualitatives**, a més d'obtenir informació sobre les característiques dels usuaris potencials, el nostre objectiu a l'hora d'estudiar-los pot ser molt divers: hem de recollir informació sobre com el producte satisfà les seves necessitats, com interactua (amb artefactes o amb altres persones) i el context en el qual duen l'avaluació. En aquest sentit, podem fer servir dues tècniques: l'entrevista i el grup d'estudi.

L'**entrevista** és la tècnica qualitativa de recollida d'informació que per excel·lència ens permet obtenir dades directament dels usuaris. Segons l'estructura que utilitzem a l'hora de recollir la informació de l'usuari tenim tres tipus d'entrevistes diferents:

- **Estructurades**: l'usuari va responent unes preguntes predissenyades sota un ordre pautat i tancat.
- **Semiestructurades**: deixem que l'usuari vagi marcant la pauta de l'entrevista fent unes preguntes més generals.
- **Obertes**: tenim uns punts que s'han de tractar però les preguntes es van fent a partir de la introducció i de com respon l'usuari. Normalment, traiem més profit de les entrevistes més obertes però això no significa que prèviament no necessitem una preparació.

Així com les entrevistes solen ser individuals, els **grups d'estudi** podrien ser com unes entrevistes grupals, tot i que hem de tenir en compte que no és el mateix una entrevista en grup que una suma d'entrevistes individuals. Per als grups d'estudi s'utilitzen tècniques de dinàmica de grups que ens permeten generar una discussió sobre els temes sobre els quals volem indagar. De l'anàlisi d'aquesta discussió n'obtindrem informació significativa sobre el que representa per als usuaris l'ús que fan d'algunes aplicacions o dispositius tecnològics en general i els obstacles amb els quals se solen trobar a l'hora de fer aquesta tasca o d'utilitzar aquests dispositius o aplicacions.

Indicadors i proves d'usabilitat

La usabilitat es mesura generalment amb una sèrie d'indicadors observables i quantificables dels quals se'n puguin obtenir uns resultats tangibles més enllà de la intuïció. A continuació, s'enumeren els més freqüents:

1. **Taxa d'acabament de la tasca**: aquest indicador es comptabilitza de manera binària (1 = tasca feta amb èxit i 0 = tasca no superada) aquesta mesura permet conèixer d'una manera simple la facilitat d'ús a l'hora de realitzar una tasca.

2. **Problemes d'usabilitat**: aquest indicador revela els problemes que li han sorgit a l'usuari a l'hora de fer una tasca. A més, pot incorporar una escala de gravetat lligada als problemes sorgits. Ajuda a conèixer la probabilitat amb la qual l'usuari es trobarà amb un problema en cada fase del desenvolupament. A més, es poden obtenir estadístiques de la quantitat de problemes que li sorgeixen a cada usuari i quins són els més freqüents.

3. **Temps d'acabament de la tasca**: serveix per mesurar l'eficiència i la productivitat. És el temps total utilitzat per dur a terme una tasca. Aquesta mesura registra les hores, minuts i segons empleats.

4. **Nivell de satisfacció de la tasca**: es tracta d'un petit qüestionari per esbrinar la dificultat que ha observat l'usuari fent la tasca, que a més servirà per comparar la dificultat entre diverses tasques.

5. **Nivell de satisfacció de la prova**: qüestionari en forma d'una o diverses preguntes que recullen les impressions que l'usuari ha percebut quant a la facilitat o dificultat de l'ús general de l'aplicació, lloc web...

6. **Errors**: aquest indicador recull qualsevol acció involuntària, error, omissió o acció no deliberada que l'usuari hagi comès en intentar realitzar una tasca. Recupera cada incidència juntament amb una descripció. Aquest indicador és de gran ajuda ja que revela els camps que causen confusió a l'usuari amb més freqüència, i en els quals comet fallades o accions equivocades que provoquen la desviació de l'objectiu proposat per a la tasca. Això últim recorda al significat arrel d'usabilitat, i per tant cal posar-hi més atenció. A més es poden ordenar per categories i classificar en funció de la seva importància.

7. **Expectatives**: en aquest indicador es demana a l'usuari que estimi el nivell de dificultat que espera trobar a l'hora de fer una tasca, per després poder comparar els resultats amb les impressions reunides en finalitzar la tasca (l'esperat amb el real).

8. **Pàgines vistes / clic**: s'utilitza en eines d'analítica web, per la qual cosa no és necessari fer un estudi d'usabilitat per recaptar dades relacionades amb aquest indicador. Serveix per a aplicacions web i llocs web. Està demostrat que hi ha una alta correlació entre els clics i el temps dedicat a cada tasca, i el primer clic és crucial per determinar si la tasca serà realitzada finalment amb èxit o no. És una de les millors mètriques per mesurar l'eficiència.

9. **Conversió**: es tracta d'una mesura d'efectivitat que també es pot trobar en eines d'analítica web. És una eina essencial en el comerç electrònic ja que revela si els usuaris poden completar les tasques d'inscripció o compra en un lloc web; també es comptabilitzen de manera binària (1 = convertit, 0 = no convertit). A l'hora de completar aquesta tasca i per tant d'aconseguir la conversió o no, normalment afecten diversos factors, entre els quals hi ha els problemes d'usabilitat, els errors i els temps.

10. **SUM** (*Single Usability Metric*): serveix per descriure la usabilitat d'un sistema mitjançant la combinació de tres indicadors d'usabilitat: la taxa d'a-

cabament de tasca, el temps d'acabament de tasca i el nivell de satisfacció de la tasca.

11. **SUS** (*System Usability Scale*): és una eina ràpida i fiable per mesurar la usabilitat. Consisteix en un qüestionari compost per deu preguntes que es puntuen mitjançant una escala Likert (d'1 a 5).

Mètodes d'avaluació de la usabilitat

L'avaluació de la usabilitat és la fase més important del procés de disseny centrat en l'usuari. Hi ha diferents mètodes, però l'elecció d'un o un altre depèn bàsicament de **tres factors**: el pressupost reservat a l'avaluació, l'adequació al tipus de projecte i les limitacions de calendari. Els mètodes són:

- **Passeig cognitiu** (*cognitive walkthrough*): els avaluadors fan una selecció d'un grup de tasques significatives i les executen per saber si pot resultar difícil per a l'usuari identificar i utilitzar l'element de la interfície més adequat per al seu objectiu i si la resposta que retorna el sistema és prou clara. Aquest mètode és molt adequat per treballar amb usuaris amb poca o cap experiència, que simplement exploren per a aprendre a utilitzar l'aplicació.
- **Anàlisi de tasques** (*task analysis*): un analista determina el conjunt d'objectius dels usuaris previstos i les tasques que els permeten aconseguir i s'ordenen d'acord amb la importància de l'objectiu i la freqüència d'execució de la tasca. A continuació, l'anàlisi suggereix com es pot fer la tasca amb més eficàcia o proposa noves tasques que puguin aconseguir més efectivament els objectius.
- **Grups d'estudi** (*focus groups*): aquest mètode és molt adequat per a obtenir opinions dels usuaris i comprovar les reaccions inicials a un disseny; també resulta molt eficient per a detectar en quina mesura el disseny difereix de les expectatives dels usuaris.
- **GOMS**: es tracta d'un conjunt de tècniques definides per Card, Moran i Newell el 1983 per a modelar i descriure el procés humà d'execució de tasques. L'acrònim GOMS es refereix a:
  - *Goals* o 'objectius' que l'usuari intenta aconseguir, especificats jeràrquicament.
  - *Operators* o 'operadors' o microoperacions que l'usuari duu a terme per a obtenir el seu objectiu.
  - *Methods*, 'mètodes' o seqüències d'operadors agrupades per a aconseguir un objectiu determinat.
- **Regles de selecció** (*selection*): s'utilitzen per a decidir quin mètode es farà servir per a resoldre un objectiu, quan hi ha diverses possibilitats.
- **Inspecció de la usabilitat**: els experts se centren especialment en les àrees que puguin resultar problemàtiques per als usuaris. El mètode d'inspecció més utilitzat en l'avaluació de sistemes interactius és el de l'avaluació heurística (*heuristic test*).

- **Test amb usuaris** (*user test*): aquest és el mètode més utilitzat quan es volen conèixer els problemes d'usabilitat amb els quals es pot trobar l'usuari final. Es basa en l'observació dels usuaris mentre executen unes tasques representatives. La repetició del test amb diversos usuaris permet descobrir quins aspectes del disseny s'han de millorar.

Representació visual dels resultats de l'avaluació

A continuació s'exposen alguns exemples de resultats d'avaluació d'usabilitat, de diferents tipologies. Aquesta primera taula és el resultat d'una **avaluació heurística** que pren com a referència alguns dels indicadors d'usabilitat més importants i els puntua a través d'un qüestionari amb la numeració 0, 1 i 2; on 2 seria la puntuació més favorable (vegeu la taula 1.2)

Taula 1.2. Valors percentuals obtinguts dels aspectes avaluats

Aspecte avaluat	Sí	De vegades	No
Navegació	54,2	37,5	8,1
Funcionalitat	71,6	18,3	6,6
Control d'usuari	37,5	62,4	0,0
Llengua i contingut	64,6	26,6	8,7
Ajuda en línia	65,3	24,6	10,0
Informació del sistema	24,9	69,6	5,0
Accessibilitat	47,3	45,3	7,3
Coherència	86,6	10,8	2,5
Prevenció d'errors	37,1	57,1	5,7
Claredat arquitectònica	66,5	25,0	10,8

Una altra representació de resultats podria ser l'obtinguda a través de l'**observació dels usuaris** fent unes tasques determinades. En aquest cas es marcaria com un 0 les tasques no aconseguides, amb un 1 les aconseguides amb dificultat i amb un 2 les superades (vegeu la taula 1.3).

Taula 1.3. Grau d'acompliment de les tasques en percentatges

Tasca	Tipus de tasca	Puntuació
5	Cerca d'un terme en dos àmbits	80,55
1	Claredat de la presentació de la informació (idiomes)	78,35
3	Cerca de col·locacions	66,65
4	Claredat de resultats	65,75
8	*Feedback*	62,95
2	Cerca d'un terme	61,65
7	Cerca de lema	50,95
6	Cerca d'informació no lingüística	50,85

Finalment, també es podria fer una **entrevista individual** a cadascun dels usuaris i puntuar les respostes de més negativa a més positiva. Sovint, en una entrevista

sorgeixen algunes dades subjectives importants que en el test són més difícils d'observar (vegeu la taula 1.4).

Taula 1.4. Resultats percentuals obtinguts de l'entrevista

Aspecte avaluat	Puntuació
Rapidesa	68,45
Facilitat	67,95
Satisfacció	56,60
Efectivitat	55,65
Utilitat de l'ajuda	33,50

1.3.7 Accessibilitat i com millorar-la

S'entén l'accessibilitat com la possibilitat que es pugui **accedir a un producte i utilitzar-lo**, independentment de les limitacions pròpies de l'individu o de les derivades del context d'ús. Dins de les limitacions pròpies de l'individu, hi pot haver deficiències visuals, auditives, motrius, cognitives i de llenguatge, però també pot tenir limitacions derivades del context d'ús i del dispositiu d'accés emprat (maquinari i/o programari), com pot ser l'idioma, l'experiència o els coneixements previs. Per exemple, comparteixen el mateix problema de visualització els usuaris amb visió reduïda i els que, sense patir discapacitat visual, utilitzen pantalles petites o hi accedeixen des d'entorns plens de fum.

La distinció entre **usabilitat**, que implicaria la facilitat d'ús, i **accessibilitat** no solament és difícil, sinó en molts casos innecessària; l'accessibilitat ha de ser entesa com a "part de..." i al mateix temps "requisit per a..." la usabilitat.

A continuació, exposem algunes **recomanacions generals** per fer un lloc més accessible:

- Tots els elements visuals, imatges o animacions, han de comptar amb una descripció de la seva funció.
- Incloure subtítols i transcripcions dels sons i descripcions dels vídeos.
- Fer servir text que tingui sentit quan es llegeixi fora de context.
- Aplicar una organització apropiada de la pàgina, amb encapçalaments, llistes i estructures consistents, taules solament per presentar dades tabulars, així com fulls d'estil en cascada on sigui possible.
- Donar alternatives accessibles als *scripts*, *applets* i *plug-ins* per als casos en què les característiques actives siguin inaccessibles o no suportades.
- Fer les taules de manera que es puguin llegir línia per línia, i afegir un resum.

I aquestes són algunes de les **eines per a millorar l'accessibilitat**:

- Una guia d'usuari, també coneguda com a *Manual d'usuari*: és un document de comunicació tècnica destinat a donar assistència a les persones que utilitzen un sistema en particular. En general, aquest document està redactat per un escriptor tècnic, com per exemple els programadors del sistema o els directors de projectes implicats en el seu desenvolupament, o el personal tècnic, especialment en les empreses més petites.

- El maneig d'excepcions: és una tècnica de programació que permet al programador controlar els errors ocasionats durant l'execució d'un programa informàtic. Quan hi ha un cert tipus d'error, el sistema reacciona executant un fragment de codi que resol la situació, per exemple retornant un missatge d'error o retornant un valor per defecte.

- La tolerància a errades o commutació per error (*failover*): es refereix a la capacitat d'un sistema de seguir funcionant, fins i tot en cas que es produeixi alguna errada en el sistema.

- Els lectors de pantalla (*screen*): són un programari que permet la utilització del sistema operatiu i les diferents aplicacions mitjançant l'ocupació d'un sintetitzador de veu que "llegeix i explica" el que es visualitza en la pantalla, la qual cosa suposa una ajuda per a les persones amb greus problemes de visió o completament cegues.

- L'audiodescripció: consisteix a proveir tota la informació de mitjans audiovisuals, en forma de text. A diferència de la descripció de l'àudio, la descripció del vídeo no està limitada a pauses en el diàleg existent. Les descripcions completes proveeixen tota la informació visual, inclosos contextos visuals, accions, expressions dels actors i un altre material visual. Addicionalment, es poden descriure i transcriure sons no parlats (riures i veus fora de pantalla, entre uns altres) de tots els diàlegs.

1.3.8 Tipus de diàlegs entre l'usuari i el SO

A continuació veurem diferents tipus de diàleg que serveixen per a establir la comunicació entre l'usuari i el sistema operatiu. Un **quadre de diàleg** és un tipus de finestra que permet una comunicació simple entre l'usuari i el sistema informàtic. El tipus quadre de diàleg més simple únicament informa l'usuari, és a dir, mostra un text (i eventualment objectes gràfics) i ofereix l'opció de tancar el quadre. Un exemple de quadre d'error més comú és el que es produeix quan es detecta un virus.

Després hi ha quadres de pregunta o confirmació, que a més de mostrar informació ofereixen alternatives a l'usuari. La més senzilla és una opció binària com acceptar/cancel·lar o permetre/impedir. Però hi ha versions més complexes amb més opcions. Per exemple, si l'usuari intenta tancar un editor de text i el document obert té canvis sense guardar, un quadre de diàleg complet podria mostrar quatre opcions: "tancar sense guardar", "guardar i sortir", "cancel·lar el tancament i

seguir editant" i "guardar amb un altre nom", aquesta última amb una caixa de text on introduir-hi el nom alternatiu.

Entre els **diàlegs per a menús**, trobem:

- Menús tipus índex: mitjançant una llista de paraules o frases molt curtes que descriuen o indiquen el que l'opció pot fer.
- Menús de petites icones: aquells que descriuen les accions que el programa ofereix en un moment determinat. Trobar aquestes icones no és tasca fàcil en general. Els dibuixos haurien d'indicar l'acció que la seva selecció desencadena.
- Menús niats: els menús es poden presentar com a menús de menús o menús niats. Cada opció del menú obre un altre menú i la selecció final es fa en les fulles de l'arbre de menús. Aquest recorregut d'un arbre de seleccions no ha de ser molt profund, màxim 3 nivells. Generalment els submenús s'obren a la dreta ja cap avall.
- Menús emergents (*pull-down*): aquest menú es desenrotlla cap avall en trepitjar el cap del menú.

D'altra banda, tenim els **diàlegs per accés directe WYSIWYG** (l'acrònim de *what you see is what you get*; en català, 'el que veus és el que obtens'), que s'apliquen als processadors de text i altres editors de text amb format (com els editors d'HTML) i permeten escriure un document veient-ne directament el resultat final, freqüentment el resultat imprès. S'utilitza en contraposició a altres processadors de text, avui dia poc freqüents, en els quals s'escrivia sobre una vista que no mostrava el format del text, fins a la impressió del document. En el cas d'editors d'HTML aquest concepte s'aplica als que permeten escriure la pàgina sobre una vista preliminar similar a la d'un processador de textos, i en aquest cas el programa s'ocupa de generar el codi font en HTML.

També tenim els **diàlegs per als comandaments**. Un comandament (calc de l'anglès *command*; 'ordre' o 'mandat') és una instrucció o ordre que l'usuari proporciona a un sistema informàtic, des de la línia de comandaments (com una *shell*) o des d'una trucada de programació. Pot ser intern (contingut en el mateix intèrpret) o extern (contingut en un arxiu executable).

La línia de comandaments o consola (de l'anglès, *shell*) permet a l'usuari ordenar instruccions mitjançant la pulsació de la tecla Enter en el teclat, llançant l'execució de processos interns a l'intèrpret de comandaments o de programes externs. En realitat, la línia de comandaments no és més que una de les formes en què l'usuari interacciona amb l'ordinador, juntament amb la interfície gràfica. En la línia de comandaments, l'usuari escriu la instrucció o el programa que vol que executi l'ordinador. Aquesta interacció la duu a terme un programa que se sol anomenar intèrpret de comandaments.

1.3.9 Normatives ISO-UNE i tècniques del W3C-WAI

L'ISO, l'Organització Internacional per a l'Estandardització, és una organització independent, no governamental, els membres de la qual són els organismes de normalització dels 163 països membres. És el desenvolupador mundial més gran d'**estàndards internacionals voluntaris** i facilita el comerç mundial en proporcionar normes comunes entre les nacions. S'han establert prop de vint mil normes, que abasten des dels productes manufacturats i la tecnologia fins a la seguretat alimentària, l'agricultura i la cura de la salut.

L'ús d'estàndards facilita la creació de productes i serveis que siguin segurs, fiables i de bona qualitat. Els estàndards ajuden les empreses a augmentar la productivitat i reduir al mínim els errors i els residus. Permeten **comparar directament productes de diferents fabricants**, facilitar que noves companyies puguin entrar en nous mercats i ajudar en el desenvolupament d'un comerç global amb bases justes. Les normes també serveixen per a protegir els consumidors i els usuaris finals dels productes i serveis, i això garanteix que els productes certificats compleixen els estàndards mínims establerts a escala internacional.

Una **Norma UNE** és un estàndard tècnic creat pels comitès tècnics de normalització, o **CTN**, dels quals en formen part totes les entitats i agents implicats i interessats en els treballs del comitè. Per regla general, aquests comitès solen estar formats per AENOR, fabricants, consumidors i usuaris, administració, laboratoris i centres d'investigació. Després de la seva creació, les normes tenen un període de sis mesos de prova durant el qual es revisen públicament, i després la comissió les redacta definitivament, sota les sigles UNE. S'actualitzen periòdicament. Les normes es numeren seguint una classificació decimal. El codi que designa una norma està estructurat de la següent manera (vegeu la figura 1.7):

- **A**: comitè tècnic de normalització del qual depèn la norma.
- **B**: número de norma emesa per aquest comitè, complementat quan es tracta d'una revisió (R), una modificació (M) o un complement (C).
- **C**: any d'edició de la norma.

Figura 1.7. Norma UNE

Norma	A	B	C
UNE	1	032	82

D'altra banda, el ***World Wide Web Consortium*** **(W3C)** és un consorci internacional que treballa per a desenvolupar i promocionar estàndards per al *World*

Wide Web. El dirigeix Tim Berners-Lee, creador del "WWW" i autor de les especificacions de *Uniform Resource Locator*, HTTP i HTML, que són les principals tecnologies d'aquesta xarxa. El consorci s'administra conjuntament entre el MIT (als Estats Units), l'ERCIM (a Europa) i la Universitat de Keio (al Japó). D'altra banda, el W3C també té distribuïdes oficines arreu del món.

El W3C fou creat inicialment per a **garantir una major compatibilitat** i acord entre els diferents membres de la indústria, a l'hora d'adoptar noves tecnologies web. Abans de la seva creació, per exemple, ja hi havia problemes de compatibilitat per culpa de diferents versions d'HTML específiques de cada venedor. Cal dir que aquests problemes encara perduren.

Un document del W3C segueix diferents **etapes de "maduració"** abans d'esdevenir una recomanació pròpiament dita: *Working Draft* (WD) o 'esborrany de treball', *Candidate Recommendation* (CR) o 'recomanació candidata', *Proposed Recommendation* (PR) o 'proposta de recomanació' i, finalment, una W3C *Recommendation* (REC) o 'recomanació'.

Logotip del W3C de recomanacions d'accessibilitat.

Les recomanacions es poden actualitzar separadament en **documents d'errata**, fins que hi hagi prou canvis significatius per a produir una nova edició de la recomanació. Aquest és el cas de l'**XML**, que es troba actualment en la tercera edició. D'altra banda, el W3C també publica diferents tipus de notes informatives, que en principi no caldria considerar com a estàndards.

A diferència de l'Organització Internacional de Normalització, o altres organismes d'estandardització internacionals, el W3C no té cap **programa de certificació**. Es va considerar que, ara per ara, no era aconsellable començar-ne un. Això no obstant, sense que el consorci ho comprovi i se'n faci responsable, es permet que qui respecti les recomanacions pugui demostrar-ho, per exemple, incloent el logotip del W3C al seu lloc web.

La ***Web Accessibility Initiative*** **(WAI)** o 'Iniciativa per a l'Accessibilitat Web' és una branca del *World Wide Web Consortium* que vetlla per l'accessibilitat de la web. Publica les Guies d'accessibilitat al contingut web, ja que la idea general del WAI és crear una sèrie de regles clares.

El grau d'accessibilitat s'estableix en nivells denominats A, AA, i AAA, corresponent, respectivament, a criteris mínims d'accessibilitat, ampliats, i màxims. Tècnicament, la **implementació de l'accessibilitat** es duu a terme mitjançant pautes de lògica estructural de documents, contingut "auto-explicatiu" i semàntica addicional, amb la intenció de permetre, a una audiència d'usuaris el més extensa possible amb, diferents nivells de dotació tecnològica i capacitat sensorial, accedir a la informació que s'intenta representar i transmetre.

S'incideix especialment en el fet que les capacitats tecnològiques capdavanteres (entengui's, per exemple, animacions amb Adobe Flash, Javascript o AJAX) s'usin amb la moderació o consideració suficients per arribar al màxim conjunt possible d'usuaris amb una funcionalitat suficient, sense desvirtuar el concepte d'accés, enfront del d'avenç tecnològic que estigui de moda, i prestant especial cura per oferir informació alternativa.

1.3.10 El document de disseny del joc (GDD)

El document de disseny del joc o GDD (*game document design*) és la base per al futur desenvolupament del joc. El GGD d'un videojoc seria l'equivalent a **"la Bíblia"** d'una sèrie o programa de televisió. Els elements que inclou un GGD són:

1. **Concepte**:
    - Títol: el títol del joc ha de ser un nom que capti l'atenció del jugador i del lector del document. A grans trets, ha d'incloure el concepte del joc.
    - Estudi/Dissenyadors: el nom de l'estudi i/o del dissenyador o dissenyadors del document. Gènere: el gènere es refereix al tipus de joc que serà: simulació, FPS, rol...
    - Plataforma: quin maquinari es requereix per jugar-hi. Ordinador personal, Xbox 360, PS3...
    - Versió: la versió del document. Ha de ser un número i no una data. (Vegeu el camp d'Historial de Versions).
    - Sinopsi de jugabilitat i contingut: en un o dos paràgrafs, cal descriure l'essència de jugar el joc i incloure part del contingut que tindrà: història, personatges, objectiu...
    - Categoria: comparar amb un o diversos jocs existents i emfatitzar en les diferències i característiques principals d'aquest joc.
    - Llicència: descriure si el joc està basat en un llibre o en una pel·lícula. Si és original, es pot ometre aquest camp o descriure per què es pot convertir en una franquícia.
    - Mecànica: descriure la jugabilitat i el control del joc. Què fa el jugador? Què fa servir per a aconseguir els seus objectius?
    - Tecnologia: fer una llista del maquinari i el programari que es requereix per produir el joc, des de llenguatge de programació fins a editor de sons.
    - Públic: A qui va dirigit el joc? Qui hi jugarà? Es pot descriure una demografia, com ara nens o adolescents, però és més senzill descriure un tipus de jugador, ja sigui casual, competitiu o veterà.
2. **Historial de versions**: el document de disseny és un artefacte que sempre estarà subjecte a canvis, per tant, és essencial un control per a les diferents versions del document i dels canvis que s'han fet. El número de versió varia en funció de si és un canvi mínim o un de molt radical. L'historial no s'inclou en un document que sotmet a revisió una empresa o grup de desenvolupadors, perquè inclou dates, i s'ha d'evitar que jutgin la idea del joc com un concepte vell.
3. **Visió general del joc**: ha d'establir la visió i l'enfocament del joc que guiarà el projecte fins al final del procés. El resum ha d'esmentar el més interessant,

els avantatges i l'originalitat del joc. Per què hi jugaria la gent? L'estructura dels paràgrafs és similar a la d'un assaig: una introducció ha d'abastar tots els aspectes importants, mentre que els paràgrafs següents han de detallar el que s'ha esmentat en la introducció. Al final, la conclusió ha de deixar al lector entusiasmat i emocionat per jugar al joc.

4. **Mecànica del joc**: aquesta secció descriu essencialment el que el jugador pot fer i com pot fer-ho. Són les regles del joc, que defineixen què ha de fer el jugador i quines conseqüències tindrà la seva acció.

    - Càmera: cal descriure el tipus de càmera que s'utilitzarà. És a dir, quina perspectiva té el jugador davant el que està veient en el joc, si és 3D o 2D, vista isomètrica, en primera persona... Perifèrics: quins perifèrics utilitzarà el jugador per aconseguir els objectius esmentats? Cal incloure tots els que siguin aplicables: teclat, ratolí, *gamepad*, micròfon...
    - Controls: cal descriure els botons i tecles que invoquin les accions esmentades en la secció de mecànica del joc.
    - Puntuació: cal explicar de quina manera el joc controla els assoliments del jugador, i també incloure si hi ha una taula de puntuacions que les compari entre els jugadors, ja sigui de manera local o en línia.
    - Guardar/Carregar: descriure com el jugador guarda el progrés dels objectius assolits en el joc i com pot continuar amb els objectius pendents. De la mateixa manera, descriure els dispositius d'emmagatzematge que s'usaran o si el joc té un sistema de contrasenyes.

5. **Estats del joc**: un estat del joc es refereix al lloc on es troba el jugador durant el joc, és a dir, si el jugador és al *Menú Principal*, està jugant un *Joc Multijugador*, és al *Menú de Pausa*... Els diagrames han de representar visualment les relacions entre els estats, si del Menú Principal es pot anar al Menú d'Opcions, com es fa? Què s'executa? Quina interfície mostra?

6. **Interfícies**: les interfícies donen la pauta a la interactivitat que té el jugador amb el joc, en aquesta secció s'ha de descriure l'aparença del joc, és a dir, colors i temàtica. És important deixar una impressió visual en el jugador i òbviament ha d'estar relacionada amb el concepte del joc.

    - Nom de la pantalla: el nom de la pantalla, si és el Menú Principal o l'HUD (Heads-up Display).
    - Descripció de la pantalla: per a què serveix aquesta interfície?
    - Estats del joc: cal fer una llista de tots els estats de joc que invoquin aquesta pantalla i també dels que s'hi puguin invocar. Imatge: una imatge que mostri en concepte com es veuria la pantalla.

7. **Nivells**: els jocs es divideixen en nivells o en mapes seqüencials dins dels quals s'ha de complir amb certs objectius per progressar. Hi ha jocs en els quals els nivells solament canvien en funció de la dificultat i els objectius segueixen sent els mateixos. S'han de descriure aquests canvis en aquesta secció.

- Títol del nivell: el nom del nivell.
- Trobada: descriure si és el primer nivell, un tutorial o un bonus. En altres paraules: quan arriba a aquest nivell el jugador? Descripció: una descripció detallada del nivell.
- Objectius: què ha de fer el jugador per acabar el nivell? Aquest camp també ha d'incloure si el jugador ha de resoldre certes endevinalles o derrotar un enemic per progressar. Progrés: descriure què passa quan el jugador acaba el nivell.
- Enemics: si el nivell té enemics als quals s'ha d'enfrontar el jugador, es posen en aquest camp. Si no n'hi ha, aquest camp es pot ometre.
- Ítems: s'han d'enumerar els objectes que el jugador o els enemics poden fer servir i que apareixen en aquest nivell. Si no n'hi ha, aquest camp es pot ometre.
- Personatges: els personatges que apareixen en el nivell. Es pot ometre si no hi ha personatges.
- Música i efectes de so: cal descriure la música d'aquest nivell i els efectes de so ambient.
- Referències de BGM i SFX: cal escriure totes les referències aplicables pel que fa a la música de fons i efectes de so descrits en la secció de música i efectes de so.

8. **Progrés del joc**: cal enumerar de manera seqüencial o per mitjà d'un diagrama de flux els esdeveniments o nivells que el jugador ha de passar per progressar en el joc. Hi ha jocs que tenen diferents maneres de joc. En aquest cas, es requereixen diverses llistes i/o diagrames.

9. **Personatges**: els personatges principals i secundaris que apareixeran en el joc; i també els enemics, que obstaculitzen el progrés del jugador, i poden ser màquines, altres personatges, monstres... Aquesta secció es pot ometre si el joc no té personatges. A banda, també esmentaríem al GDD tots els **ítems** o objectes especials que ajuden el jugador a arribar als objectius i progressar en el joc. Així, caldrà exposar en el GDD:

   - Noms: el nom dels personatges i dels enemics.
   - Descripció: descriure detalladament el físic del personatge i dels enemics, si és humà o extraterrestre, la roba, accessoris... i incloure una fotografia o dibuix conceptual del personatge.
   - Concepte: cal descriure la conducta i el comportament, i també els motius del personatge i de l'enemic. Cal esmentar, també, si són principals o secundaris. A banda, el concepte es pot relatar com una història del personatge, detallant-ne les relacions amb altres personatges del joc.
   - Trobada: quan apareix el personatge i l'enemic en el joc.
   - Armes: les armes del personatge i de l'enemic.
   - Ítems: els objectes del personatge i de l'enemic.
   - Personatge no jugable: si el personatge no és controlable pel jugador, cal descriure el seu propòsit per al joc i/o per al jugador.

- Habilitats: els personatges i els enemics tenen certes habilitats fora de les accions comunes. S'han de descriure.

10. **Guió**: en aquesta secció s'inclouen tots els diàlegs del joc. Poden ser molt variats o inexistents, en funció de la naturalesa del joc. El guió ha d'incloure encapçalaments, noms, diàleg, acció i transicions.

11. **Assoliments**: cal descriure els diversos assoliments o fites que el jugador obté mentre progressa en el joc. Poden atorgar medalles, personatges secrets o punts extra.

12. **Codis secrets**: s'han de descriure els codis secrets que el jugador pot introduir i com s'introdueixen.

13. **Música i sons**: la música i/o sons que s'utilitzaran en el joc, nom, descripció juntament amb un número de referència. Si és música de fons, la referència ha de començar amb una ema seguida d'un número en seqüència. Si és un efecte de so, ha de començar amb essa.

14. **Imatges de concepte**: totes les imatges que mostrin algun possible nivell, personatge, objecte s'han d'incloure en aquesta secció i han d'estar numerades i amb títol.

15. **Membres de l'equip**: informació de les persones que treballaran en el projecte, el nom, el paper o papers que tindran i els mitjans pels quals se'ls pot contactar.

16. **Detalls de producció**: abans d'entrar a l'etapa de producció, cal definir en el document alguns detalls del projecte; com ara:

    - Data d'inici: quan comença l'etapa de producció del projecte?
    - Data de finalització: quan acaba l'etapa de producció del projecte?
    - Pressupost: una estimació aproximada del pressupost del joc.

1.3.11 Prototip gràfic i prototip funcional

El **prototip gràfic o horitzontal** consisteix en preparar gràficament les parts visibles més importants del projecte per tal de fer-nos una idea de com quedarà. Hi solen aparèixer les diferents pantalles, menús i altre contingut que considerem significatiu, com per exemple les imatges que siguin canviants amb la interacció de l'usuari.

Quan preparem un prototip, pensem en com es veuran distribuïts els elements més enllà de la seva integració a la plataforma o el llenguatge que farem servir per a programar-lo. Aquest tipus de prototip serveix per a diverses coses importants:

- L'equip de producció pot comprovar si la proposta s'adapta realment a les necessitats del sistema i les capacitats dels recursos disponibles.

- Permet fer avaluacions d'usabilitat per a corregir problemes abans de passar a nivells avançats de desenvolupament.

D'altra banda, el **prototip funcional o vertical** és una versió més desenvolupada que l'anterior, però sense arribar al resultat final. Hem de saber triar molt bé quines opcions s'han de desenvolupar en aquest prototip, ja que és un mitjà fonamental per a la detecció d'errors en el disseny funcional.

L'usuari pot navegar pel sistema però no pot accedir a tots els continguts, així que ens permet fer avaluacions d'usabilitat per a detectar possibles problemes. També podem presentar-lo al client perquè en aquesta fase encara és factible corregir possibles errades.

1.3.12 Aplicació d'unitat d'estil estètica i narrativa

En aquesta fase es defineixen les característiques gràfiques de la interfície; seguint, amb les variables que escauen a cada cas, els passos següents:

1. Anàlisi del llibre d'estil (si n'hi ha) o de les propostes que hagi aportat el client.
2. Documentació gràfica que tenis per integra-la abans de definir el disseny: fotos, gràfics, il·lustracions...
3. *Benchmarking* d'altres continguts similars o que s'hagin dirigit al mateix tipus d'usuari. És molt important fixar-se en altres produccions per agafar idees o, fins i tot, evitar caure en errades que ja s'han comès prèviament.
4. Gamma cromàtica. Un cop tinguem els aspectes anteriors ja podrem decidir la nostra gamma predeterminada, que s'haurà de respectar durant tot el projecte.
5. Tipografia que, juntament amb la gamma cromàtica, determina el to del projecte.
6. Elecció dels principals elements que configuren el disseny: el fons, els blocs de text i els logotips.
7. Elecció del elements secundaris: les opcions, els textos de contingut, les imatges i altres elements ornamentals.

'Benchmarking'

El *benchmarking* és una tècnica per millorar aspectes d'una empresa. Es basa a analitzar les característiques i funcionament d'altres empreses que són considerades com rellevants al mercat, per aplicar-les a la mateixa empresa.

És important tenir present que aquesta accepció de *benchmarking* com a "comparativa" entre diversos casos és pròpia d'Espanya i no equival del tot al sentit que se li dona en anglès, on el concepte de *benchmark* fa referència a buscar el millor cas o exemple representatiu per tenir-lo com a referent específic (un de sol, i no pas diversos on emmirallar-se).

2. Arquitectures de desenvolupament i de destinació en projectes AMI

Per desenvolupar eficientment projectes d'aplicacions multimèdia interactives (AMI) cal conèixer un seguit de tècniques i eines que faciliten enormement el treball. Si sou capaços i capaces de representar la funcionalitat d'una manera **simple i visual** serà més fàcil treballar-hi en equip. A més a més, l'èxit d'un projecte AMI depèn en gran mesura tant de l'**entorn de desenvolupament** escollit com de l'equip encarregat de produir el producte. Finalment, com en qualsevol altre entorn, cal tenir en compte la **seguretat** de tot el desenvolupament.

2.1 Representacions de la capacitat i funcionament del sistema

Reflexioneu un moment sobre algun cop en què heu volgut explicar un concepte complex a un grup de persones. Segur que **tenir material visual de suport us ha resultat fonamental** per poder transmetre la idea. Tot i així és molt probable que us hagi calgut respondre a alguna pregunta o que algun concepte no hagi quedat clar a algú. Aquest, de fet, és el problema més comú a l'hora de **treballar en equip**, o d'interpretar el que un client vol exactament. Hi ha un llenguatge, però, que permet posar en comú totes les idees d'una manera metòdica i tècnica, per evitar problemes de malentesos. Aquest és el **llenguatge unificat de modelatge (UML)**.

2.2 Modelatge de sistemes: eines, tècniques i procediments

El modelatge de sistemes és un conjunt d'eines, tècniques i procediments que s'utilitzen per descriure els processos d'un sistema, per exemple un programari informàtic, una aplicació interactiva, o fins i tot una màquina. Com que els processos poden ser complexes i amb moltes variables, el modelatge ens permet analitzar i tractar tota la complexitat del sistema. L'objectiu és crear un model on estiguin descrites els diferents aspectes, com són la interacció de l'usuari amb l'aplicació, quins són els esdeveniments i quines accions es produeixen, quina és la relació entre aquestes accions. Per construir el model s'utilitzen eines visuals (diagrames), que mostren de manera molt clara aquests conceptes que són abstractes.

El **llenguatge unificat de modelatge (UML)** és un llenguatge visual per especificar, construir i documentar els artefactes dels sistemes. A la taula 2.1 trobareu el vocabulari vinculat a l'UML.

TAULA 2.1. Glossari de termes comuns al vocabulari d'UML

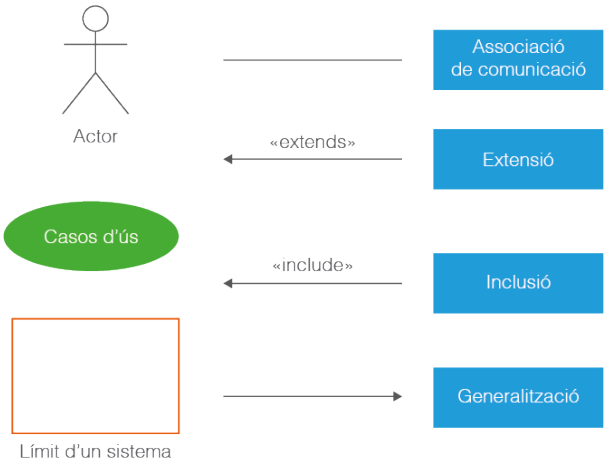
Vocabulari d'UML	Descripció
Compatibilitat amb sintaxi abstracta	Els usuaris poden moure models a través de diferents eines, fins i tot si usen diferents notacions.
Metamodel de magatzem comú (CWM)	Interfícies estàndards que s'usen per permetre l'intercanvi de metadades de magatzem i intel·ligència de negocis entre eines de magatzem, plataformes de magatzem i repositoris de metadades de magatzem en entorns heterogenis distribuïts.
Compatibilitat amb sintaxi concreta	Els usuaris poden continuar usant una notació amb la qual estiguin familiaritzats a través de diferents eines.
Nucli	En el context d'UML, el nucli comunament es refereix al "paquet central", que és un metamodel complet particularment dissenyat per a una alta reutilització.
Unitat de llenguatge	Consisteix en una col·lecció de conceptes de modelatge estretament vinculats que proporciona als usuaris la capacitat de representar aspectes del sistema en estudi segons un paradigma o formalisme en particular. Consisteix en una col·lecció de conceptes de modelatge estretament vinculats que proporciona als usuaris la capacitat de representar aspectes del sistema en estudi segons un paradigma o formalisme en particular.
Nivell 0 (L0)	Nivell de compliment inferior per a la infraestructura UML. Una sola unitat de llenguatge que fa possible el modelatge de tipus d'estructures basades en classes que es troben en els llenguatges més populars de programació orientats a objectes.
Meta Object Facility (MOF)	Una especificació de modelatge d'OMG que brinda la base per a les definicions de metamodels en la família de llenguatges MDA d'OMG.
Metamodel	Defineix el llenguatge i els processos a partir dels quals formar un model.
Construccions de metamodels (LM)	Segon nivell de compliment en la infraestructura UML. Una unitat addicional de llenguatge per a estructures més avançades basades en classes, usades per construir metamodels (per mitjà de CMOF), com ara el mateix UML mateix. L'UML solament té dos nivells de compliment.
Arquitectura dirigida per models (MDA)	Un enfocament i un pla per aconseguir un conjunt coherent d'especificacions de tecnologia dirigida per models.
Llenguatge de restriccions per a objectes (OCL)	Un llenguatge declaratiu per descriure regles que s'apliquen al llenguatge unificat de modelatge. L'OCL complementa l'UML proporcionant termes i símbols de diagrames de flux que són més precisos que el llenguatge natural, però menys difícils de dominar que les matemàtiques.
Object Management Group (OMG)	És un consorci sense finalitats de lucre d'especificacions per a la indústria de la computació, els membres de la qual defineixen i mantenen l'especificació UML.
UML 1	Primera versió del llenguatge unificat de modelatge.
XMI	Una especificació basada en XML de formats d'intercanvi de models corresponents.

2.2.1 Diagramació, nivells apropiats de detall

El llenguatge unificat de modelatge (UML) va ser creat per forjar un llenguatge de modelatge visual comú i semànticament i sintàcticament ric per a l'arquitectura, el disseny i la implementació de sistemes de programari complexos, tant en estructura com en comportament. Representa la **notació estàndard** i semàntica essencial per al modelatge de sistemes. L'UML té aplicacions més enllà del desenvolupament de programari, per exemple, en el flux de processos en la fabricació.

L'UML es pot usar per modelar diferents tipus de sistemes: sistemes de programari, sistemes de maquinari i organitzacions del món real. Usa elements i els associa de diferents maneres per formar diagrames que representen aspectes estàtics o **estructurals** d'un sistema, i diagrames **de comportament** que capten els aspectes dinàmics d'un sistema (vegeu la figura 2.1).

FIGURA 2.1. Elements d'un diagrama de casos d'ús



Els diagrames UML estructurals són:

- **Diagrama de classes**: és el diagrama UML més comunament usat, i la base principal de tota solució orientada a objectes. És un tipus de diagrama estàtic que descriu l'estructura d'un sistema mostrant les seves classes, atributs i les relacions entre ells. Les classes s'agrupen per crear diagrames de classes en crear diagrames de sistemes grans.
- **Diagrama de components**: mostra la relació estructural dels elements del sistema de programari, molt freqüentment empleats en treballar amb sistemes complexos amb components múltiples. Els components es comuniquen per mitjà d'interfícies.
- **Diagrama d'estructura composta**: els diagrames d'estructura composta s'usen per mostrar l'estructura interna d'una classe.
- **Diagrama d'implementació**: il·lustra el maquinari del sistema i el seu programari. És útil quan s'implementa una solució de programari en múltiples màquines amb configuracions úniques.
- **Diagrama d'objectes**: mostra la relació entre objectes per mitjà d'exemples del món real i il·lustra com es veurà un sistema en un moment donat. Atès que les dades estan disponibles dins dels objectes, aquests poden usar-se per aclarir relacions entre objectes.
- **Diagrama de paquets**: hi ha dos tipus especials de dependències que es defineixen entre paquets: la importació de paquets i la fusió de paquets. Els paquets poden representar els diferents nivells d'un sistema per revelar l'arquitectura. Es poden marcar les dependències de paquets per mostrar el mecanisme de comunicació entre nivells.

I entre els diagrames UML de comportament, tenim:

- **Diagrames d'activitats**: fluxos de treball de negocis o operatius representats gràficament per mostrar l'activitat d'alguna part o component del sistema. Els diagrames d'activitats s'usen com una alternativa als diagrames de màquina d'estats.

- **Diagrama de comunicació**: similar als diagrames de seqüència, però s'enfoca en els missatges que es passen entre objectes. La mateixa informació es pot representar usant un diagrama de seqüència i objectes diferents.
- **Diagrama de panorama d'interaccions**: hi ha set tipus de diagrames d'interaccions. Aquest diagrama mostra la seqüència en la qual actuen.
- **Diagrama de seqüència**: mostra com els objectes interactuen entre si i l'ordre de l'ocurrència. Representen interaccions per a un escenari concret.
- **Diagrama de màquina d'estats**: similar als diagrames d'activitats, descriuen el comportament d'objectes que es comporten de diverses maneres en el seu estat actual.
- **Diagrama de temporització**: igual que en els diagrames de seqüència, es representa el comportament dels objectes en un període de temps donat. Si hi ha un sol objecte, el diagrama és simple. Si hi ha més d'un objecte, les interaccions dels objectes es mostren durant aquest període de temps particular.
- **Diagrama de cas d'ús**: representa una funcionalitat particular d'un sistema. Es crea per il·lustrar com es relacionen les funcionalitats amb els seus controladors (actors) interns/externs.

2.2.2 Modelatge de requisits des de la perspectiva de l'usuari

En el llenguatge unificat de modelatge (UML), un **esquema de cas d'ús** pot resumir els detalls dels usuaris del vostre sistema (actors) i les seves interaccions amb el sistema. Per construir-ne un, s'utilitza un conjunt específic de símbols i connectors. Un esquema de cas d'ús eficaç pot ajudar l'equip a parlar i debatre sobre:

- Escenaris on el sistema o l'aplicació interacciona amb persones, organitzacions o elements externs.
- Punts forts del sistema o l'aplicació que ajuden els actors a aconseguir fites.
- L'abast del vostre sistema.

Així, quan s'han **d'aplicar els diagrames de casos d'ús**? Un diagrama de casos d'ús no entra en gaire detall; per exemple, no esperem que modifiqui l'ordre en què es realitzen els passos. En canvi, un diagrama de casos d'ús adequat mostra una visió d'alt nivell de la relació entre els casos d'ús, els actors i els sistemes. Els experts recomanen que s'utilitzin diagrames de casos per completar un cas d'ús textual més descriptiu.

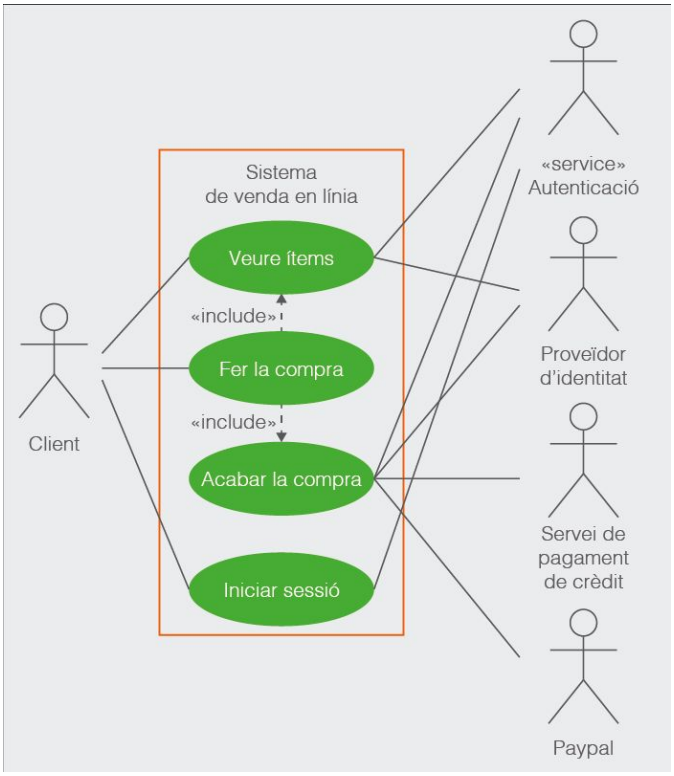
L'UML és el ***kit*** **d'eines** de modelatge que podem utilitzar per crear els diagrames.

Els casos d'ús es representen amb una forma ovalada etiquetada. Les figures de pal representen actors en el procés, i la participació de l'actor en el sistema es modela amb una línia entre l'actor i el cas d'ús. Per representar el límit del sistema, dibuixa un quadre al voltant de l'ús del cas en si.

Els **diagrames de casos** d'ús UML són ideals per a (vegeu la figura 2.2):

- Representar els objectius de les interaccions entre el sistema i l'usuari.
- Definir i organitzar requisits funcionals en un sistema.
- Especificar el context i els requisits d'un sistema.
- Modelatge del flux bàsic d'esdeveniments en un cas d'ús.

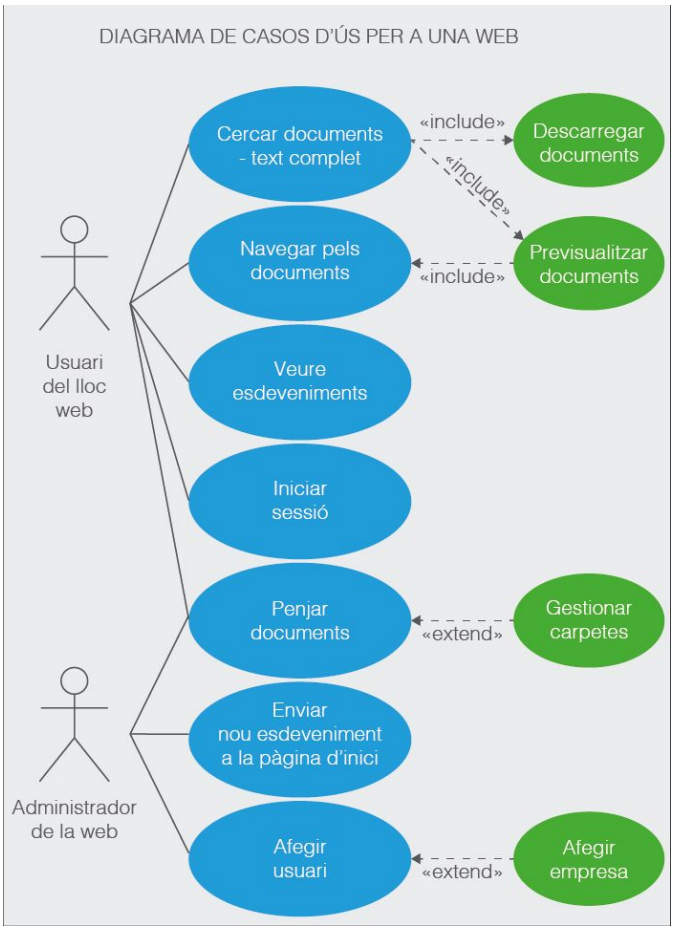
Figura 2.2. Diagrama de casos d'ús per a una botiga 'online'



Per respondre la pregunta "què és un diagrama de casos d'ús?", primer hem d'entendre els blocs de construcció i com utilitzar els components del diagrama de casos. Els components comuns inclouen (vegeu la figura 2.3):

- **Actors**: els usuaris que interactuen amb un sistema. Un actor pot ser una persona, una organització o un sistema extern que interactuï amb la vostra aplicació o sistema. Han de ser objectes externs que produeixen o consumeixen dades.
- **Sistema**: seqüència específica d'accions i interaccions entre actors i el sistema. Un sistema també es pot denominar escenari.
- **Objectius**: el resultat final de la majoria dels casos d'ús. Un diagrama amb èxit hauria de descriure les activitats i variants utilitzades per assolir l'objectiu.

Figura 2.3. Diagrama de casos d'ús per a una web



Llavors, com cal **utilitzar símbols i notació de diagrama de casos**? La notació per a un diagrama de casos d'ús és bastant senzilla i no implica tants tipus de símbols com en el cas d'altres diagrames UML. A continuació en veurem alguns exemples:

- **Casos d'ús**: ovals en forma horitzontal que representen els diferents usos que un usuari pot tenir.
- **Actors**: figures de pal que representen les persones que realment utilitzen els casos d'ús.
- **Associacions**: una línia entre actors i casos d'ús. En diagrames complexos, és important saber quins actors estan associats amb els casos d'ús.
- **Caixes de límits del sistema**: una casella que estableix un abast del sistema per utilitzar casos. Tots els casos d'ús fora de la caixa es considerarien fora de l'àmbit d'aquest sistema. Per exemple, Psycho Killer està fora de l'abast de les ocupacions en l'exemple de motoserra que es troba a continuació.
- **Paquets**: una forma UML que us permet col·locar diferents elements en grups. Igual que amb els diagrames de components, aquestes agrupacions es representen com a carpetes d'arxius.

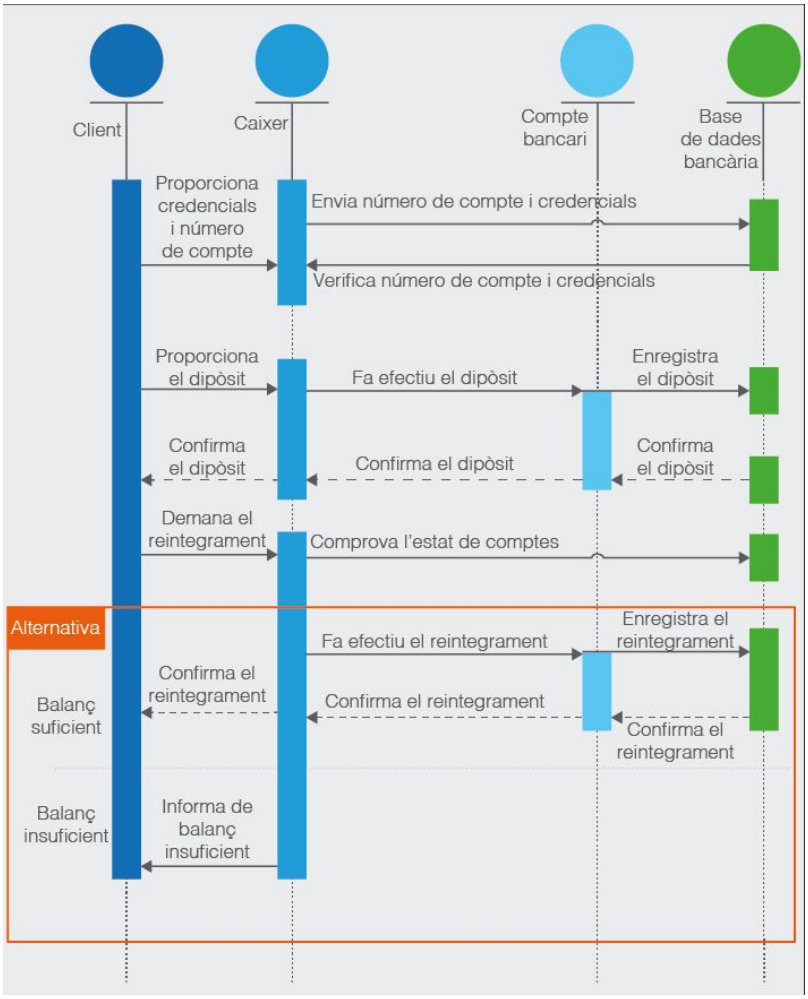
2.2.3 Modelatge de les seqüències dinàmiques d'acció i relacions: diagrames de seqüència

Els **diagrames de seqüència** són una popular solució de modelatge dinàmic. El modelatge dinàmic s'enfoca en les interaccions que tenen lloc dins del sistema. Els diagrames de seqüència s'enfoquen específicament en les "línies de vida" d'un objecte i com es comuniquen amb altres objectes per realitzar una funció abans que la línia de vida acabi.

Per comprendre què és un diagrama de seqüència, és important saber quin és el rol de l'UML. L'UML o el llenguatge unificat de modelatge és un conjunt d'eines de modelatge que dirigeix la creació i notació de molts tipus de diagrames, inclosos els diagrames de comportament, diagrames d'interacció i diagrames d'estructura.

Tant els desenvolupadors de programari com els empresaris usen aquests diagrames per comprendre els requisits d'un sistema nou o documentar un procés existent. Els diagrames de seqüència de vegades es coneixen com a diagrames d'esdeveniments o escenaris d'esdeveniments (vegeu la figura 2.4).

Figura 2.4. Diagrama de seqüència del sistema bancari



Els diagrames de seqüència poden ser diagrames de referència útils per a les empreses i altres organitzacions. Els **usos dels diagrames de seqüència** són:

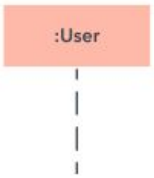
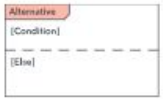
- Representar els detalls d'un cas d'ús en UML.
- Modelar la lògica d'una operació, una funció o un procediment sofisticats.
- Veure com les tasques es mouen entre els objectes o components d'un procés.
- Planificar i comprendre la funcionalitat detallada d'un escenari actual o futur.

Els diagrames de seqüència estan formats pels següents **elements** (vegeu la taula 2.2). A banda, també podem trobar els **símbols de missatges de seqüència UML**, que són paquets d'informació que es transmeten entre els objectes i poden reflectir l'inici i l'execució d'una operació o l'enviament i la recepció d'un senyal (vegeu la taula 2.3).

TAULA 2.2. Elements d'un diagrama de seqüències (I)

Element	Descripció	Imatge
Símbol d'objecte:	Aquesta figura de caixa representa una classe o objecte en UML. Demostra com es comportarà un objecte en el context del sistema. Els atributs de les classes no han d'aparèixer en aquesta figura.	
Casella d'activació:	Simbolitzada amb una figura rectangular, una casella d'activació representa el temps necessari perquè un objecte finalitzi una tasca. Com més temps porti la tasca, més llarga serà la casella d'activació	
Símbol d'actor:	Es mostren amb una figura de vareta. Els actors són entitats que interactuen amb el sistema, però que són externs a aquest.	

Taula 2.2 (continuació)

Element	Descripció	Imatge
Símbol de paquet:	També conegut com a marc, és una figura rectangular que s'usa en la notació UML 2.0 per contenir els elements interactius del diagrama. Aquesta figura conté un petit rectangle interior per etiquetar el diagrama.	
Símbol de línia de vida:	Una línia vertical discontínua que representa el pas del temps a mesura que s'estén cap avall. A més del temps, representa esdeveniments seqüencials que li ocorren a un objecte durant el procés de representació grafica. Les línies de vida poden començar amb una figura rectangular etiquetada o un símbol d'actor.	
Símbol de bucle d'opció:	Una figura rectangular que conté una etiqueta més petita. Aquest símbol s'empra per modelar escenaris del tipus "Si... llavors...", és a dir, una circumstància que solament succeirà en determinades condicions.	
Símbol d'alternatives:	S'usen per simbolitzar una decisió (que, en general, és mútuament exclusiva) entre dues o més seqüències de missatges. Per representar alternatives, empra la figura rectangular etiquetada amb una línia discontínua a l'interior.	

Taula 2.3. Elements d'un diagrama de seqüències (II)

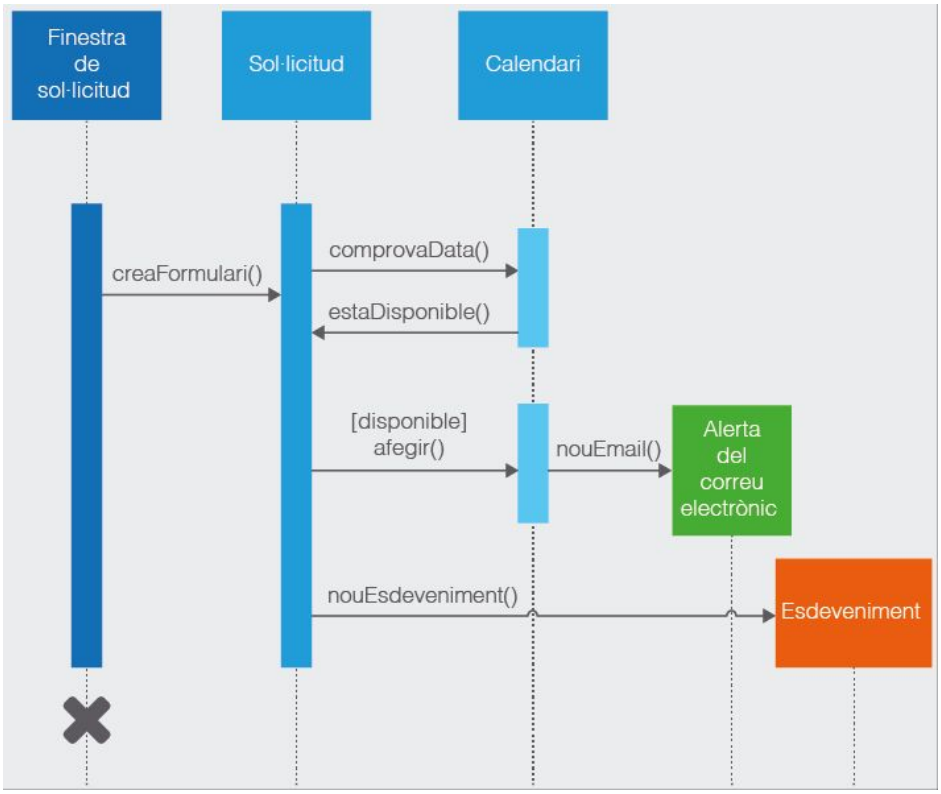
Element	Descripció	Imatge
Símbol de missatge sincrònic:	Representat per una línia contínua i una punta de fletxa sòlida. Aquest símbol s'utilitza quan un remitent ha d'esperar una resposta a un missatge abans de prosseguir. El diagrama ha de mostrar el missatge i la resposta.	
Símbol de missatge asincrònic:	Representat per una línia contínua i una punta de fletxa simple. Els missatges asincrònics són aquells que no necessiten una resposta perquè el remitent avanci. Solament la trucada s'ha d'incloure en el diagrama.	
Símbol de missatge de resposta asincrònic:	Representat per una línia discontínua i una punta de fletxa simple.	
Símbol de crear missatge asincrònic:	Representat per una línia discontínua i una punta de fletxa simple. Aquests missatges s'envien a les línies de vida per crear-se per si sols.	
Símbol de missatge de resposta:	Estan representats amb una línia discontínua i una punta de fletxa simple. Aquests missatges són les respostes a les trucades.	

TAULA 2.3 (continuació)

Element	Descripció	Imatge
Símbol d'eliminar missatge:	Estan representats per una línia contínua i una punta de fletxa sòlida, seguida d'un símbol X. Aquests missatges indiquen la destrucció d'un objecte i estan situats en la seva ruta de la línia de vida.	

La figura 2.5 mostra un **exemple de diagrama de seqüència** que desglossa el sistema de creació i anunci d'un nou esdeveniment en un calendari.

FIGURA 2.5. Diagrama de seqüència del sistema de creació i anunci d'un nou esdeveniment en un calendari



2.2.4 Modelatge del comportament dinàmic d'objectes o classes: diagrames d'estats

Un diagrama d'estats, de vegades conegut com a *diagrama de màquina d'estats*, és un tipus de diagrama de comportament en el llenguatge unificat de modelatge (UML) que s'especialitza a mostrar transicions entre diversos objectes.

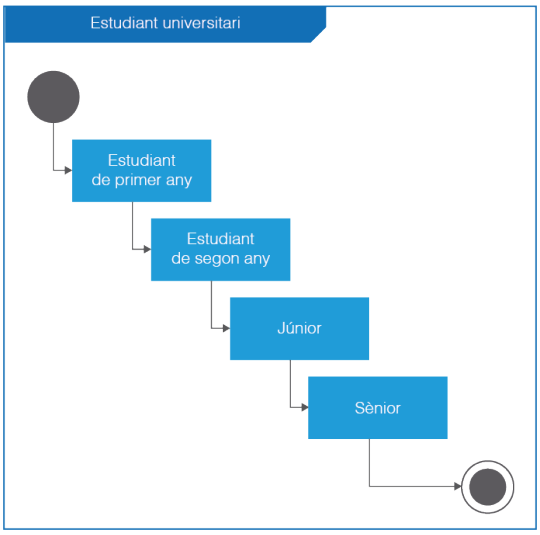
Així, **què és un diagrama d'estats en UML?** Una màquina d'estats és tot el que pugui tenir diferents estats. En molts casos, quan parlem d'estats, parlem dels diferents estats d'un objecte. Els diagrames complexos poden tenir molts estats

diferents. Per entendre millor objectes difícils, de vegades té sentit entendre tots els diferents estats possibles d'un objecte i com arriba l'objecte a aquest estat. Els estats són les diferents combinacions d'informació que pot contenir un objecte i no com es comporten.

Els principals elements que representen els diagrames d'estat són els **estats** i les **transicions**.

Els estats es capten per mitjà de rectangles arrodonits que s'etiqueten amb el nom de l'estat. Les transicions es marquen amb fletxes que flueixen d'un estat a un altre, i mostren com canvien els estats. A la figura 2.6 podeu veure aquests dos elements en acció en un diagrama bàsic.

FIGURA 2.6. Exemple de diagrama d'estats



Les **aplicacions dels diagrames d'estat** són les següents:

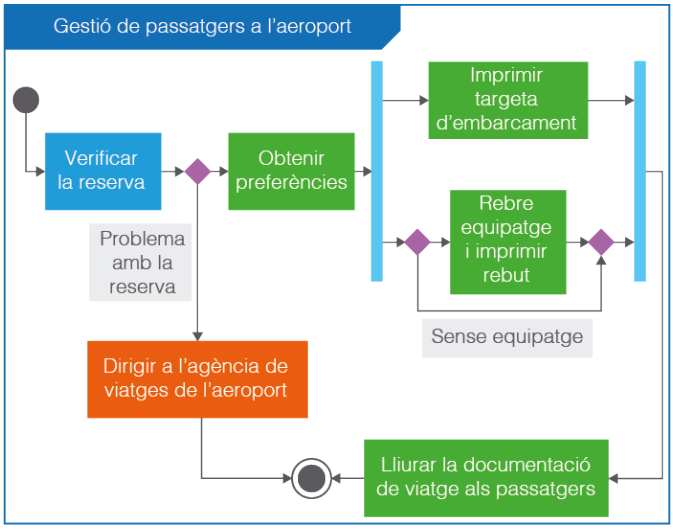
- Representar objectes basats en esdeveniments en un sistema reactiu.
- Il·lustrar escenaris de casos d'ús en un context de negocis.
- Descriure com es mou un objecte a través de diversos estats al llarg de la seva existència.
- Mostrar el comportament general d'una màquina d'estats o el comportament d'un conjunt relacionat de màquines d'estats.

Pel que fa als **components dels diagrames d'estats**, es poden incloure moltes figures diferents en un diagrama d'estats, particularment si tries combinar-ho amb un altre diagrama. Aquesta llista és un resum de les figures més comunes que pots trobar (vegeu la figura 2.7):

1. Estat compost: un estat que conté subestats anidats.
2. Pseudoestat d'opció: un símbol de diamant que indica una condició dinàmica amb resultats potencials ramificats.

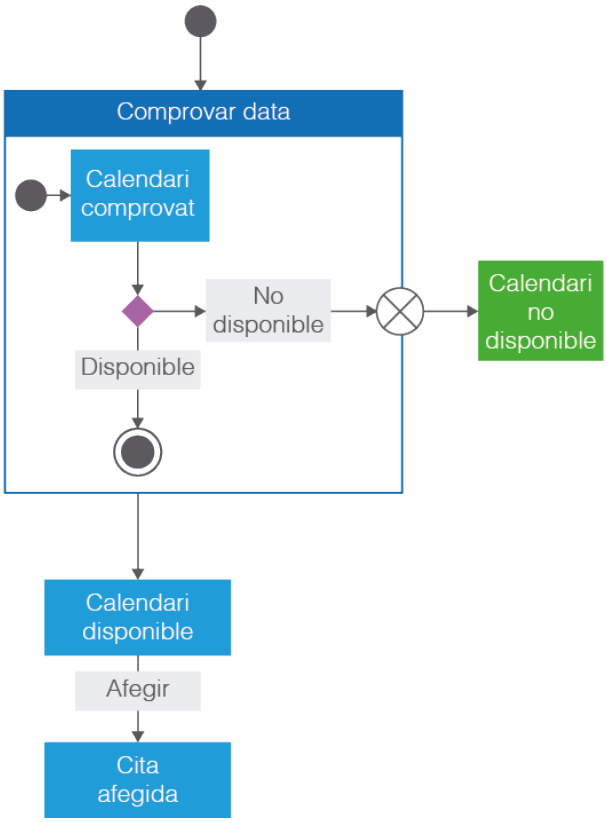
3. Punt de sortida: el punt en el qual se surt d'un estat compost o d'una màquina d'estats. Es representa amb un cercle amb una X a l'interior.
4. Esdeveniment: una instància que activa una transició. S'etiqueta amb nom a dalt de la fletxa de transició aplicable.
5. Estat final: un marcador per al primer estat del procés. Es mostra mitjançant un cercle fosc amb una fletxa de transició.
6. Protecció: una condició booleana que permet o deté una transició. S'escriu a dalt de la fletxa de transició.
7. Estat: un rectangle arrodonit que indica la naturalesa actual d'un objecte.
8. Subestat: un estat contingut dins de la regió d'un estat compost.
9. Transició: una fletxa que corre d'un estat a un altre que indica un estat canviant.
10. Comportament transicional: un tipus de comportament resultant que ocorre durant la transició d'un estat. S'escriu a dalt de la fletxa de transició.
11. Disparador: un tipus de missatge que mou activament un objecte d'estat en estat. S'escriu a dalt de la fletxa de transició.

Figura 2.7. Exemple dels components dels diagrames d'estats



El següent diagrama d'estats (figura 2.8) mostra els diferents estats d'un objecte del calendari, el qual s'ha diagramat previament per mostrar el flux d'algú que defineix un esdeveniment en un calendari.

Figura 2.8. Exemple de diagrama d'estats



La taula 2.4 mostra el significat dels diferents símbols i notacions que podem trobar a un diagrama d'estats.

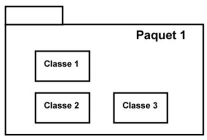
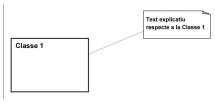
Taula 2.4. Símbols i notació per a diagrames d'estats

Símbol	Significat
	El cercle amb un punt a l'interior significa que un procés està acabat.
	El cercle amb una "X" a l'interior significa que un procés està sent evitat.
	Els quadres d'estat representen els diferents estats en els quals pot estar una màquina durant un procés.
	Un punt negre representa l'inici d'un procés.

2.2.5 Elements d'ajuda, sense valor semàntic, utilitzats en els diagrames

En els diagrames s'utilitzen tres elements d'ajuda que no tenen valor semàntic; ens referim als paquets, les notes i els estereotips (vegeu la taula 2.5).

TAULA 2.5. Elements d'ajuda utilitzats als diagrames

Element d'ajuda	Descripció	Exemple
Paquets:	En algunes ocasions hi ha la necessitat d'organitzar els elements d'un diagrama en un grup. Tal vegada es vol mostrar que certes classes o components són part d'un subsistema en particular. Per fer-ho, es poden agrupar en un paquet, que es representa per una carpeta tabular.	
Notes:	És freqüent que alguna part del diagrama no presenti una explicació clara de per què és allí o la manera en què treballa. Quan això passi, la nota UML serà útil. La nota té una cantonada doblegada i s'adjunta a l'element del diagrama connectant-lo mitjançant una línia puntejada.	
Estereotips:	Alguns sistemes requereixen elements fets a mida que no es troben a l'UML. Per això, els estereotips o clixés permeten prendre elements propis de l'UML i convertir-los en uns altres que s'ajustin a les necessitats. Es representen com un nom entre dos parells de parèntesis angulars.	<<nom>>

2.3 Modelatge de l'estructura estàtica del sistema: el diagrama de classes

Un mètode ben acceptat en el món del desenvolupament de projectes AMI per relacionar les idees i les descripcions del projecte és fer servir un diagrama de classes. Aquesta eina és una de les més importants dins del disseny de *software*, ja que us servirà fonamentalment per a poder treballar en equip i poder **planificar el desenvolupament** d'un projecte. A més a més, estableix d'una manera clara i molt visual la dificultat de programació i quines són les parts crítiques de qualsevol projecte.

> Un **diagrama de classes** és simplement un recull de les classes que interpretem com a necessàries, i les dependències entre elles, donada una definició d'un problema per resoldre o una aplicació per desenvolupar.

Com és d'esperar, no sempre sereu capaços de definir perfectament un diagrama de classes abans de començar. Això és perquè no es pot esperar tenir sempre tota la informació, i tota correcta, des del primer moment. De fet, sovint us trobareu amb **imprevistos** a mig projecte, amb nous requeriments un cop començat el desenvolupament o, simplement, millorareu la comprensió de la tasca a mesura que passi el temps i us endinseu en el projecte.

És per això que, tot i que la construcció del diagrama de classes és un pas fonamental en el disseny de les vostres aplicacions, i que hauria de ser una de les primeres tasques a fer, no us heu de quedar amb un diagrama immutable. En tot cas, un cop tingueu una primera versió d'aquesta eina, una bona pràctica és distribuir la programació dins el vostre equip, assignant una classe per programador i rebre comentaris, per analitzar si cal **introduir millores**.

El primer pas per començar a dissenyar el diagrama de classes és **entendre el problema** profundament. Heu d'assegurar-vos de fer totes les preguntes necessàries i tenir tots els conceptes clars fins al màxim de detall abans de començar a dissenyar el diagrama.

Entendre bé el problema

Diversos científics han dit en més d'una ocasió que, si haguessin d'afrontar un problema, dedicarien el 80% del temps a entendre'l, i el 20% a resoldre'l. Si bé aquests percentatges no han de ser indicatius en absolut, l'esperit d'aquest pensament es manté.

El segon pas és **definir les classes**; per fer-ho, heu de tenir clar el conceptes de classe i instància. Una classe és la definició d'una idea o concepte, mentre que una instància és un exemple concret del concepte. Per exemple, tots tenim al cap què és una cadira. Això seria una classe. És simplement una idea, una definició. D'altra banda, cada dia seiem en multitud de cadires reals diferents; a l'autobús, a l'oficina, a la cuina tenim cadires que existeixen i que són exemples concrets del concepte que tenim al cap; això són instàncies.

Quan dissenyeu un diagrama de classes, heu de pensar únicament en classes i no en instàncies. Un exemple clàssic són **els usuaris**. Us pot interessar guardar perfils d'usuari, dels quals voldreu saber-ne el nom, l'edat, l'adreça electrònica... Quan definiu la classe "usuari", no heu de pensar en un usuari en concret, amb un nom concret o una adreça concreta. De fet, aquestes dades es coneixen com a **atributs** i són tot allò que defineix una classe des del punt de vista del que interessa a l'aplicació.

Per tant, hi ha multitud de possibles atributs per definir un usuari o una persona, però per a una aplicació concreta n'hi pot haver prou coneixent-ne només tres o quatre. Per exemple, per a una aplicació per conèixer a gent a qui agradi la cervesa, dels usuaris potser només en necessiteu saber l'edat, el nom, el gènere, la preferència de cervesa i algunes fotografies. A priori, no us cal informació com ara la religió, el pensament polític, l'estat civil... A mesura que aneu desgranant les classes, podeu anar-ne detallant els atributs. En tot cas, es pot anar detallant iterativament, millorant progressivament el diagrama.

L'*Unified Modeling Language*, o **UML**, és la convenció més acceptada per definir diagrames de classes. En UML les classes es representen amb capses, on el títol de cada capsa és el nom de la classe. A l'hora d'escollir un nom per a una classe, heu de procurar escollir-ne un que sigui clar i concís, que transmeti el concepte de la millor manera possible. Per **convenció**, no s'utilitzen verbs ni adjectius, sinó que cal fer servir substantius. Els atributs es representen també dins de la capsa, a sota del títol. Per als atributs també fem servir noms, i no adjectius o verbs.

'Unified Modeling Language' (UML)

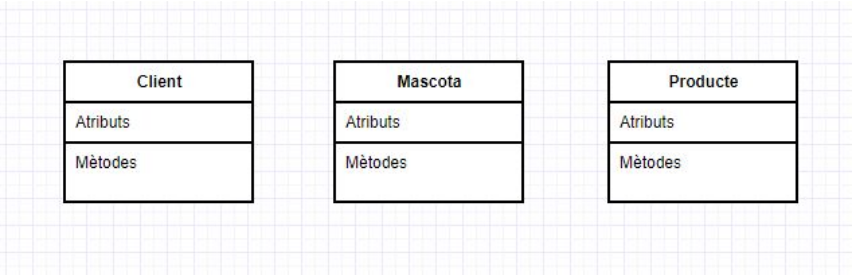
L'UML, o 'llenguatge de modelatge unificat', es va crear als anys noranta combinant diverses tècniques i diversos autors. Consta de diverses versions; l'última és la 2.0, que es va publicar l'any 2005.

Exemple de diagrama de classes

Imagineu que voleu fer un projecte per a desenvolupar una botiga virtual per a mascotes. Això sí, voleu poder oferir un servei de qualitat, i no només vendre els productes, sinó enregistrar informació sobre els clients i sobre les seves mascotes, per poder fer suggeriments de productes complementaris.

Analitzem ara les entitats fonamentals que apareixen en aquest exemple. Hem parlat de

mascotes, de clients i de productes. Aquestes seran les classes del vostre diagrama per a una botiga virtual, que, provisionalment, quedarà com a la imatge següent:



Com veieu, el títol de les capses correspon al nom de la classe, que és una paraula clara i concreta. Segueix la convenció que sigui un substantiu en singular.

Per poder estar segurs que esteu definint classes, podeu fer-vos algunes **preguntes de comprovació**; per exemple, esteu visualitzant un concepte, o més aviat és un exemple concret? Us podeu trobar amb ocurrències diferents del mateix concepte? És a dir, si, per exemple, definim la classe "client", us podeu trobar amb dos o més clients diferents que compleixin la definició de client. Si no queda clar que és una classe, probablement heu de repensar i repassar la definició del problema per comprovar-ho.

2.3.1 Diferents tipus d'atributs

Davant d'un diagrama de classes, a l'hora de visualitzar les classes, a més del títol, observeu que cada capsa té dos apartats més, que són per als **atributs** i els **mètodes**.

Pel que fa als atributs, us podeu trobar amb multitud d'informació que pot descriure una classe. Per exemple, un cotxe es pot descriure per la seva potència en CV, pel seu pes en kg, el seu preu en € i el seu consum en l/km. Tot i ser unitats diferents, des del punt de vista d'un diagrama de classe, aquests atributs són del mateix tipus: números. La notació per als atributs també és simple; es posen dins de la segona capsa, primer el nom de l'atribut i després el tipus, en anglès. Els atributs poden ser:

- "Integer" (números enters)
- "String" (cadenes de caràcters)
- "Boolean" (atributs lògics que poden ser només o cert o fals)
- "Float" (números amb decimals)

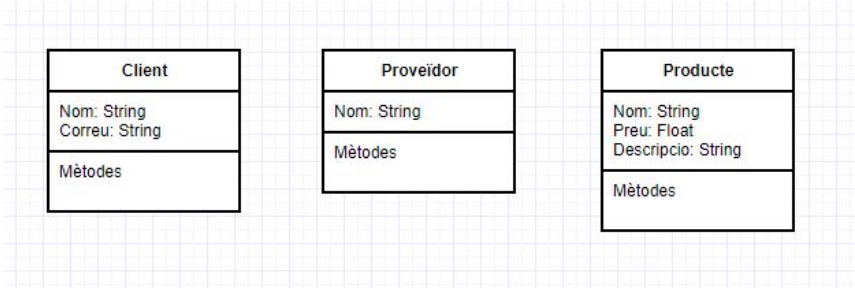
Lògicament, es pot donar el cas que aquests tipus no siguin suficients. Un exemple ben habitual són les **dates**. Tan vàlid és dir 12/10/1984 com 12 d'octubre de 1984, o d'altres maneres de representar la mateixa data. D'altra banda, no totes les combinacions amb característiques similars formen dates correctes. Per exemple, 35 d'octubre de 1984 no és una data correcta.

L'UML ens permet definir els nostres propis **tipus d'atributs**; tants com vulguem. Això sí, han de ser possibles de formar a partir de combinacions dels atributs bàsics descrits.

Per poder saber que realment esteu identificant un atribut, us heu de fer algunes **preguntes de comprovació**, per exemple, saber si defineix d'alguna manera la classe, i si és necessari donat el context del nostre problema. No volem afegir mai complexitat innecessària a un projecte, per tant, ens hem de restringir només a explicitar els atributs que siguin rellevants per al projecte. D'altra banda, si veiem que allò que estem dissenyant no formaria part d'una definició de la classe, pot ser que no sigui un atribut, sinó que sigui un altre element, com ara una classe diferent o una relació.

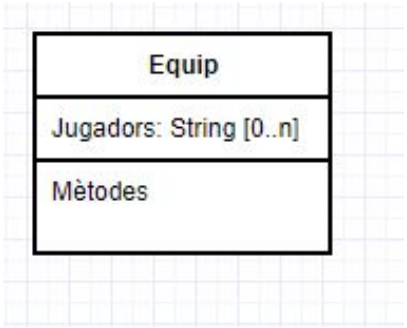
Exemple de creació d'atributs

Imagineu-vos que teniu una petita empresa que distribueix productes de diferents fabricants. Per tant, les classes que tindreu seran "client", "producte" i "fabricant". En aquest cas, a priori no teniu prou informació sobre els atributs que necessiteu. Podeu fer una mica d'investigació de camp preguntant a altres empreses que tinguin programaris similars, i us trobareu que típicament es vol guardar el nom i el correu dels clients. A banda, també voleu guardar el nom dels proveïdors, el nom de cada producte, el seu preu i una descripció. El resultat ens donaria una imatge similar a la següent, amb els atributs necessaris per al projecte:



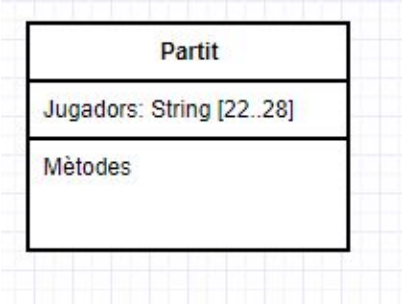
Ben sovint us trobareu amb classes que tenen un atribut que és una **llista d'atributs**; per exemple, un equip d'esports pot tenir un nombre indeterminat de jugadors. La notació en aquest cas és: indicar el tipus, seguit del símbol [, el mínim d'elements de la llista, dos punts seguits i el màxim, seguit del símbol]. A la figura 2.9, podeu veure el cas d'un equip que pot estar buit i en principi no tenir un màxim de jugadors.

Figura 2.9. Atribut del tipus llista



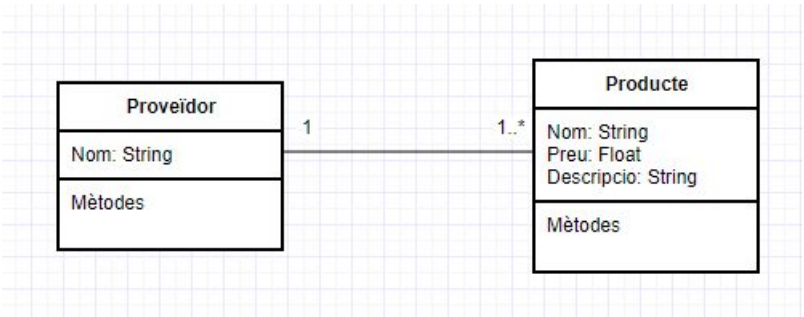
D'aquesta manera, si esteu definint un equip d'un esport concret, amb unes regles de mínims i màxims, també ho podreu fer. Per exemple, un partit de futbol onze ha de jugar amb una alineació inicial d'onze jugadors per equip; a més a més, cada equip pot fer tres canvis. Per tant, en un partit hi poden jugar des de 22 jugadors fins a 28. Com a resultat, afegiríem un atribut **de mida variable** amb un rang definit, tal com mostra la figura 2.10.

FIGURA 2.10. Atribut llista de mida variable amb un rang definit



D'altra banda, us podeu trobar que voleu que una classe faci referència a una altra classe. Així, les **relacions entre classes** són un cas diferent dels atributs; perquè justament la informació ja està representada al diagrama de classes amb una classe i, per tant, afegir-ho com a atribut seria redundant. Per exemple, imagineu un projecte on teniu proveïdors que fabriquen diversos productes, i voleu saber el nom del proveïdor, quins productes fabrica, el nom dels productes, el preu i una petita descripció. Aquesta relació se simbolitza unint les dues classes amb una línia, com podeu veure a la figura 2.11.

FIGURA 2.11. Relacions entre classes



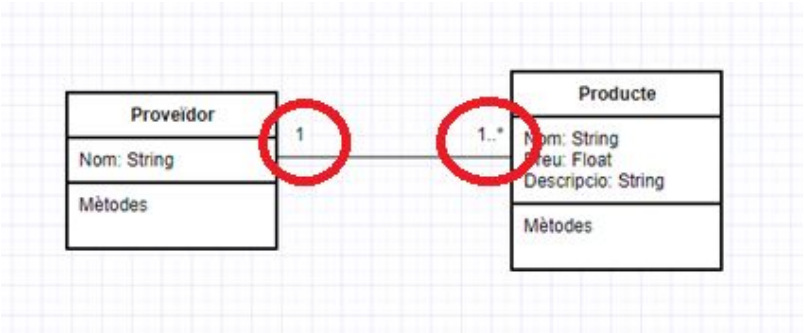
Continuant amb aquest exemple de relació, assumiu que després de fer diverses comprovacions us trobeu amb les següents premisses: un proveïdor ha de subministrar com a mínim un producte, i un màxim indefinit de productes. Si un proveïdor no us ofereix cap producte, no té sentit guardar la seva informació, perquè no està aportant res al vostre negoci. D'altra banda, no té sentit en general que un mateix producte l'ofereixin diversos proveïdors. Per tant, podeu assumir que cada producte és subministrat per un i només un proveïdor.

Aquestes restriccions s'anomenen **cardinalitat de les relacions**. Per representar-les es fan servir unes indicacions al voltant dels extrems de la línia d'associació, a prop de les capses de les classes. Es llegeix de la següent manera:

- Una instància de la classe que està lluny de l'indicador pot estar relacionada amb un mínim i un màxim d'instàncies, expressades a l'indicador.
- El mínim i el màxim estan separats per dos punts i seguit i, si són el mateix valor, s'indica amb un únic número.
- D'altra banda, si no és un número definit, s'indica amb un asterisc, i si no hi ha cap restricció (és a dir, pot ser 0 o més) s'indica amb un zero, dos punts seguits i un asterisc.

A l'exemple de la figura 2.12 veieu com un proveïdor pot tenir un o més productes associats, mentre que un producte ha de tenir necessàriament un proveïdor.

Figura 2.12. Cardinalitats de relacions entre classes

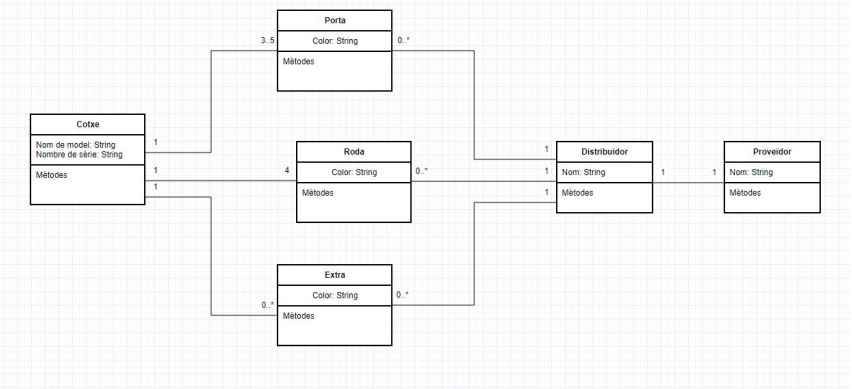


És important tenir clara la cardinalitat de les relacions per diverses raons. Una d'elles és per claredat i llegibilitat, especialment a mesura que el projecte creix en complexitat. Quan tingueu diagrames de classes amb més de vint classes, conèixer la cardinalitat us facilitarà entendre la imatge global. També és important perquè té un impacte clar a l'hora d'**implementar el projecte**. Si una relació té cardinalitat d'1, us heu d'assegurar que no es pot instanciar una classe sense validar aquesta relació. Per tant, després de definir que dues o més classes es relacionen, heu de definir la cardinalitat de la relació pensant quantes instàncies com a màxim i com a mínim es relacionen.

El terme *instanciar* és un terme tècnic que ve de l'anglès *to instantiate*, i vol dir 'crear un exemple de' o 'crear una mostra concreta de'.

Exemple de diferents tipus de cardinalitats

Per veure un exemple amb diversos tipus de cardinalitats podeu imaginar-vos el disseny de classes que representa la cadena de muntatge d'un cotxe amb diverses versions. Els cotxes poden tenir tres o cinc portes, tindran sempre quatre rodes, i poden venir amb extres o no. Cada peça, portes, rodes o extres, arriba a la cadena de muntatge a través d'un distribuïdor, que a la vegada treballa amb un únic proveïdor. A més a més, sabeu que cada proveïdor treballa amb un únic distribuïdor. L'esquema resultant seria el de la imatge següent:



Definir els mètodes

A més dels atributs, les classes també poden tenir mètodes. Un mètode és tota aquella operació d'una classe que pot ser útil per a una altra classe. Serveix per descriure d'una manera simplificada la **funcionalitat de la classe**, amb les entrades i sortides que tindrà cada mètode.

Per exemple, si des de la classe "client" voleu accedir a informació de preu de la classe "producte", heu d'escriure un mètode dins de la classe producte que retorni el preu. D'aquesta manera, no només podreu accedir a la informació des de la classe "client", sinó des de qualsevol classe.

Com que són operacions, la **convenció** és que els noms dels mètodes siguin verbs. Els casos més típics són obtenir valors i guardar valors, i la convenció per a aquests casos és fer servir els verbs anglesos *get* (obtenir) i *set* (guardar), respectivament. A més a més, la gran majoria de llenguatges de programació no accepten espais entre paraules per definir el nom d'un mètode, per tant per assenyalar que són dues paraules diferents feu servir les majúscules.

Per exemple, per guardar un nom, afegireu un mètode "setNom", i per obtenir aquest mateix nom, el mètode es dirà "getNom". També hi poden haver mètodes amb altres **finalitats** molt diferents, que no tinguin a veure amb obtenir o guardar informació. Per exemple, imagineu una situació on un càlcul sigui molt complex, i voleu executar-lo només quan sigui necessari. En aquest cas, definireu un mètode "executarCalcul", i el fareu servir quan convingui.

D'altra banda, heu d'explicitar quins són els **tipus de dades** que necessita el vostre mètode, o dades d'entrada, i quin tipus de dades retorna. Per fer-ho, la convenció és posar les dades d'entrada entre parèntesi, i el tipus de dades després de ':'. Per al retorn, també escriureu ':' i el tipus, després del parèntesi. Per exemple, un mètode per calcular una multiplicació serà: "multiplicar(a:Float, b:Float) : Float". D'aquesta manera, sabreu quin tipus de dades heu de passar al mètode, i quin tipus de dades ens retornarà.

Els mètodes poden tenir un tipus de dades "Void". Aquest tipus vol dir que és buit, que o bé no necessita cap paràmetre o bé no retorna res. Es fa servir per deixar clar que no és que faltin paràmetres o hi hagi algun error, sinó que el mètode no té o bé paràmetres o bé valor de retorn, o cap d'aquests.

És cert que això no sempre conforma una informació completa. Per exemple, un mètode per a calcular una arrel quadrada hauria de controlar que no tingués una entrada negativa, perquè el càlcul no seria possible. La manera de resoldre-ho formalment és prou complexa, i se surt de l'abast d'aquest curs, tot i que és possible. De totes maneres, no se sol fer servir a la pràctica, ja que comporta un esforç que en moltes ocasions iguala o supera el de fer la implementació directament.

Lògicament, això no ens interessa, ja que l'objectiu principal del diagrama de classes és esquematitzar amb un nivell d'abstracció prou alt el disseny d'un projecte de *software*, abans de començar a implementar. La solució en aquest cas, de manera pràctica, seria escriure noms de mètodes prou entenedors per fer explícita la funcionalitat, i escriure les possibles restriccions informalment en comentaris.

En resum, els **diagrames de classes** són esquemes visuals que es construeixen a partir dels requeriments del problema que cal resoldre. Aquests requeriments són identificar correctament les classes i les seves associacions, a més dels atributs i mètodes de cada classe.

2.3.2 Modelatge dels detalls concrets de la implementació del sistema: diagrames de classes i components

Els diagrames de classes són una eina fonamental a l'hora de dissenyar programari; us permet prendre decisions i començar a veure l'arquitectura del projecte de programari abans de començar la implementació. A més a més, com que és una eina visual, és fàcil de compartir en un equip i iterar sobre el disseny. Finalment, resol d'una manera senzilla una primera aproximació al **repartiment de tasques**, perquè les classes més complexes es veuen a simple vista, i es poden assignar a individus o petits equips.

El terme iterar és molt comú en el desenvolupament de projectes, i es refereix a acabar una versió (pot ser incompleta) que tingui sentit com a entitat, i tornar a començar per millorar-la o afegir-hi funcionalitat.

Tenir tota la casuística possible resolta en un diagrama de classes és complex. Hi ha una gran variabilitat de possibilitats, i preveure tots els casos amb un nombre reduït d'eines és molt complicat; per això heu de tenir-les clares quan treballeu en **escenaris més complexos**. És a dir, només tenint clar com es relacionen dues classes i com es simbolitza un diagrama de classes, podeu veure i entendre casuístiques més elaborades: les relacions ternàries o múltiples i les herències.

Relacions ternàries

En primer lloc, per distingir bé els casos, hem de definir què són les relacions binàries. Una **relació binària** és aquella que involucra únicament dues classes; es diferencia d'un atribut perquè és una entitat que necessita la seva pròpia classe. En el cas, però, que vulgueu relacionar tres classes, us trobeu amb una **relació ternària**. Si relaciona quatre o més classes, serà una **relació múltiple**. Aquests tipus de relacions són importants i tenen una casuística pròpia, i una convenció també per a representar-les dins del diagrama de classe.

Les relacions ternàries són ben comunes. Molts cops us les trobareu com una manera d'enregistrar ocurrències d'una relació. Per exemple, quan vulgueu enregistrar una compra, típicament s'enllaça un client, un producte i una factura. D'aquesta manera, en el vostre diagrama de classes heu de poder expressar la cardinalitat de la relació, és a dir, indicar que un client i múltiples productes s'assignen a una factura. Per fer-ho, teniu les mateixes eines que en una relació binària, concretament els números al voltant de les classes. Finalment, una relació ternària s'indica amb un rombe que uneix línies cap a cada classe. A més a més, s'hi pot posar també una petita descripció d'un mot.

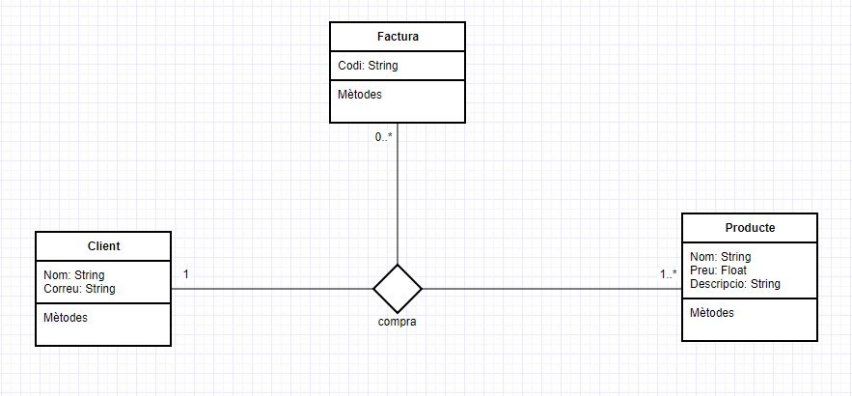
Exemple de relació ternària

El següent exemple es tracta d'una relació ternària que indica que un client pot comprar diversos productes, i això resulta en una factura.

A més d'identificar les classes, que són "factura", "producte" i "client", heu d'observar que tenen una relació ternària. Al diagrama de classes hi heu d'afegir les unions, i encara quedarà per determinar la cardinalitat. Per determinar la d'una classe, us heu de preguntar quantes instàncies d'aquella classe poden existir per cada instància de les altres dues; és a dir:

- donat un client i un producte, quantes factures hi pot haver;
- donat un client i una factura, quants productes hi pot haver, i
- donada una factura i un producte, quants clients hi pot haver.

Per lògica, us trobeu que donat un producte i un client, podeu tenir 0 o més factures, ja que un client pot comprar més d'una vegada un mateix producte, o no comprar-lo. Això ens indica que la cardinalitat a la banda de les factures és de "0..*". De la mateixa manera, donat un client i una factura, hi poden haver múltiples productes, tot i que com a mínim en tindreu 1, ja que no té sentit que es faci factura per no comprar cap producte. Per tant, la cardinalitat de la banda del producte serà "1..*". Finalment, donada una factura i un producte, només es pot tenir un client i, per tant, la cardinalitat de "client" serà d'1. Com veieu a la següent imatge, aquesta relació ternària conforma un petit diagrama de classes:



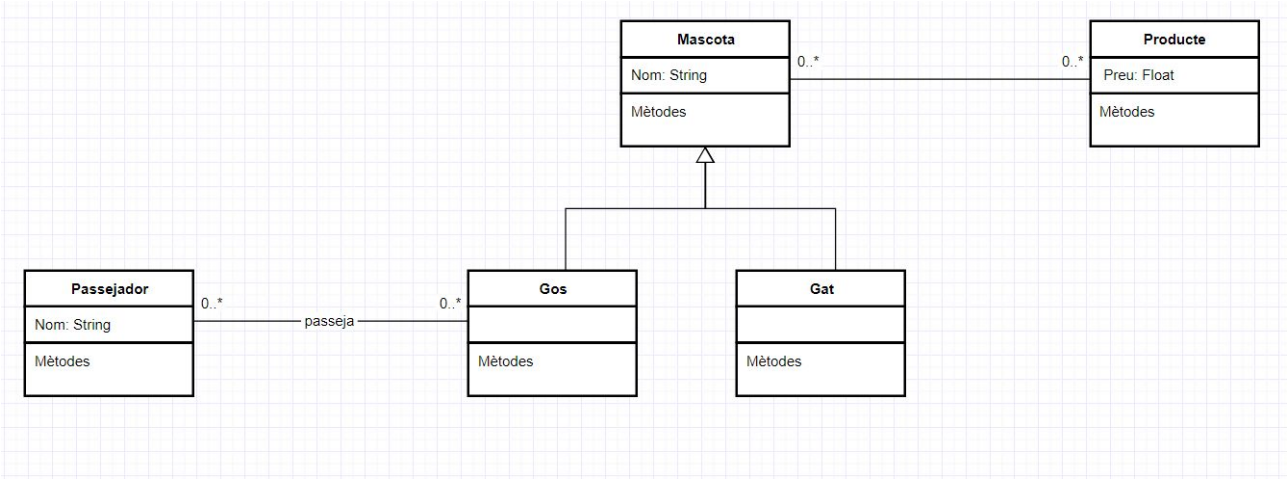
Les herències

Les herències són un altre tipus de relació en escenaris complexos. En moltes ocasions us podeu trobar amb classes que formen part d'un mateix concepte; per exemple, un gos i un gat poden ser dues classes diferents, però es poden abstreure en una classe mascota; més enllà, qualsevol mascota és un ésser viu. Tot sovint us trobareu que volem tenir atributs, mètodes i relacions que són **comunes a dues o més classes**, i per tant us convindrà agrupar-les per reduir esforç i possibles errors per redundància. Aquesta és justament la missió de les herències.

La manera de **representar una herència** és mitjançant una línia que connecti les classes, però on un dels extrems és un triangle. Aquest triangle se situa a la classe més global, i les classes més concretes simplement se situen en la línia, com qualsevol altra relació. En aquest cas, no cal indicar la cardinalitat perquè no és una relació entre dues classes diferents, sinó una indicació que és el mateix concepte, on hi ha algunes particularitats. Per tant, no té sentit parlar de cardinalitat.

Seguint amb l'exemple de les mascotes, podeu definir un diagrama per a un cas on tinguem un passejador de gossos, productes per a mascotes, i gossos i gats. Els productes per a mascotes afecten tant gats com gossos, i tant gats com gossos tenen un nom; per tant, es pot abstreure. D'altra banda, els passejadors de gossos no treballen amb gats, per tant, no es poden relacionar amb la classe "mascota", sinó directament amb la classe "gos". Com veieu, no hi ha cap problema per fer associacions, tant amb la classe genèrica com amb les més específiques (vegeu la figura 2.13).

FIGURA 2.13. Exemple d'herència "gos" i "gat" de la classe "mascota".



A l'hora de referir-se a cada classe en una herència, es poden expressar com classes pare i fills, per a les classes abstractes i concretes, respectivament. També es pot anomenar superclasse la classe pare, i subclasse la classe fill. Les classes del mateix nivell es defineixen com germanes, o *siblings* en anglès. No hi ha un límit de nivells ni un límit de fills per nivell; podeu tenir capes d'abstraccions il·limitadament. És a dir, no hi ha cap restricció per tenir una classe que tingui un nombre molt elevat de fills, on cada fill tingui al seu torn un nombre elevat de fills, i així successivament. Això sí, heu de procurar sempre que tot tingui coherència i no abusar del detall innecessàriament. Recordeu que si voleu que el diagrama sigui útil, heu d'incloure-hi només la informació rellevant per cada problema en tot moment.

Pràctica amb diagrames de classes

Tot i que la teoria d'UML és molt més extensa, i abasta molts més casos, n'hi ha prou amb **conèixer la base** perquè sigui útil i per poder-la fer servir com una eina professional. Sabent com es planteja un diagrama de classes, quins elements formen una classe, com relacionar les classes i descriure detalls com la cardinalitat, ja teniu un coneixement suficient per a molts projectes. El problema pot ser posar-ho en pràctica. Per passar aquesta dificultat, l'única manera és practicant a base d'exemples. Per això, practicarem fer diagrames de classes amb exemples en profunditat.

Imagineu que heu de desenvolupar un projecte per a una empresa de màrqueting anomenada Mar. Per a Mar és fonamental conèixer:

- El nom de les empreses amb què treballen i també el CIF, per poder facturar.
- Com que es dediquen al màrqueting, els treballadors de Mar necessiten tenir informació sobre els consumidors dels clients amb qui treballen, per poder ajustar les ofertes apropiadament. En concret, necessiten tenir el telèfon, l'edat i el nom dels consumidors, per fer entrevistes i enquestes.

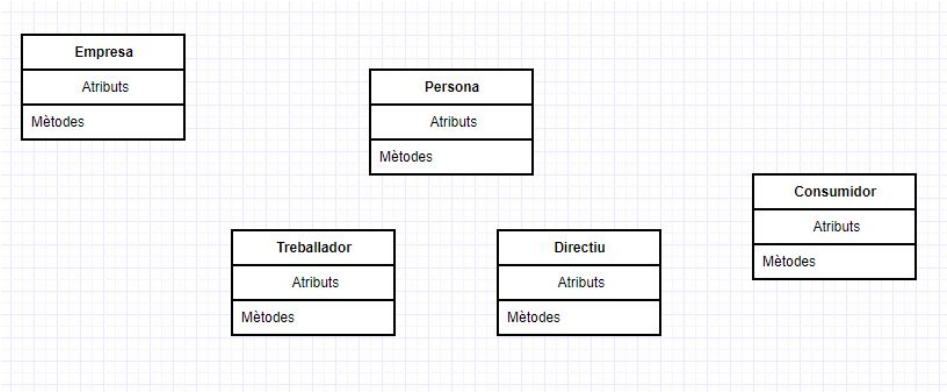
- Per tenir una idea del valor dels projectes, Mar també necessita conèixer el nom i sou brut dels treballadors dels seus clients. D'aquesta manera podrà posar un preu ajustat a la necessitat de cada client.
- Finalment, Mar necessita saber qui són els directius dels seus clients, per poder adreçar-se a ells i poder enviar informació comercial rellevant.

De tota aquesta informació, primer heu de poder **extreure les classes**. Vistos tots els conceptes que es mencionen a l'enunciat, queda clar que "client" és un dels més rellevants. Per comprovar que és una classe, us podeu preguntar si és un concepte (que sí que ho és) i si és només un exemple concret o una descripció. Mar podria tenir com a clients el client A i el client B, per tant el concepte client no és un una ocurrència concreta, sinó que és una classe.

També necessiteu conèixer els consumidors dels clients de Mar. Aplicant la mateixa lògica anterior, us adonareu que "consumidor" és una classe. Aquí, però, ja es pot veure un problema, i és que "client" i "consumidor" són dues paraules molt properes, i a primera vista no queda massa clar què és què. Per això podeu **canviar el nom** d'alguna de les dues classes per un altre que sigui també correcte, i que us permeti tenir un diagrama de classe més llegible. En aquest cas, una bona opció pot ser canviar "client" per "empresa". Si bé és cert que Mar té clients, també ho és que aquests són empreses. Conceptualment aquest canvi no introdueix cap problema i, com es pot comprovar, aporta claredat.

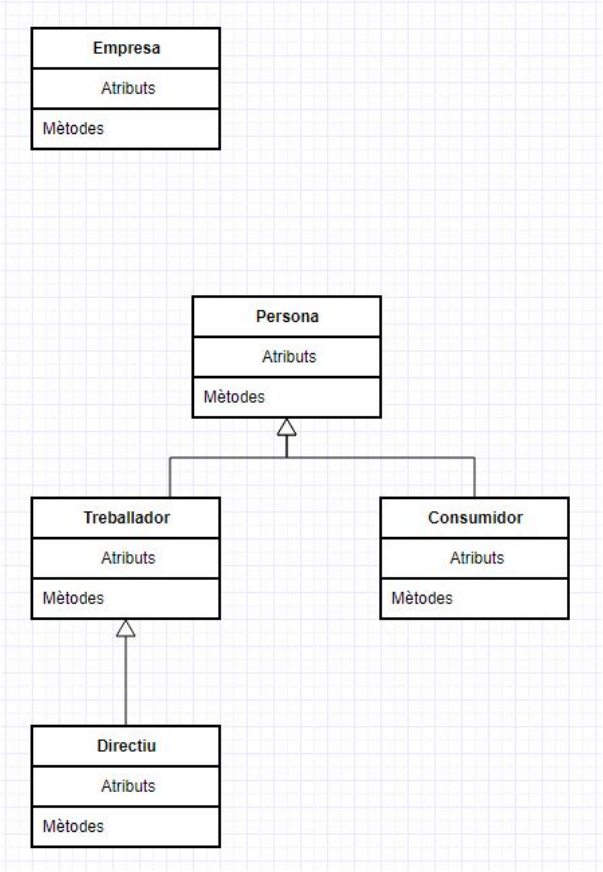
Altres conceptes que apareixen a l'enunciat són els "treballadors" i "directius". Aquests conceptes es poden generalitzar, ja que tant els treballadors com els directius són persones. A més a més, els consumidors també són persones. Per tant, podem abstreure aquestes tres classes amb la classe "persona". Això sí, els directius són una subclasse de "treballador", no de "persona" directament. Això ens dona el diagrama següent (vegeu la figura 2.14):

Figura 2.14. Classes identificades donat l'enunciat



Però, com ho fem ara per **relacionar les classes**? Doncs, com hem comentat, tant "directiu", com "consumidor" i "treballador" són subclasses de "persona", però "directiu" és, de fet, subclasse de "treballador". Això ens dona el diagrama següent (vegeu la figura 2.15):

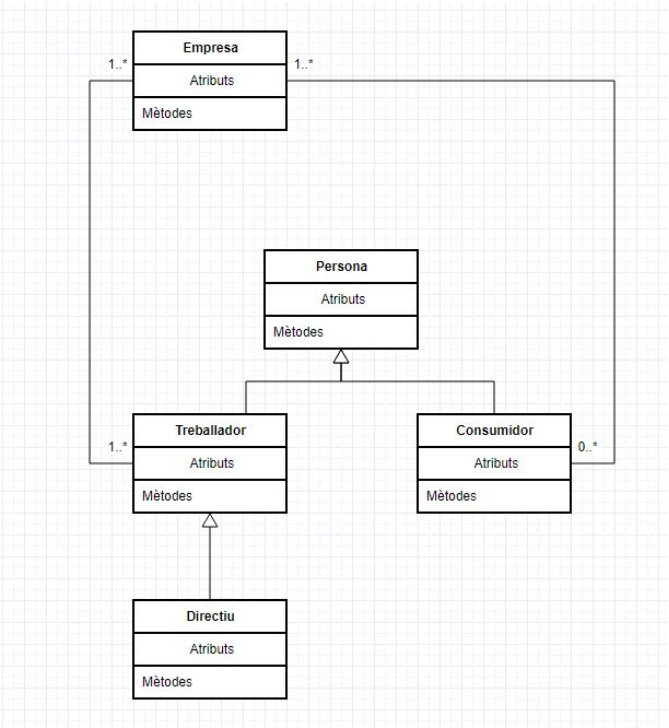
Figura 2.15. Relacions entre classes; herència identificada



Ara heu de veure com es relaciona "empresa" amb les altres classes. L'enunciat comenta que cada empresa té treballadors i consumidors; per tant, ha d'estar relacionada amb aquestes dues classes. Una empresa legalment ha de tenir algú que l'administri i no hi ha un màxim legal per nombre d'empleats; per tant, la cardinalitat a la banda de "treballador" serà "1..*" (vegeu la figura 2.16).

Aquí, si sou observadors, podeu veure una diferència entre el món real i el nostre diagrama de classes. Tècnicament, un administrador d'una empresa no pot ser un treballador contractat com la resta, sinó que ha de ser autònom. En tot cas, des del punt de vista dels requeriments d'aquest projecte, una persona que faci feina per a una empresa serà tractada com una persona treballadora per aquesta empresa. Formalitzar els detalls no sempre ens convé perquè no aporta res al diagrama i per tant només afegeix una complexitat innecessària.

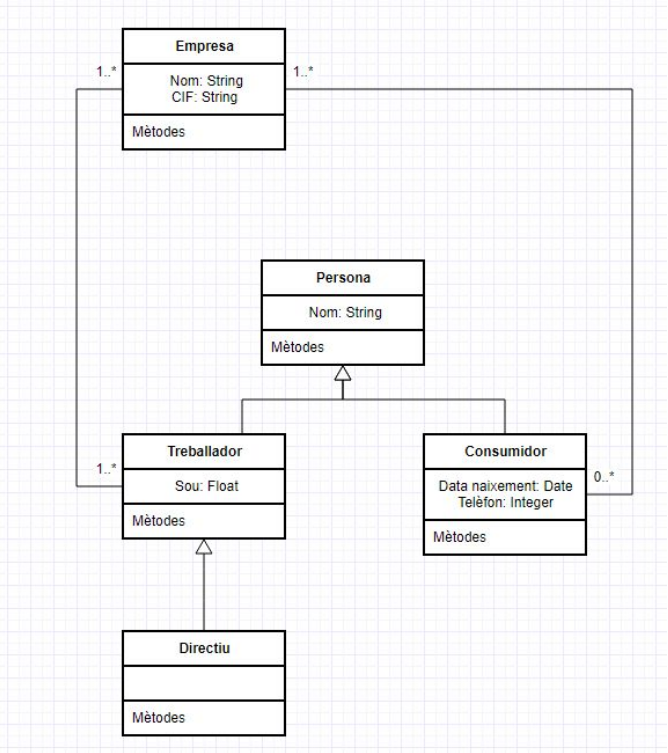
FIGURA 2.16. Relacions entre classes identificades amb la seva cardinalitat



Un cop especificades les classes i establertes les relacions, queda **definir els atributs i els mètodes**. En aquest cas, molts dels atributs es demanen explícitament. En concret, l'enunciat diu que heu de guardar el nom i el CIF de les empreses. Per tant, aquests són els atributs, i com que són una combinació de caràcters alfanumèrics, tots dos són de tipus "String".

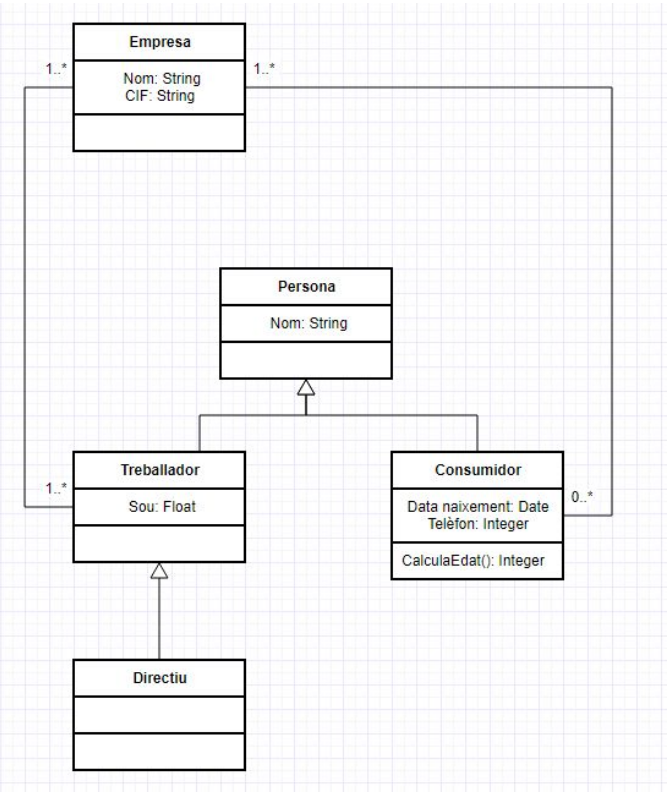
També és un requeriment conèixer el nom de totes les persones, siguin clients o treballadors. A més, dels consumidors se'n necessita el telèfon i l'edat. Aquí us podeu trobar amb un problema: si simplement guardeu l'edat, cada any haureu d'actualitzar les dades. I, si fos necessari ser precís, no podríeu saber-ho fàcilment d'aquesta manera. Per tant, la millor opció és guardar la data de naixement i dissenyar un mètode per calcular l'edat. Finalment, dels treballadors se'n necessita el sou, que serà un atribut de treballador. Per tant, el diagrama quedarà així (vegeu la figura 2.17):

FIGURA 2.17. Atributs identificats a les classes



I, respecte als mètodes, per acabar, com hem dit abans, en necessiteu un per calcular l'edat. Així doncs, el diagrama final resultarà de la següent manera (vegeu la figura 2.18):

FIGURA 2.18. Resultat final del diagrama de classes de Mar



En resum, caldrà que tingueu en compte que:

1. Els diagrames de classes en UML poden tenir diferents tipus d'associacions entre les classes. Les classes no només poden relacionar-se entre elles de forma binària, sinó ternària o múltiple.
2. Cal entendre com funcionen aquestes associacions, com es representen i quins casos cobreixen; a més de poder determinar la cardinalitat d'aquestes relacions.
3. A més a més, les herències permeten abstreure i formar factor comú entre múltiples classes i poder d'aquesta manera reduir la complexitat i redundància del diagrama. És fonamental saber determinar les herències, també, perquè això té un impacte directe sobre la programació del projecte.

2.3.3 Esquemes i models de bases de dades

Un dels reptes més habituals, als quals qualsevol desenvolupador s'enfronta a l'hora de dissenyar una aplicació, és **tractar les dades**. En moltes ocasions haureu d'emmagatzemar o accedir a dades generades per usuaris o per altres aplicacions; caldrà, doncs, considerar-ho en detall.

Les dades en una aplicació es poden perdre fàcilment. És a dir, si tanqueu una aplicació d'una manera inesperada, o bé el dispositiu on s'està executant es queda sense bateria, o hi ha qualsevol tipus de problema, les dades que no s'hagin guardat en un sistema permanent es perdran. Mantenir aquests **valors enregistrats** d'una manera estable i segura requereix un esforç a l'hora de programar, i té un cost també en el rendiment d'una aplicació tant en pèrdua de velocitat, com en augment de consum de bateria com en espai en disc dur. Per tant, heu de tenir molt clar:

- quines dades són les necessàries per al nostre sistema,
- eliminar redundàncies i
- tenir clar quin sistema d'emmagatzemament de dades fer servir.

Hi ha múltiples maneres d'assegurar la persistència de dades: podeu fer servir fitxers i abocar-hi les dades directament; pot ser amb un format propi, o bé formats estàndard, podeu fer servir alguna base de dades o inclús connectar-vos remotament a algun servei al núvol i que es gestioni d'alguna de les maneres anteriors. Cadascuna de les opcions té, lògicament, una sèrie d'**avantatges i inconvenients**:

- Els fitxers tenen l'avantatge de poder ser llegits per humans, si s'escull un format adequat. Amb un bolcat directe, per exemple serialitzant les dades, us trobareu que carregar i guardar dades pot ser realment senzill. Això sí, no és llegible per humans i, si canvieu el format de les classes, la serialització

ja no és possible, i per tant no és gens flexible. Connectar-nos al núvol pot ser una gran opció per a aquells projectes que vulguin ser independents del dispositiu on s'executi el nostre projecte, i inclús admetre aplicacions multidispositiu.

- D'altra banda, significa haver de programar també la connexió i resoldre múltiples problemes derivats de les casuístiques que es poden donar, com per exemple fallades de xarxa i edició simultània de dades. A més a més, no soluciona el fet que al núvol igualment hi heu de ser capaços de guardar les dades d'alguna manera, que acaba sent una de les altres opcions mencionades.
- Finalment, les bases de dades suposen una solució molt estesa que es troba en un punt mitjà. Una base de dades és un sistema estructurat d'emmagatzemament de dades, on el gestor de la base de dades és qui s'encarrega d'executar les entrades i sortides de les dades. Aquest gestor és independent dels projectes que es desenvolupin i que requereixin dades, i hi interactua a través d'un llenguatge estàndard.

Al llarg dels anys han existit diversos tipus d'estàndards de llenguatges per interactuar amb bases de dades, i també diferents tipus de bases de dades. Des de fa un temps, l'àmplia majoria de les bases de dades són de tipus relacional.

Una **base de dades relacional** es pot entendre com un conjunt de taules on cada taula té un nombre de columnes, o camps, definit per nosaltres, i tantes files com dades contingui, on cada fila és una entrada única de dades.

Així, per exemple, podríeu tenir la taula "persona", on guardar el nom, el cognom i l'edat d'una persona. Aquests atributs serien les columnes de la taula i una fila seria una entrada concreta, com ara "Joan, Coloma, 33". Lògicament, hi podria haver un nombre elevadíssim de files.

Cada taula, a més a més, cal que tingui un identificador únic, de manera que només coneixent aquell valor es pugui conèixer tota la entrada d'una manera inequívoca. Aquest identificador rep el nom de **clau primària**.

Si voleu, per exemple, guardar les dades d'una persona, no n'hi ha prou amb guardar el nom i els cognoms, perquè no podeu assegurar que no hi hagi una altra persona que es digui de la mateixa manera. Per tant, es pot afegir un **camp extra** que pot no ser estrictament necessari per a una aplicació, com per exemple el DNI, o bé es pot afegir una columna generada automàticament que fa la funció de clau primària. Fins i tot es pot donar el cas que una combinació de dos o més camps sigui única, i dissenyar la clau primària com una combinació d'aquestes columnes.

Hi ha molts casos en els quals no hi ha cap camp extra que sigui necessari i que pugui fer de clau primària. En aquests casos es genera un camp automàtic, que es sol marcar com a "id(auto)".

Sovint es necessita que les taules facin referència les unes a les altres. Per exemple, podeu necessitar una base de dades d'on puguem extreure quins cotxes pertanyen a algunes persones. Els cotxes tenen un identificador únic, que és la matrícula, i les persones es poden identificar amb el seu número de DNI. Ara bé, com relacionem aquestes dues taules? La resposta és amb la **clau forana**.

Una clau forana realment és una columna que fa referència a una clau primària d'una altra taula. Per exemple, a la taula de persones hi afegiu una nova columna, que és la clau forana de la taula dels cotxes, on hi poseu valors que ens permetin trobar les dades del cotxe propietat de cada persona.

Aquestes operacions es fan a partir del llenguatge anomenat ***Structured Query Language* (SQL)**, o 'llenguatge de consulta estructurat'. És un llenguatge molt ben estructurat i definit amb el qual es poden crear, manipular i consultar bases de dades relacionals. Conèixer SQL és, per tant, fonamental per dissenyar aplicacions amb persistència de dades.

Hi ha diverses maneres d'assegurar la **permanència de les dades**. Probablement, la més comuna és fer servir bases de dades relacionals amb SQL. Això, però, no treu que s'ha de fer una anàlisi de quines dades són necessàries i fer un esquema de l'estructura de la base de dades.

2.4 Arquitectures, plataformes i entorns tecnològics

Tot projecte AMI complex ha de tenir en compte no només el repte de programar el codi, sinó tot l'entorn necessari per aconseguir-ho, assegurant que es compleixen els objectius. Cal tenir en compte els usuaris finals, per produir un producte que satisfaci les seves necessitats i no d'altres. A més a més, no tots els suports físics tenen les mateixes característiques. Finalment, cal que els usuaris sàpiguen trobar i instal·lar l'aplicació.

2.4.1 Arquitectura de producció o desenvolupament

Qualsevol projecte ha de tenir una previsió de quins són els **recursos necessaris** per dur-lo a terme: quants programadors calen? Quina és la experiència mínima necessària per a poder-lo desenvolupar? Quines llicències i entorns de desenvolupament calen? Totes aquestes preguntes sorgeixen per a qualsevol projecte d'una envergadura raonable. Normalment, el sentit comú i l'experiència són les millors conselleres per resoldre els dubtes.

En tot cas, per començar a poder fer aquestes estimacions, es poden seguir algunes guies. Una de molt clara és tenir en compte el diagrama de classes. Cada classe, cada mètode, cada atribut i relació afegeix una complexitat que es transforma

en **temps de desenvolupament**. Per tant, diagrames més extensos volen dir desenvolupaments més llargs.

A més del nombre de classes, cal distingir la dificultat de desenvolupament de cadascuna (per exemple, si una classe té a veure amb una interfície gràfica, molt probablement es trigarà substancialment més). No hi ha una regla clara i fixa, perquè cada projecte té les seves particularitats, sobretot a la part d'interfícies.

Una bona pràctica és agrupar les classes amb **components visuals**. I també ho és agrupar els elements interactius de cada interfície. Per cada element interactiu haureu de controlar el tipus de valors que s'introdueixen, controlar errors, capturar esdeveniments... Per tant, primer, a l'hora de dissenyar interfícies, heu d'intentar sempre minimitzar els **elements interactius** i, d'altra banda, per estimar el cost en temps de desenvolupament d'una classe, cal tenir en compte que les interfícies més complexes seran més lentes de programar.

Una bona manera i ràpida de fer una estimació bàsica d'un temps de desenvolupament d'una aplicació és programar una classe de **complexitat mitjana**, comparativament amb la resta. Mesurant quant es triga a acabar aquesta classe, es pot extrapolar amb la resta.

Un altre mètode, més costós, però més acurat, és elaborar primer totes o bona part de les pantalles d'interacció amb l'usuari i fer el **cicle d'ús complet** tot i no programar tota la funcionalitat. D'aquesta manera us podeu assegurar que no us deixeu elements importants per falta de previsió o per un disseny que no ha tingut en compte tots els casos possibles. Un cop fet aquest prototip, podeu estimar molt millor el temps necessari per a completar la resta de la funcionalitat.

En un equip sempre hi cal algú amb experiència en desenvolupament de programari per dur a terme projectes. Aquesta persona liderarà l'equip i serà l'encarregada d'escollir la resta de l'equip i distribuir les tasques.

Respecte a les **llicències**, heu de considerar no només tot allò que quedarà inclòs dins del projecte, sinó també les eines necessàries per coordinar un equip. Per exemple, serveis com Trello o Bitbucket, per poder sincronitzar tasques, són essencials en projectes d'una mida raonable, i també heu de comptar amb la compra de les llicències de sistemes operatius, ordinadors on desenvolupar-los, possiblement editors per programar, serveis per veure estadístiques d'usuari...

És molt complex produir programari; cal prestar atenció als **requisits tècnics** i **preveure'n les capacitats**. Com més experiència, més fàcil serà poder predir necessitats i fer estimacions de temps; així i tot, seguint una sèrie de pautes i amb ajuda d'algunes eines, com el diagrama de classes, se'n pot tenir una idea més clara.

2.4.2 Arquitectura de destinació o desplegament

Abans de començar a dissenyar visualment un projecte, heu de pensar en l'usuari final. No serveix de res tenir uns algoritmes excel·lents i una programació elegant i robusta, si qui ha d'utilitzar el programari no el fa servir. Per tant, en tot moment heu de tenir present les possibles **limitacions** dels futurs usuaris; preguntar-vos: són professionals acostumats a processos molt concrets i definits? Són menors d'edat? Són menors de deu anys? Són persones grans? Es tracta d'usuaris amb disfuncions cognitives, com alteracions de la personalitat? Cal tenir en compte aquestes respostes perquè no només afectaran visualment el vostre projecte, sinó també tot el context que l'envolta.

Per exemple, fer un projecte per a una escola de primària és molt més sensible, perquè heu de poder assegurar que no hi haurà contingut que pugui ferir la sensibilitat dels alumnes ni alliberar les seves dades personals. A banda, depenent de l'edat, és possible que facin falta dispositius més robustos per aguantar caigudes i un tracte físic més exigent. Tot i no ser sempre responsables d'aquests dispositius, sí que hem de ser conscients que tot sovint aquests aparells tenen unes limitacions tècniques importants, que el seu pes sol ser més elevat, i que es poden perdre; per tant, la informació cal que estigui xifrada i protegida per contrasenya.

La millor manera d'assegurar que la vostra aplicació tindrà èxit és posant-vos a la pell dels vostres usuaris. A més, heu d'intentar **preveure les situacions més extremes**. Per fer-ho, us heu d'imaginar dos o més perfils extrems d'ús de la vostra aplicació. Per exemple, imagineu-vos que feu un projecte d'una aplicació per a una llar de gent gran. Un usuari extrem seria aquell a qui li encanta l'aplicació i la vol fer servir a totes hores, però té problemes de visió i de memòria. Heu de dissenyar la interfície visual de manera que tots els elements interactius siguin prou visibles i prou intuïtius perquè aquest usuari pugui gaudir-ne. Un cop fet, comprovareu que ho heu aconseguit, mostrant la interfície a algú amb aquestes característiques, i deixant que sigui ell mateix qui sigui capaç de moure's dins l'entorn. Per fer aquesta prova, no cal ni tan sols desenvolupar la interfície, sinó que es pot fer amb papers impresos; d'aquesta tècnica se'n diu **disseny centrat en l'usuari**.

> **Un disseny que afavoreixi l'usuari**
>
> Una coneguda cadena de menjar ràpid va dissenyar i optimitzar les seves cuines en una pista de tenis. Va dibuixar amb guix els elements d'una cuina i van veure com interactuaven els treballadors que simulaven fer les tasques de cada part. En poques hores, i sense pràcticament cap inversió, van aconseguir trobar la distribució ideal per poder servir el màxim de menjar en el mínim temps.

Tornant a l'exemple de la llar de gent gran, un altre usuari extrem podria ser aquell a qui no li agrada la tecnologia i no hi troba utilitat, a la seva vida. En aquest cas, potser es podria fer una guia impresa en paper, amb una bona explicació feta amb cura sobre què li aporta l'aplicació i com pot començar-la a fer servir de manera senzilla. Tots aquests elements poden ser determinants en l'èxit del projecte, i variaran en cada situació. Per tant, més que una guia concreta, l'important és sempre posar-se a la perspectiva dels nostres usuaris, buscar els casos més extrems i trobar quines serien **solucions adients a cada cas**.

Des del punt de vista tècnic, heu d'assegurar-vos que controleu els dispositius on s'executarà la nostra aplicació. No sempre serà possible controlar el detall de cada dispositiu, però sí que heu de tenir clar uns mínims. Per exemple, el sistema operatiu és determinant a l'hora de començar a concretar els equips de desenvolupament. Hi ha tecnologies que funcionen a tots els sistemes operatius, però n'hi ha moltes que simplement no. Per tant, heu de conèixer quin és l'entorn final on s'espera poder gaudir de l'aplicació; a dia d'avui hi ha bàsicament tres grans mons:

- Les aplicacions d'**escriptori**, on el sistema operatiu Windows és el gran dominador, i, tot i que s'està esforçant per desenvolupar i omplir una botiga d'aplicacions, el cert és que els usuaris estan acostumats a descarregar-les d'internet o a instal·lar-les amb suport físic. Com a conseqüència, tindreu poc control sobre l'entorn final. No sabreu si el dispositiu tindrà panell tàctil, si tindrà una pantalla gran o petita, el país, l'idioma...
- Els **dispositius mòbils**, que, en canvi, estan ocupats per dos grans sistemes operatius, iOS i Android, i els dos tenen associats una botiga d'aplicacions molt sofisticada on els desenvolupadors tenen tot tipus d'eines per decidir quina versió es pot distribuir, en quina geografia, tipus de dispositiu... Aquest és el cas més favorable, ja que teniu un control prou específic.
- El **web**, que és el cas contrari: es pot executar sobre qualsevol sistema operatiu, diversos exploradors, i pràcticament tota la combinatòria de maquinari possible. Per tant, és el més complicat de controlar, però també l'entorn on és més fàcil arribar al màxim de públic i el que ofereix millor compatibilitat amb tot tipus de dispositius. Hi ha eines per fer compensar la manca de control sobre el maquinari. La més comuna és tenir diverses versions del mateix contingut, i decidir quina versió enviar segons el dispositiu. Ja que vosaltres teniu el control de la distribució al servidor que escolliu, ho podeu aprofitar per adaptar-vos a cada situació.

Finalment, tot sovint us trobareu que el projecte s'ha de poder fer servir en dispositius de plataformes i sistemes operatius diferents, i a més s'han de poder comunicar entre dispositius. Per solucionar aquest problema d'**interoperabilitat**, moltes empreses ofereixen serveis al núvol a través d'internet. Aquesta solució té diversos avantatges, com el fet que les dades es guarden de manera persistent i que la responsabilitat sobre la protecció de les dades recau sobre una altra entitat. Com que és una tasca fonamental, crítica i normalment molt complexa, molts cops és millor que se'n responsabilitzi algú altre. A canvi, és clar, té un cost econòmic que pot arribar a ser important segons el nombre d'usuaris.

És fonamental conèixer l'entorn final del nostre projecte, és a dir, els **requisits d'accessibilitat** segons el tipus d'usuari i els **mètodes de distribució** per tal de trobar la millor compatibilitat amb la plataforma de suport de la informació. Us trobareu amb multitud de problemes que podreu resoldre amb proves inicials, un disseny adient i restringint en la mesura que pugueu els dispositius on s'executarà l'aplicació.

2.4.3 Desplegament d'AMIs

Les plataformes actuals tenen característiques molt diferents l'una respecte de l'altra, tant pel que fa al rendiment com a eines de desenvolupament i desplegament. Així, és important conèixer, almenys, unes línies generals per no confondre les capacitats de cadascuna.

Les **arquitectures mòbils** com els telèfons intel·ligents o les tauletes opten en la seva majoria per arquitectures *hardware* basades en **ARM**. Aquests processadors són cada any més potents, però en rendiment no arriben encara a rivalitzar amb ordinadors d'escriptori. Això és perquè s'enfoquen més a ser eficients energèticament per allargar la vida de les bateries. A més a més, els sistemes operatius de dispositius mòbils (majoritàriament Android i iOS) contenen una sèrie de mesures cada cop més estrictes per evitar que les bateries s'exhaureixin massa d'hora. En aquest sentit, els usuaris són cada cop més sensibles a desinstal·lar aplicacions que saben que consumeixen en excés. Per tant, desenvolupant projectes per a aquestes plataformes cal tenir molta cura amb aquest aspecte.

ARM

ARM és una empresa de disseny de processadors. Els seus dissenys els compren gairebé tots els fabricants de xips de baix consum, com els destinats a dispositius mòbils.

A més, aquests dispositius normalment són molt interessants perquè contenen una gran quantitat de **sensors**. Gairebé tots disposen de múltiples càmeres, sensors de proximitat, de llum, GPS, acceleració, brúixola... sense oblidar-nos de la connectivitat telefònica i d'internet. Tot aquest conjunt de sensors desemboca en informació que, interpretada correctament, pot donar lloc a aplicacions molt sorprenents. Per exemple, les aplicacions de realitat augmentada fan servir pràcticament tots els sensors d'un telèfon mòbil intel·ligent.

A més a més, el ràpid creixement d'aquesta plataforma, i el fet que està pràcticament ocupada pel duopoli Android i iOS, ha despertat un enorme interès per eines de **desenvolupament multiplataforma**. Per a aplicacions en 3D hi ha entorns com Unity 3D o Unreal Engine, amb els quals es pot desenvolupar una mateixa aplicació que no requereixi canvis o que aquests siguin mínims entre plataformes. Per a aplicacions en 2D actualment tenim entorns com PhoneGap, Ionic, Native script o React Native.

D'altra banda, les **plataformes d'escriptori** (tant en ordinadors de sobretaula com en portàtils) tenen, en general, millors prestacions de computació: memòries més ràpides, un disc dur amb molta més capacitat, processadors més ràpids i pantalles més grans. També se'ls exigeix més, la qual cosa resulta en una fluïdesa menor de la que podrien tenir. Aquest fet és molt notable en moltes pàgines web, que tenen versions d'escriptori i mòbils. Les versions mòbils tenen unes capacitats molt menors i mostren molta menys informació d'un cop. Això sí, si obriu aquestes versions en un dispositiu d'escriptori veureu una pantalla tan poc atractiva que no us resultarà útil, i exigireu més al vostre dispositiu. D'altra banda, en el sector professional segueix sent pràcticament imprescindible desenvolupar per a una plataforma d'escriptori. La distribució, en aquest cas, serà o a través de suport físic, com els CD, memòries USB o bé a través de descàrregues per internet i codis d'activació.

En un futur no massa llunyà és més que probable que es convergeixi cap a una única plataforma. En aquests moments ja hi ha propostes de Windows amb arquitectures ARM, i versions d'Android que funcionen en arquitectures de sobretaula. Per tant, la discussió sobre cada plataforma i sobre la distribució està subjecta a molts canvis.

2.4.4 Llançament d'un producte. Aspectes que cal tenir en compte

Tots els sistemes intel·ligents i les plataformes estan convergint cap al mateix concepte: un sistema operatiu desenvolupat per la mateixa empresa que controla també quines aplicacions s'instal·len a través d'una botiga d'aplicacions. Hi ha diverses propostes, més o menys restrictives, per a aquest mateix concepte. Per exemple, tenim les plataformes de sobretaula, dominades per Windows, on el maquinari no està controlat per la mateixa empresa, i les aplicacions es poden instal·lar a través de diverses botigues i no només des de la seva. A l'altre extrem hi ha els dispositius iOS, que únicament poden instal·lar aplicacions de la botiga, que pertany a la mateixa empresa que controla tant el sistema operatiu com el *hardware*.

A banda dels ordinadors, els telèfons mòbils i els diversos equips d'electrònica de consum, altres sistemes d'exhibició i **noves plataformes** estan guanyant importància. Els últims anys, diverses empreses s'han esforçat molt per ser rellevants a la TV interactiva. Tenim les propostes de Smart TV de cada fabricant, a més de les d'Apple TV o Android TV. Les videoconsoles també han evolucionat per deixar de ser una simple plataforma de jocs i convertir-se en centres multimèdia amb multitud d'aplicacions i no totes dedicades al joc. Tot i això, també aquestes plataformes tenen en comú que fan servir el mateix concepte doble de **sistema operatiu-botiga d'aplicacions** (en el cas de les videoconsoles, les botigues físiques també son rellevants, però la tendència és que deixin de ser-ho).

Per tant, ens podem preguntar: què és allò que les fa tan útils? Com podem convèncer els usuaris que facin servir la nostra aplicació i no la dels competidors? Tot i que cada botiga té característiques específiques pròpies, en general fan una feina similar:

- Des del punt de vista de l'usuari el valor és molt clar: facilita molt la feina de trobar **aplicacions adients**, ja que està tot agrupat en un mateix lloc, està categoritzat, i té una sèrie de puntuacions i comentaris dels usuaris que fan referència a la qualitat de l'aplicació. D'aquesta manera es pot comparar d'una manera més fiable que amb els comentaris i descripcions del fabricant.
- D'altra banda, per als desenvolupadors també hi ha avantatges, ja que només posant l'aplicació a la botiga tenen accés a un **mercat immens**, de centenars de milions d'usuaris distribuïts per tot el món, on tota la feina de monetitzar, establir confiança i distribuir ja està feta. A més, ofereixen un seguit d'eines molt útils per poder fer proves i una distribució efectiva controlant errors i

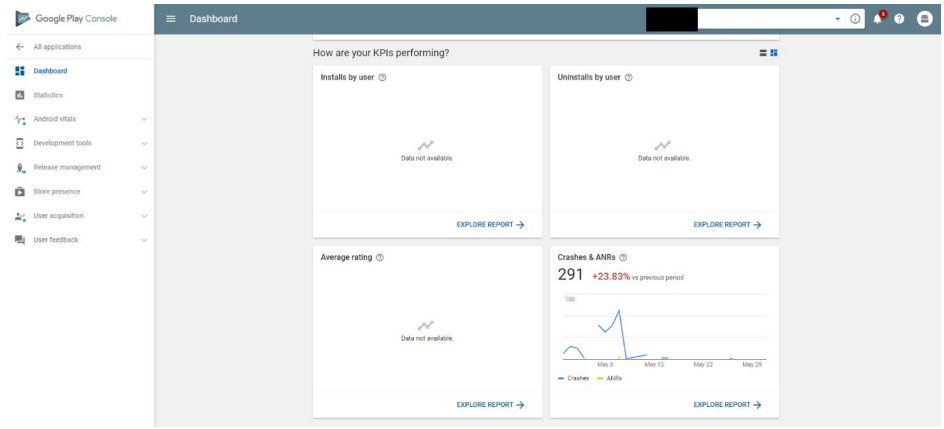
possibles problemes legals. Per això aquest model funciona i la tendència és que totes les plataformes convergeixin al mateix concepte.

Totes les plataformes estan convergint cap a oferir **formes centralitzades** de distribució d'aplicacions. El concepte és una botiga d'aplicacions on tant els usuaris com els desenvolupadors poden gaudir de diversos serveis que els milloren l'experiència.

A continuació veureu un exemple d'una botiga d'aplicacions dins l'apartat de desenvolupadors, també conegut com la **consola per desenvolupadors**. En concret, veureu el cas de Google Play, que és segurament la més accessible per a les persones. El cost per poder començar a publicar aplicacions és de 25 dòlars, que es paguen un sol cop i donen accés il·limitat. Un cop pagat, us trobeu que podeu gestionar i afegir múltiples aplicacions dins la mateixa consola. Si n'afegiu una, entrareu en una consola on veureu un panell amb la informació més rellevant.

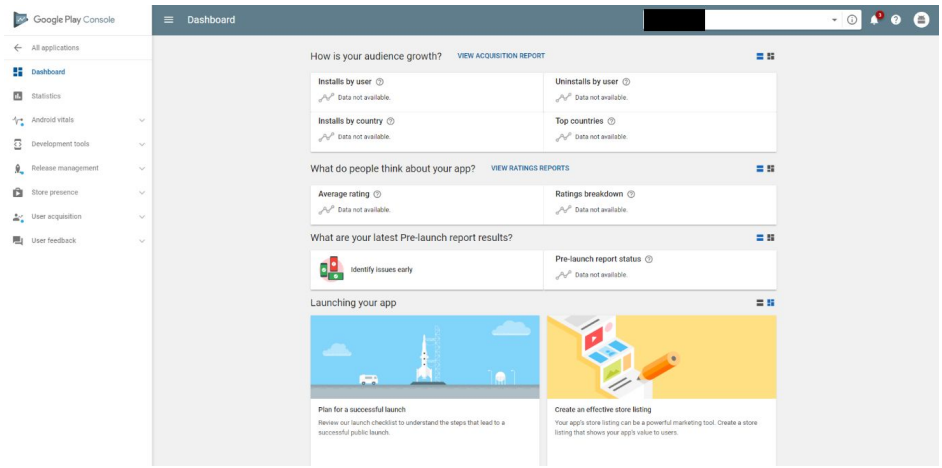
El primer que veureu són quatre *Key Performance Indicators* (KPI), que són indicadors rellevants del rendiment de la nostra aplicació. En concret, trobeu informació sobre el nombre d'instal·lacions per usuari, desinstal·lacions, una mitjana sobre la valoració dels usuaris i el nombre de *crashes* o problemes han tingut els usuaris (vegeu la figura 2.19).

FIGURA 2.19. Pàgina inicial de la consola de desenvolupadors de Google Play, que mostra alguns KPI



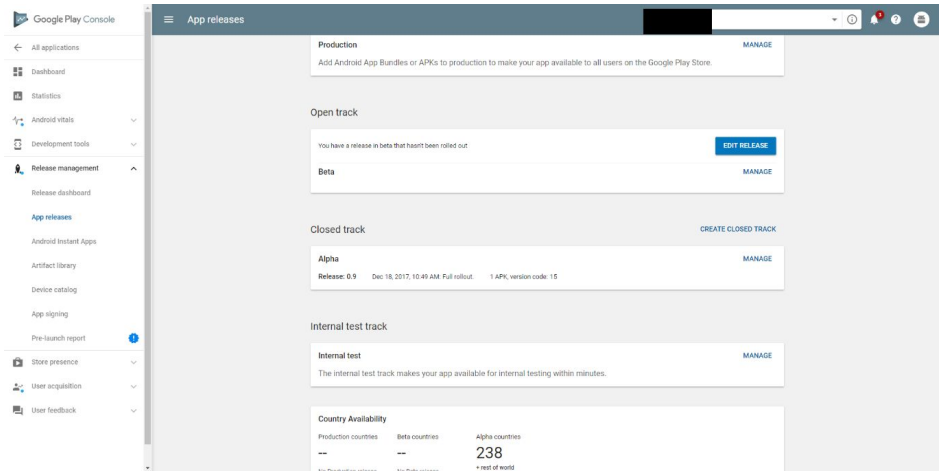
Tot seguit, teniu un apartat relacionat amb el creixement del nombre d'usuaris, amb mètriques sobre instal·lacions per país i un rànquing de països. Finalment, també hi ha informació sobre el llançament de l'aplicació i enllaços a articles que parlen de bones pràctiques (vegeu la figura 2.20).

FIGURA 2.20. Més KPI de la pàgina inicial de la consola de desenvolupadors de Google Play



A més d'aquesta secció, podeu veure la de *Release Management* (vegeu la figura 2.21). Aquí és on es pot començar a veure el potencial de les eines proporcionades pels desenvolupadors. Normalment, els equips tenen alguna persona o inclús un departament dedicat (potser amb d'altres tasques) a encarregar-se dels llançaments. En aquest sentit, pot fer quatre **tipus de llançaments** diferents:

FIGURA 2.21. 'Release Management': espai per definir diferents tipus de llançaments, dins la consola de desenvolupadors de Google Play



- Un **llançament Internal** serveix per a fer proves internes, dins de l'equip de desenvolupadors. Cal pujar l'APK a la botiga d'aplicacions, i d'allà començar a omplir tots els camps requerits per tenir la pantalla de descàrrega de l'aplicació des de la banda de l'usuari tal com es vol presentar. Als desenvolupadors els és molt útil per a fer totes les proves necessàries abans d'arriscar-se a fer servir usuaris reals amb errors. Presentar tot el procés és massa llarg i depèn completament de cada botiga, per tant, aquí no l'afegirem. Però el concepte és el mateix a totes bandes, i l'objectiu també.

- Els **llançaments Alpha** serveixen per a fer proves amb grups de persones controlats. Les consoles per a desenvolupadors ofereixen maneres per a construir llistes tancades de provadors. En el cas de Google Play, només calen les adreces de correu. A partir d'aquesta llista, els provadors poden

descarregar l'aplicació com si ja estigués disponible per a tothom. La diferència respecte als Internal és que en aquest cas els provadors poden ser de fora dels equips de proves, o inclús de fora de l'empresa que desenvolupa el projecte.

- Avançant en aquesta lògica, els **llançaments Beta** són molt semblants als Alpha, però el grup és una mica més ampli. Podem obrir aquest tipus de llançaments al públic general, a totes aquelles persones que es vulguin apuntar al programa de llançaments Beta. Si comuniqueu de manera entenedora que aquestes versions no són necessàriament estables i que contenen funcionalitat que no sempre serà definitiva, els usuaris seran tolerants a errades i a dissenys poc clars. L'avantatge d'aquesta manera de distribuir les aplicacions és que es pot monitorar de manera fidedigna l'ús real de cada part de l'aplicació. Es pot controlar si una nova funció agrada o no als usuaris, o si un nou disseny genera rebuig o atrau usuaris. A més a més, rebreu comentaris específics i inclús suggeriments de millora. Fins i tot, si l'aplicació és popular, es filtrarà a la premsa i podreu veure clarament la reacció del públic.

APK

Un APK, o *Android Application Package*, és el format de les aplicacions d'Android. Qualsevol aplicació en un dispositiu Android és realment un APK instal·lat.

Per apurar i donar el màxim de flexibilitat als desenvolupadors, les consoles els permeten **ajustar alguns límits**. Per exemple, podeu controlar quins països tenen accés a cada versió. També podeu limitar les versions de sistema operatiu que poden instal·lar l'aplicació, o inclús restringir la mida de pantalla. La raó per la qual podeu voler restringir aquests paràmetres és per a evitar, per exemple, problemes legals en alguns països. Recentment a la Unió Europea hi ha hagut un canvi important de legislació respecte a la privacitat de les dades. En aquest cas, moltes empreses han optat per tenir versions diferents segons la geografia fins a ajustar-se a la normativa i poder convergir a una única versió. D'altra banda, hi ha algunes aplicacions on el disseny és una peça tan fonamental que una diferència important sobre la mida i resolució d'una pantalla resulta una experiència d'usuari molt diferent. Per tant és millor tenir dues versions diferents segons aquests paràmetres. Si bé això és cert per a tots els tipus de llançaments, és normalment en les fases Beta on fareu les proves necessàries i establireu alguns límits. Lògicament, en els llançaments finals l'ideal és no posar límits i tenir tots els casos previstos i provats.

- Finalment, un **llançament *Production*** és molt similar a un de Beta, però sense restricció d'usuaris. Es poden definir encara criteris de restricció sobre algunes característiques, però qualsevol usuari que compleixi els criteris podrà baixar i instal·lar l'aplicació. Una eina addicional que solen oferir les consoles de desenvolupadors és una distribució per onades o per etapes. És a dir, oferir la nova versió primer a un percentatge definit dels usuaris, escollits a l'atzar, obrir una finestra temporal per analitzar-ne l'impacte abans de seguir oferint la nova versió a més usuaris, i apujar aquest percentatge si tot és correcte. D'aquesta manera les empreses eviten descobrir massa tard que una versió nova té algun error que afecta una quantitat massa gran d'usuaris, i poden reaccionar a temps.

A més d'aquestes qüestions, per respondre a la pregunta "què podeu fer per convèncer els usuaris que facin servir la nostra aplicació i no la d'un competidor",

la resposta és que teniu un seguit d'eines per fer-ho. Algunes formen part de les consoles de desenvolupador, com tenir cura de la presentació de l'aplicació i fer servir descripcions amb paraules clau; d'altres són serveis externs, com ara les analítiques de Firebase o els anuncis en xarxes socials.

Els usuaris tenen diverses maneres d'escollir quines aplicacions instal·len als seus dispositius. Una d'elles és la **posició dins d'una cerca**. Les cerques poden ser per nom o per algun concepte que descrigui les propietats de l'aplicació. Com més amunt estigui, més probabilitats hi ha d'obtenir descàrregues. És una dinàmica molt semblant a la dels resultats d'un cercador web. Per pujar en el rànquing, cal treballar bé la descripció de la nostra aplicació, incloent les paraules clau que els usuaris fan servir a les cerques. Per trobar aquestes paraules, es pot fer una anàlisi dels competidors. Copiant el text i cercant les paraules més repetides, es pot extreure quines són les paraules amb les quals els usuaris troben les seves aplicacions. A partir d'aquí, cal fer una reflexió sobre si la vostra aplicació té alguna diferència que cal incloure, o si cal fer servir les mateixes paraules clau.

Un altre factor que cal tenir en compte és la **valoració dels usuaris**. Molts usuaris deixaran comentaris sobre l'aplicació, i una valoració numèrica. De manera molt senzilla, com més comentaris i valoracions tingui una aplicació, més confiança genera, i com més positius siguin, més probabilitat d'obtenir descàrregues. Una manera simple d'obtenir les primeres valoracions és animar a tots els coneguts a descarregar-se l'aplicació i provar-la. Més enllà d'això, es pot enviar a mitjans de premsa, blogs especialitzats, etc.

A més a més, per minimitzar els comentaris negatius les consoles ofereixen dues possibilitats: la primera és fer llançaments de tipus Beta, on els usuaris ja entenen que algunes característiques no funcionen correctament, i la segona és poder contestar els missatges. Demanar disculpes per errors i agrair comentaris positius generarà una dinàmica positiva amb els usuaris, tot i que cal preveure que constitueix un gran esforç, magnificat per la diversitat d'idiomes en què ens podem trobar les ressenyes.

Finalment, la **presentació de l'aplicació dins la botiga**, just abans de la descàrrega, és fonamental. Com més esforç es dediqui a presentar visualment l'aplicació, més atractiva serà. En aquests casos, comptar amb un bon treball de disseny marca la diferència.

A part d'aquests factors, la qüestió subjacent és millorar l'aplicació per fer-la o bé més útil, o bé més atractiva o bé més rendible, per aconseguir augmentar l'ús de cada usuari, el nombre d'usuaris o la monetització per usuari. En aquest sentit, cal poder monitorar bé l'ús de l'aplicació mitjançant **eines d'anàlisi**. Són normalment serveis externs que s'integren dins del desenvolupament de l'aplicació (donada la varietat i complexitat que tenen, no caben dins del temari d'aquest curs, però podeu trobar més informació a firebase.google.com o appscatter.com/).

2.5 Selecció dels equips i les eines de producció

Un cop tingueu tots els requeriments clars sobre el tipus d'usuaris finals, l'arquitectura dels dispositius i l'experiència d'usuari, a més de tota la documentació preparada, com ara el diagrama de classes, l'estructura de la base de dades o les interfícies d'usuari, ja podreu començar a desenvolupar el projecte AMI. En aquest moment inicial, cal preparar tot l'entorn necessari, no només per escriure codi, sinó per fer proves i avançar feina de manera eficient. Per programar i fer proves calen essencialment dues coses: programari i maquinari. Caldrà, doncs, prestar especial atenció a la seva elecció, i també la de les diverses eines que necessitareu.

Com que **comprar equips** té un cost, cal tenir molt present el tipus de projecte per ajustar d'una manera adient la màquina amb la qual treballar. Si cada cop que es vol provar alguna cosa es triguen minuts a veure el resultat, es perdrà agilitat, s'acabaran ajornant o fins i tot evitant les proves. A més a més, l'afegit de tots els minuts d'espera al llarg d'un projecte sumaran una quantitat molt més alta del que ens podem imaginar. Tot això té un cost que en molt poc temps supera amb escreix el cost de molts equips. Per tant, no cal tenir por de sobredimensionar les màquines amb les quals es vol treballar.

Dit això, cal sobredimensionar particularment aquells components que seran més crítics depenent del projecte. Per exemple, en general el rendiment els jocs depèn de la targeta gràfica. Per tant, un projecte amb una gran càrrega gràfica hauria de ser desenvolupat amb màquines que tinguin bones targetes gràfiques. Aquells equips que es dediquin a compilar projectes o a exportar-los a diferents plataformes, haurien de tenir processadors, memòries i discs durs més potents i ràpids.

Com **escollir programari** és menys evident. Cada projecte té unes necessitats concretes, i cal abordar-les individualment. No obstant això, com a mínim cal un editor de codi, un sistema operatiu, un gestor de repositoris, un sistema per poder anotar tasques per fer i errors per corregir, i tot el que calgui per a fer proves del projecte. Cada element de la llista anterior depèn en gran mesura dels requisits del projecte, per tant és difícil generalitzar. Això sí, cal també assegurar la compatibilitat de totes les eines; siguin de creació i retoc visual, o d'integració i desenvolupament.

2.5.1 Eines de producció i desenvolupament

És molt probable que un projecte també necessiti algunes fonts visuals, com **icones, textures i imatges**. En altres casos caldran fonts més específiques, com animacions, malles en 3D o so. Per internet hi ha multitud de recursos, molts cops gratuïts, per trobar aquestes fonts. Aquests en són uns exemples: www.flaticon.com/free-icons/library, www.turbosquid.com, freesound.org/browse/tags/sound-effects/. Cal sempre llegir bé els drets d'ús per a cada recurs i assegurar-se de complir tots els requisits que mencionin.

Per editar aquests arxius, hi ha disponibles una gran quantitat de programes, i molts són multiplataforma, de codi obert i gratuïts. Per exemple, el Blender (www.blender.org) és un estàndard per a treballar amb malles i animacions en 3D; l'Audacity (www.audacityteam.org) és un dels editors de so més utilitzats i coneguts; i, finalment, el GIMP (www.gimp.org/) conforma una alternativa molt potent i gratuïta a programes molt coneguts d'edició d'imatges. En resum, tot i que trobar, produir i editar fonts és costós en temps i esforç, és ben possible fer-ho fins i tot de manera gratuïta.

Respecte a la **integració i el desenvolupament de programari**, el component més essencial és escollir un editor adient, ja que és, molt probablement, l'eina amb la qual els programadors s'hi passen més temps, i és que els editors fan molta feina per facilitar la feina als desenvolupadors. Per exemple, completen automàticament bona part del codi, reduint els errors i augmentant la productivitat. A més a més, ressalten les paraules clau i algunes estructures del codi per facilitar-ne la visualització, i també afegeixen tabulacions automàticament per a les estructures niades, tot seguint les bones pràctiques de cada llenguatge. Hi ha tota una varietat d'editors disponibles, i en molts casos escollir el millor depèn del llenguatge i l'entorn de desenvolupament. El Visual Studio visualstudio.microsoft.com és un dels més complets i utilitzats, i una versió reduïda i gratuïta és el Visual Studio Code code.visualstudio.com/.

A més d'un editor adient, cal tenir un control de versions. Serveis com Github, amb el protocol Git, permeten guardar i restablir versions; a més de poder treballar en equip sobre el mateix codi sense molestar la resta dels companys o companyes.

2.6 Operació i seguretat de l'entorn de producció o desenvolupament

Tot i que el desenvolupament de programes no sembla *a priori* un entorn de risc, el cert és que hi ha perills que son fàcils d'evitar i cal fer-ho. Per exemple, hi ha diverses complicacions i malalties associades a una mala postura, i és evident que els desenvolupadors es passen moltes hores asseguts davant d'una pantalla. Però a més dels riscos físics, també hi ha precaucions que cal prendre respecte a les dades. Cal assegurar-se que les dades no es perden i que es fa tot el possible per protegir-se d'intrusos i de còpies il·legals.

2.6.1 Legislació sobre prevenció de riscos

Els convenis col·lectius i la legislació actual sobre prevenció de riscos tenen apartats concrets sobre la feina d'oficina. Els programadors són, probablement, les persones a qui més s'adreça aquesta legislació, per la quantitat d'hores en postures similars. No parar atenció a aspectes com l'altura de la pantalla respecte als ulls o una postura adequada pot provocar **articulacions adolorides** al cap d'un temps o dolor crònic. Actualment hi ha solucions, com les taules per treballar dret. Fins i

tot es poden trobar al mercat solucions per poder treballar en una cinta mecànica mentre es camina.

Cada persona té les seves preferències i potser també d'altres limitacions paral·leles, per tant, davant del dubte cal preguntar als experts. Les empreses d'una certa mida estan obligades a impartir cursos de prevenció de riscos on es pot preguntar lliurement qualsevol dubte. Si no, sempre es pot preguntar al metge de capçalera.

2.6.2 Aspectes ambientals i eficiència energètica

Actualment pràcticament tota la comunitat científica està d'acord que el clima canvia degut a l'acció humana. Consumim en global molta energia i ho fem alliberant residus al medi ambient. Les conseqüències d'aquest **canvi climàtic** poden ser absolutament devastadores si no es fa res per evitar-ho.

Molts governs i institucions públiques han marcat alguns objectius a escala nacional i global, però és a les mans de tots contribuir a pal·liar el canvi climàtic. És per això que moltes de les grans multinacionals han anunciat plans per neutralitzar el seu impacte a la natura. Per exemple, Apple i Google ja s'han compromès a obtenir tota la seva energia de **fonts renovables**, i Samsung ha anunciat recentment que ho aconseguirà al 2020. Aquestes són notícies transcendentals, ja que són empreses amb desenes de milers de treballadors i una gran quantitat de servidors funcionant les 24 hores del dia, amb la qual cosa el consum que generen és realment elevat.

Tot i que seria desitjable, no totes les empreses han fet aquest pas. De totes maneres, com a individu és possible contribuir a **reduir el consum**. Per exemple, moltes de les màquines tenen diferents modes energètics, i no sempre cal escollir el més potent. En moltes ocasions es poden activar modes ecològics o de baix consum. A més a més, també es poden escollir equips i components en funció del consum. També, és clar, és un bon costum apagar les màquines quan no es preveu utilitzar-les. Aquesta mesura, a més d'estalviar energia i diners, evita atacs maliciosos dels pirates informàtics (*hackers*).

A més de bones pràctiques, també es poden avaluar les tecnologies en funció del consum. Un exemple molt clar és **Blockchain**. Aquesta és una tecnologia d'emmagatzemament distribuït que té unes característiques molt atractives per a segons quines aplicacions. Això sí, és realment ineficient en l'aspecte del consum energètic. De fet, es calcula que Bitcoin, una de les aplicacions de Blockchain més conegudes i utilitzades, consumeix més que l'energia consumida per molts països junts. Si tenim en compte que tots els projectes que es poden implementar amb Blockchain també es poden implementar amb altres tecnologies com el núvol, una manera molt eficaç de millorar l'eficiència energètica és no fer servir Blockchain, si no és imprescindible.

2.6.3 Permisos d'accés a la informació: controlats i discrecionals

L'accés a la informació és una part clau del desenvolupament en la gran majoria de projectes, més encara en aquells que tractin dades personals dels usuaris. De fet, recentment la Unió Europea ha dictat noves normatives respecte a les **dades personals**. Però no és l'únic cas en què cal tenir molta cura de les dades. Entorns com l'educació amb menors o l'àmbit sanitari són especialment sensibles, i aquells projectes relacionats amb la defensa o amb contractes restrictius respecte a les dades són especialment sensibles. El primer que cal que tingueu clar i definiu de seguida és:

- quin tipus de dades és sensibles i
- quin tipus d'usuaris hi pot accedir.

Per exemple, en un joc hi poden haver diversos perfils: els jugadors, els observadors, els organitzadors i els creadors del joc. Cadascun d'ells ha de poder tenir accés només a les dades que li corresponen. De fet, un dels recursos més utilitzats pels jugadors que volen fer trampes és veure el joc com un observador, per tenir accés a informació que d'altra manera no podrien veure. D'altra banda, típicament els creadors del joc poden voler conèixer l'ús del joc per part dels jugadors per poder-lo mantenir correctament, o per millorar-ne algun aspecte. Això sí, no han de poder veure dades personals o contrasenyes. És important tenir-ho clar, perquè el fet de crear i distribuir una aplicació no ha de donar peu a poder veure la informació privada dels usuaris. Fins i tot amb consentiment explícit d'un usuari només en casos excepcionals s'hi hauria de poder accedir.

El mètode més utilitzat per assegurar accessos correctes és **crear grups d'usuaris**, i marcar a les dades sensibles quins grups d'usuaris hi tenen accés. A més a més, sempre cal xifrar aquestes dades per prevenir que els atacants puguin copiar les bases de dades i saltar-se aquestes restriccions. Després de casos ben sonats d'atacs amb èxit, les empreses han entès que no poden treballar d'una altra manera. Avui en dia ja és excepcional trobar que els atacants han pogut entrar a bases de dades i les han pogut llegir sense problema.

També comença a ser la norma utilitzar **sistemes d'autenticació** amb més d'un pas. En aquest cas, cada pas significa part de la clau. Per exemple, una autenticació en dos passos pot ser: una cosa que l'usuari sap, i una cosa que l'usuari té. Per la ràpida expansió dels telèfons intel·ligents, la combinació sol ser una contrasenya i una interacció al telèfon. Ja es comença a parlar, però, d'autenticacions en tres passos: una contrasenya, una interacció al telèfon i un reconeixement biomètric. Si bé mai és possible garantir la privacitat dels usuaris, amb una autenticació tan sofisticada serà excepcional ser vulnerables als atacs.

3. Organització i catalogació de continguts, fonts i mòduls d'informació

Per ser eficient a l'hora de desenvolupar i mantenir un projecte d'aplicacions multimèdia interactives (AMI) complex, cal tenir tota la **informació organitzada** amb uns criteris clars. Aquesta necessitat es torna més evident quan es treballa en equip. No només ha de quedar **normalitzat** l'accés al contingut, sinó que el contingut que es produeixi també ha de seguir unes pautes per tal que sigui tan **homogeni** com sigui possible. En cas contrari, l'usuari final es podria trobar amb productes poc cuidats, on s'evidenciaria la feina de múltiples persones de manera poc cohesionada.

Finalment, és molt important conèixer de manera tècnica com fer servir les eines adients per mantenir segura tota la informació, guardar còpies de seguretat i poder **treballar en paral·lel** en millores sense molestar la resta de l'equip.

3.1 Valoració de la consistència, pertinència i qualitat dels continguts i/o fonts

En qualsevol projecte de desenvolupament mitjanament complex es generen una **gran quantitat de documents** associats, al marge del desenvolupament i el codi. Per exemple, en jocs hi ha els documents de disseny del joc o GDD (*Game Design Documents*), que resumeixen el concepte d'un joc, amb prou informació per poder posar d'acord tot un grup i establir-los com una guia bàsica. Aquests documents poden ser molt extensos i plens d'il·lustracions. Altres casos, com ara documents per aplicar a projectes europeus, poden ser inclús més caòtics. Aquest tipus de projectes típicament requereix que diverses institucions, públiques i privades i localitzades en diferents països, **treballin conjuntament** en els mateixos documents.

És pràcticament impossible que una sola persona escrigui tot el material. Això implica que els continguts **s'elaboren en grup**. Cada persona pot tenir una opinió i estil diferent, i per tant cal posar-ho en comú. En projectes amb càrrega visual alta, com poden ser els videojocs, aquest és un factor determinant, ja que des del punt de vista de l'usuari cal que tot el joc tingui **un estil homogeni**. També des del punt de vista de la programació és important, ja que cal definir un patró clar per a les convencions dels noms de classes, atributs i mètodes. En cas contrari, el codi es pot tornar caòtic i es pot perdre molt de temps buscant noms.

A més de les qüestions estètiques i pràctiques, poden sorgir també multitud de **problemes tècnics**. Els projectes com un videojoc poden tenir una altíssima complexitat, amb multitud de disciplines treballant pel mateix objectiu. Tots els

equips han d'estar coordinats respecte als tipus de fitxers, programaris, versions i sistemes operatius que cal fer servir, per evitar incompatibilitats. És més, fins i tot amb tots aquests aspectes sota control, poden sorgir problemes com l'orientació arbitrària d'objectes 3D, els centres o la quantitat de triangles de les malles.

Cal mantenir uns **criteris clars** en qüestions estètiques (seguir un estil homogeni), pràctiques (treballar conjuntament) i tècniques (evitar incompatibilitats), perquè si no es fa hi pot haver, fàcilment, errors costosos en temps o pèrdua de rendiment per excés de detall.

3.1.1 Unitat estilística (estètica i narrativa)

Des del punt de vista de la unitat estilística estètica, el més important és **definir jeràrquicament** l'estil del projecte. Una de les primeres decisions que s'ha de prendre és si el projecte és en 2D o en 3D. A partir d'aquesta resolució moltes preguntes estilístiques ja poden quedar resoltes. Per exemple, el nombre de polígons per objecte no és rellevant en projectes en 2D. En aquest cas, el següent pas és decidir el **tipus d'art**: és un projecte retro que imita antigues consoles de 8 bits o s'utilitza la màxima resolució possible?, quin és l'espai de color que es farà servir?, es tracta d'un projecte amb un estil visualment més aviat fosc o es vol donar un aspecte més de còmic? Totes aquestes preguntes i més qüestions similars han de trobar resposta.

Els projectes en 3D tenen complexitats afegides. A més de preguntar-se el mateix que en el cas de projectes en 2D, els equips que desenvolupin l'aplicació s'han de posar d'acord en altres aspectes tècnics. El **nombre de triangles** que es mostrarà per pantalla està directament relacionat amb el rendiment d'una aplicació en 3D. Això vol dir que si tenim massa personatges, personatges amb massa detall o escenes massa complexes, l'aplicació perd fluïdesa i l'experiència d'usuari es veu perjudicada ràpidament. També aspectes com la il·luminació i la interactivitat són un factor fonamental que clarament afecta el rendiment. Aquestes qüestions, però, estan lligades a la **qualitat visual** resultant.

Per exemple, una aplicació en 3D amb unes malles de molt baixa resolució poligonal, una il·luminació amb poc detall i colors plans és coherent. En canvi, si tingués una il·luminació i unes textures realistes, però malles amb un mínim de polígons, no quedaria gens coherent. Per tant, cal tenir en compte els requisits globalment i definir criteris des del punt de vista estètic i tècnic.

La **unitat estilística narrativa** és probablement més senzilla d'establir en forma de normes, però la més difícil de concretar detalladament. Cal decidir els temps verbals que es faran servir (quin passat, per exemple), si hi ha narrador o no, si l'enfoc és més en els diàlegs o en la descripció del context, si les descripcions són extenses o curtes... Com veieu, alguns d'aquests elements no són sempre senzills d'acotar. Per exemple, és difícil acotar què vol dir una descripció extensa.

També hi ha factors molt específics que es poden definir per homogeneïtzar documents narrativament. Per exemple, les **referències a d'altres texts**. En documents tècnics és del tot habitual referir-se a contingut ja publicat per reforçar algun argument o per comparar alguna idea, entre d'altres motius. Per no haver de repetir tot el contingut de la font, es pot marcar una referència per tal que el lector pugui trobar fàcilment el text referenciat. Aquestes referències gairebé sempre tenen dues parts:

- Una que queda inserida en el moment en què es necessita fer la referència i
- una altra que és la indicació completa de la font, que queda normalment cap al final, agrupada amb tota la resta de referències del document en una secció que es diu **bibliografia**.

Com que és una operació molt habitual, hi ha diversos estàndards que marquen de manera molt precisa tant el format de la referència com el de la bibliografia. Per exemple, hi ha el format American Psychology Association (APA) o el Vancouver, que són molt diferents visualment. Com que és bastant feixuc estar pendent de tots els detalls de format de les referències, hi ha moltes eines per a gestionar-les.

Referències bibliogràfiques

Trobareu informació sobre com citar i gestionar la vostra bibliografia al següent enllaç: goo.gl/ZYdxPg.

3.1.2 Requisits d'adaptació, d'edició o de reelaboració

Un cas que es pot donar freqüentment és començar un projecte que es basi en **materials ja existents**. En el món dels videojocs aquest cas és del tot comú perquè molts jocs tenen segones, terceres o més parts. En general, en aquests casos, què cal fer-ne dels documents i arxius ja existents? Per contestar aquesta pregunta, es distingeixen clarament dos casos:

- **Documents explicatius**: en aquest cas, cal homogeneïtzar els criteris entre projectes, per tal d'aconseguir una unitat estilística, ja sigui reescrivint i adaptant els documents antics al format nou, o bé modificant els criteris del nou per aconseguir el mateix objectiu. La decisió depèn fonamentalment de la importància dels nous criteris, i de com de fragmentats estiguin els documents anteriors. Es tracta d'estalviar feina innecessària.
- **Fonts d'arxius d'altres menes**: en el cas dels arxius que no siguin text (imatges, vídeo, so, animacions...) és molt probable que calgui refer-los. La tecnologia avança molt ràpidament, i projectes d'un o dos anys enrere es poden veure antics. De fet, els productors de videojocs o pel·lícules d'animació normalment ja compten que rarament es pot reciclar contingut i dissenyen equips i pressupostos partint d'aquest fet. Per tant, cal afrontar-ho amb la mentalitat de fer-ho tot de nou.

Les excepcions a aquest últim cas són àmplies, però. Les bandes sonores ben produïdes i reconegudes no només es poden reciclar sinó que és molt positiu fer-ho. Repetir banda sonora si és bona causa una associació positiva amb els

entusiastes dels productes que no és fàcil de reproduir. També en general es poden reciclar tots aquells arxius per omplir decoració. Per exemple, reciclar animacions de personatges en segon o tercer pla pot tenir un impacte visual mínim i **estalviar molt temps de producció**. Finalment, aquells projectes molt específics visualment admeten molt fàcilment integrar els arxius existents. Els videojocs amb temàtica 8 bits en són un bon exemple, ja que per molt que la tecnologia evolucioni, els 8 bits de color seguiran sent els mateixos.

Cal dir que, en general, el sentit comú i l'experiència donen respostes senzilles i intuïtives a aquesta qüestió. Depenent del tipus d'arxiu i dels projectes, caldrà adaptar o refer part del material.

3.1.3 Formats adequats d'arxiu

Hi ha moltes aplicacions que treballen amb fitxers per manegar dades. Un exemple molt clar són les **aplicacions ofimàtiques**. El problema més comú sorgeix quan es fan servir diferents aplicacions o diferents versions que no són compatibles entre elles, o que causen errors. Tot i que cada cop es va resolent més, aquest era el problema més comú de les aplicacions ofimàtiques fa uns anys.

L'arrel del problema és que tot sovint s'introdueixen millores en les aplicacions que **no estaven previstes** de bon començament. En aquests casos, les novetats només són compatibles amb les versions noves. D'aquesta manera, els fitxers generats amb les versions més modernes deixen de ser compatibles amb les versions antigues. Si bé és impossible preveure exactament el futur, el cert és que hi ha bones pràctiques per evitar aquests problemes. La més comuna és **establir una única versió** de les aplicacions feta servir per tots els membres d'un equip per a un projecte, i mantenir la versió durant tot el procés.

Alhora, es busca que els desenvolupaments dels projectes tinguin **cicles curts** per diverses raons, i aquesta n'és una: un desenvolupament que duri massa pot començar amb eines modernes i que en un moment donat aquestes eines ja estiguin fora de mercat. En el millor dels casos, el resultat serà una aplicació pobra visualment, però pot arribar el cas que fins i tot calgui cancel·lar el projecte. Un efecte tan negatiu es podria evitar actualitzant les versions a mig projecte, però les conseqüències poden ser imprevisibles i causar molts problemes i retards. En la gran majoria dels casos, és molt millor planificar **separar en diverses versions** cada projecte, limitant-ne l'abast i reduint-ne el cicle de desenvolupament.

3.1.4 Criteris d'avaluació, llistes de control i verificació

Avaluar continguts és un procés semblant en molts casos a llançar una aplicació al mercat. Es tracta d'anar **iterant i millorant** o afegint contingut fins a tenir versions que tinguin prou qualitat per donar-les per bones. Aquest últim pas, donar per

bona una versió d'un document o un fitxer, requereix tenir **criteris definits**. Per exemple, podem diferenciar entre:

El terme *iterar* és molt comú en el desenvolupament de projectes, i es refereix a acabar una versió (pot ser incompleta) que tingui sentit com a entitat, i tornar a començar per millorar-la o afegir-hi funcionalitat.

- **Criteri d'acabament d'un document de text**: abans de començar a omplir un document de text, caldria tenir clar quines parts el formen. Formar un índex i definir aproximadament un gruix esperat és una bona pràctica. D'aquesta manera, podem establir un criteri mínim d'acabament.
- **Criteri de correcció d'una animació**: una animació de caminar es pot considerar completa quan es pugui veure un cicle continu sense salts, de manera que no es vegi la diferència entre passes. També quan totes les animacions secundàries (mans, cara, esquena...) estiguin acabades.

Però la casuística possible de criteris és tan gran que és molt complicat descriure-ho en un curs com aquest. En resum, el que cal tenir en compte és que és fonamental **mantenir una consistència** dins de tot el material generat per un equip. No només des del punt de vista estètic, sinó també tècnic. Cal definir criteris estètics i tècnics i assegurar-se que se segueixen dins de l'equip.

3.2 Determinació dels mòduls d'informació del producte AMI

Un projecte audiovisual multimèdia interactiu (AMI) sol tenir una **càrrega narrativa** important; per exemple, els videojocs amb mode història s'han de plantejar **com afrontar la història** que volen explicar. En molts casos es tracta d'explicar una història o ambientar l'usuari en un cert context. És important determinar quin tipus de modalitat narrativa cal utilitzar: lineal o interactiva.

Trobareu més informació sobre la narrativa linela o interactiva a la secció "Narrativa lineal i interactiva: estructura seqüencial determinada i modular" dins de l'apartat "Determinació d'objectius, estil gràfic i narratiu d'un projecte" d'aquesta mateixa unitat formativa.

A les **seqüències lineals** està predefinit el camí que seguirà l'usuari i el final que s'espera. No heu de confondre seqüències lineals amb narratives lineals. Una narrativa lineal es refereix a una progressió temporal de principi a fi sense salts en el temps o referències al passat o futur. Es pot donar que tinguem salts temporals, com en jocs ben coneguts com *Prince of Persia*, però seguir seqüències lineals. Es pot donar més o menys sensació de llibertat al jugador, però el cert és que haurà de passar per **un nombre definit d'etapes** fins a acabar el joc, i l'únic que definirà la progressió serà l'habilitat del jugador, a més del temps invertit.

El gran avantatge d'aquesta modalitat és que els projectes són més senzills en gairebé tots els casos. Són més fàcils de produir, de programar, de mantenir, de comunicar... No tot és perfecte, però, i és que els jugadors trobaran a faltar aquesta llibertat de moviments. A més a més, alguns gèneres de jocs, com els RPG, difícilment encaixen amb seqüències lineals.

En canvi, les **seqüències interactives** són bastant més complexes, però encaixen en gairebé tot tipus de gèneres de joc, i els jugadors ho saben apreciar. De fet, des de fa un temps moltes de les produccions amb més tirada comercial opten per aquesta modalitat. Aquí cal definir bé l'abast de la interactivitat, com es produeix, i quines conseqüències té.

Per exemple, el *Knights of the old republic* ofereix una narrativa interactiva limitada: el jugador pot fer accions bones i prendre el camí de la llum, o accions dolentes i prendre el camí de l'obscuritat. La diferència no és gaire gran ja que el joc té les mateixes etapes i només afecta cap al final, on és capaç de fer servir algunes habilitats en comptes d'unes altres. El cas oposat, un dels exemples més interessants de narrativa interactiva amb més possibilitats per al jugador, és el joc *The Stanley parable*. En aquest cas, hi ha una combinatòria realment gran d'accions que pot fer el jugador i que porten a finals totalment diferents.

La tria de la **narrativa** final té un gran impacte dins del desenvolupament i hi pot afegir una gran complexitat. Per tant, cal rumiar quina és la millor opció. A més a més, cal determinar quina **interactivitat** s'atorga a l'usuari, i quina és la manera de controlar l'acció dins del projecte.

3.2.1 Graus d'interactivitat i de control

La qüestió fonamental és definir quin **tipus d'interacció** pot tenir el jugador. Es poden agrupar en dos tipus:

- **Un únic final**: aquelles produccions que tinguin un únic final poden també introduir interactivitat per donar la **sensació de llibertat** d'escollir el destí del jugador. Un mètode que ja comença a ser comú és definir un mapa ampli on el jugador és lliure d'anar on vulgui. En punts concrets del mapa, però, s'activen diferents mecanismes d'interacció. Aquest és un exemple d'una manera híbrida d'abordar les seqüencies lineals introduint interactivitat.
- **Multitud de finals**: els projectes que tenen diferents finals introdueixen situacions on el jugador ha de prendre una decisió que afecta la resta del joc. En aquest cas, un dels problemes és que pot ser que molts jugadors **no gaudeixin mai de bona part del joc**, ja que no sempre repetiran per fer diferents eleccions.

Pel que fa a la **interacció** en si amb els elements d'un joc, el procés més senzill és examinar jocs i projectes AMI amb temàtiques semblants. Les maneres que té un usuari d'interactuar amb l'entorn estan molt ben estudiades, i trobar-ne una de nova que funcioni bé és molt molt difícil. No només és difícil de desenvolupar i posar en marxa, sinó de comunicar als usuaris.

Malauradament, la majoria d'usuaris **rebutgen haver d'aprendre noves interaccions**, i aquest fet sol ja pot tombar un projecte amb una gran qualitat. En molts casos és preferible invertir recursos en la resta del projecte i fer servir modes d'interacció ben coneguts.

3.3 Classificació, reestructuració i organització de la informació

Les produccions amb diferents tipus d'arxius i dades, com podrien ser els videojocs, són un repte perquè abasten una gran quantitat de casos. Per això és important tant l'organització de la informació, com la seva classificació, catalogació i indexació.

Catalogar consisteix en descriure un contingut de manera succinta o establir un criteri d'organització per tal que es pugui trobar fàcilment. Per exemple, un criteri de catalogació pot ser ordenar alfabèticament o de manera jeràrquica: ordenar per tipus d'arxiu (documents, multimèdia, animació...) i dins de cada categoria, fer-ho per nom. En qualsevol cas, per ser eficient, la catalogació ha de complir una sèrie de requisits:

- Ha de ser unívoca, és a dir, que seguir els criteris ha de portar a un únic arxiu.
- Ha d'aconseguir una uniformitat del contingut.
- Ha de poder fer que sigui senzill tant guardar com extreure contingut, des d'un sol sistema o des de sistemes diferents.

Poder **trobar la informació necessària en qualsevol moment** és vital per a la gran majoria de projectes. Per fer-ho, és necessari tenir eines i criteris per organitzar i catalogar les dades.

Un cop establerts els criteris, cal ser sempre **metòdics amb les dades i arxius**. És molt fàcil que a mesura que creix un projecte s'hi comencin a afegir molts arxius, i la prioritat deixi de ser tenir les dades ben organitzades. Això, però, a la llarga és molt perjudicial per als equips. D'altra banda, ser massa estrictes amb l'estructura pot causar una rigidesa innecessària i una lentitud o sobretreball que només endarrereixen el projecte.

Una manera d'evitar aquests problemes és que els equips facin servir repositoris, i els individus puguin fer totes les proves necessàries en local o amb les branques de desenvolupament, i després organitzar-ho tot abans d'incloure-ho a les branques comunes. Una altra manera és fent servir eines d'administració de mitjans digitals.

3.3.1 Eines d'administració de mitjans digitals

Administrar fonts de diversos tipus és més complex del que pot semblar a primera vista. No només es tracta de tenir-los disponibles, sinó d'assegurar-se que només hi puguin accedir els membres d'un equip que siguin necessaris, que es guardin diverses versions, que es puguin compartir...

A més a més, no sempre es vol restringir l'ús dels arxius per un únic projecte o inclús un únic grup. Arxius de gran qualitat tenen un **valor intrínsec** al marge de cada projecte, i es poden oferir a d'altres persones a canvi de diners o d'algun acord. D'aquesta manera, podeu ingressar diners o expandir la vostra marca no només amb productes, sinó amb els arxius que conformen aquests productes.

Teniu diverses eines de ***Digital Asset Management*** **(DAM)** que fan aquesta feina. Per exemple, Bynder (www.bynder.com) té una versió gratuïta. Aquesta eina permet guardar arxius al núvol, compartir-los amb més usuaris i fer cerques avançades. També hi ha eines integrades dins d'eines de desenvolupament. Per exemple, Unity incorpora una botiga on desenvolupadors i artistes poden penjar la seva feina de manera gratuïta o cobrant a *Unity Asset Store*.

A més a més d'oferir aquests intercanvis, els DAM solen oferir també estadístiques per tal que els autors puguin saber què millorar, i també eines per poder parlar amb els usuaris.

3.3.2 Reagrupament i reestructuració de la informació

Tot i ser meticulosos amb les pautes i els criteris definits, és molt possible trobar-se que cal canviar alguns aspectes de l'organització. És impossible predir-ho tot de bon començament, i per tant poden sorgir **canvis de requisits a mig desenvolupament**, o bé simplement veure que una proposta no funciona.

Donada la necessitat de fer canvis, cal afrontar-ho de manera ordenada. El primer pas és establir quins són els problemes que han sorgit, o quines noves necessitats són necessàries. Després cal consensuar amb l'equip les noves normes i finalment assegurar-se que tothom les ha entès.

La casuística possible és massa àmplia per fer una descripció detallada de totes les passes necessàries en aquests processos. Això si, podem destacar bones pràctiques a la implementació de reestructuracions, sobretot per a aquells projectes que són públics, o que tenen milers d'usuaris o més:

- La primera bona pràctica és **escollir un moment per executar el canvi que no sigui crític**. L'ús de qualsevol aplicació massiva o dels fitxers d'un projecte amb molts membres fluctua. Els treballadors deixen de treballar després de la seva jornada laboral, i els usuaris tenen en general un comportament diferent els dies festius que els dies feiners. Heu d'escollir un moment on es pugui predir que la demanda d'informació serà mínima, ja que molt probablement caldrà restringir o tallar l'accés durant el procés.
- La segona pràctica és **no introduir canvis ni reestructurar** just abans d'un període on no hi haurà ningú per mantenir-ho. Per exemple, els divendres solen ser dels dies prohibits a moltes empreses per a publicar canvis, ja que si hi ha un error, durant el cap de setmana no hi haurà ningú per arreglar-lo.

3.3.3 Metadades: processament i recuperació de la informació

Qualsevol arxiu conté bastant més informació que **no només el seu contingut**. Per exemple, cada arxiu té una mida, un tipus, un autor, un autor de l'última modificació, una data de creació i de modificacions, una sèrie de permisos, quin programa s'ha utilitzat per modificar o crear... A més, depenent del tipus d'arxiu, hi pot haver més metadades. Per exemple, els vídeos tenen una llargada, un còdec, un *bit rate*, una resolució...

> Les **metadades** són un recull d'informació addicional (com per exemple, data de creació, autor, mida...) que estan associades a un arxiu digital o recurs.

Tota aquesta informació pot fer-se indispensable. Per exemple, molts projectes tenen diferents versions segons el dispositiu o la gamma de dispositius als quals es dirigeixen. Per exemple, no té gaire sentit fer servir gràfics en alta definició si la pantalla del dispositiu amb què s'està gaudint no ho és.

A banda, moltes d'aquestes metadades s'omplen i es processen automàticament. Gràcies a això, es poden establir **criteris de catalogació senzills**, ràpids i eficaços que minimitzin l'acció dels usuaris. També hi ha molts sistemes automatitzats que poden llegir aquestes metadades molt ràpid, per tant és molt eficient a l'hora de fer cerques i recuperar informació.

Avui en dia les metadades són cada cop més presents gràcies a la **intel·ligència artificial**. Aquests sistemes comencen a ser capaços d'analitzar el contingut i extreure patrons i conclusions d'alt nivell. Per exemple, ja és habitual trobar serveis que puguin distingir tipus d'animals a fotografies. Aquesta informació es pot incorporar al conjunt de metadades, per accelerar les cerques.

3.3.4 Diagramació dels continguts organitzats

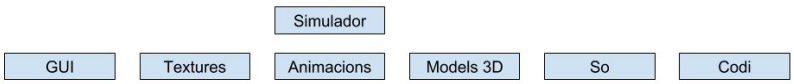
Com que els criteris de catalogació poden ser jeràrquics, es poden dibuixar perfectament en diagrames senzills i ben entenedors. Aquesta és una molt bona manera de **comunicar-ho** a un equip o al públic en general.

Vegeu ara un exemple d'un diagrama d'organització dels fitxers d'un videojoc. Imagineu que es vol fer un joc de simulació de futbol. Com podeu imaginar, hi haurà una gran diversitat de fonts. Contindrà models en 3D dels jugadors, les seves textures, les seves animacions, el so, la intel·ligència artificial... A més, el projecte tindrà el so dels comentaristes, en diferents idiomes, les textures de les interfícies, tutorials...

La pregunta pot ser: com cal començar a organitzar-se? No hi ha una única resposta vàlida. Depèn de les dinàmiques de treball dels equips. En projectes tan grans els equips solen ser petits i tenir líders que es comuniquen amb altres

líders d'equips, i les estructures de fitxers poden reflectir aquesta organització. Per tant, la figura 3.1 podria ser una solució inicial, de manera simple.

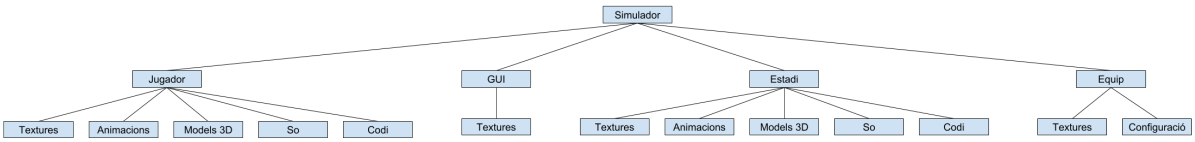
Figura 3.1. Solució inicial de diagrama d'organització de continguts



El problema d'aquesta organització és que no facilita la col·laboració. Per exemple, simular un futbolista és més que un model 3D. Li calen les animacions, les textures... De fet, a l'hora de dissenyar i provar el comportament d'un futbolista cal poder visualitzar el resultat i fer les modificacions apropiades. Per tant, tot i que aquests criteris són clars, **no faciliten els projectes multidisciplinaris**.

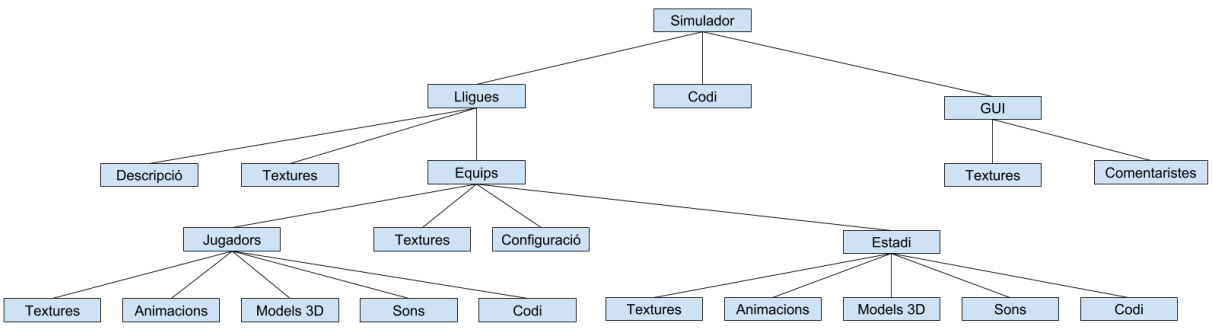
Una alternativa més adient en aquests casos és formar **unitats autocontingudes**. Per exemple, tot el que és necessari perquè un equip pugui dissenyar un jugador de manera completa. En el diagrama de la figura figura 3.2 es pot veure un exemple d'aquesta organització.

Figura 3.2. Solució de diagrama d'organització de continguts seguint els fluxos de producció



Però encara es pot millorar. Els equips que treballen amb models 3D o textures normalment ho fan de manera indiferent de si es tracta d'un model o un altre. A més a més, un partit de futbol no té només un jugador, sinó fins a 28. I també els equips solen estar en una lliga, que al seu torn també té unes característiques. Per tant, es pot jerarquitzar encara més. En el diagrama de la figura figura 3.3 es poden veure en plural totes aquelles classificacions que tindran múltiples instàncies. Per exemple, un equip tindrà múltiples jugadors, i una lliga múltiples equips.

Figura 3.3. Solució de diagrama d'organització de continguts aplicant jerarquies



En resum, és fonamental **determinar els mòduls d'informació dels projectes AMI**. Establir criteris per organitzar i classificar la informació de manera consistent i sistemàtica no només és una bona pràctica, sinó que és clau per establir processos eficients. Penseu que no hi ha una única solució a cada context, sinó que depèn de les dinàmiques de cada equip o conjunt d'equips. El mateix projecte amb un abast més petit o més gran pot requerir diferents maneres d'estructurar els fitxers i diferents criteris d'organització.

3.4 Els drets d'autor i les seves característiques

Per protegir un projecte o una idea hi ha els següents mecanismes:

- **Patent**: serveix per garantir el dret exclusiu d'ús d'una idea per part del grup que obté la patent. A canvi, cal publicar com a part de la patent una explicació entenedora que descrigui com arribar a desenvolupar la idea o solució. El requisit fonamental és que aquesta idea sigui nova (no hagi estat publicada anteriorment) i que no sigui trivial. A més, cada geografia (cada país o regió) té algunes particularitats, però el concepte és el mateix. Una particularitat molt propera és la de la Unió Europea: no es poden patentar aplicacions directament. Una manera de poder patentar aplicacions és que formin part d'una solució més global, com per exemple si està associada a una màquina.
- ***Copyright***: és semblant a una patent, però és menys restrictiu. Es pot protegir contra la còpia pràcticament qualsevol concepte. Ja pot ser la lletra d'una cançó o una aplicació, no hi ha restriccions. Això sí, el projecte ha de ser nou. També és un intercanvi, en el sentit que es garanteix aquesta protecció a canvi de publicar el concepte.
- ***Trademark***: no protegeixen idees, sinó la marca. Per exemple, es pot protegir un disseny, un nom, un logotip... Cal tenir en compte, però, que si no es fan servir es perden.
- ***Trade secret***: no és realment una protecció, sinó que és mantenir en secret i no divulgar les idees clau. Lògicament, aquesta no és una protecció ferma perquè els secrets es poden filtrar i acabar en mans de competidors. Aquests casos se solen donar quan membres dels equips o treballadors d'una empresa canvien d'empresa o creen la seva pròpia, perquè ningú pot ser obligat a oblidar el que ha après. En aquestes situacions la forma més habitual de protecció és un contracte amb clàusules que prevenen treballar per competidors durant un cert temps, i també responsabilitzar legalment els treballadors per mantenir els secrets.

Com es pot veure, les opcions per protegir un projecte o són diverses, i cadascuna té **avantatges i inconvenients**, com també camps concrets d'actuació. En aquest sentit, ens podem fer una sèrie de preguntes:

1. La primera pregunta és si **es pot assumir** que allò que es vol protegir sigui públic. Per exemple, en contractes amb agències de defensa, o amb tecnologies capdavanteres, aquesta no és una opció. Per tant, les patents i el copyright no es poden aplicar. Els *trademarks*, en aquest cas, són irrellevants.

2. La segona pregunta és si ja s'ha publicat i si **serà útil la protecció**, tenint en compte que es converteix en domini públic. Qualsevol publicació explicativa pot ser suficient per anul·lar una sol·licitud de patent. El *copyright*, en canvi, funciona de manera diferent. Aquest últim mètode de protecció protegeix allò que se sol·licita, de manera pràcticament literal. És a dir, que si voleu que el codi de la nostra aplicació tingui *copyright* cal enviar el codi i es protegirà contra codis massa semblants. Val a dir que la frontera respecte al fet que un codi s'assembli a un altre és molt fina, i no sempre es convertirà en una protecció útil. També cal tenir en compte que el codi gairebé mai s'estabilitza en una versió fixa, sense canvis. Per tant, en l'evolució d'aquest codi és molt probable que es perdi l'essència del *copyright*, o que calgui fer-lo de nou. A més a més, convertir el codi en públic vol dir exposar-se a que els competidors aprenguin, el millorin i hi trobin errades. Per tant cal valorar si aquestes són les proteccions més adients, tenint en compte com es transforma en un bé públic.

3. La tercera pregunta és **quina zona geogràfica** ens interessa protegir. Resulta bastant costós sol·licitar i mantenir patents a cada país. Per tant, si no teniu previst entrar i operar en alguns països, podem deixar-los de banda i enfocar els recursos en les zones que sí que seran rellevants. Això sí, com que hem fet públiques les nostres dades, ens arrisquem a que competidors d'àrees desprotegides desenvolupin els seus projectes als seus països d'origen. En el cas, però, de voler ampliar la protecció dels nostres projectes a d'altres països, hi ha diversos mètodes.

4. Finalment, cal preguntar-se si **realment cal invertir recursos** en protecció de drets d'autor. En una indústria tan ràpida com la tecnològica, aquests processos lents i costosos poden alentir el desenvolupament dels productes, que al final són el que decidiran el futur de cada projecte. És important fer un pas enrere i tenir ben clar què és important protegir i què no, i sempre consultar experts.

En resum, hi ha diversos mètodes per protegir les nostres creacions. Cadascun d'ells té les seves característiques i limitacions, per tant, cal examinar quin tipus de protecció és possible i adient. També cal examinar detalladament quines són les zones geogràfiques que cal protegir i prendre les decisions adients.

No sempre protegir tot el possible serà **la millor opció**, ja que implica invertir en uns recursos aliens al producte que, d'altra manera, es podrien invertir en el producte en si.

3.5 Sistemes d'emmagatzematge, còpies de seguretat i control de versions

Feu ara un exercici de memòria i recordeu tots els suports físics on hàgiu guardat arxius. Des de CD, DVD, *pen drives*, discs durs, emmagatzemament al núvol, i més antigament també disquets o cassets. Segur que heu tingut a casa molts d'aquests sistemes, però amb el pas del temps ja no serieu capaços de recuperar la informació. En alguns casos, perquè s'ha perdut el suport, en d'altres perquè ja no funciona, o fins i tot perquè ja no teniu cap aparell capaç de llegir el contingut, perquè han quedat **desfasats**.

Tot sistema de transmissió i emmagatzemament de dades està **subjecte a errades**. Les transmissions es poden tallar degut a una àmplia varietat de motius, des de físics (tallades de cables, que se'n vagi l'electricitat, tempestes electromagnètiques...) fins a per errades humanes, passant per atacs maliciosos. L'emmagatzemament té el mateix problema, i és que qualsevol sistema al final es pot trencar i pot sofrir el pas del temps. De fet, tots els suports físics d'emmagatzemament tenen una garantia i uns períodes estimats de duració. Pel desgast que pateixen els materials al fer escriptures i lectures, com més operacions d'aquesta mena es facin, abans deixaran de funcionar. Tot i semblar que un disc dur està en perfectes condicions, poden haver-hi parts que no funcionin correctament. El resultat serà que de tant en tant la informació no es podrà guardar o llegir correctament.

Quan hi ha un problema amb un suport físic, tant de lectura com d'escriptura, el resultat és que quan es volen llegir les dades el resultat és incoherent. L'aplicació encarregada d'utilitzar les dades o el sistema operatiu, doncs, en el millor dels casos emet un missatge d'error dient que les dades no es poden interpretar. En el pitjor, però, causarà un error i l'aplicació o el sistema operatiu sencer es pararan. La primera conseqüència d'aquest procés és que com a programadors heu de ser conscients que aquests casos es poden donar. Per tant, heu de controlar sempre cada lectura per possibles **errors d'integritat**.

La solució més òbvia a aquest problema podria ser tenir **còpies de seguretat**, guardant diversos cops cada fitxer en diferents carpetes o suports físics. Això, però, no és una solució gens eficient, ja que voldria dir ocupar el doble d'espai sempre, mentre que aquests errors són molt poc freqüents. A més a més, no seria una solució fiable del tot, ja que si de dos fitxers se'n llegeixen dades diferents, tampoc es tindria cap criteri per decidir quin és el fitxer correcte. I, evidentment, afegir un tercer fitxer només faria el sistema més complex.

Una altra solució podria ser anar escrivint sempre **fitxers diferents**, i conservar sempre el fitxer anterior. En aquest cas, perdre un dels fitxers antics no suposaria cap problema, però no serviria de res en cas de pèrdua del més recent. I també suposa un ús encara menys eficient dels recursos, ja que potencialment hi hauria una infinitat de còpies de qualsevol fitxer.

Aquests sistemes estan àmpliament superats pels **sistemes moderns** de còpies de seguretat, de protecció de dades amb redundància i pels repositoris. Amb aquests

sistemes no només es poden guardar fitxers amb molta més seguretat respecte a fallades, sinó de manera eficient, i fins i tot transparent de cara al sistema operatiu. Les còpies de seguretat i els repositoris funcionen de manera molt diferent, però són complementaris. Si bé conèixer el detall complet de tots els sistemes no és necessari d'entrada, és important conèixer-ne la base.

Com a programadors heu de vetllar per la **integritat i la disponibilitat de la informació**. Hi ha dos problemes extremadament comuns que, en el pitjor dels casos, poden enfonsar qualsevol empresa o projecte: la pèrdua de dades i introduir canvis que continguin errades. Per solucionar aquests problemes teniu dues eines ben provades al món real: les còpies de seguretat i els repositoris.

3.5.1 Sistemes de suport i recuperació de dades

El cas ideal de protecció contra la pèrdua de dades és aquell que **no requereix cap tipus d'intervenció** de l'usuari ni del programador. Però, com aconseguir-ho? Recordeu l'origen de les fallades d'escriptura o lectura: tot es basa al final en el fet que els suports físics es degraden al cap del temps i de l'ús. Però no tot deixa de funcionar de cop. De fet, la gran majoria de vegades, només una petita part dels suports físics deixa de funcionar. Per exemple, un disc dur internament està dividit en petites parcel·les de memòria permanent, que es diuen sectors. I cada disc dur té milions de sectors. Doncs un sector dins d'un disc dur es pot espatllar i donar sempre error. I un fitxer pot ocupar diversos sectors, milers, fins i tot, i per tant deixar de ser llegible només pels errors produïts en un sector. Com s'evita aquest problema?

Els sistemes ***Redundant Array of Independent Discs*** **(RAID)** s'aprofiten del fet que no tot deixa de funcionar de cop, i que dos discs durs no comencen a donar errades alhora. De fet, es pot esperar que hi hagi un marge prou gran de solapament on, si un falla, l'altre encara funcionarà durant un temps suficient per recuperar la informació i arreglar el disc dur que falli.

Hi ha diferents configuracions de RAID, segons les necessitats i els components que es configurin, però el concepte en el fons és el mateix. La idea és tenir la informació distribuïda i **de manera redundant** en diversos discs durs. Però oi que tenir redundància amb les dades porta a errors i problemes? Sí, la redundància amb les dades és molt problemàtica i té un cost, però en aquest cas la gestió es fa en l'àmbit del maquinari i a un nivell molt primitiu dels controladors dels discs durs. D'aquesta manera, fins i tot el sistema operatiu pensa que és un únic disc dur i no una unitat més complexa. I com que és compatible amb discs durs de gammes baixes, l'augment de cost pel fet de necessitar més discs durs per la mateixa quantitat d'informació no és tan alt, i es pot justificar bé pels beneficis que se n'obté.

Cal dir també que no totes les configuracions RAID estan pensades per **salvaguardar la integritat de les dades**, i per tant cal que les conegueu i les sapigueu aplicar quan convingui:

- Per exemple, la primera configuració RAID, el **RAID 0**, està pensada per a augmentar la velocitat d'accés a les dades. La manera com ho fa és fent servir dos discs durs i posant la meitat de la informació en cadascun. D'aquesta manera, quan es vol llegir un fitxer, es demana la informació als dos discs durs alhora, i el resultat és que la informació arriba el doble de ràpid. Com a nota important, en aquest sistema, si es perd un disc dur o inclús una part, es perd tota la informació.
- **RAID 1** és una configuració simple però eficaç a l'hora d'assegurar la protecció de les dades. En aquest cas, el sistema simplement utilitza un dels disc durs com a còpia exacta de l'altre. Aquí el sacrifici és que necessitarem el doble de discs durs, i que el sistema només podrà oferir tant d'espai com la mida del disc dur més petit. A canvi, si s'espatlla qualsevol dels dos discs durs es pot continuar i recuperar les dades sense cap problema. La resta de configuracions treballen a base de tenir com a mínim tres discs durs, on un d'ells emmagatzema el resultat d'una combinació de les dades dels altres discs durs. L'avantatge d'aquestes configuracions és que no cal duplicar la capacitat dels discs durs, sinó que amb només un disc dur extra ja n'hi ha prou.
- Finalment, també hi ha configuracions niades, en les quals es poden dividir parts dels discs durs per configurar-les com algun tipus de RAID, mentre que la connexió entre els diversos discs durs es configura com un altre RAID. Per exemple, es pot tenir un **RAID 0+1**, on dos discs durs estan configurats en RAID 1, i alhora cada disc dur està subdividit en RAID 0 internament.

Respecte als sistemes físics que s'utilitzen actualment en els entorns laborals, la resposta és que depèn molt de les **necessitats de cada projecte o servei**, i concretament de si les dades es necessiten habitualment. Per exemple, en entorns com els bancs, en la gran majoria de casos es necessita només una informació recent. Les transaccions antigues, aquelles fetes anys enrere, gairebé mai es consulten (reflexioneu si cerqueu habitualment les vostres transaccions més antigues, el més probable és que només consulteu el que hagi passat en els darrers mesos o setmanes). En canvi, dades com el perfil d'usuari d'una xarxa social han d'estar llestes en qualsevol moment, i a més la velocitat d'accés pot determinar l'èxit de la xarxa. Imagineu si faríeu servir una xarxa social on haguéssiu d'esperar un minut per cada perfil que volguéssiu mirar. I tampoc seria un problema crucial trobar-se amb un perfil no actualitzat.

Aquestes diferències marquen el tipus d'emmagatzemament i l'estratègia de seguretat i recuperació de dades. En aquells casos on les dades són **crítiques, però no necessàries** en qualsevol moment, els suports físics que es fan servir habitualment són discs durs amb configuracions RAID 1 i còpies de seguretat amb sistemes de cintes magnètiques. Els sistemes magnètics fa temps que no es veuen dins l'àmbit particular i domèstic, però és una tecnologia molt provada, barata i

que ocupa molt poc espai físic per byte (són molt semblants a les cassets de ja fa uns anys). En canvi, els sistemes que necessiten donar **resposta immediata** actualment fan servir discs durs SSD professionals en configuracions RAID 3+0 o 5+0. També és habitual que ho combinin amb sistemes de seguretat basats en cintes, però d'una manera menys metòdica i regular que l'altre cas.

En empreses més petites, però, des de fa un temps la solució és més senzilla. Es poden fer servir serveis d'**emmagatzemament al núvol**. Aquests serveis típicament consisteixen en definir una carpeta concreta on tot el que hi hagi dins es copia a un espai remot, establint una connexió via internet. Aquest espai remot està gestionat per empreses com Google o Dropbox, i de manera transparent per a l'usuari fan servir els mateixos sistemes que les empreses grans amb necessitats importants d'accés a les dades (configuracions RAID o còpies de seguretat amb cintes, per exemple). En aquests casos, un avantatge afegit és que els arxius o carpetes es poden compartir amb diversos usuaris, i que qualsevol canvi es sincronitza després amb tots els usuaris.

En tot cas, qualsevol equip preocupat per protegir les dades pot seguir una sèrie de bones pràctiques ben senzilles, com ara:

- La **regla del 3-2-1**, que indica que heu de procurar que hi hagi tres còpies de les dades més sensibles, en dos dispositius diferents, on almenys un sigui fora del lloc habitual de treball. Si només teniu dues còpies d'un arxiu i són diferents, no sereu capaços d'identificar la correcta, però amb tres sí. Aquest procés es pot automatitzar amb *scripts* o programes específics, independentment de les possibles configuracions RAID. De fet, ni la millor configuració de seguretat és capaç de protegir contra possibles errades humanes, com esborrar arxius per error. Per tant, és important tenir aquestes tres còpies i ser metòdic o automatitzar-ho.
- Copiar les dades en dos dispositius diferents protegeix contra les fallades en un dispositiu concret. Si bé és cert que les configuracions RAID poden ser molt robustes pel que fa a errors, no ofereixen cap protecció contra **robatoris, pèrdues, actes de vandalisme, o desastres naturals** entre d'altres. Justament per això també és important que com a mínim un d'aquests dispositius estigui en un altre lloc físicament.
- Per implementar aquestes bones pràctiques, es poden fer servir **serveis en el núvol**, que ja fan la feina automàticament. En ocasions algunes empreses tindran normatives que els impedeixin utilitzar núvols comercials, però això no és cap problema. Hi ha múltiples solucions ja preparades, com per exemple www.resilio.com, per tenir serveis amb la mateixa funcionalitat, però sota el control total de l'administració de l'empresa que ho configuri.

Cal preguntar-se també **amb quina freqüència** cal fer còpies de seguretat, si se'n fan. No hi ha una única resposta: depèn molt de cada cas. Per exemple, en entorns amb equips de poques persones on la velocitat de canvi no és gaire gran, es pot fer **setmanalment**. En el pitjor dels casos es perdrà una setmana de feina de tots els integrants de l'equip, però aquest cas és molt poc probable, sobretot si es fa

servir la regla 3-2-1. Cada membre tindrà còpies de seguretat en local als seus dispositius, i n'hi hauria d'haver algun que no estigui al mateix entorn físic. Per tant, és poc probable i l'impacte no seria tan gran. D'altra banda, amb equips molt grans, perdre potencialment una setmana de treball seria tan car que justifica una despesa major en còpies de seguretat, i per tant fer-les **de manera diària**. Finalment, en entorns amb una generació molt alta de dades que costen molt de generar (com per exemple en centres d'investigació) les còpies de seguretat podrien ser inclús més freqüents. En tot cas, analitzeu l'entorn i reflexioneu sobre les necessitats.

En el cas, però, de perdre dades tot i tenir les proteccions adequades heu de saber reaccionar i ser capaços de **recuperar les màximes dades possibles**. Afortunadament hi ha moltes eines que ajuden en aquesta tasca. De fet, fins i tot hi ha blogs amb entrades dedicades a comparar diversos programes de recuperació de dades, com aquest: www.lifewire.com. La idea de tots aquests sistemes és que en moltes ocasions tot i que el disc dur no sigui capaç d'entendre que té informació, els sectors del disc dur la tenen. Això pot ser perquè el disc dur ha sofert algun tipus de problema parcial, i ha calgut canviar-ne alguna part. Molt sovint aquests problemes ocasionen que el disc dur deixi de funcionar correctament, tot i tenir la informació correctament emmagatzemada.

Els programes dedicats a recuperar dades solen ser molt senzills d'utilitzar. Es configuren indicant quins són els discs durs a analitzar, i per als més avançats cal indicar si hi ha algun tipus de configuració RAID. Un cop iniciat el procés només cal esperar, i normalment no triguen gaire. Cal dir que, de vegades, per a algunes dades no queda bé la lectura (al cap i a la fi és normal, esteu intentant recuperar dades d'un disc dur que havia deixat de funcionar correctament). Per tant, els programes poden generar diversos resultats amb diverses interpretacions. Com a usuaris haureu d'**examinar els resultats manualment** i determinar quin és el millor. Seria molt optimista desitjar una recuperació completa de totes les dades i sense errors, tot i que no és descartable. El cas més probable serà una recuperació de la majoria de dades, però alguns arxius es perdran.

3.5.2 Tipus de còpies de seguretat

Una còpia de seguretat es pot fer de diverses maneres. La més senzilla i evident és copiar cada arxiu sencer, i fer-ho així per cada modificació. Aquest sistema funciona i és molt fàcil d'implementar i automatitzar, però és perfectible. Té l'inconvenient de duplicar la necessitat d'espai d'un sol fitxer i de no mantenir versions antigues.

En comptes de fer una còpia completa, alguns sistemes fan **còpies de seguretat incrementals**. D'aquesta manera, només es guarden els nous valors, i no cal tornar a guardar tot el fitxer sencer. Aquesta és una pràctica molt comuna en sistemes on en principi el passat és immutable. Per exemple, les transaccions bancàries un cop fetes queden registrades i ja no poden canviar. De fet, en cas d'error els bancs enregistren una nova operació desfent l'errada, però no modifiquen les dades

inicials. Un altre exemple poden ser els videojocs. Molts estudis volen veure com els jugadors fan servir els seus productes. Per fer-ho, enregistren totes les accions rellevants a mesura que els usuaris les activen. Lògicament, aquí interessa fer còpies incrementals d'aquests registres perquè les accions passades ja no es poden canviar.

El problema d'aquest sistema és que per calcular l'estat més actual, cal llegir tots els fitxers des del començament. Això pot significar una pèrdua de rendiment important, i un temps de resposta molt baix. A més a més, si es perd fins i tot només un dels fitxers, inclús dels antics, ja no es pot calcular l'estat més actual. La solució a aquests problemes és fer còpies d'estats actuals **de manera regular**, de manera que no calgui llegir gaires fitxers fins a tenir les dades més recents.

Una **còpia de seguretat diferencial** és un concepte senzill. Es basa en guardar només els canvis entre versions d'un mateix fitxer. D'aquesta manera es minimitza l'espai necessari per a cada còpia de seguretat. Té l'avantatge respecte a les còpies de seguretat incrementals que funciona en qualsevol tipus d'entorn. No cal que siguin dades amb un ús concret, sinó que pot ser molt eficient per a fitxers tan canviants com el codi d'un projecte. De fet, és el tipus de còpia de seguretat que es fa servir en tots els sistemes de control de versions. A més a més, permet tornar fàcilment a una versió antiga i comparar-la amb l'actual. D'aquesta manera, es poden fins i tot **assignar comentaris** a cada còpia de seguretat, de manera que es pugui identificar ràpidament quins canvis representen.

De totes maneres, cal entendre que no tots els casos són òptims per a aquest tipus de còpies de seguretat. Per exemple, els fitxers binaris de dades molt canviants o les dades encriptades o comprimides representen tants canvis als fitxers que no val la pena fer còpies diferencials. L'ús més adient és per a text, perquè és comprensible pels humans, i perquè permet veure molt fàcilment els canvis.

Tant els sistemes de còpies diferencials com els sistemes de còpies incrementals poden no ser sostenibles a llarg termini. Es tendeix a **acumular massa informació innecessària**, i en algun moment cal netejar. Una pràctica comuna és establir còpies de seguretat dels arxius sencers i de l'estat actual un cop per setmana, i començar repositoris nous amb versions totalment noves.

3.5.3 Integritat i disponibilitat de la versió adequada dels productes

Tot sovint, canvis de versions de programes signifiquen **deixar de ser compatibles** amb les dades disponibles. O se n'afegeixen de noves, o es modifica el significat, o s'elimina la necessitat... Per tant, com a programadors heu de ser conscients que alguns usuaris poden estar fent servir còpies de seguretat.

Una manera de facilitar la compatibilitat enrere és guardant dins de les dades **la versió del model de dades** que esteu fent servir. És a dir, si la versió del vostre programa canvia però les dades no es modifiquen, el model de dades no canvia. Però si canvien els camps que es guarden o el tipus d'estructures, llavors es pot

considerar que és un model nou de dades. Tenint en compte això, es poden fer diferents lectures dels fitxers depenent del model de dades, i evitar així molts errors. Com a alternativa, es poden programar conversors. Són petits programes que transformen els formats antics a formats nous, de manera que les versions més noves també siguin compatibles. Aquest és el mètode més emprat en els paquets de programari més coneguts.

El terme *model de dades* es refereix a les estructures de dades que es guarden i la forma com es guarden; és a dir, quines dades es guarden i amb quina seqüència.

El cas més complicat és quan cal donar servei a diferents usuaris que poden intercanviar-se dades. Les aplicacions mòbils i els jocs són casos molt habituals perquè el ritme d'actualitzacions és prou alt, i molta gent s'ha acostumat a actualitzar sempre que es pugui totes les seves aplicacions. En canvi, hi ha molta gent que prefereix no fer-ho. Per tant, és inevitable trobar-se amb una **fragmentació de les dades**. En aquests casos, la millor solució és treballar amb connexió a internet i col·locar les dades a algun servei al núvol. Molts dels serveis inclouen bases de dades NoSQL, amb les quals es pot interoperar entre versions diferents, i ja se n'encarreguen els proveïdors dels serveis, de fer les còpies de seguretat correctament.

3.5.4 Sistemes de control de versions: diferències, estat i traça de productes

Quan un equip està treballant amb els mateixos fitxers poden sorgir **dubtes i problemes**, com ara: com es poden compartir els fitxers?, com es pot assegurar la qualitat del producte?, com cal assignar responsabilitats?, com es pot tornar a una versió estable si es troba un error?

Totes aquestes preguntes es resolen en l'actualitat amb els **repositoris de versions**. Conceptualment, no són més que un dipòsit on es guarden totes les versions de tots els fitxers de manera ordenada, perquè qualsevol persona pugui consultar la versió més recent o una d'antiga, i un membre d'un equip pugui també guardar els seus canvis en privat sense afectar tothom. Els més populars són **SVN, Mercurial i Git**. De fet, el concepte amb el qual treballen és molt similar, i cada cop es tendeix més a fer servir Git.

Aquests tres sistemes es basen en definir una sèrie de fitxers i carpetes que formen part d'un projecte, i guardar-los de manera centralitzada en un arxiu que es diu repositori. Com que està tot centralitzat es pot gestionar més fàcilment l'ús compartit. A més a més, els sistemes han de poder detectar cada cop que un usuari faci canvis. Aquests canvis, però, no es guarden automàticament sinó que són els usuaris els que decideixen quan tot està llest per enviar-ho al repositori. En aquest moment, l'usuari executa una sèrie d'ordres, depenent del protocol, i s'emmagatzemen no només les diferències dels fitxers actuals amb les versions del repositori, sinó també els comentaris que escrigui l'usuari. Aquestes petites notes serveixen per descriure d'una manera molt breu quina és la intenció dels canvis introduïts.

Vegem ara com fer servir el tipus de repositori més popular avui en dia: el Git. Primer cal descarregar Git d'internet, de la pàgina oficial: git-scm.com/downloads.

Cal fer una distinció entre Git i GUI, que veureu a la pàgina de descàrregues. **Git és el protocol** que controla totes les operacions amb el repositori. La GUI, o *Graphical User Interface*, és un programa que mostra una sèrie de pantalles per facilitar la comprensió a l'usuari del que està passant a cada moment. I és que molts usuaris no estan acostumats a treballar amb ordres i prefereixen veure gràficament les seves operacions. **Treballar amb ordres**, però, té múltiples avantatges:

- La primera és que per molt ben dissenyada que estigui una interfície, amb la pràctica escriure ordres és molt mes ràpid.
- La segona és que un és molt més conscient de què està passant. Cada proveïdor de programes de interfícies de Git té un llenguatge visual diferent, un disseny diferent i en general una usabilitat diferent. Però les ordres de Git són les mateixes en qualsevol entorn, i no es preveu que canviïn.
- Finalment, perquè és molt més fàcil treballar en remot. Connectar-se a una consola d'un ordinador al núvol és molt més senzill que carregar tot l'entorn gràfic i treballar remotament amb aquest. De fet, en moltes ocasions això no és ni possible, degut a les configuracions de l'equip o del sistema operatiu.

Per aquest motiu, a continuació veureu com fer servir **Git amb línia d'ordres**. El primer pas és crear un repositori. L'opció més senzilla és treballar amb el Git Bash. A Windows podeu utilitzar el vostre explorador d'arxius i fer clic amb el botó dret a la carpeta que contingui els arxius que voleu incloure en el repositori. De les opcions que apareixen, escolliu *Git Bash here* (en altres sistemes operatius, l'operació serà semblant). Us hauria de sortir una pantalla semblant a la de la figura figura 3.4. Si no us surt, repasseu la instal·lació de Git. Per veure tota la llista de possibles ordres i una petita descripció de què fan, podeu escriure l'ordre *git* i prémer la tecla *Enter*. La llista que us sortirà serà com la de la figura figura 3.5.

FIGURA 3.4. Pantalla del 'Git Bash'

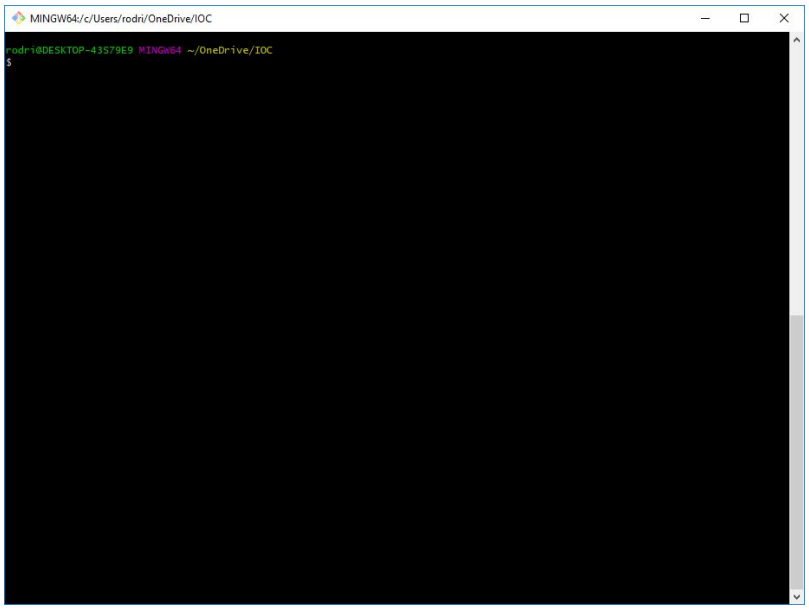
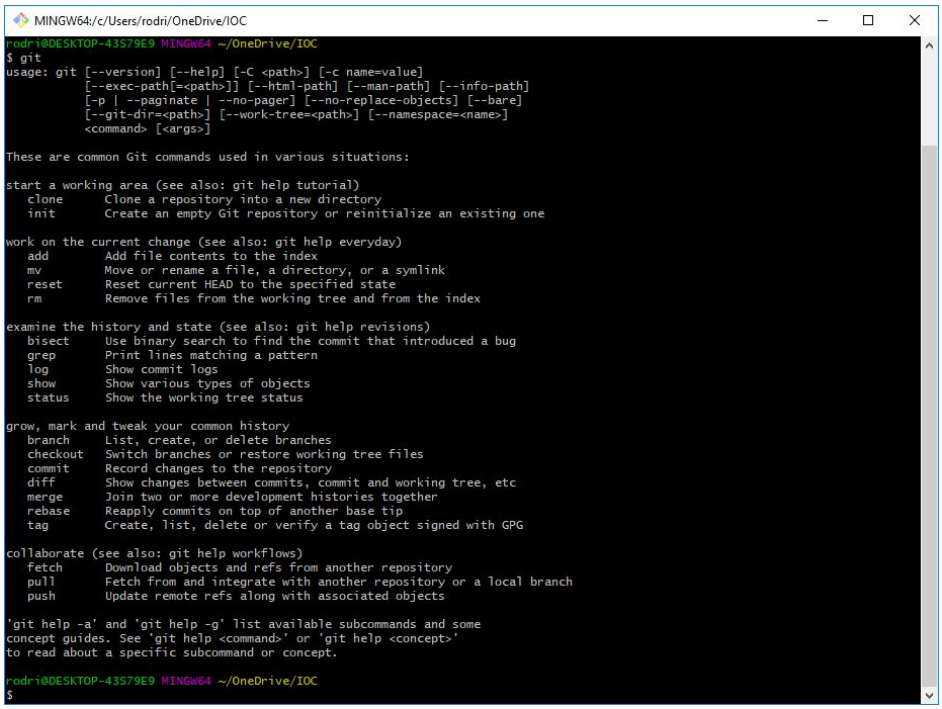
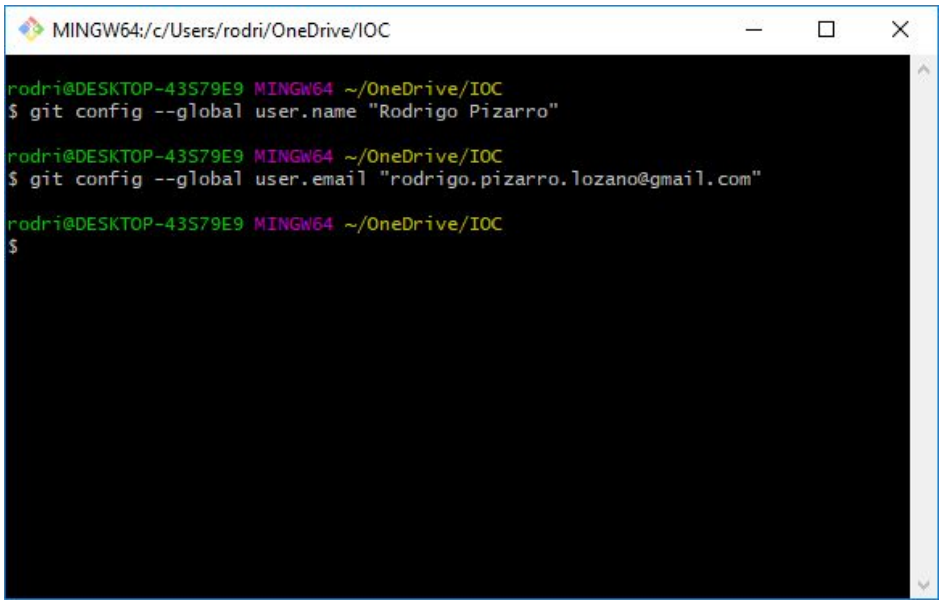


FIGURA 3.5. Ordre 'git'



El primer cop que feu servir Git, no tindreu configurat el vostre usuari. No és estrictament necessari per treballar-hi, però és útil **fer-ho un cop i ja deixar-ho fet**. Per configurar el vostre nom, heu d'escriure l'ordre: *git config –global user.name* i el vostre nom entre cometes. Per exemple, pel meu nom queda de la següent manera: *git config –global user.name "Rodrigo Pizarro"*. Un cop escrita l'ordre només cal prémer la tecla *Enter* i el nom ja quedarà registrat. La vostra adreça electrònica la podeu configurar d'una manera molt similar, i és amb l'ordre *git config –global user.email* més l'adreça entre cometes. A la figura figura 3.6 podeu veure com queda un exemple.

FIGURA 3.6. Ordre 'git config', per configurar l'adreça electrònica i el nom d'usuari



Com veieu, les ordres de Git **consten de diferents parts**. Per fer-les servir, cal escriure primer *git*, després l'acció que es vol executar, després opcionalment algun modificador assenyalat amb dos guions mitjans, seguit de la configuració de l'acció si cal, i finalment els valors que calguin.

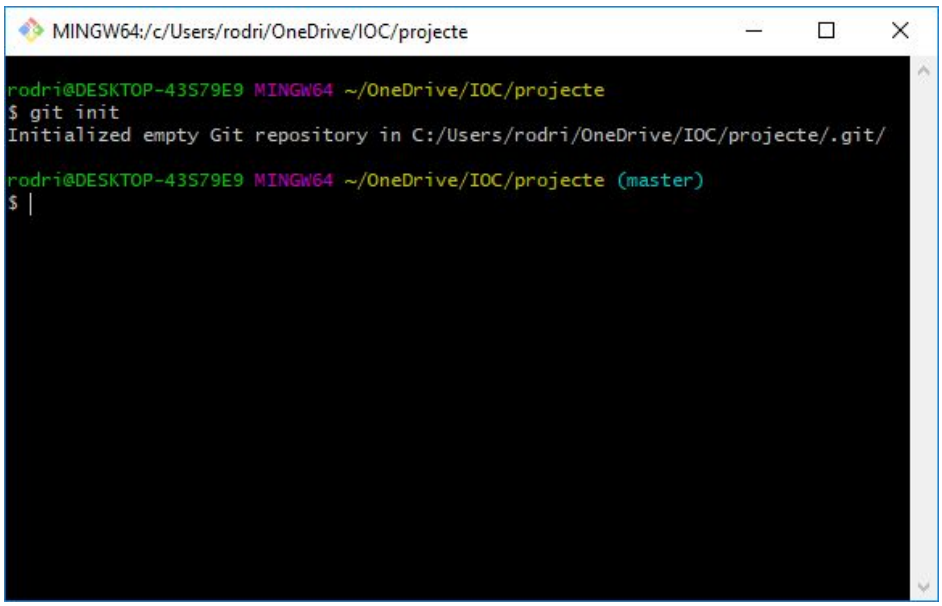
En el cas de la configuració, l'acció és *config*, l'atribut opcional és global (perquè en aquest cas no és la configuració d'un projecte concret sinó de tots els projectes) i la configuració de l'acció és *user.name*. Per això l'ordre per configurar el nom queda: *git config –global user.name "nom cognom"*. Els valors estan entre cometes perquè és una manera senzilla de marcar que és tot allò que es troba dins de les cometes. Si no s'hi posessin, no seria fàcil entendre si és un nom i cognom o només és un nom i després hi ha algun paràmetre més o és un error.

A més d'una línia d'ordres que entén les ordres de Git, el Git Bash entén també les **ordres d'UNIX**. Si bé aquest curs no se centra en les ordres d'UNIX, és interessant saber que es poden fer servir i poden ser útils. Per exemple, l'ordre *clear* esborra totes les ordres de la pantalla, i us serveix per tenir una pantalla neta i organitzar-vos millor.

Repasseu les ordres disponibles. Tal com diu la descripció, l'ordre *git init* serveix per crear un repositori nou o per reinicialitzar-ne un d'existent. Aquesta acció crearà també una carpeta privada dins del directori que s'està versionant anomenada .git i que conté una estructura interna per mantenir la configuració del repositori. La descripció detallada dels continguts d'aquesta carpeta queda fora de l'abast d'aquest temari, però cal remarcar que **els fitxers i directoris generats són necessaris** i que esborrar-los o modificar-los pot causar problemes o fins i tot fer que deixi de funcionar el sistema. De fet, una manera simple de treure un repositori d'una carpeta és esborrant el directori .git. Per tant, simplement comprovant que el directori .git s'ha creat, ja podeu continuar.

Un cop creat el repositori, veureu també que la línia d'ordres inclou al final, en un blau clar, la indicació que és a la **branca *master***, com veieu a la figura figura 3.7.

Figura 3.7. Ordre 'git init', per començar un repositori



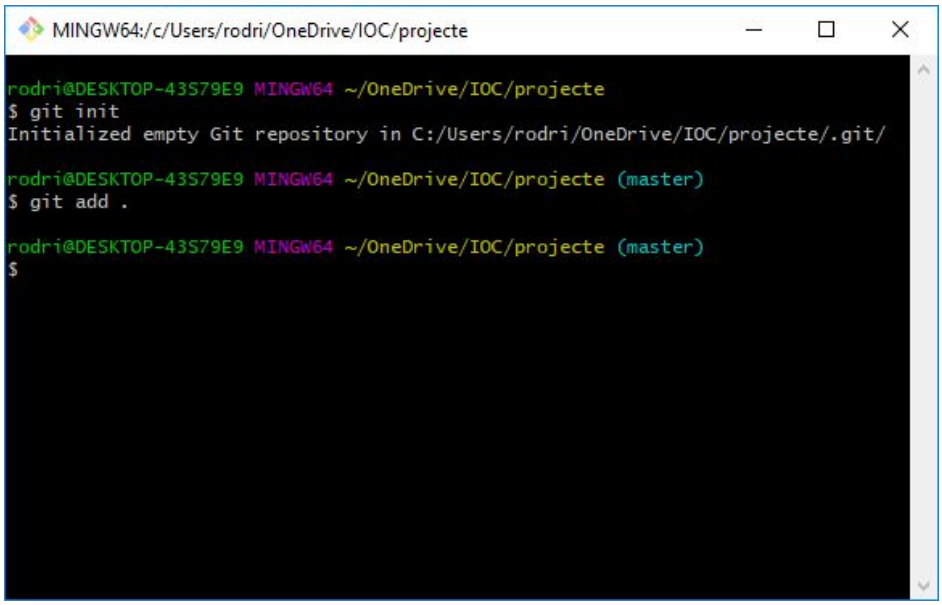
El concepte de les branques és senzill. Cada cop que algun membre d'un equip vol fer un desenvolupament i el vol guardar sense estar acabat, es crea una branca per a ell o ella i guarda la informació a la branca. D'aquesta manera, la informació queda ben guardada i no es molesta la resta de l'equip.

Un repositori en si mateix, però, no té cap utilitat. Un cop creat el que heu de fer és fer-lo servir per **guardar dades**. Per aconseguir-ho heu de seguir un procés en dues etapes:

1. La primera és dir-li a Git que voleu que observi alguns directoris o fitxers.
2. La segona és dir-li que voleu guardar l'estat en què es trobi.

Per aconseguir la **primera etapa**, dir-li a Git quins són els fitxers i directoris que formen part del projecte, heu de fer servir l'ordre *git add* i el fitxer o directori que voleu afegir al repositori. A la imatge veureu un exemple de com fer servir la comanda per **afegir el directori actual**. El directori actual se simbolitza amb un punt, i el directori pare amb dos punts seguits. Qualsevol subdirectori s'afegeix amb el nom de la carpeta. Per facilitar-vos la vida, podeu posar tots els fitxers necessaris en un directori i afegir-lo al repositori amb una única comanda. A la figura figura 3.8 es mostra com afegir la carpeta actual.

Figura 3.8. Ordre 'git add', per afegir fitxers i directoris



La raó per la qual cal dir-li a Git quins són els fitxers i directoris necessaris és perquè molts projectes fan servir fitxers de configuració que depenen de l'entorn concret on s'està executant o programant. Si aquests fitxers s'afegissin a un repositori comú per a un equip, seria impossible treballar, perquè cada membre de l'equip hauria de canviar aquests fitxers cada cop que obtingués l'estat del repositori. D'aquesta manera, la manera de treballar és **molt més senzilla**, només s'afegeixen al repositori tots aquells fitxers i directoris que són comuns, i tot allò que depèn de l'entorn s'exclou. També s'exclouen totes les dependències i llibreries de tercers que calguin per fer a servir el projecte.

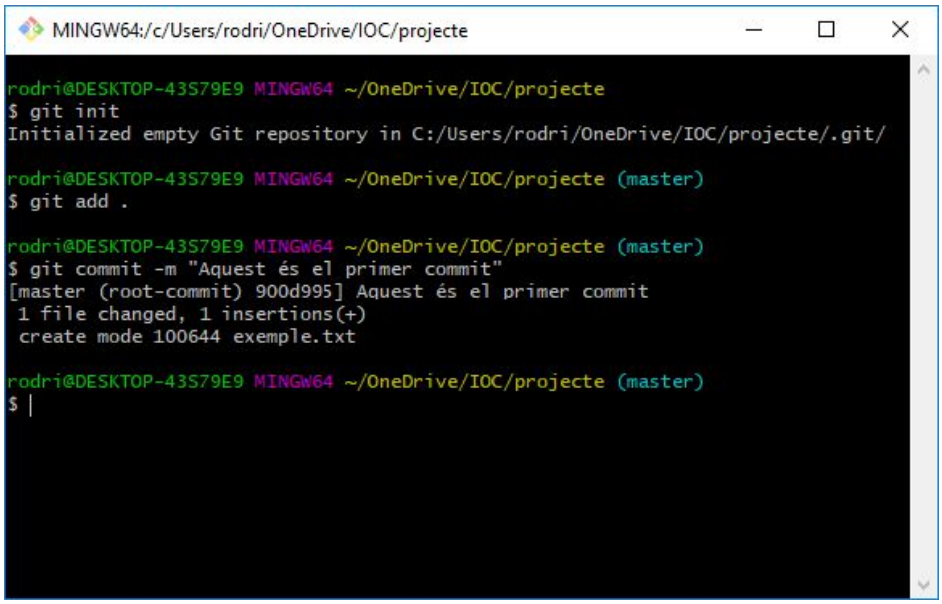
Aquesta situació és molt comuna en **projectes complexos**. Molts cops una llibreria desenvolupada per algú i distribuïda per internet ja fa la feina que es necessita, i per tant no cal reinventar la roda. És molt més senzill integrar la llibreria i fer servir les funcions dins del vostre projecte. Els problemes arriben quan la llibreria s'actualitza i pot deixar de ser compatible amb part del vostre projecte. També pot passar que alguns membres de l'equip vulguin provar versions diferents per veure els canvis o per qualsevol altre motiu. Si estiguessin dins del repositori es complicarien aquestes operacions. Finalment, quan un projecte creix, el seu repositori **creix en la mateixa mesura**, i alleugerir les operacions pot arribar a ser crític. Quan hi ha milers de fitxers en un repositori, les operacions es tornen més lentes i hi ha un impacte en l'espai ocupat en el disc dur.

La **segon etapa** per guardar els fitxers al repositori és fer servir l'ordre *git commit*. Per provar que aquesta ordre funciona, primer haureu de crear un fitxer de prova, que en aquest cas anomenarem exemple.txt i veureu què passa. El concepte de *commit* és **guardar un estat actual** al repositori. D'aquesta manera, es va treballant a base d'anar guardant els estats del projecte a mesura que es va progressant, i si algú detecta algun error o vol comprovar alguna cosa d'alguna versió anterior ho pot fer tornant a un estat anterior. La comanda està formada per les primeres paraules clau *git commit*, i després l'opció *-m* i un text per descriure què representa l'estat que s'està guardant.

Si només tenim dues o tres versions no hi ha problema per saber què representa cada versió. Però si tenim un projecte on treballen 30 persones durant mesos, serà impossible tenir un control mental de què conté cada versió. Per tant, serà imperatiu que treballeu amb uns **criteris clars** de com descriure cada versió i quins canvis aporta. Tant és així que Git no deixa fer un *commit* si no hi ha un comentari.

A la imatge figura 3.9 es pot veure el resultat de fer un primer *commit* de la carpeta on esteu treballant.

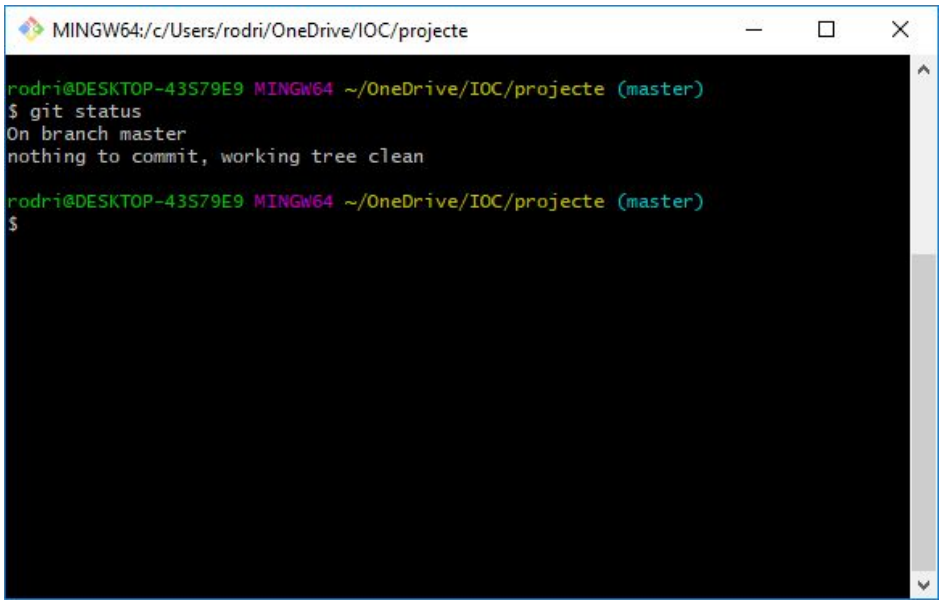
Figura 3.9. Ordre 'git commit', per afegir l'estat actual al repositori



En aquest cas hem posat un comentari d'exemple que diu que aquest és el primer *commit*. Com podeu veure, el sistema ens torna un missatge que informa que hi ha un fitxer que s'ha afegit i un fitxer que s'ha modificat. Lògicament, és el mateix fitxer.

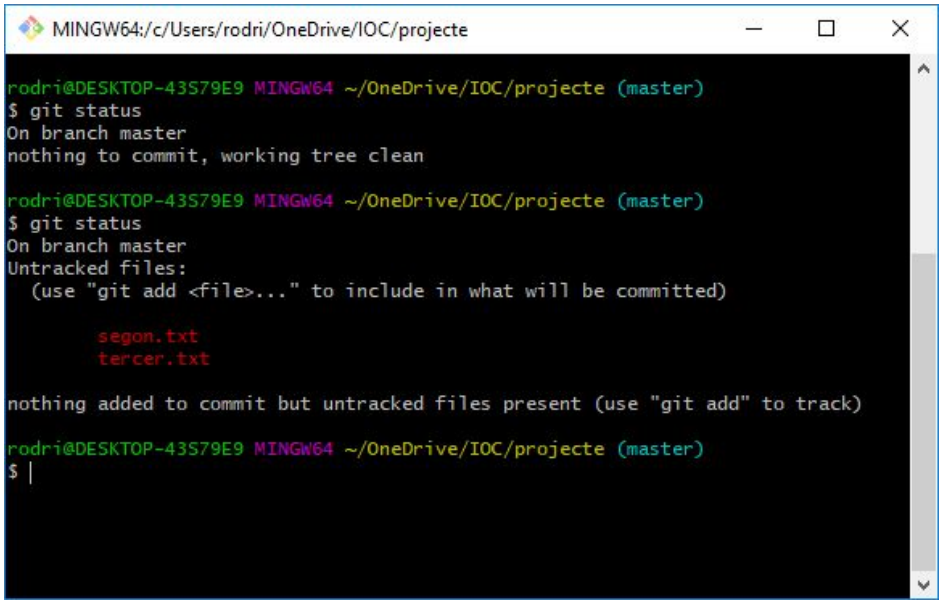
Per comprovar el resultat i l'estat del sistema, podeu fer servir l'ordre *git status*. El resultat d'aquesta ordre és la branca en la qual esteu treballant actualment i també si queda algun canvi per enviar al repositori. A la imatge figura 3.10 podeu veure com queda el resultat de *git status* a l'exemple, per exemple, si els fitxers s'han modificat o no s'han afegit al repositori.

FIGURA 3.10. Ordre 'git status', per veure l'estat en què es troben els fitxers



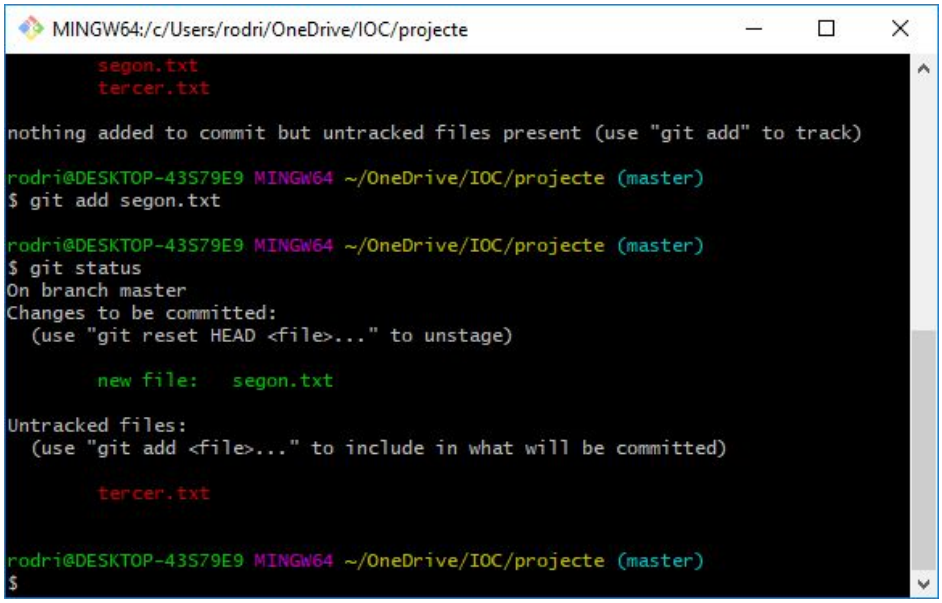
Vegeu ara què passa si afegiu un parell de **fitxers nous**. Creeu dos fitxers anomenats segon.txt i tercer.txt i torneu a la consola de *Git Bash*. Torneu a fer l'ordre de *git status*. El resultat ha de ser el mateix que el de la figura figura 3.11.

FIGURA 3.11. Modificar fitxers i veure el resultat de l'ordre 'git status'



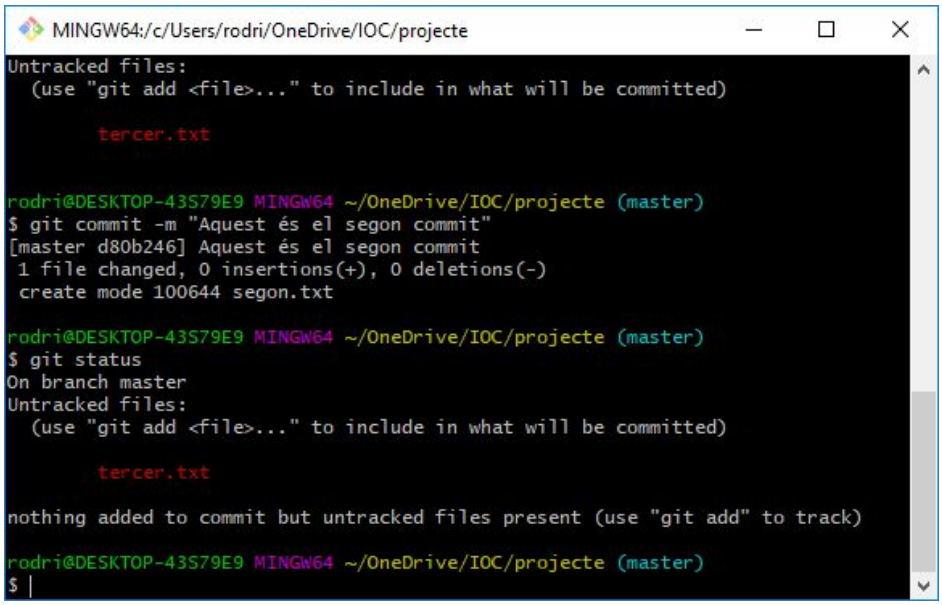
En aquest cas, podeu veure que *git status* retorna que hi ha dos fitxers que no estan seguint, que són justament els fitxers que acabeu de crear. També us indica que per afegir els fitxers heu de fer servir l'ordre *git add* i el nom dels fitxers. Proveu ara d'afegir el segon fitxer. Ho podeu fer amb l'ordre *git add* 'segon.txt'. Si després escriviu *git status* veureu que Git us indica que hi ha un fitxer amb canvis i que cal enviar-lo al repositori. També indica que hi ha un fitxer, tercer.txt, que no està afegit al repositori. A la figura figura 3.12 es veu com hauria de quedar:

Figura 3.12. Modificar un fitxer després d'afegir-lo i veure el resultat de l'ordre 'git status'



Ara, per guardar l'estat, heu de fer servir l'ordre *git commit -m* i el comentari entre cometes. En el moment en què feu el *commit*, l'estat quedarà guardat al repositori, i Git interpretarà que no queda res per fer. L'únic que no farà serà cap operació amb el fitxer tercer.txt, perquè interpreta que no és rellevant per al repositori. De fet, si després de fer el *commit* feu un *git status*, us sortirà que tot està actualitzat, però que hi ha un fitxer anomenat 'tercer.txt' al directori que no està afegit al repositori. El resultat de les dues operacions es pot veure a la figura figura 3.13.

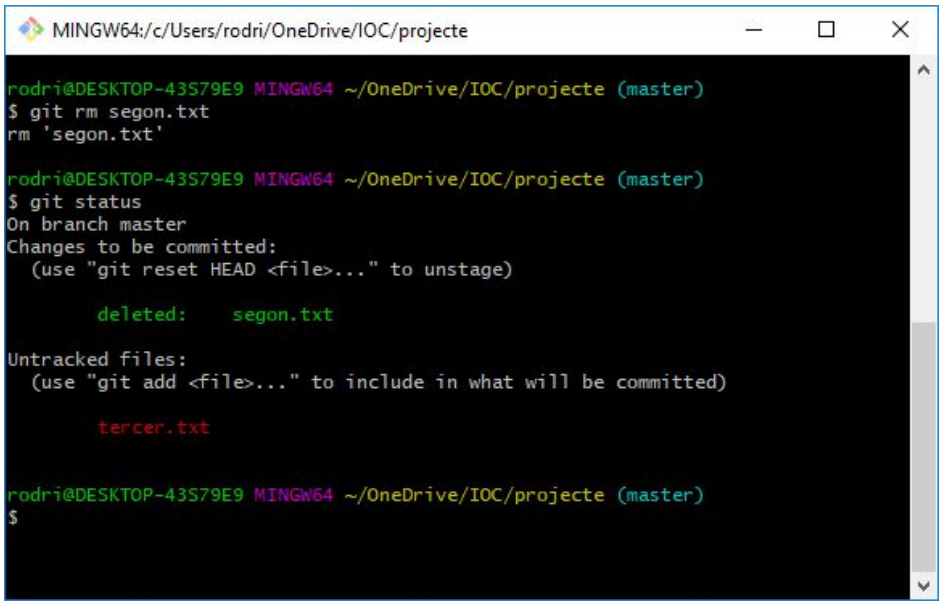
FIGURA 3.13. Ordre 'git commit'



De fet, tots els passos pels quals passen els fitxers tenen **un nom tècnic concret**. Els fitxers que hi ha al directori però no estan afegits al repositori (no s'ha fet cap ordre amb ells i es veuen en vermell quan es crida l'ordre *git status*) estan en un estat que es diu ***working copy***. Un cop s'executa l'ordre *git add* per a un fitxer, passa a l'estat de ***staging area***. En aquest moment, encara no està guardat al repositori, però Git està configurat per afegir-lo a la següent ordre *git commit*, i es veuen en verd amb l'ordre *git status*. Un cop executada l'ordre *git commit* per a un fitxer, el fitxer és al repositori. El fitxer tornarà a ser al *working copy* si es modifica; caldrà executar l'ordre *git add* per afegir-lo a *staging area*, i tornarà a estar sincronitzat amb el repositori quan es faci un *git commit*. Aquests últims passos són il·limitats sempre que no s'esborri un fitxer del repositori amb l'ordre corresponent.

Això sí, cal tenir present que això són els estats lògics pels quals passen els fitxers dins de Git. El fitxer en si mateix **no pateix canvis**. És a la carpeta on s'hagi creat i no es modifica ni es mou ni es destrueix amb les ordres de *git add* o *git commit*. Hi ha una ordre de Git, però, que sí que afecta realment els fitxers. L'ordre *git rm* més el nom del fitxer **esborra el fitxer** realment de la carpeta on estigui. Aquesta ordre és equivalent a esborrar el fitxer manualment a través del sistema operatiu. A la figura figura 3.14 podeu veure un exemple de l'ordre, que també esborra el fitxer segon.txt de la carpeta.

Figura 3.14. Ordre 'git rm'



La particularitat de l'ordre *git rm* és que aquest canvi encara no s'ha enviat al repositori. És a dir, el repositori ha anat guardant progressivament totes les versions a mesura que heu anat fent *git commit*, i per tant un canvi com esborrar un fitxer es guarda també quan s'executi l'ordre *git commit*. Per ser més precisos, el fitxer realment no s'esborra del repositori, sinó que s'indica que ja no hi ha aquell fitxer dins del projecte. Però si es volgués recuperar per qualsevol motiu, es podria fer amb l'ordre corresponent.

3.5.5 Repositoris i còpies de treball: resolució de conflictes

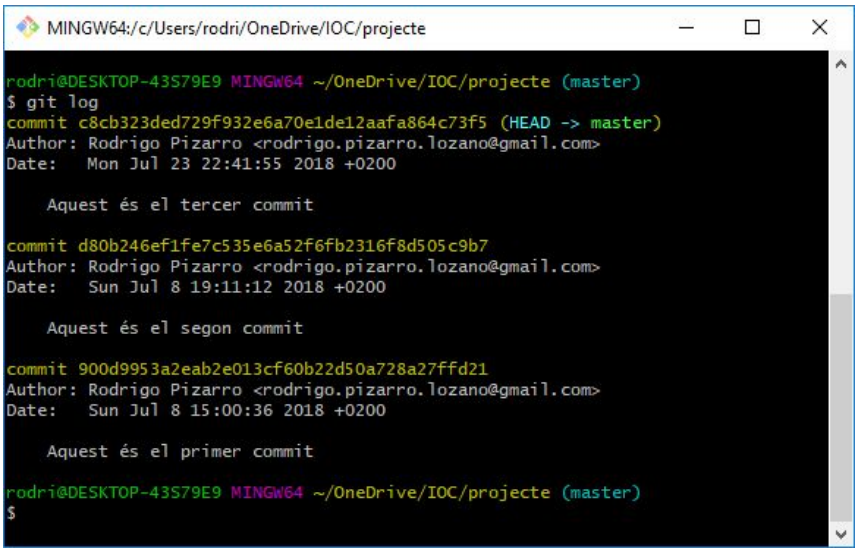
Una de les operacions més interessants de tot el sistema de control de versions i repositoris és la de **tornar a una versió més antiga**. Quan ens adonem que hi ha errors i que és més fàcil solucionar-los tornant enrere, Git ens ho facilita amb l'ordre *git checkout*. Aquesta ordre serveix per a posar estats del repositori al *working copy*, és a dir, posa fitxers que s'hagin guardat anteriorment dins del repositori a la carpeta del projecte que s'estigui versionant.

Per poder indicar a Git quina és la versió que es vol recuperar, cal indicar-li el codi de la versió. Aquest codi apareix amb l'ordre *git log*, al costat de cada *commit*. Aquests codis són autogenerats, i són prou llargs perquè sigui pràcticament impossible que hi hagi dos *commits* amb el mateix codi. Normalment amb els **primers caràcters** ja n'hi ha prou per **identificar una versió**, i amb aquests primers caràcters s'indica a l'ordre *git checkout* quina és la versió que es vol recuperar.

Exemple d'ús de l'ordre 'git checkout'

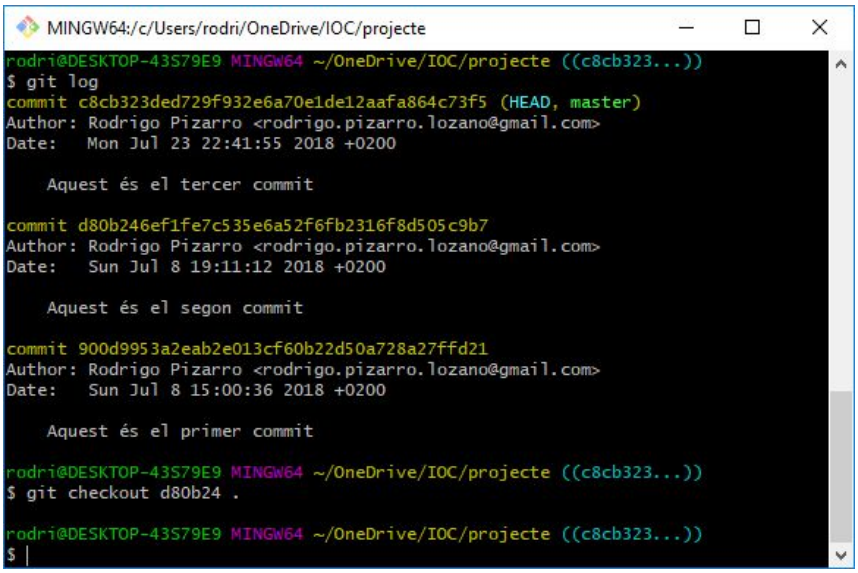
Feu ara un projecte amb tres versions dins del repositori, fent servir l'ordre *git commit* tres cops. Hauria de quedar com a la figura següent:

Figura 3.15. Projecte amb tres 'commits'



Ara feu servir l'ordre *git checkout* amb el codi de la segona versió. En aquest cas el codi per al segon *commit* del projecte d'exemple és d80b24. Cal també indicar quins fitxers es volen recuperar, i per recuperar-los tots es pot escriure un punt, com a la figura següent:

Figura 3.16. Projecte amb la segona versió recuperada



Però, què passa si algú vol guardar una versió que encara és inestable o no ha estat provada? Per això hi ha les **branques**: qualsevol persona pot crear una branca per guardar canvis sense afectar els altres; la idea és que després les branques o s'abandonen perquè ja es veu que no porten a cap millora, o bé es fusionen amb els canvis de la resta. Cada branca es pot anomenar de la manera que es vulgui, tot i que les bones pràctiques indiquen que el nom hauria de ser indicatiu d'allò en què s'està treballant. La branca principal es diu *master*, i es fa servir com la branca de la qual tothom està d'acord que és la més estable, i l'objectiu de totes les branques és que els canvis més rellevants acabin a la branca *master*.

Per crear una branca, podeu fer servir l'ordre *git branch* més el nom de la branca. Aquesta comanda crea una branca com una còpia de la branca on estigueu (és a dir,

tot l'estat serà el mateix), però, no fa cap més canvi. Si voleu començar a treballar amb la nova branca, heu de fer *git checkout* més el nom de la branca. Per fer les dues operacions alhora, podeu fer servir l'ordre *git checkout -b* més el nom de la branca.

Una pràctica molt comuna és crear branques per cada funcionalitat nova que es vulgui afegir, i assignar-les a cada persona. D'aquesta manera, no s'entorpeix el treball de ningú. Per agregar tots els canvis en una versió comuna cal **fusionar les branques**. Per fer-ho, cal fer servir l'ordre *git merge* més el nom de la branca que volem fusionar. Aquesta ordre afegeix tots els canvis de la branca que es vol fusionar amb la branca pare. És a dir, una branca sempre es crea a partir d'una altra, que és la branca pare. Al fusionar-la, es veuen tots els canvis i, si no hi ha conflictes, s'afegeixen a la branca pare. Un conflicte és un canvi en un fitxer que no sigui evident de fusionar.

Exemple de conflictes a l'hora de fusionar branques

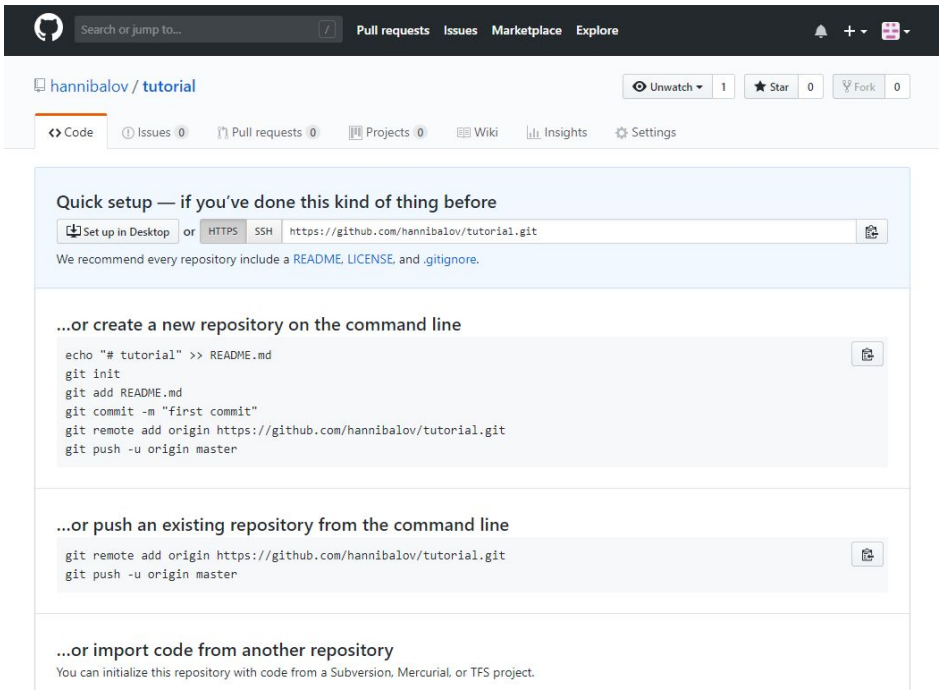
Tenim una branca que es diu pare i una branca que es diu fill i un fitxer que es diu A.txt, i dues persones en un equip. Suposeu que una persona edita el fitxer A.txt a la branca pare, fa *commit* i l'altra persona de l'equip també edita les mateixes parts del fitxer A.txt i també fa *commit*, però a la branca fill. Ara, al fusionar la branca fill el problema és que Git no sap escollir si ha de quedar-se amb la versió d'A.txt de la branca pare o de la branca fill.

Si no es resolen els conflictes, no es poden fusionar les branques. Per **resoldre els conflictes** hi ha moltes eines. Un exemple d'ordre per resoldre-ho és *git mergetool –tool=kdiff3*. Totes les eines funcionen de manera similar. Bàsicament ensenyen el canvi d'una branca, el canvi de l'altra, i cal que l'usuari decideixi quin canvi es converteix en definitiu.

Treballar amb repositoris adquireix una dimensió molt més important quan es **treballa en equip**. Quan membres d'un grup poden fer canvis i veure els canvis dels altres és quan s'entén la rellevància de tot el sistema de control de versions. Per organitzar-ho, hi ha molts serveis que ofereixen poder treballar amb repositoris Git allotjats al núvol. Vegeu ara com fer servir **Github** amb les ordres:

1. El primer pas és crear-se un compte de Github per tenir un usuari.
2. Un cop creat, podeu crear un repositori públic dins de Github per fer proves com veieu a la figura figura 3.17.
3. Després, simplement cal seguir les ordres que s'indiquen a la pàgina i ja tindreu el repositori al vostre ordinador.

FIGURA 3.17. Repositori creat a Github



El concepte de serveis com Github és que ofereixen una **sincronització dels repositoris**, és a dir, qualsevol usuari que tingui accés a un repositori remot pot anar fent canvis en local a la seva màquina i, quan vegi que té una versió que vol compartir, pot enviar els canvis del seu repositori local al repositori remot. Per fer-ho, pot fer servir la comanda *git push -u origin* i la branca que vulgui pujar.

Per veure quins canvis han fet els altres, cal primer descarregar-los del servei en remot. Això es fa amb l'ordre *git pull*, i el que fa és fusionar el repositori local amb el repositori remot. D'aquesta manera, es podran veure tots els canvis que s'hagin fet en altres ordinadors i que s'hagin pujat al repositori remot.

En resum, treballar en projectes en equip és una feina complexa i que pot generar errors. Eines com els repositoris ens permeten no només tenir còpies de seguretat, sinó també una manera molt adequada per resoldre problemes i per no destorbar en la feina d'altres companys o companyes.

Producció i avaluació del projecte

José Manuel Solís

Índex

Introducció

Per treballar en l'àmbit dels productes multimèdia, cal estar familiaritzat amb els processos de desenvolupament que duen a terme els equips humans que els creen i els estàndards de qualitat que apliquen per tal d'obtenir productes que puguin posar-se a disposició del públic o els clients amb una certa confiança en les seves característiques. La finalitat d'aquesta unitat és presentar aquests processos, així com les principals tendències actuals dins d'aquest àmbit.

En l'apartat **"Planificació del seguiment i realització del projecte"**, es presenta el concepte de procés de desenvolupament d'un projecte i s'identifiquen els rols principals dins d'un equip que treballa a aquest entorn i la forma en què els productors planifiquen el seu treball.

En l'apartat **"Avaluació del projecte"**, es presenten els rols i els processos implicats en la fase de comprovació d'un projecte, així com els conceptes principals que es fan servir quan es treballa amb aquest aspecte.

Per assolir amb èxit els coneixements exposats en aquesta unitat, és convenient fer les activitats i els exercicis d'autoavaluació que es proposen en cada apartat.

Resultats d'aprenentatge

En finalitzar aquesta unitat, l'alumne/a:

1. Determina les arquitectures tecnològiques de producció o desenvolupament i de destinació o desplegament (usuari final) dels projectes audiovisuals multimèdia interactius, relacionant les especificacions tècniques amb els requisits d'operació i de seguretat.

- Segmenta els diagrames dels models inicials en seccions o capes per mostrar la lògica de l'aplicació, el disseny de la interfície d'usuari i les classes implicades en l'arxivament de dades.
- Documenta els detalls de la implementació del sistema (diagrames de classe i components) i de la distribució general del maquinari necessari (diagrames d'implementació).
- Documenta l'arquitectura tecnològica de producció o desenvolupament, tenint en compte l'anàlisi de les capacitats previstes, les especificacions de caràcter tècnic, la disponibilitat de les bases de dades, els permisos d'accés a la informació i els sistemes de comunicació entre el personal tècnic.
- Documenta l'arquitectura tecnològica de destinació o desplegament (usuari final), atenent els requisits d'accessibilitat, compatibilitat i interoperabilitat entre plataformes.

2. Determina els paràmetres i els procediments de gestió de projectes, sistemes de posada a punt d'equipaments i d'eines, connectivitat i comunicacions, i assegurament de la qualitat i seguretat de la informació de l'entorn de producció.

- Determina els mòduls d'informació del projecte (agrupacions de fonts de texts, gràfics, sons, imatges fixes i imatges en moviment) segons les especificacions, per garantir-ne la fluïdesa de processament, integritat informativa, mida, posició i funció al producte.
- Determina els continguts, els aspectes i les característiques de les fonts, mòduls d'informació, pantalles, nivells i diapositives.
- Estableix les relacions entre els mòduls d'informació i la seva ubicació al producte audiovisual, en funció de les tècniques narratives i estètiques.
- Elabora els esbossos o maquetes de cada pantalla, nivell i diapositiva del producte audiovisual multimèdia, en funció de les tècniques narratives i estètiques.
- Respecta la legislació vigent entorn dels drets d'autor i la propietat intel·lectual, d'acord amb les característiques particulars del projecte que es desenvoluparà.

- Estableix el sistema d'organització i de catalogació de fonts conforme als requisits d'operació i de seguretat acordats.
- Determina protocols de realització de còpies de seguretat per garantir la integritat i disponibilitat de la informació.
- Determina el sistema de control de versions per garantir la integritat i la disponibilitat de la versió adequada dels productes.
- Estableix el sistema d'actualització del reposador des de còpies de treball, preveient possibles conflictes.

1. Planificació del seguiment i realització del projecte

Per fer la planificació i el seguiment d'un projecte audiovisual multimèdia cal primer de tot entendre quina és la composició d'un **equip de desenvolupament**, les funcions de cadascun dels seus membres i les formes d'organització principals que existeixen a les empreses.

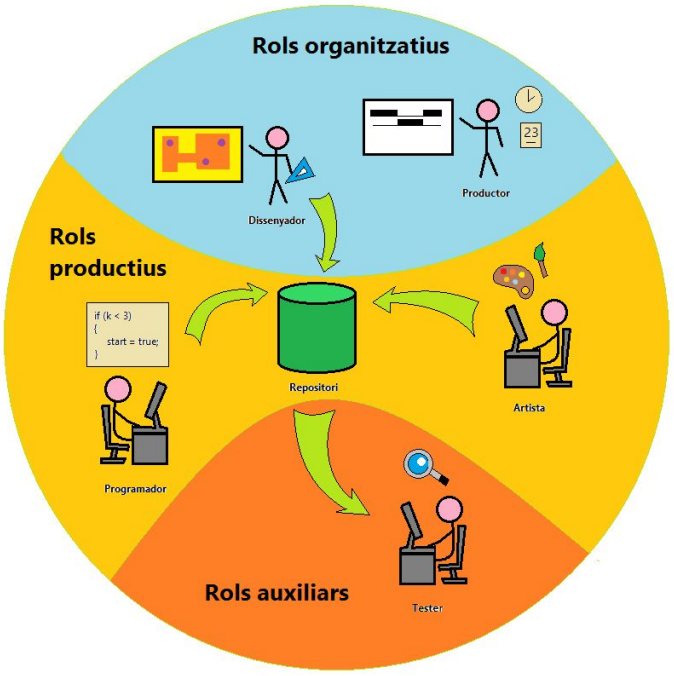
D'altra banda, per poder assegurar que l'equip està desenvolupant el projecte correctament, cal **organitzar el treball en unitats** que puguem caracteritzar i situar en un pla de treball. Aquesta és la feina principal del productor.

Finalment, cal entendre les **diferents etapes** que formen part del cicle de desenvolupament d'un projecte per tal de saber en quines tasques cal que l'equip s'enfoqui en cada moment.

1.1 Equips de desenvolupament

Portar a terme un projecte audiovisual multimèdia és una **tasca complexa** que requereix el concurs de professionals de diverses disciplines que s'han d'organitzar com un equip de desenvolupament, cadascun d'ells assumint un **rol diferent** (vegeu la figura 1.1).

FIGURA 1.1. Rols en un equip de desenvolupament



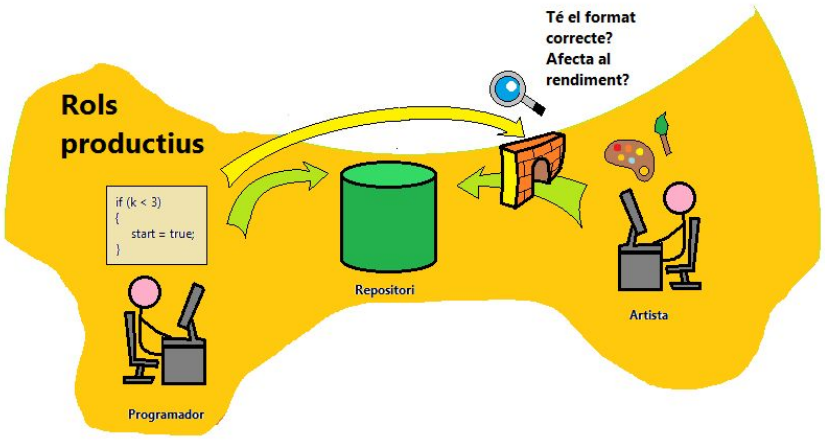
Podem classificar els rols dins d'un equip de desenvolupament en diferents grups. Els rols **productius**, els rols **organitzatius** i els rols **auxiliars**. Cada rol delimita les responsabilitats que té el membre de l'equip i el tipus de tasques que pot dur a terme.

1.1.1 Rols productius

Físicament, un producte audiovisual multimèdia està format per uns arxius que comprenen **recursos** com gràfics 2D i 3D, so, música i text i **codi** que els integra en entitats com botons o personatges, dotant-los de comportament. Els rols productius són aquells que tenen com a funció principal crear tots aquests arxius, tot partint d'unes **referències** establertes pels directors dels seus respectius departaments o en absència d'aquests, de les indicacions d'altres rols que assumeixin les competències en aquesta àrea. El rol productiu està integrat pels programadors i els artistes.

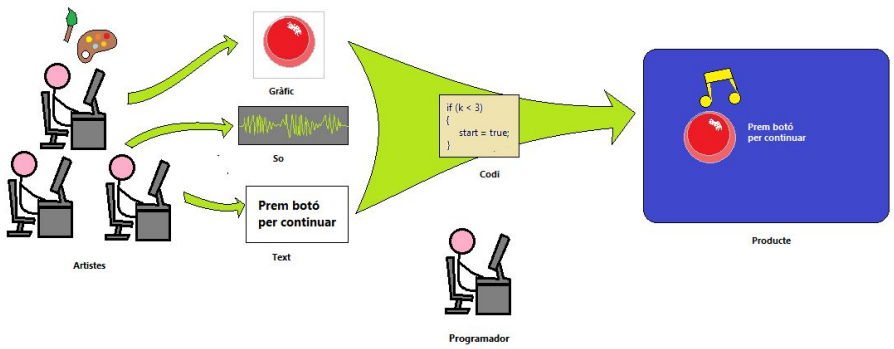
Els **programadors** són els encarregats de dissenyar i produir el codi del projecte. Duen a terme diverses tasques. Per exemple, **integren** els recursos que produeix la resta de rols productius al repositori del projecte, i s'asseguren que tinguin un format adequat per assolir els **paràmetres de qualitat i rendiment** establerts, per exemple controlen que les imatges no excedeixin una resolució màxima per tal que l'aplicació pugui mostrar-les amb fluïdesa (vegeu la figura 1.2).

Figura 1.2. Validació dels recursos per part del programador



A més, **produeixen codi** que carrega els recursos a l'aplicació i els agrupa en entitats; per exemple, creant un botó que agrupi un gràfic i un arxiu de so (vegeu la figura 1.3).

FIGURA 1.3. Assemblatge de les entitats a partir dels recursos



També creen el **comportament** d'aquestes entitats, a partir d'elements com **diagrames de casos d'ús** i de **seqüència** que expressen els **requisits** del projecte, per exemple, fent que en prémer un botó un personatge salti, com s'estableix a un diagrama de seqüència concret.

Teniu una explicació sobre els diagrames de casos d'ús a l'apartat "Determinació d'objectius, estil gràfic i narratiu d'un projecte" de la unitat "Disseny i planificació del projecte".

Finalment, els programadors construeixen o gestionen una **estructura *software*** que permet fer la integració de recursos de forma còmoda per la resta de rols productius i dota de serveis comuns a les entitats, per exemple creant un sistema d'animació comú a totes elles. Són exemples d'aquestes estructures *software* els motors o les llibreries gràfiques i editors de joc.

A banda dels programadors, dins dels rols productius, els **artistes** agrupen tots aquells rols que produeixen algun tipus de recurs diferent del codi. Típicament s'identifica amb aquest terme les persones encarregades de la producció dels gràfics, però també hi ha artistes de so, músics i escriptors que entrarien dintre d'aquesta categoria. La característica més comuna a tots ells és que, tot i que a la pràctica hi ha rols intermedis, no tenen un perfil tan acusadament tècnic com els programadors, raó per la qual aquests solen haver de revisar aquest aspecte del seu treball.

1.1.2 Rols organitzatius

Al marge dels rols productius, trobem una sèrie de rols organitzatius, que es distingeixen perquè la seva funció no és tant produir contingut que hagi d'integrar-se al producte final com **transmetre** a la resta de membres de l'equip els **paràmetres i objectius** que ha de seguir el projecte. El rol organitzatiu està integrat pels dissenyadors i els productors.

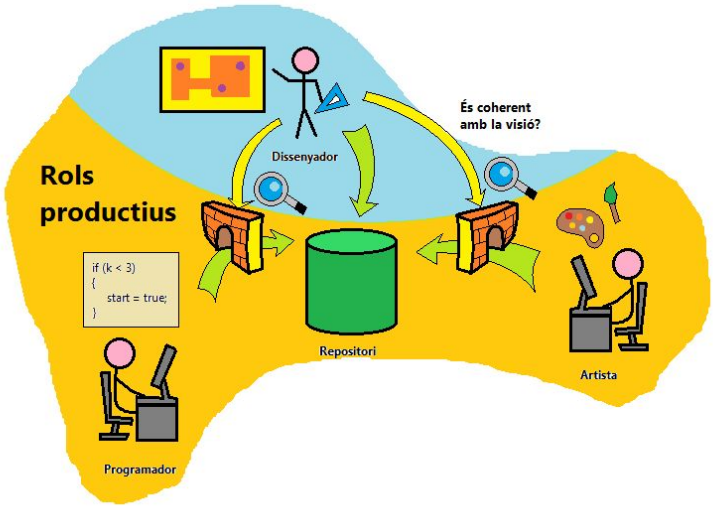
La funció dels dissenyadors és mantenir la visió del resultat que s'està cercant i assegurar-se que els diferents rols productius són coherents amb aquesta visió. Entre les **qualitats desitjables per a un dissenyador** distingim:

En alguns àmbits, els termes *dissenyador* i *grafista* sovint es confonen.

- Ser capaç de visualitzar en tot moment el resultat final que es cerca i preveure la reacció que tindrà l'usuari davant qualsevol element nou que s'incorpori al projecte.

- Demanar a l'equip que es facin correccions si es detecten desviacions que puguin comprometre aquesta visió, tot i ser permeable també a noves idees que puguin millorar el resultat (vegeu la figura 1.4).
- Comunicar la visió, produint versions prototípiques dels recursos i redactant documents que en descriguin el comportament.

FIGURA 1.4. Validació per part del dissenyador



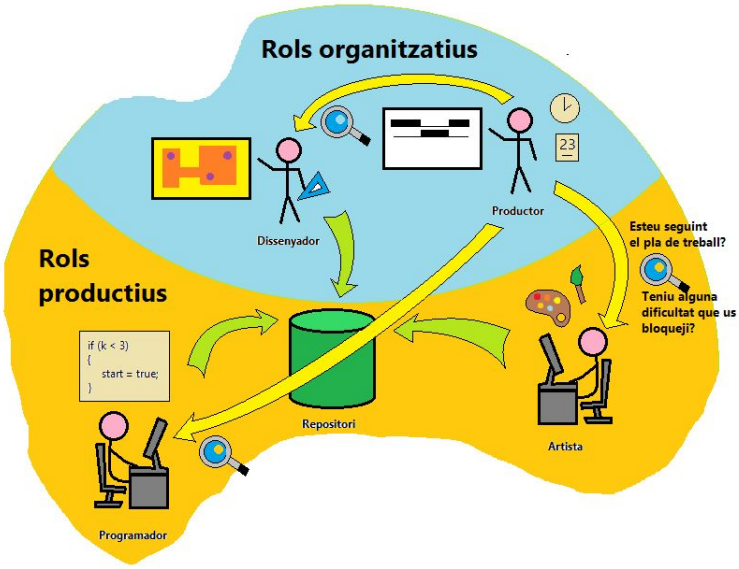
D'altra banda, els productors són les persones encarregades de gestionar el temps, els materials i les persones assignades a les diferents tasques. Entre les **funcions del productor** hi ha:

- Assegurar que els objectius del projecte s'assoleixin dintre del temps assignat i fent un ús òptim dels recursos humans i materials.
- Estructurar el treball que cal fer en elements com a objectius i tasques, expressar-lo en un pla de treball i comunicar-lo a l'equip.
- Fer el seguiment de la posada en pràctica del pla, i revisar-lo i adaptar-lo si les circumstàncies del projecte canvien (vegeu la figura 1.5).

Parlarem extensament del rol del productor al punt "La tasca del productor: elaboració, comunicació i supervisió d'un pla de treball", d'aquest mateix apartat.

En alguns equips, el productor i el dissenyador poden ser la mateixa persona.

Figura 1.5. Supervisió de les tasques per part del productor



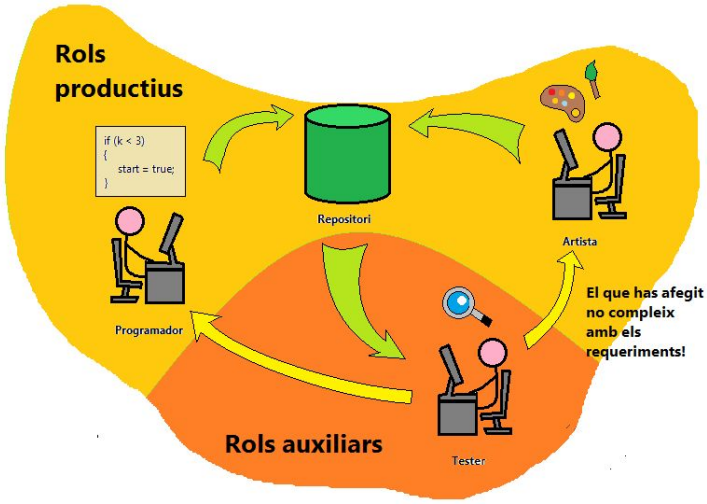
1.1.3 Rols auxiliars

Els rols auxiliars s'especialitzen a fer algun tipus de **funció concreta** dins el projecte. El més important és el de *tester*, que es podria traduir per 'verificador' i assumeix el paper d'un usuari que experimenta amb el producte. Així, el rol auxiliar està integrat pels *testers*, aquests faciliten que la resta de l'equip de desenvolupament pugui avançar en les seves tasques sense haver de verificar que aquestes estan completament lliures d'errors, ja que ells s'encarreguen específicament de fer aquesta funció; tot i que també **tenen les funcions següents**:

Parlarem extensament del paper dels *testers* a l'apartat "Avaluació del projecte", d'aquesta mateixa unitat.

- Organitzar plans de test sistemàtics, verificant que el comportament del producte multimèdia s'ajusta a les condicions establertes en els requisits del projecte.
- Verificar que els recursos visuals i sonors es mostren sense defectes apreciables.
- Redactar informes d'errors que especifiquin amb precisió les situacions en què el producte no es comporta com s'espera o ho fa amb algun defecte (vegeu la figura 1.6).
- Fer el seguiment de les correccions aplicades per l'equip, verificant que efectivament es corregeixen els problemes trobats.

FIGURA 1.6. Verificació dels requeriments del producte per part dels 'testers'



1.1.4 Jerarquies i especialitats

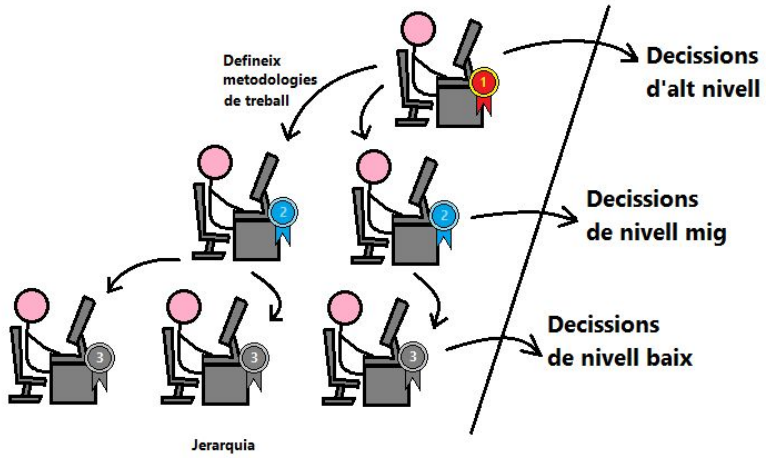
Un equip de desenvolupament pot estar format per un nombre de membres que pot o no coincidir amb els rols requerits per fer les tasques. Això pot donar lloc a l'aparició de jerarquies i especialitats. Poden donar-se **diferents situacions**:

- Que els membres de l'equip tinguin exactament els rols requerits pel projecte, amb la qual cosa no cal prendre cap decisió.
- Que hi hagi més rols requerits que membres de l'equip amb rols coincidents, i en aquest cas algun dels membres de l'equip n'haurà d'assumir més d'un. Per exemple, es pot donar el cas que un artista també faci de programador.
- Que hi hagi més membres de l'equip amb un rol que rols requerits al projecte, situació en la qual poden donar-se conflictes, ja que els límits de les responsabilitats de cadascun es poden superposar.

Una excepció a aquest últim cas es dona quan les tasques que s'han de fer són homogènies i fàcilment divisibles, de manera que no cal establir cap diferència entre membres que fan el mateix rol. Un cas típic podrien ser uns *testers* que es divideixen els nivells d'un joc tot i fer una feina semblant. Si no estem en aquest cas, cal establir algun tipus de **diferenciació** entre els membres que tenen el mateix rol, establint jerarquies i especialitats.

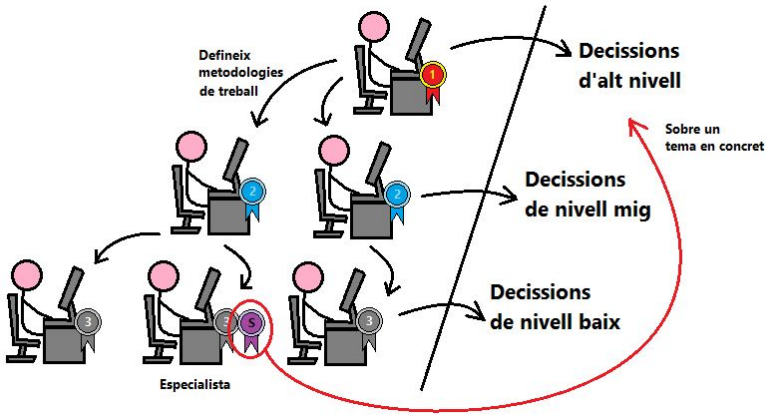
Una **jerarquia** defineix **qui té més pes** a l'hora de definir aspectes comuns com metodologies de treball i decisions importants sobre les tasques. Exemples d'això serien la distinció entre grafistes júnior i sènior o entre artistes i directors d'art. El criteri per establir la jerarquia hauria de ser el **grau d'experiència** i/o **coneixements** de cada membre de l'equip, tot i que a l'àmbit privat és una decisió que pot dependre d'altres factors (vegeu la figura 1.7).

FIGURA 1.7. Jerarquies



D'altra banda, una **especialitat** reconeix una àrea en què un membre de l'equip té més coneixements, de manera que el seu criteri **pot tenir prioritat** sobre el d'altres, fins i tot si jeràrquicament tenen posicions superiors. Seria el cas per exemple d'un programador especialista en comunicacions per xarxa, que pot ser la màxima autoritat entre els programadors, però només en aquest aspecte (vegeu la figura 1.8).

FIGURA 1.8. Especialitats



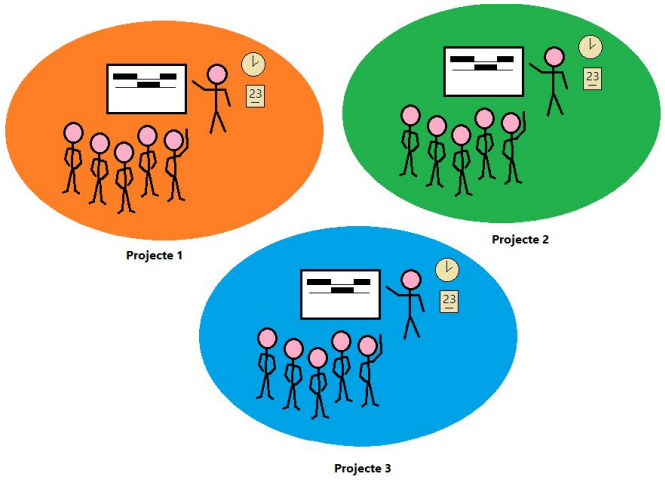
1.1.5 Estils organitzatius

En funció del nombre total de persones que formin part d'una empresa, del nombre i durada dels projectes que es duguin a terme, i de com siguin de fixos els processos de producció, podem trobar dos **estils organitzatius** principals, el treball per equips i el treball per departaments.

En el **treball per equips** s'assigna cada projecte a un equip, que s'ha d'encarregar de totes les tasques que generi. En aquest estil de treball poden establir-se unes jerarquies entre els diferents membres en general no gaire profundes que a vegades poden existir només en el context del projecte. És un estil on s'afavoreix el contacte entre persones amb rols diferents i s'afavoreix per tant el treball **interdisciplinari**,

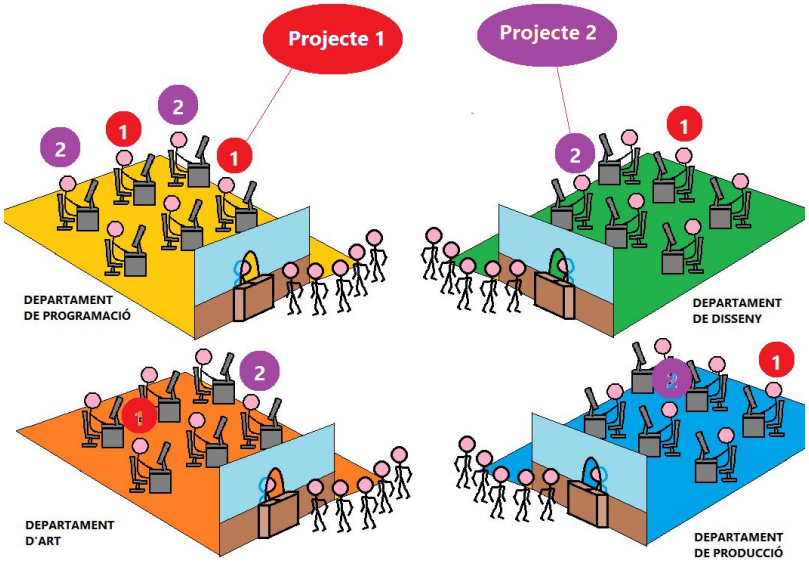
l'equip es reuneix amb freqüència i és relativament senzill canviar les prioritats de les tasques i fer-ne el seguiment (vegeu la figura 1.9).

FIGURA 1.9. Treball per equips



En el **treball per departaments** les persones que treballen a l'empresa es divideixen per grups anomenats *departaments*. Cada departament agrupa tots els treballadors que assumeixen un **mateix rol o rol concret**, i s'estableixen unes jerarquies profundes i relativament estàtiques entre ells. Els projectes els assumeix col·lectivament tota l'empresa, i s'assignen recursos dins de cada departament per fer les diferents tasques (vegeu la figura 1.10).

FIGURA 1.10. Treball per departaments



En aquest cas, la comunicació es **burocratitza** i hi ha poc contacte entre membres amb diferents rols, de manera que resulta més difícil canviar les prioritats i fer el seguiment dels projectes. A canvi, l'empresa pot resultar en conjunt més productiva, ja que totes les tasques d'un mateix tipus les realitza sempre **el mateix grup humà**, que amb el temps troba els processos més òptims per fer-les.

Treball per departaments vs. treball per equips

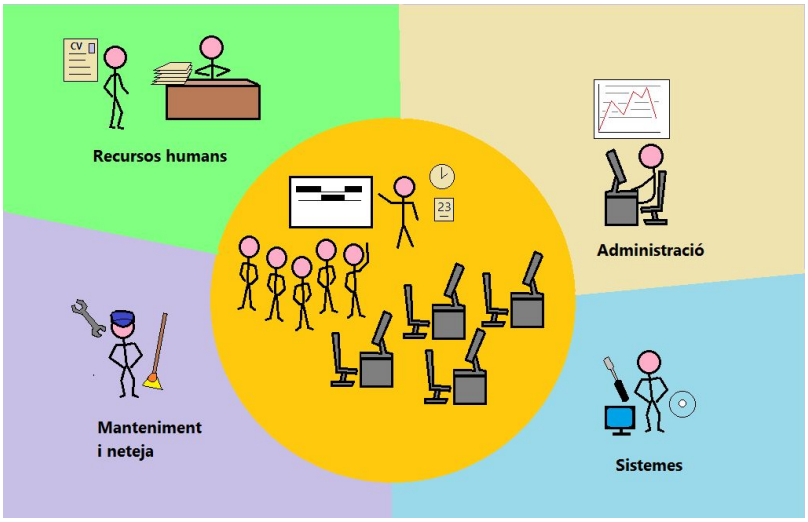
Cada estil és adequat per a un tipus de projecte. El treball per departaments és propi de sectors on els projectes són molt semblants entre si i les tasques que ha de fer cada un d'ells són fixes i conegudes, per exemple el sector editorial, mentre que el treball per equips és més adequat per a projectes on hi ha un grau més alt d'experimentació i els processos de producció no són coneguts a priori, com el sector dels videojocs de petit format, on cada projecte és diferent.

1.1.6 L'entorn de l'equip de desenvolupament

L'equip de desenvolupament és aquell que s'encarrega d'elaborar el producte audiovisual multimèdia, bé perquè ho fa directament produint els seus components, cas dels rols productius, bé perquè organitza el treball o dirigeix altres rols, cas dels rols organitzatius o bé perquè assumeix una funció dins el procés de desenvolupament, cas dels rols auxiliars.

Ara bé, aquest equip existeix dins un entorn humà i material, que el proveeix del context físic i els recursos que necessita, a més d'**una sèrie de serveis** que, si bé no tenen a veure amb cap aspecte concret de la producció, sí que faciliten i afavoreixen que aquesta es desenvolupi de la millor manera possible (vegeu la figura 1.11).

Figura 1.11. Entorn de l'equip de desenvolupament



Aquest entorn pot afavorir el desenvolupament del projecte, però també el pot perjudicar, i produir **externalitats** com pressions per part dels inversors o decisions sobre la forma de dur a terme les tasques preses per persones que no en tenen un coneixement directe, i és responsabilitat també dels rols organitzatius del projecte, especialment del productor, protegir l'equip de desenvolupament i **aïllar-lo** d'aquest tipus d'influències.

De manera anàloga als rols que podem trobar en un equip de desenvolupament, a l'entorn de l'equip hi ha també una sèrie de persones que treballen a l'empresa i que també adopten rols i formen jerarquies i especialitats. La tasca d'aquest equip humà és gestionar les necessitats dels diferents projectes que es duen a terme a l'empresa, i usualment s'organitzen en departaments (vegeu la figura 1.11):

- El **departament d'administració** gestiona els recursos econòmics. Autoritza decisions com l'adquisició de materials o la contractació de personal i serveis i vetlla pel balanç entre ingressos i despeses. L'equip de desenvolupament pot comunicar a aquest departament necessitats com l'**adquisició de nou material**, tot i que també pot rebre pressions si no s'estan assolint els objectius econòmics.
- En les empreses tecnològiques el **departament de sistemes** vetlla pel **manteniment** dels equips informàtics, les xarxes de comunicacions i els recursos compartits, com servidors, discs durs i impressores. També s'encarrega de la instal·lació i el manteniment del programari i assessora sobre l'adquisició de nou material. Els membres de l'equip de desenvolupament poden notificar-li de qualsevol incidència que tinguin amb els equips i informar-lo de les necessitats que tinguin quant a nou programari o equipament.
- El **departament de recursos humans** s'encarrega d'aspectes relacionats amb el **tracte amb les persones** que formen part de l'empresa. Els membres de l'equip de desenvolupament han de notificar-li qualsevol circumstància personal que pugui afectar el desenvolupament del projecte, així com de les absències previstes o preferències quant als períodes de vacances. Entre les seves funcions hi ha:
  - Supervisar i organitzar els **processos de selecció** de personal per tal de formar els equips de desenvolupament amb persones qualificades per a la feina que han d'afrontar.
  - Gestionar els **possibles conflictes personals** i diferències que puguin sorgir entre els membres dels equips de desenvolupament i procurar que aquestes es resolguin de forma diplomàtica.
  - Vetllar perquè no es produeixin **situacions abusives** a l'empresa, com assetjament o tracte desconsiderat.
  - Gestionar circumstàncies comunes com baixes i absències.
- El **departament de manteniment i neteja** vetlla per l'estat òptim de les **instal·lacions**, i s'encarrega de resoldre les incidències que tinguin a veure amb serveis com l'aigua i el subministrament elèctric així com de mantenir unes condicions higièniques adequades a l'espai de treball.

1.2 L'organització del treball

Heu de tenir clar que **treballar no és equivalent a produir**, ja que si el treball es fa de manera desorganitzada, no es pot garantir que s'estiguin assolint uns productes amb unes condicions concretes, sinó només que s'està fent una **despesa d'energia** i recursos, i això pot o no pot afavorir el desenvolupament del projecte. Per aquest motiu, cal estructurar el treball, organitzant-lo en unitats, de manera que no hi hagi ambigüitats quant a l'aspecte concret del projecte en què s'està treballant i els resultats que s'esperen d'aquest treball.

Una vegada estructurat el treball en aquestes unitats, podem **caracteritzar-les**, és a dir, establir una sèrie de propietats que mesurin aspectes com la dificultat o l'extensió de cadascuna d'elles, de manera que es pugui **planificar** el treball per fer-lo amb ordre i de la forma més convenient.

Per tant, els **elements o fases** que es tenen en compte a l'hora d'organitzar el treball són:

1. Les unitats de treball
2. Els nivells de producció
3. La caracterització d'un objectiu
4. La caracterització d'una tasca

1.2.1 Unitats de treball

Les unitats de treball principals que podem distingir són la tasca i l'objectiu. Una **tasca** consisteix en l'aplicació d'un esforç per part d'un o més membres d'un equip de desenvolupament durant un cert temps per tal d'assolir un objectiu (vegeu la figura 1.12). Podem distingir entre:

- Tasques **simples** o individuals, quan un sol membre de l'equip pot dur-les a terme.
- Tasques **complexes** o d'equip, quan no es poden dur a terme directament, sinó que cal descompondre-les en tasques individuals que comunament s'assignaran a diferents membres de l'equip (vegeu la figura 1.13).

FIGURA 1.12. Tasca

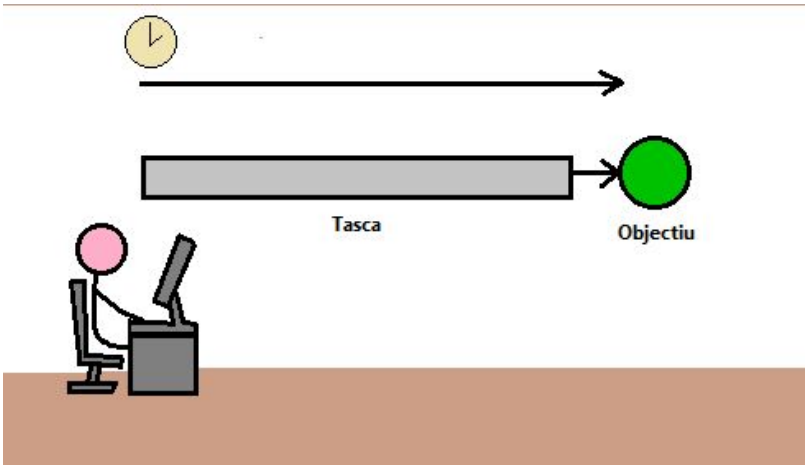
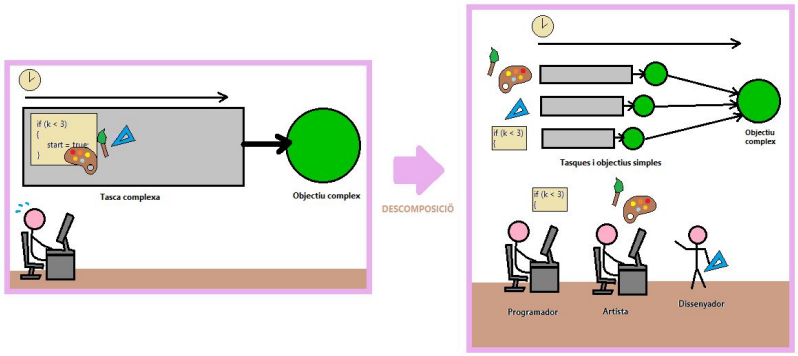


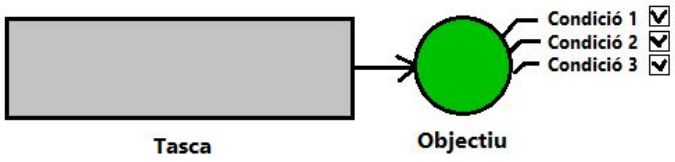
FIGURA 1.13. Descomposició d'una tasca complexa



Veurem com descompondre aquestes tasques al punt "La tasca del productor: elaboració, comunicació i supervisió d'un pla de treball", d'aquest mateix apartat.

D'altra banda, un **objectiu** consisteix en una sèrie de condicions que volem que es compleixin una vegada s'hagi dut a terme una tasca. Aquestes condicions són independents del mètode emprat per fer-la i s'han de poder expressar sense ambigüitat, per poder verificar de forma positiva si l'objectiu s'ha assolit o no (vegeu la figura 1.14).

FIGURA 1.14. Objectiu

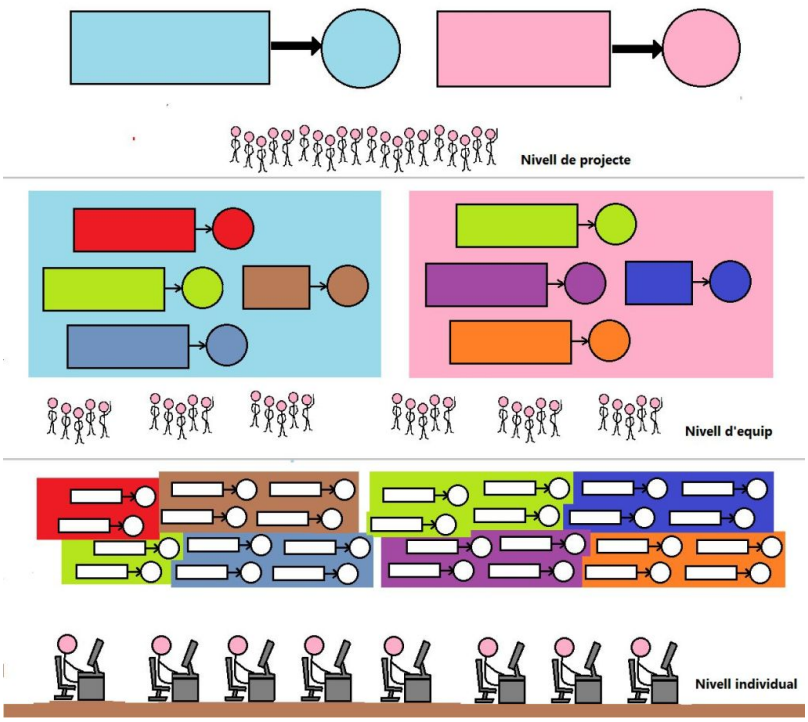


Moltes vegades podem fer servir **tasca i objectiu** de forma intercanviable, ja que tota tasca implica un objectiu i tot objectiu implica una tasca; però no els podem unificar perquè són conceptes diferents.

1.2.2 Nivells de la producció

Les tasques i els objectius existeixen a **diferents nivells** dins d'un projecte. Aquests nivells formen una **jerarquia** on les tasques i objectius de nivell superior es descomponen en tasques i objectius dels nivells inferiors. És important notar que en aquesta jerarquia, les tasques i objectius dels nivells alts són abstraccions que ens ajuden a guiar el desenvolupament, ja que les úniques tasques que tenen un impacte directe al projecte són les del nivell més baix. En general, podem distingir tres nivells: individual, d'equip i de projecte (vegeu la figura 1.15).

Figura 1.15. Els nivells de la producció



El **nivell individual** comprèn les tasques i objectius que s'assignen directament als membres d'un equip de desenvolupament i que aquests van desenvolupant i assolint **diàriament**. Aquestes tasques suposen una modificació directa dels arxius i documents que formen el projecte i són les que afegeixen els canvis que fan créixer. Típicament les tasques d'aquest nivell es completen en **terminis curts**, de l'ordre d'hores o dies i és molt senzill i ràpid verificar que s'han assolit els objectius establerts. A aquest nivell, per gestionar les tasques, n'hi pot haver prou amb una **llista**.

Una tasca o objectiu passa a ser de **nivell d'equip** quan no la pot assolir un sol membre individualment, bé sigui per la seva **complexitat** o perquè suposa un volum de treball massa gran, de manera que per assolir-les primer s'han de **descompondre** en tasques individuals tot formant un pla de treball. Una vegada les tasques individuals que formen aquest pla s'hagin dut a terme, s'avalua si els objectius de la tasca de nivell d'equip s'han assolit.

Aquestes tasques es desenvolupen en terminis de **durada intermèdia**, de l'ordre de dies o setmanes i verificar que se n'han assolit els objectius pot ser més costós, de manera que poden requerir algun tipus de *testeig* o verificació. A aquest nivell, se solen organitzar les tasques i objectius en **elements compartits** com pissarres i taulells.

Examinarem el paper del *testeig* o verificació a l'apartat "Avaluació del projecte" d'aquesta unitat.

Però, al punt més alt de la jerarquia de tasques i objectius trobem el **nivell de projecte**. Inclou els aspectes no ja associats a una part concreta del projecte, sinó al projecte com a conjunt. A aquest nivell sovint es parla més d'objectius que de tasques, i en podem distingir dos tipus:

- Objectius que es pacten directament amb les parts interessades o ***stakeholders*** del projecte, és a dir, els inversors i clients, anomenats ***requisits***.

- Objectius que l'equip de desenvolupament considera **necessaris** per complir aquests requisits.

Això dona lloc a un **pla de treball a llarg termini**, on els punts de verificació dels objectius s'anomenen *fites*. Es distingeix entre **fites internes**, quan els objectius els revisa només l'equip i **fites externes**, quan la revisió la fan també les persones interessades.

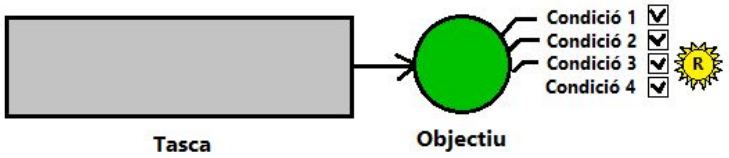
Requisits i tasques a nivell de projecte

Quant als requisits, com qualsevol altre tipus d'objectiu, són simplement **condicions** que ha de complir el projecte (vegeu la figura 1.16), tot i que en aquest cas, a més, han de ser **directament apreciables** per l'usuari, ja que la revisió la faran persones externes a l'equip de desenvolupament, i en general podem agrupar-los en dos:

- Els requisits **funcionals**, que tenen a veure amb què fa el producte interactiu, és a dir, **com s'ha de comportar** davant d'una acció de l'usuari. Si es fa servir una metodologia basada en diagrames UML, aquests requisits es poden definir a través del diagrama de casos d'ús i els diagrames de seqüència.
- Els requisits **no funcionals** tenen a veure amb la *qualitat* amb què es duen a terme els requisits funcionals. Per exemple, podem posar com a requisit no funcional que una aplicació interactiva multimèdia sempre respongui a una acció de l'usuari en un temps no superior a un segon.

Teniu una explicació sobre aquests diagrames a l'apartat "Determinació d'objectius, estil gràfic i narratiu d'un projecte" de la unitat "Disseny i planificació del projecte", d'aquest mateix mòdul.

FIGURA 1.16. Condicions que són requisits



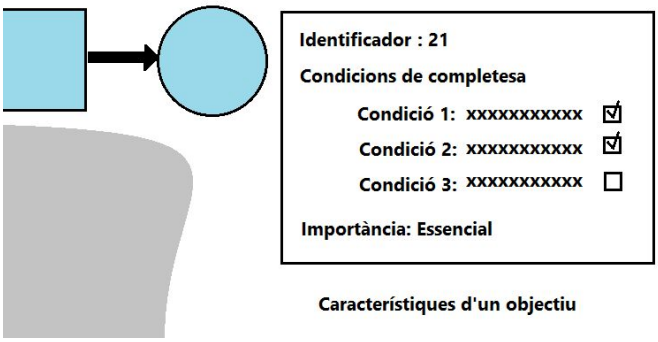
Quant a les tasques necessàries per complir amb aquests objectius, aquestes esdevenen **tasques de nivell de projecte**, que s'han de descompondre en tasques d'equip i organitzar-se també en un pla de treball, i a aquest nivell sol prendre la forma d'un **full de ruta**. Aquests tipus de tasques es duen a terme en **terminis llargs**, de l'ordre de mesos o anys, i per verificar que s'han assolit correctament sol ser necessari algun tipus de *testeig* sistemàtic, ja que comprovar tots els casos que es poden donar sol ser inabordable per a una persona o inclús per a un equip petit.

Les tasques i objectius de nivell de projecte se solen pactar i discutir en **reunions** amb les parts interessades i s'expressen de forma molt general a través d'eines com ara **presentacions**.

1.2.3 Caracterització d'un objectiu

La caracterització dels objectius consisteix a considerar certes **propietats** que ens permetin organitzar-los millor en un pla de treball. Aquestes propietats poden ser més o menys importants segons si es tracta d'un objectiu de nivell individual, d'equip o de projecte, però sempre estan presents (vegeu la figura 1.17).

FIGURA 1.17. Caracterització d'un objectiu



Tot i que pot semblar redundant, és molt important caracteritzar sempre qualsevol element dins d'un pla de treball amb un **identificador** únic, ja que això ens permetrà fer-hi **referència** sense ambigüitat des d'altres punts del pla de treball, per exemple quan volem expressar que un objectiu concret és requisit per poder-ne assolir un altre.

Condició de completesa

La condició de completesa és la característica més important d'un objectiu, ja que si l'expressem correctament, ens permet avaluar sense ambigüitat si l'objectiu **s'ha assolit o no**. S'han de poder expressar, doncs, com una sèrie d'afirmacions objectives i **verificables**, idealment amb una prova que doni un resultat del tipus sí o no.

És extremadament important poder fer aquesta verificació, especialment als nivells més baixos de la jerarquia de nivells de producció, ja que això ens permet **afirmar amb seguretat** que el producte està complint amb les condicions que es demanen. Si definim les condicions de completesa de forma ambigua, el producte no es podrà construir mai amb seguretat, ja que no podrem verificar que les seves parts funcionen correctament i, per tant, es donaran per tancats objectius que no s'han assolit, i això farà que el conjunt eventualment esdevingui inestable i es col·lapsi.

Així doncs, **no són bones condicions** de completesa aquelles que incloguin:

- Apreciacions o opinions personals, per exemple que el resultat sigui "bonic", "simpàtic" o "xulo", ja que això és un criteri subjectiu i no verificable, que a més pot variar en el temps, inclús quan reflecteix l'opinió d'una mateixa persona.
- Afirmacions tautològiques pròpies del llenguatge publicitari, per exemple "el joc que esperen els jugadors", ja que són expressions buides de contingut i, per tant, no verificables.
- Afirmacions difuses, com per exemple "que tingui un millor rendiment", ja que no hi ha un límit clar que ens permeti verificar si l'afirmació es compleix o no.

En canvi, **són bones condicions** de completesa:

- Aquelles que incloguin fets directament observables, per exemple que, si premem el botó de saltar, el personatge salti.
- Expressions que incloguin magnituds quantificables, per exemple que el temps de càrrega d'un escenari no superi en cap cas els cinc segons.

Així i tot, la verificació pot esdevenir una tasca complexa, ja que els productes interactius multimèdia tenen un nombre molt alt d'estats interns que poden fer que no sempre reaccionin igual davant d'una mateixa acció de l'usuari. Per aquest motiu sovint els objectius s'han de **verificar estadísticament**, és a dir, cal arribar a un grau de confiança estadística molt gran a partir de verificar-los **repetidament** en diferents circumstàncies.

Un altre aspecte que també és important mencionar dintre de les condicions d'assoliment d'un objectiu és que aquestes **sempre s'han d'establir per anticipat**, és a dir, mai s'han de redefinir quan el treball ja s'ha començat, ja que això comporta introduir un nou objectiu que no queda pròpiament registrat. En aquests casos és millor **cancel·lar** completament l'objectiu inicial o esperar que s'assoleixi l'objectiu anterior i llavors plantejar el canvi com un nou objectiu.

Importància d'un objectiu

La importància d'un objectiu és la mesura d'en quin grau aquest és **essencial** de cara a la consecució del projecte. Tenint en compte aquest aspecte podem categoritzar els objectius en objectius essencials, recomanables i opcionals:

Teniu una explicació del paper dels requisits en el punt "Requisits i tasques a nivell de projecte" d'aquest apartat.

- Els **objectius essencials** s'han d'assolir **sempre**, ja que formen part de les **expectatives mínimes** que té l'equip o les persones interessades sobre el projecte. Per exemple, si fem un joc de plataformes, tothom esperarà que el personatge pugui saltar. Dintre dels objectius essencials s'inclouen també els **requisits** pactats amb les persones interessades.
- Els objectius **recomanables** són aquells que asseguren una **bona qualitat** del projecte, és a dir, que milloren l'experiència de l'usuari tot i no ser

absolutament essencials. Per exemple podem fer que si el nostre personatge salta, a més tingui una animació que faci el salt més atractiu.

- Els objectius **opcionals** són aquells que fan que el projecte vagi més enllà de les expectatives i resulti més estimulant per a l'usuari, però que en cas de necessitat es poden **descartar**. Usualment es classifiquen en aquesta categoria els objectius que suposen un **cert risc** i poden comprometre altres objectius. Per exemple, podríem fer que el personatge tingués un doble salt, però si ja tenim construït un escenari, això suposarà revisar-lo per assegurar que mai pugui sortir del camí que hem previst.

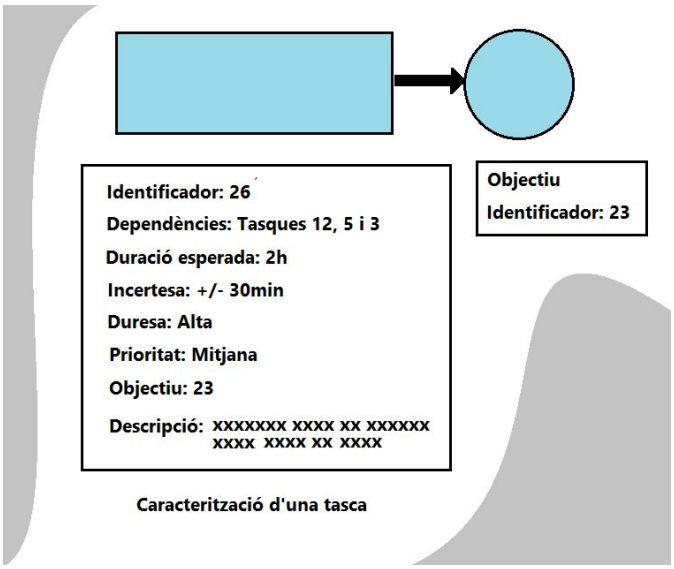
Si caracteritzem els objectius per importància, com a productors, podrem decidir entre **diferents estratègies** que tindran un impacte al producte final:

- Si prioritzem els objectius essencials i descartem la resta, estarem adoptant una estratègia conservadora. Obtindrem un producte amb les característiques mínimes però que possiblement no destacarà enfront d'altres d'un perfil similar i no introduirem cap mena d'estímul a l'usuari per preferir-lo, a canvi d'estalviar riscos.
- Si a més dels objectius essencials complim una bona quantitat d'objectius recomanables, adoptarem una estratègia equilibrada i el nostre projecte millorarà en comparació a d'altres amb un perfil similar, tot i no suposar una diferència radical respecte a ells.
- Si prioritzem els objectius opcionals, adoptarem una estratègia innovadora i el nostre projecte destacarà enfront de la competència tot i assumir un risc més alt.

1.2.4 Caracterització d'una tasca

De manera anàloga a un objectiu, també podem caracteritzar una tasca. Com que les tasques són **activitats físiques** i no només condicions abstractes, la seva caracterització inclou **aspectes pràctics** com la durada i el procediment a emprar. Aquestes característiques es fan servir per avaluar els recursos que suposaran, i situar-les al lloc més convenient dins dels plans de treball (vegeu la figura 1.18).

FIGURA 1.18. Caracterització d'una tasca



Com qualsevol altre element d'un pla de treball, una tasca ha de poder identificar-se **sense ambigüitat**, de manera que s'hi pugui fer referència des d'altres punts d'aquest pla, per exemple per expressar que una altra tasca en depèn. Una tasca es caracteritzarà per les seves dependències, la durada, la dificultat, la prioritat, l'objectiu i la descripció.

Dependències d'una tasca

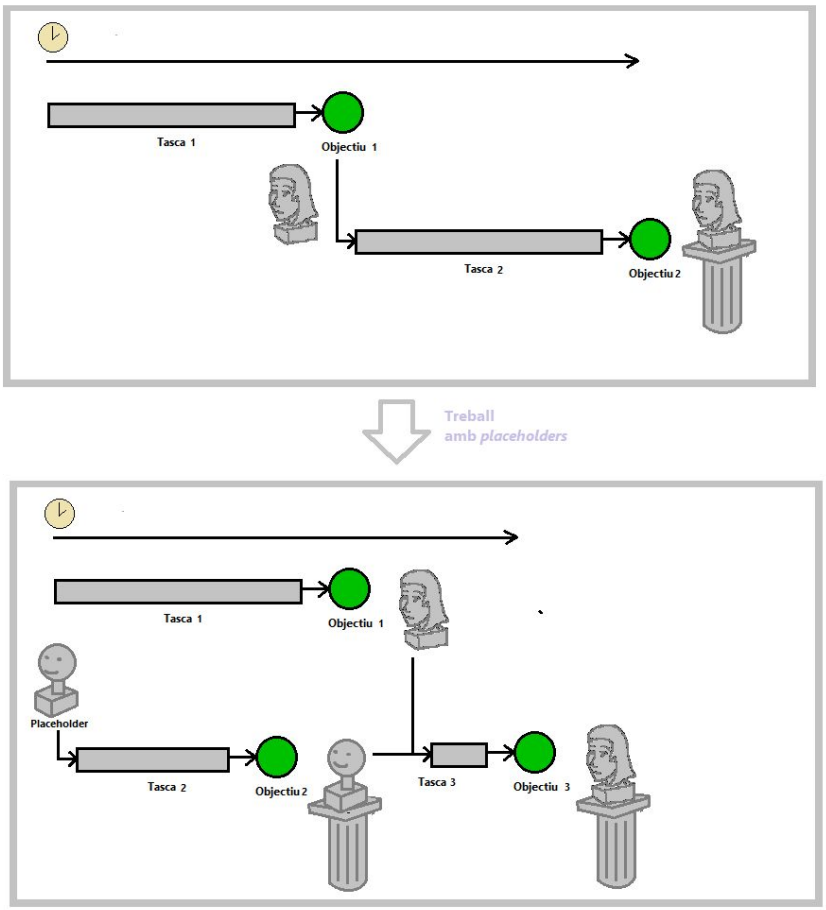
Una tasca pot incloure dependències, és a dir, **altres tasques** que s'hagin d'haver assolit per tal que es pugui dur a terme. Això pot ser degut al fet que necessita alguns dels recursos produïts per aquestes tasques o bé perquè requereix que es compleixi alguna condició que forma part dels objectius d'aquestes. La tasca finalitzarà després de les seves dependències.

> L'existència de dependències té un impacte molt important en un pla de treball, ja que **estableix un ordre obligatori entre les tasques**, limitant el grau de paral·lelització que es pot fer i establint implícitament una durada mínima.

Visualitzar aquest tipus d'estratègies és més senzill fent servir un diagrama de Gantt.

Una manera de minimitzar aquesta dependència de tasques anteriors consisteix a fer servir versions simplificades o ***placeholders*** dels recursos que encara no s'han produït, de manera que la tasca pugui iniciar-se encara que no s'hagin assolit totes les seves dependències. Aquesta estratègia fa que la tasca es pugui desenvolupar en paral·lel a les tasques de què depèn, però a canvi afegeix una tasca addicional que sigui la integració dels recursos definitius una vegada s'hagin produït i la verificació de què els objectius s'han assolit (vegeu la figura 1.19).

FIGURA 1.19. Ús dels 'placeholders'



Durada esperada d'una tasca

Per poder situar una tasca dins d'un pla de treball, una de les mesures més importants és la seva durada. En general, establir-la amb precisió resulta difícil, ja que pot dependre de factors com la **complexitat** tècnica de la tasca, el grau d'experimentació que suposa per a qui l'ha de dur a terme i els possibles **contratemps** que puguin sorgir durant el seu desenvolupament.

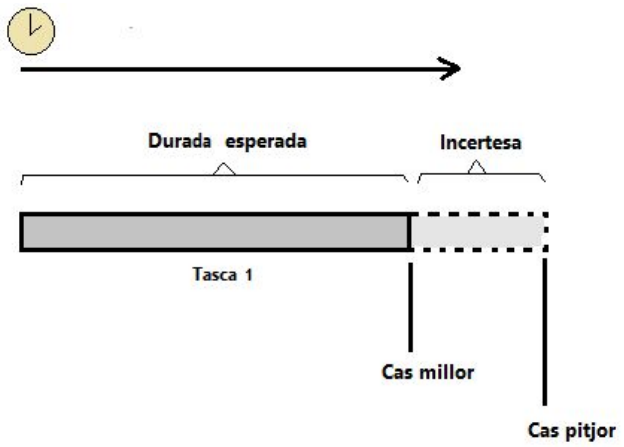
Tot i que amb l'experiència els productors i membres de l'equip poden donar unes mesures força confiables, hem de tenir en compte que totes les durades de les tasques en un pla de treball **sempre són estimacions**. En concret, la durada esperada representa una estimació de la durada en el millor cas, és a dir, si no es dona cap circumstància excepcional que dificulti la consecució de la tasca.

Incertesa d'una tasca

Sobre la base de la durada esperada s'estima una altra mesura de la durada, que és la incertesa. Representa el temps que estimem que es pot demorar la tasca si ens trobem circumstàncies desfavorables. Amb això podrem determinar la durada mínima i màxima de la tasca.

Un aspecte que cal tenir en compte és que a mesura que les tasques es desenvolupen, el **grau de coneixement** sobre la tasca augmenta i el **grau d'incertesa** en la durada es redueix. Per tant, resulta útil que el productor revisi periòdicament aquestes estimacions amb l'equip. També cal tenir en compte que en el cas de les tasques d'equip o de projecte, la incertesa s'obté combinant les incerteses de les tasques de nivells inferiors de què estan compostes (vegeu la figura 1.20).

FIGURA 1.20. Incertesa en la durada d'una tasca



Duresa o dificultat d'una tasca

La duresa o dificultat és una mesura de la intensitat de l'**esforç** que suposa per a un membre de l'equip de desenvolupament o un equip dur a terme la tasca. Deixar que el mateix equip caracteritzi les tasques d'aquesta manera permet que el productor estimi el **grau de desgast** que estan patint; per exemple, tractar de no sobrecarregar-los assignant-los massa tasques difícils en un interval curt de temps.

Prioritat d'una tasca

Ordre de prioritats

Podem entendre la prioritat com una mena de "volant" que fa servir el productor per dirigir el treball de l'equip, que seria "el cotxe".

La prioritat defineix **l'ordre** en què s'ha de dur a terme una tasca concreta respecte a altres tasques. El valor concret que prengui la prioritat d'una tasca en un moment donat és sempre resultat d'una **circumstància concreta** del procés de desenvolupament i establir-les és responsabilitat del productor. Podem distingir diferents criteris per establir les prioritats.

Una opció que podem fer servir per establir el valor de la prioritat d'una tasca és **relacionar-lo directament amb la importància** del seu objectiu, de manera que si un objectiu és essencial, les tasques relacionades amb ells esdevinguin més prioritàries. Encara que aquesta sembli una estratègia lògica, si la mantenim de forma rígida, pot passar que els objectius no essencials quedin permanentment relegats i no es puguin assolir.

Un altre factor que determina les prioritats de les tasques són els **aspectes psicològics** del treball. En concret, les persones tenim una sensació de satisfacció que sovint depèn més de la **quantitat** de tasques que completem que de la **dificultat** de cascuna d'elles, de manera que com a productors, a vegades cal prioritzar les tasques més senzilles, encara que no siguin importants, per tal que

l'equip guanyi confiança i pugui enfrontar les tasques més difícils amb millor disposició.

Finalment, els projectes existeixen en un **entorn** que pot produir unes **externalitats**, a les quals també cal adaptar les prioritats. Per exemple, pot passar que un recurs necessari, com un equip informàtic, s'espatlli o s'aproximin les dates en què un membre de l'equip vol agafar vacances, i cal re prioritzar les tasques per tenir en compte aquesta circumstància.

Objectiu i descripció d'una tasca

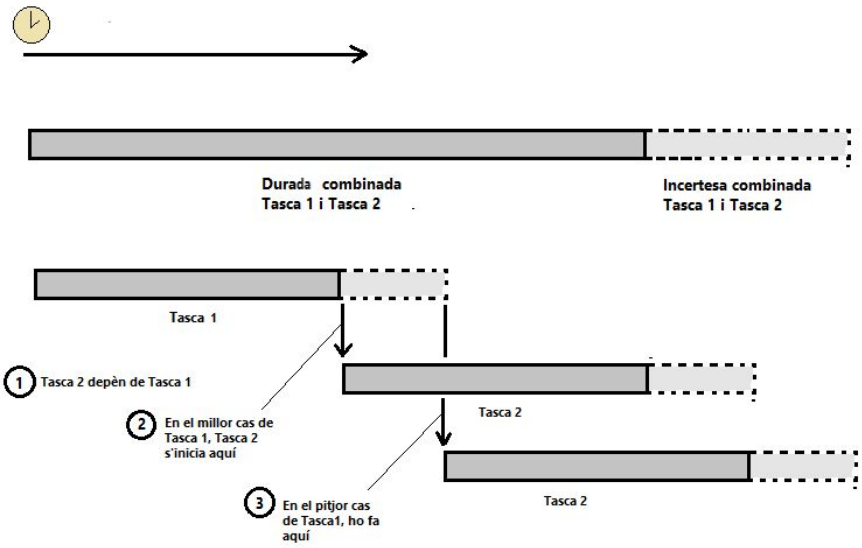
Vegeu la secció "Objectius" i "Caracterització d'un objectiu" d'aquest apartat per tenir una explicació més detallada del que és un objectiu.

No hem de confondre l'objectiu amb la descripció. Tota tasca té associat un **objectiu** que conté les condicions que s'han de poder verificar una vegada es dugui a terme.

A diferència d'un objectiu, que està format per condicions independents del procediment que es faci servir per assolir-les, les tasques han d'anar acompanyades d'una **descripció** que especifiqui clarament **l'abast** de la tasca i, opcionalment, el **procediment** que s'ha de seguir per dur-les a terme.

En el cas d'una tasca individual, aquesta descripció pot consistir en una sèrie de passes concretes, mentre que en una tasca d'equip la descripció consisteix en un pla de treball (vegeu la figura 1.21).

FIGURA 1.21. Durada de tasques combinades



Especialment al principi dels projectes, pot succeir que l'objectiu d'una tasca sigui establir quin és el millor procediment per fer una altra tasca.

1.3 La tasca del productor: elaboració, comunicació i supervisió d'un pla de treball

El productor és la figura encarregada de l'organització de l'equip i estructuració del treball, així com del seguiment i la revisió de les tasques i objectius d'un projecte. Tot i que també té un paper en les tasques individuals, la seva funció és especialment important en les tasques d'equip i de projecte.

Durant la major part del temps, la tasca d'un productor consisteix a elaborar i fer el seguiment dels plans de treball, però també té altres funcions, com **afavorir** en tot el possible el desenvolupament de les tasques i **anticipar** els recursos que puguin necessitar els membres de l'equip per dur-les a terme. Addicionalment, el productor també ha de facilitar la **comunicació** entre els membres de l'equip i organitzar **reunions** on es puguin determinar quines són les millors estratègies per assolir els objectius.

Davant d'una tasca o objectiu complex el productor ha de ser capaç, amb l'ajut de l'equip, d'elaborar un pla de treball que permeti abordar-la mitjançant un seguit d'etapes. Aquest pla constarà d'una descomposició de la tasca o l'objectiu en tasques i objectius més senzills que eventualment esdevindran tasques individuals directament assolibles pels membres de l'equip. Les **etapes d'un pla de treball** són cinc:

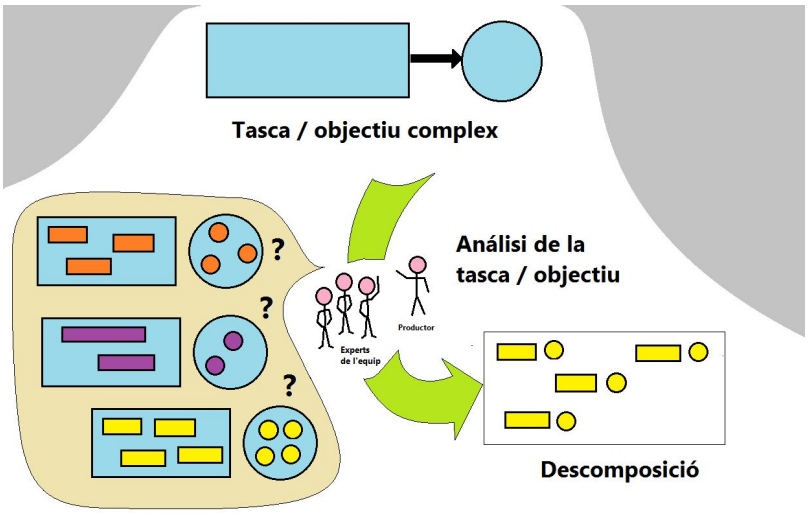
1. Anàlisi i descomposició
2. Reunió de l'equip
3. Elaboració del pla
4. Comunicació del pla
5. Supervisió del pla

1.3.1 Anàlisi de l'objectiu i descomposició de la tasca

La primera etapa consisteix a analitzar l'objectiu o tasca que cal fer per arribar a una **comprensió** dels seus possibles components. Per fer això el productor pot haver de **consultar** amb els membres de l'equip amb més experiència, que idealment també seran els responsables de dur la tasca a terme fins a trobar una descomposició que pugui dur-se a la pràctica (vegeu la figura 1.22).

Si l'objectiu suposa una novetat per al mateix equip, pot ser útil consultar també **altres equips** que hagin abordat tasques similars o dedicar un temps a fer **recerca** sobre el procés de desenvolupament d'altres projectes similars en el passat.

FIGURA 1.22. Anàlisi d'una tasca complexa

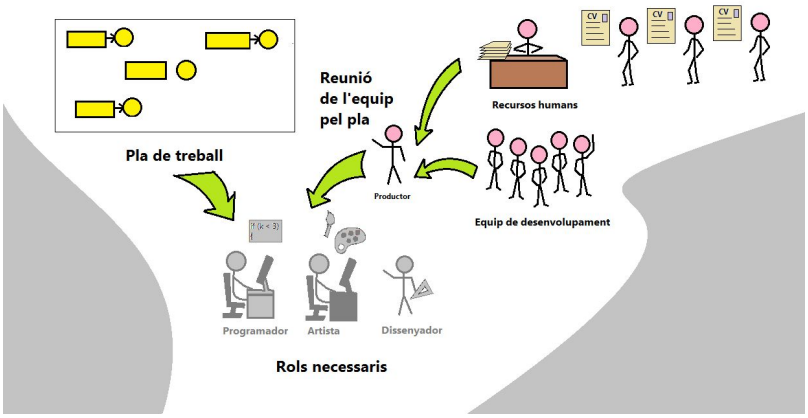


Reunió de l'equip

Una vegada establertes les tasques i objectius, el productor ha de reunir l'equip humà necessari per dur-les a terme. Podem trobar dos casos: o bé l'equip compta amb membres suficients que puguin assumir els rols adequats a les tasques o bé hi ha algun tipus de mancança.

En aquest segon cas, caldrà o bé reforçar l'equip amb **noves incorporacions** o bé crear **equips *ad hoc*** per afrontar aquesta tasca en concret. Si fem això últim, alguns membres de l'equip de desenvolupament hauran de sortir del seu rol habitual per tal d'assumir les tasques (vegeu la figura 1.23).

FIGURA 1.23. Reunió de l'equip que ha de portar a terme un pla de treball



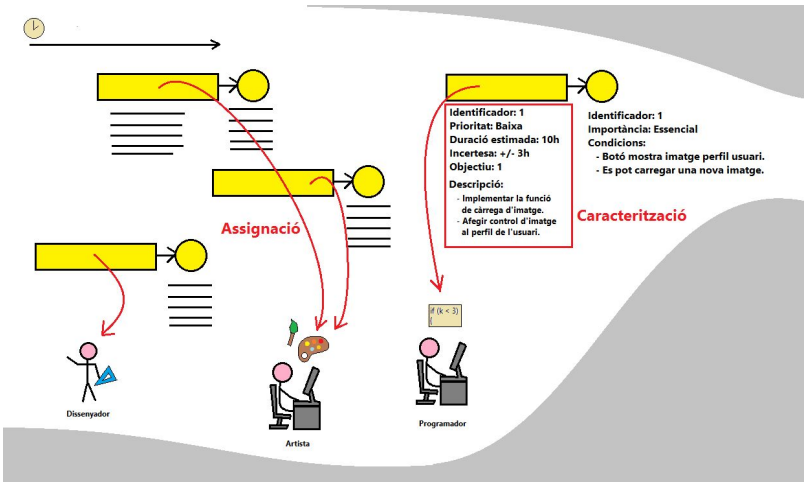
Equips vs. departaments

Aquesta forma àgil d'organitzar un equip, per a un objectiu concret, és pròpia del treball per equips, i molt difícil d'aconseguir quan es treballa per departaments.

Elaboració del pla

Una vegada s'han determinat les tasques i la composició de l'equip, el productor ja pot començar a elaborar el pla de treball, assignant les tasques als membres de l'equip i determinant amb el seu ajut algunes característiques d'aquestes, com la durada i la incertesa (vegeu la figura 1.24).

FIGURA 1.24. Elaboració del pla i assignació de tasques



L'estil brúixola i l'estil mapa

Arribats a aquest punt, podem distingir dos estils a l'hora d'elaborar el pla; que podríem anomenar *brúixola* i *mapa*.

Un productor estil mapa establirà des del principi un pla de treball detallat on hi hagi poc marge a la improvisació i el tractarà de dur a terme independentment de les circumstàncies que es donin durant la seva posada en pràctica. Aquest estil és adequat quan les tasques a fer són conegudes o poden definir-se amb precisió des del principi i l'entorn de treball de l'equip és força estable.

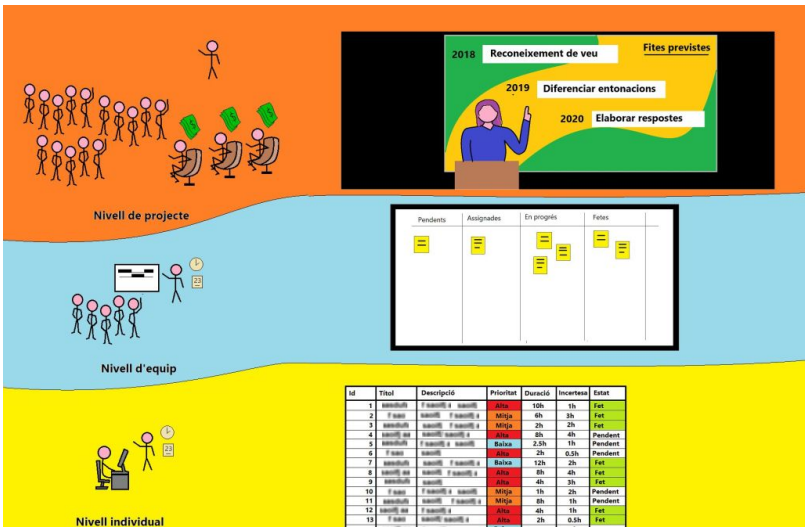
Un productor estil brúixola, en canvi, farà un pla mínim per tal d'assolir els objectius inicials i no planificarà res més per avançat, sinó que esperarà que aquests objectius s'assoleixin i només llavors estendrà el pla de treball a nous objectius. Aquest estil és adequat quan el projecte implica un grau d'experimentació gran o l'entorn de treball sovint produeix externalitats que afecten l'equip de desenvolupament.

Comunicació del pla

Una vegada el pla s'ha elaborat, cal comunicar-lo a l'equip. La forma en què es fa aquesta comunicació depèn del nivell en què s'estigui treballant. En el cas de les tasques d'equip, es pot elaborar algun tipus de diagrama en **suports compartits**, com ara pissarres o taulells físics o virtuals, que mostri les dependències entre les diferents tasques, i exposar-lo a una reunió semiformal amb l'equip, i individualment, com una **llista de tasques** amb les seves prioritats.

En el cas que el pla tracti amb tasques de **nivell de projecte**, es pot expressar també amb diagrames que mostrin les fites a llarg termini davant la totalitat de l'equip o les persones interessades amb format de presentació (vegeu la figura 1.25).

Figura 1.25. Comunicació del pla de treball



Supervisió del pla

Una vegada s'ha definit el pla de treball i s'ha comunicat a l'equip, es posa en pràctica i la tasca del productor passa a ser la seva supervisió. Això vol dir **comunicar-se** amb una certa freqüència amb els membres de l'equip que l'estan duent a terme, **comprovar** que els objectius de les tasques s'estan complint i tractar de **detectar** els obstacles que puguin comprometre'l. Per tant, la supervisió es fa a nivell individual, d'equip i de projecte.

A **nivell individual**, aquesta tasca es pot fer simplement parlant un per un amb els membres de l'equip que estan duent a terme les tasques i comprovant *in situ* que aquestes s'estan completant correctament, així com interessar-se pels problemes que pugui tenir amb les eines o preguntar-los si requereixen l'ajut d'una altra persona per dur a terme les seves tasques. Aquest tipus de supervisió és recomanable fer-la **freqüentment**, per exemple un o més cops al dia.

A **nivell d'equip**, la supervisió se sol fer en el context de reunions semiformals on cada membre de l'equip informa del progrés de les tasques que està fent i dels obstacles que està trobant. Aquestes reunions poden fer-se amb una **freqüència moderada**, per exemple un cop al dia o un cop a la setmana. Una altra manera de supervisar les tasques d'equip és parlar individualment amb cadascun dels membres que fa les subtasques i fer d'enllaç entre ells, de manera que no hagin d'interrompre el que estan fent.

A **nivell de projecte**, la supervisió del pla es fa en reunions formals on s'informa dels resultats i poden mostrar-se versions del projecte a les persones interessades per tal que puguin comprovar directament que els objectius acordats s'estan assolint. Aquestes reunions es fan amb una **freqüència baixa**, per exemple un cop al mes o un cop cada pocs mesos, proporcional a l'extensió total del projecte.

'Micromanagement'

El *micromanagement* és un estil de treball on el productor supervisa fins a les tasques més petites de cada membre de l'equip. Tot i que en algunes organitzacions es fa servir per a maximitzar la productivitat, es considera un estil molest per a les persones que duen

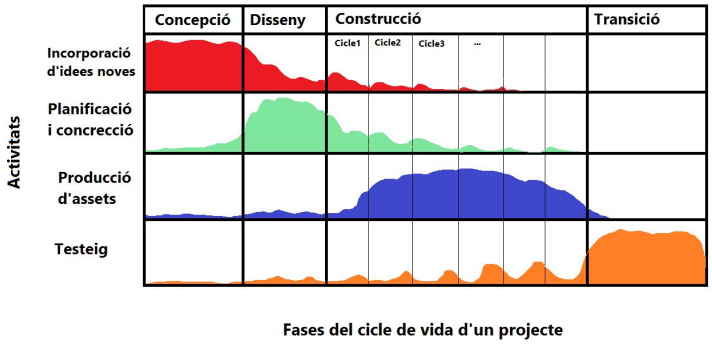
a terme les tasques, ja que dona la impressió d'una manca de confiança per part del productor i a algunes persones els pot crear inseguretat.

1.4 El cicle de vida d'un projecte

Tots els projectes passen per un cicle de vida on l'equip primer es dedica predominantment a unes tasques i després a unes altres. Aquests cicles es poden dividir en certs períodes, anomenats **etapes**. Com a productors, fins a cert punt és possible **marcar el pas** d'una etapa a l'altra, donant directives a l'equip, per la qual cosa al llarg del temps s'han plantejat **diferents models** que estableixen quin és l'ordre més convenient en què s'ha de donar segons la naturalesa del projecte.

Cadascuna de les etapes d'un projecte està determinada per un **objectiu general** diferent i la predominança d'algunes activitats respecte a altres dins l'equip (vegeu la figura 1.26). Per exemple, la verificació (o *testeig*) està poc present a les etapes inicials i molt present a les finals.

FIGURA 1.26. Etapes del cicle de vida d'un projecte



Metodologia RUP

La divisió en etapes que us proposem segueix bastant de prop l'establerta a la metodologia RUP, un sistema de gestió de projectes estàndard proposat per IBM i força implantat a les empreses que desenvolupen grans projectes de programari.

1.4.1 Etapa de concepció

A l'etapa de concepció s'estableix la **idea inicial** que serveix com a punt de partida del projecte i s'intenta arribar a una imatge de la forma que tindrà el producte una vegada desenvolupat. Les activitats que predominen més són l'elaboració de documents i imatges de concepte que permeten establir les **línies d'estil** que se seguiran posteriorment. Durant aquesta etapa els rols més importants són els especialitzats a definir aquesta **visió general** del projecte, com els dissenyadors o els directors d'art.

En termes generals, podem dir que l'objectiu de l'etapa de concepció és determinar què es vol fer.

Etapa de disseny

Una vegada s'ha establert quina és la idea inicial i s'ha arribat a una visió clara del resultat que s'està cercant, s'entra a l'etapa de disseny. En aquesta etapa es **desenvolupen** els objectius establerts a l'etapa anterior fins a **concretar-los** en estratègies concretes que permetin assolir-los amb les eines i recursos que té l'equip.

En aquesta etapa, l'activitat predominant és, sobretot, el disseny; tant per part dels dissenyadors com dels programadors. Els primers **concreten les idees** establertes en l'etapa de concepció en documents en què es detallen el comportament i l'aspecte dels diferents elements, per exemple els enemics que podem trobar a un joc interactiu o les pantalles que formaran una pàgina web, mentre que els programadors **defineixen l'estructura i els serveis** del sistema que donarà suport a aquests elements.

L'objectiu de l'etapa de disseny és establir *com es portarà a la pràctica el projecte*.

1.4.2 Etapa de construcció

Quan un projecte entra a l'etapa de construcció es coneix tant el que cal fer com la manera de fer-ho, però encara no s'ha començat a construir el producte pròpiament i, per tant, no s'han establert **fluxos de treball eficients** per fer les diferents tasques. L'objecte d'aquesta etapa és, doncs, establir aquests fluxos. El rol més determinant durant aquesta etapa és el de productor, que ha d'organitzar l'equip, elaborar plans de treball i fer el seguiment de les tasques, així com els rols productius, que han d'elaborar els diferents recursos que formen el cos del projecte.

Dintre d'aquesta etapa podem distingir unes subetapes que podem anomenar **cicles de producció** i que es corresponen als terminis que es marca l'equip per assolir els diferents objectius. Típicament als primers cicles els mètodes de producció emprats són força ineficients però, segons se succeeixen nous cicles, es van trobant mètodes de treball millors fins que la producció adopta un ritme regular i òptim quant a temps i recursos emprats.

L'objectiu de l'etapa de construcció és optimitzar el quan.

1.4.3 Etapa de transició

Una vegada s'ha construït el cos del projecte, s'entra en l'etapa de transició, on es posa èmfasi a **tancar el projecte**; és a dir, verificar que compleix amb els seus objectius i està preparat per ser lliurat als usuaris. En aquesta etapa els rols amb més activitat són els de verificació (*testeig*) i en principi l'equip no afegeix nou contingut al projecte sinó que es dedica a **corregir** tots els defectes que aquests trobin.

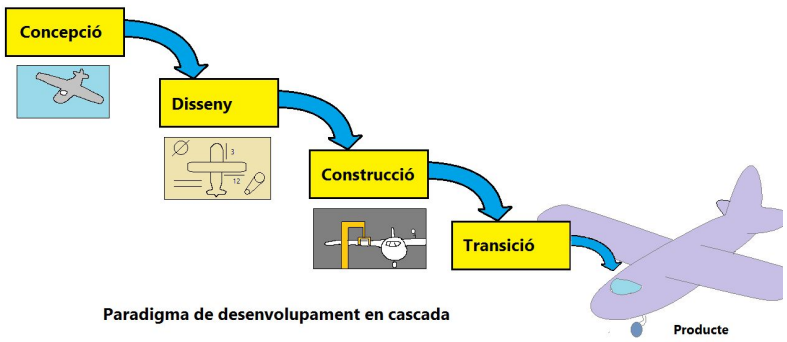
1.4.4 Paradigmes organitzatius

Al llarg del temps, s'han proposat diferents paradigmes organitzatius per al desenvolupament de projectes. Cadascun d'ells estableix quines etapes ha de tenir aquest desenvolupament i en quin ordre s'han de succeir. Els paradigmes més importants són dos, el desenvolupament en cascada i el desenvolupament iteratiu.

El **desenvolupament en cascada** estableix una successió d'etapes per les quals passa el projecte un sol cop i en ordre. L'assumpció que fa és que quan s'ha concebut el projecte, s'ha concebut completament i, quan s'ha produït, s'ha produït completament, i no admet la possibilitat de fer un pas enrere (vegeu la figura 1.27). És un paradigma que s'inspira en les cadenes de producció industrials, on el treball que cal fer en cada moment està perfectament definit i el producte viatja d'una etapa a l'altra **de forma lineal**.

És un model adequat per a projectes on els paràmetres que els defineixen són **fixos i coneguts** des del principi, de manera que és poc probable que les circumstàncies del projecte canviïn al llarg del seu desenvolupament. En projectes on no es donen aquestes condicions, però, implica una sèrie de problemes, ja que assumeix que les decisions que es prenen a una certa etapa **no s'hauran de reconsiderar mai**, i si les circumstàncies del projecte canvien en un moment concret, això pot ser necessari.

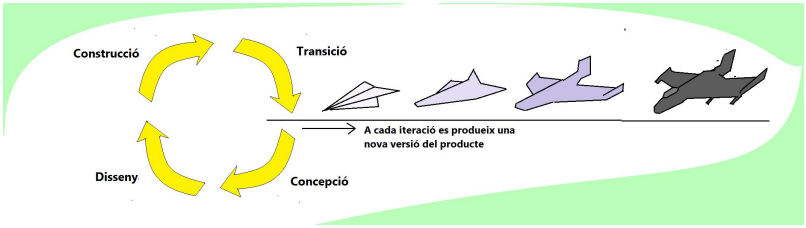
FIGURA 1.27. Desenvolupament en cascada



El **desenvolupament iteratiu** és un paradigma on les etapes es visiten **en ordre, però cíclicament**, de manera que l'equip pot fer ajustos cada vegada que les visita, incorporant l'experiència adquirida als cicles anteriors. Al final de cada cicle, es produeix una versió del producte, que tindrà una funcionalitat mínima als primers cicles i s'anirà ampliant fins a ser completa en els últims (vegeu la figura 1.28).

És un paradigma adequat quan les circumstàncies del projecte **varien amb facilitat** o no es poden determinar des del principi quines són les tasques que cal fer. Sota aquest paradigma, les primeres versions del producte s'anomenen prototips, cosa que fa que també es conegui com a **prototipatge**.

FIGURA 1.28. Desenvolupament iteratiu



2. Avaluació del projecte

Quan es prepara un projecte s'elaboren una sèrie de **requisits**, és a dir, característiques que ha de complir una aplicació interactiva i que s'acorden amb el client o l'equip. Si aquests requisits estan ben formulats, és a dir, es redacten en forma de proposicions ben formades i no de manera vaga o genèrica, una vegada arribats al final del projecte, aquests es podran verificar, és a dir, es podrà fer un test que determini si el requisit s'ha assolit o no.

Procés de desenvolupament

És una mena de procés en què incorporem i verifiquem característiques diàriament, tot construint una base sòlida que ens permet assolir eventualment els requisits generals.

A una escala diferent, quan treballem en un projecte, també **establim objectius** diaris o setmanals que inclouen també requisits, i d'igual manera que hem de poder verificar que els requisits generals del projecte s'han assolit quan aquest acabi, també hem de poder verificar que els objectius/requisits diaris o setmanals efectivament s'han portat a terme.

Característica o 'feature'

Una característica o *feature* és quelcom que té valor per a l'usuari i que aquest desencadena mitjançant algun tipus d'acció, per exemple poder obtenir un mapa que li indiqui la gasolinera més propera.

> En el cas del desenvolupament d'una aplicació interactiva, la regla més important que hem de mantenir és que l'**única verificació completament vàlida** és aquella que fem en el dispositiu final i en les mateixes circumstàncies que es trobarà l'usuari i que, si és possible, estigui feta per una persona diferent del desenvolupador. Qualsevol altra forma de verificació ens donarà un grau de seguretat més o menys gran, però mai complet.

La verificació es pot fer en l'àmbit individual, d'equip, o de forma sistemàtica i organitzada per professionals especialitzats en un procés anomenat **verificació o *testeig***, i la mesura de la qualitat d'una aplicació interactiva està directament determinada per la duresa i l'exigència del procés de verificació i validació a què ha estat sotmesa. Entendre això marca la diferència entre un equip de desenvolupament amateur i un de professional.

El fet d'haver de verificar les característiques o requisits en el dispositiu final, junt amb les molt diverses circumstàncies que ens podem trobar a una aplicació interactiva fa que el procés de verificació o *testeig* sigui costós tant en termes de temps com de recursos humans i materials.

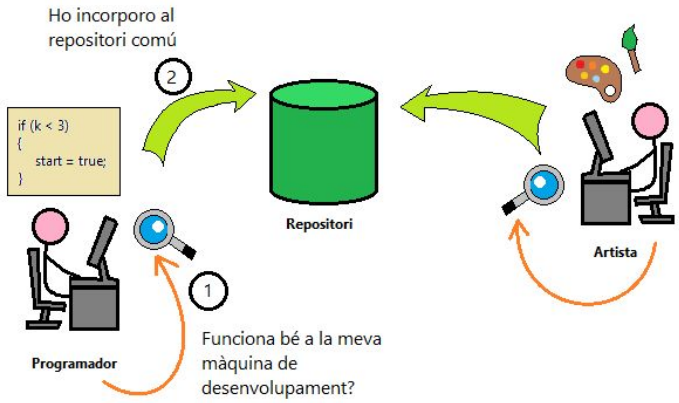
2.1 Proves internes

Al marge del procés de verificació, durant el desenvolupament d'un projecte es fan una sèrie de proves que, si bé no constitueixen un procés de verificació o *testeig* sistemàtic, sí que permeten que, en començar-lo, el projecte ja sigui relativament sòlid i funcional. Bàsicament, hi ha **quatre tipus** de proves: individual, sobre un component o sistema concret, d'equip i de projecte.

La **prova individual** és aquella que fem en l'àmbit personal, abans d'incorporar al projecte una nova característica en què hem estat treballant (vegeu la figura 2.1). En aquesta circumstància hem d'entendre que el fet d'haver completat les tasques necessàries per a implementar una característica **no implica** que aquesta funcioni realment, de manera que abans d'incorporar-la al projecte, per exemple incorporant-la al repositori, hem de fer una prova o test individual. Caldrà tenir en compte que:

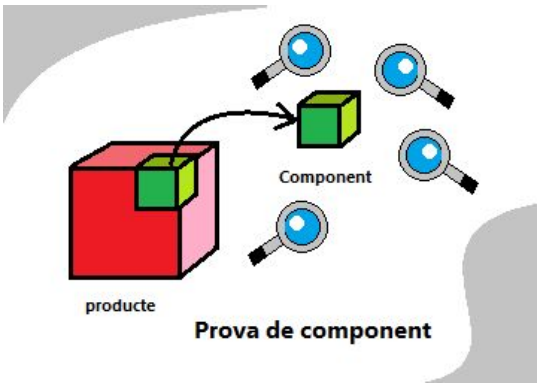
- Lògicament, aquest no pot ser un test complet, ja que durant el dia a dia del desenvolupament d'un projecte no disposem d'una quantitat suficient de temps, i possiblement no puguem tenir accés al dispositiu final, per exemple una *tablet*, de manera que hem de fer aquesta prova sobre un **nombre limitat de casos** i al dispositiu que tinguem més a mà, a l'ordinador amb què estem fent el desenvolupament o a un **simulador**.
- En el cas que no sigui una característica el que estem incorporant sinó un contingut, per exemple un gràfic o un clip d'àudio que en principi no hauria de generar cap error, cal igualment fer una prova d'integració, és a dir, incorporar el recurs als arxius de l'aplicació interactiva i verificar, idealment al dispositiu final, que es veu o sona correctament.

Figura 2.1. Prova individual



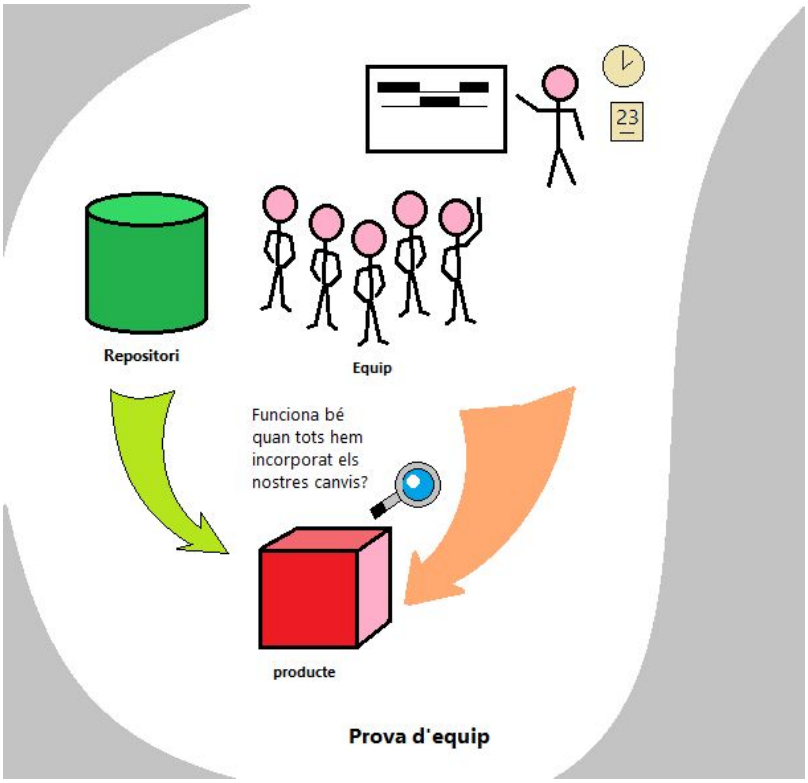
En el cas dels programadors, es poden fer **proves sobre un component o sistema concret**. Com a mínim, hauria de contenir una trucada a cadascun dels seus mètodes i, si és possible, algun tipus de prova automatitzada (vegeu la figura 2.2).

FIGURA 2.2. Prova de component



La **prova d'equip** és necessària perquè un equip de desenvolupament pot estar format per moltes persones que incorporen característiques i continguts contínuament, i es pot donar la circumstància que una característica que funcionava bé en la prova individual falli perquè una altra persona ha incorporat una característica que hi interfereix d'alguna manera (vegeu la figura 2.3).

FIGURA 2.3. Prova d'equip



- Aquest problema es pot prevenir organitzant el desenvolupament en àrees i sistemes amb límits ben definits, de manera que tinguin el mínim grau d'**interdependència** entre ells. Aquest repartiment de responsabilitats en l'àmbit d'equip correspon als caps de producció i en l'àrea tècnica, a un programador amb coneixements d'arquitectura de *software*.

- Per tal de detectar aquests possibles problemes, cal fer una **prova col·lectiva** de l'aplicació amb una certa periodicitat, per exemple setmanalment. D'a-

questa manera s'assegura que les característiques i continguts incorporats pels membres de l'equip no tan sols funcionen correctament individualment sinó també quan treballen en conjunt. Com que aquesta prova es fa amb una freqüència molt menor, moltes vegades es pot fer sobre el **dispositiu final** i en aquestes circumstàncies té un grau molt alt de confiança.

Finalment, tenim la **prova de projecte**, ja que cada cert temps, amb motiu d'una fita important dintre de la planificació del projecte, per petició d'un client, o bé perquè s'apropa la data anunciada de publicació, pot caldre presentar l'aplicació. En aquestes circumstàncies, serà necessari que l'aplicació estigui moderada o completament lliure d'errors, de manera que cal fer un esforç específic per detectar i solucionar els detalls que poden quedar pendents. Dit d'una altra manera, és un moment en què està justificat fer un procés de verificació.

En termes de disseny de *software*, interdependència fa referència al mínim d'acoblament possible.

Exemple de prova de càrrega sobre un component

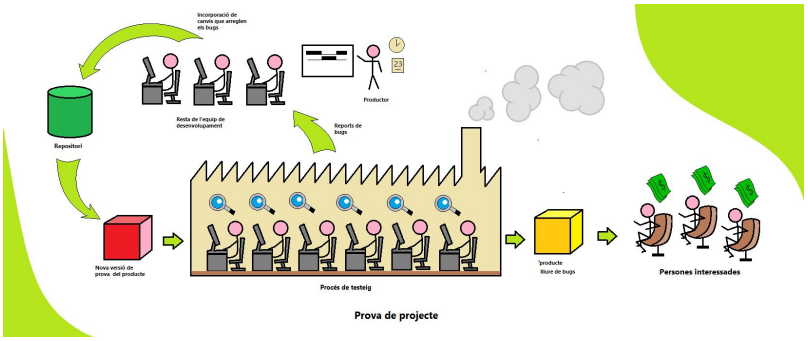
Donar una càrrega de treball molt alta a un sistema de l'aplicació o a un component, per exemple una aplicació que mostri un mapa, fent-li representar una quantitat molt alta d'icones per veure si així i tot el component continua funcionant. El raonament que hi ha darrere d'aquesta prova és que si el sistema o component suporta una càrrega de treball molt superior a la que haurà de suportar realment, podem tenir una confiança bastant gran que suportarà una càrrega de treball normal.

2.2 El procés de verificació (o 'testeig')

En desenvolupaments de projectes, és habitual utilitzar la terminologia anglesa *test*, *testing* i *tester*; en català hauríem de dir 'verificar, verificació i verificador', però potser serà més habitual trobar expressions com 'testejar', 'testeig' o 'tester'.

El **procés de verificació** (o *testeig*) és la comprovació i validació, sistemàtica i controlada, que les característiques i/o requisits d'una aplicació s'han assolit i funcionen correctament (vegeu la figura 2.4).

FIGURA 2.4. Prova de projecte amb procés de verificació o 'testeig'



La **dificultat més gran** la trobarem a l'hora de validar que l'aplicació funciona correctament, ja que en el cas d'una aplicació interactiva, per verificar que una característica funciona correctament hauríem de comprovar-la en totes les circumstàncies possibles, cosa que sovint no podem fer donada la gran quantitat d'estats en què pot trobar-se un sistema informàtic i la gran **diversitat d'accions** que pot fer l'usuari en un moment donat.

Per aquest motiu, parlarem del concepte de *comportament típic* i posarem com a **objectiu** no assegurar una experiència lliure d'errors en tots els casos, sinó assegurar que amb un grau de confiança molt alta qualsevol usuari que faci un ús raonable de l'aplicació no trobarà cap error.

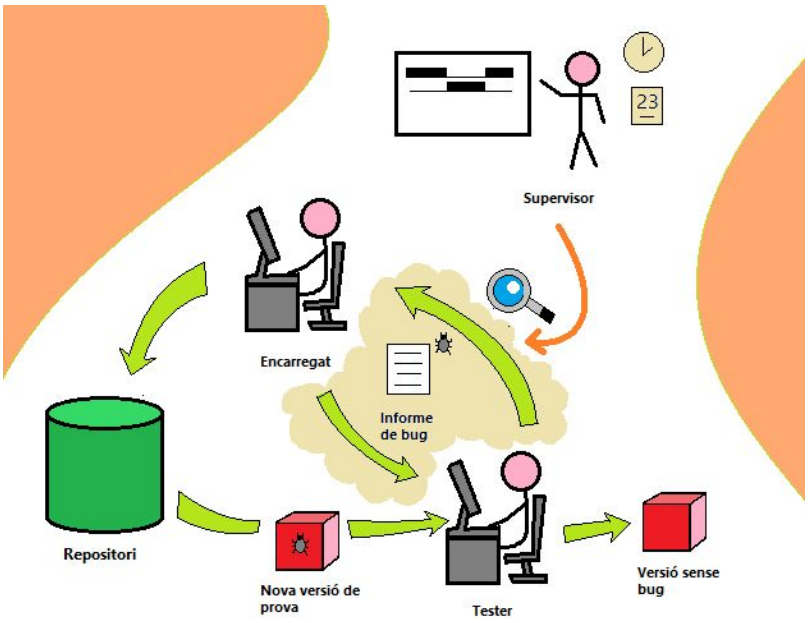
Com que és un procés col·lectiu que té una extensió en el temps i implica diferents persones, des del punt de vista de la producció caldrà **definir rols**, organitzar el treball en **àrees** i crear unes **dinàmiques de treball** que ens permetin assegurar que en finalitzar el procés haurem verificat totes les àrees i totes les característiques del projecte.

Per a aquest procés, podem ajudar-nos d'**eines específiques** per a la gestió del procés de verificació, en aquest cas bases de dades i llistes de *bugs* que ens ajudaran a mantenir organitzada la informació que anirem generant.

2.2.1 Els rols: el verificador, l'encarregat i el supervisor

En un procés de verificació es donen principalment tres rols diferents, el rol de **verificador (*tester*)**, que detecta els *bugs*, el rol d'**encarregat**, és a dir, la persona que s'encarrega de solucionar el *bug* i el rol de **supervisor**, que valida que el procés es desenvolupa correctament i pot prendre decisions en casos especials que excedeixen les responsabilitats del *tester* i l'encarregat (vegeu la figura 2.5).

Figura 2.5. Rols al procés de verificació



El 'tester' o verificador

El *tester* o 'verificador' és la persona encarregada de **detectar *bugs***, és a dir, funcionaments erronis a l'aplicació interactiva, bé perquè no es compleix algun

El 'bug'

El terme *bug* fa referència a que un dels primers errors detectats a un ordinador va ser causat per la presència d'un tipus d'insecte (en anglès, *bug*) entre els circuits de l'ordinador Mark II, el 1946.

requisit, bé perquè alguna característica o recurs no funciona o no es mostra correctament.

El fet que el *tester* hagi de verificar que els requisits i característiques es compleixen impliquen que **ha de conèixer** quins són aquests requisits. Per exemple, en el cas d'una aplicació que mostri diferents àrees d'un parc, el *tester* ha de conèixer quantes àrees haurien de visualitzar-se per poder detectar que en falta una.

D'altra banda, com que el concepte de "funcionar bé" o "veure's bé" té un component **subjectiu**, el *tester* ha de tenir algun tipus de guia que li indiqui quins comportaments de l'aplicació són acceptables i quins no.

El verificador no hauria de verificar mai àrees i característiques del projecte a l'atzar, sinó concentrar-se en una àrea concreta o verificar una característica en particular cada vegada. Això implica la presència d'un pla de verificació que especifiqui en cada moment **quin *tester* té assignada** una àrea o una característica; garantint així que l'esforç de verificació s'aplica amb la intensitat apropiada a cada àrea o característica, i no en queda cap sense provar.

Temps de càrrega

Cada *tester* pot tenir una impressió diferent de si un procés de càrrega de recursos és o no massa llarg, però si definim a un document que el temps de càrrega no hauria de ser mai superior a, per exemple, deu segons, estem donant una mesura *objectiva* de quan un procés de càrrega és massa llarg.

Tant la comunicació dels requisits de l'aplicació o joc als *testers*, com proveir-los de les **guies d'estil** que han de seguir (a vegades, són les mateixes empreses propietàries d'una plataforma les que les elaboren), així com l'organització del pla de verificació, són responsabilitat del supervisor.

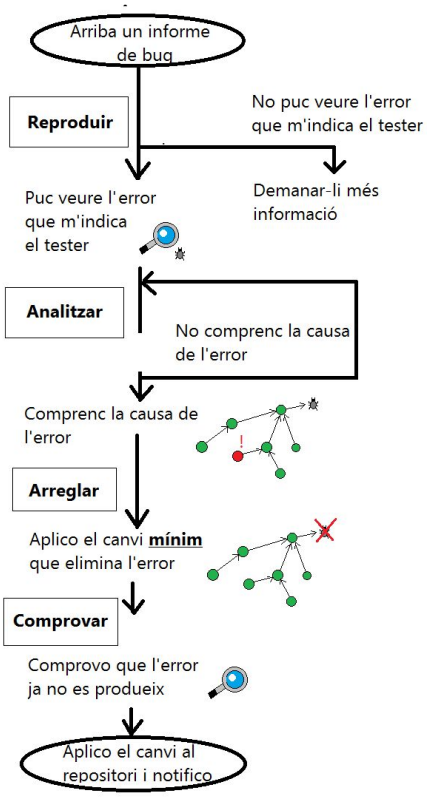
L'encarregat

L'encarregat és la persona encarregada d'**arreglar un *bug***. En principi, ha de ser un membre de l'equip de desenvolupament amb els coneixements necessaris per arreglar-lo, tot i que pot succeir que un *bug*, al qual inicialment se li assigna a un membre de l'equip, acabi **reassignat** a un altre, ja sigui perquè aquesta persona considera que l'altre membre té uns coneixements més adequats, o bé perquè el *bug* s'ha **categoritzat** incorrectament, és a dir, s'ha estimat que era responsabilitat d'un departament o rol dins de l'equip de desenvolupament al qual no pertany.

Les responsabilitats de l'encarregat són quatre, en aquest ordre (vegeu la figura 2.6):

1. **Reproduir** el *bug*, és a dir, ser capaç pels seus propis mitjans de provocar la situació d'error de la qual l'està informant el *tester*.
2. **Analitzar** quines són les **causes** de l'error, és a dir, entendre perquè està succeint.
3. Aplicar el **canvi mínim** que faci que l'error no es doni.
4. **Comprovar** que l'error ja no es produeix.

FIGURA 2.6. Resolució d'un bug



Una vegada aplicat el canvi, encara que pot tenir una certa seguretat en haver arreglat el *bug*, no és responsabilitat seva **verificar-lo**, ja que això sempre correspon a la persona que ho ha detectat; en aquest cas, el *tester*.

Una altra raó per la qual els canvis que fa l'encarregat sempre han de ser mínims, és perquè no provoquin més *bugs*; és a dir, no s'ha d'aprofitar que s'arregla un *bug* per introduir noves característiques a l'aplicació interactiva.

El supervisor

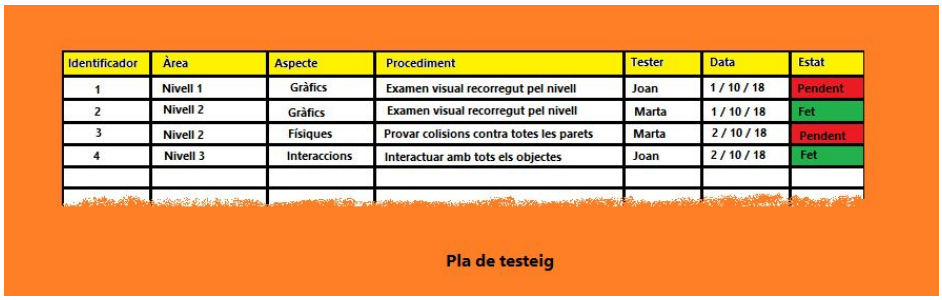
El supervisor és una persona **amb un cert rang** dins el projecte que supervisa l'activitat dels verificadors i els encarregats, i pot prendre accions en aquelles situacions que sobrepassin l'àmbit de responsabilitat d'aquests. Si es dona una d'aquestes situacions, ha de consultar els administradors del projecte i l'equip de desenvolupament per determinar l'acció que cal fer. Fora d'aquestes situacions, només ha d'observar.

Una responsabilitat del supervisor és conèixer bé els **requeriments** i **estàndards de qualitat** que ha de complir el projecte per tal de comunicar-los als *testers*, i també l'extensió del projecte i les àrees que comprèn, per tal d'elaborar un **pla de verificació** que els abasti completament.

2.2.2 Elaboració d'un pla de verificació

Una vegada arribats a un punt suficientment avançat del projecte, on ja tenim una certa seguretat que tant les característiques de l'aplicació interactiva com els continguts i nombre de pantalles que inclou no patiran grans **alteracions**, podem començar amb un procés de verificació sistemàtica que parteixi d'un **pla de verificació** (vegeu la figura 2.7). Abans d'aquest punt, l'equip de desenvolupament ha d'haver fet les corresponents proves internes, i també pot continuar amb elles en paral·lel mentre es duu a terme el pla.

Figura 2.7. Pla de verificació



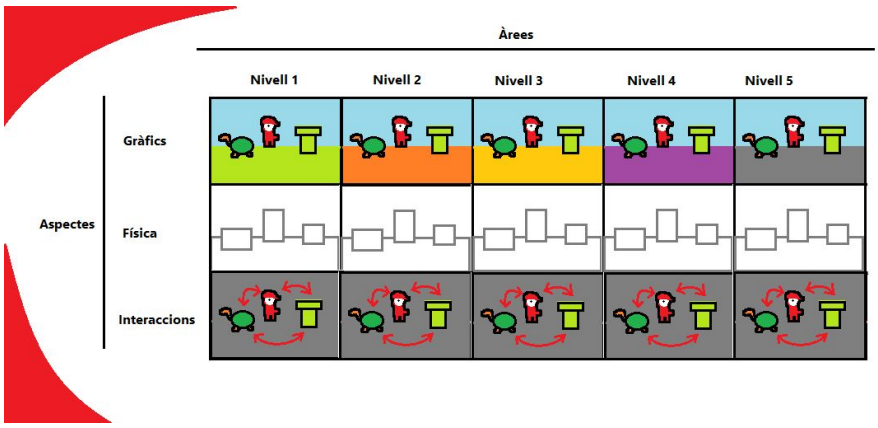
Identificador	Àrea	Aspecte	Procediment	Tester	Data	Estat
1	Nivell 1	Gràfics	Examen visual recorregut pel nivell	Joan	1 / 10 / 18	Pendent
2	Nivell 2	Gràfics	Examen visual recorregut pel nivell	Marta	1 / 10 / 18	Fet
3	Nivell 2	Físiques	Provar colisions contra totes les parets	Marta	2 / 10 / 18	Pendent
4	Nivell 3	Interaccions	Interactuar amb tots els objectes	Joan	2 / 10 / 18	Fet

Pla de testeig

Trobareu una explicació sobre els tipus de requisits que es poden definir a un projecte al punt "El nivell de projecte", de l'apartat "Planificació i realització del projectes" d'aquesta unitat.

Per fer aquest pla, haurem de fer una llista de les **característiques**, **àrees** i **aspectes** que volem verificar (vegeu la figura 2.8), així com establir els **requisits de qualitat** que han de complir. Aquests requisits han de ser formals i objectius, per tal d'eliminar qualsevol tipus d'ambigüitat a l'hora de definir si un comportament detectat és erroni o no. Finalment, haurem d'establir unes **dates** i assignar-les a diferents **grups de verificadors** per tal que puguin començar a treballar.

Figura 2.8. Aspectes i àrees



Àrees, característiques i aspectes

En el curs de la seva activitat, l'usuari d'una aplicació interactiva passa per una sèrie d'**àrees**, és a dir **zones diferenciades** on pot romandre un cert **temps** i fer una sèrie d'**accions** (vegeu la figura 2.8). Una forma d'organitzar el pla de verificació és fer una llista d'aquestes àrees i assignar-les a verificadors diferents per tal que cap zona de l'aplicació interactiva quedi sense ser verificada.

Dintre d'una àrea l'usuari pot fer tota una sèrie d'**accions**, i és responsabilitat del *tester* provar-les **totes**, i també totes les **combinacions** d'accions, per exemple obrir el menú d'opcions abans i després d'examinar un objecte.

D'altra banda, les **característiques** són requisits que ha de complir l'aplicació, i és responsabilitat de la verificació comprovar que aquestes característiques no només **estan presents** sinó que funcionen en **qualsevol circumstància**.

Característiques d'una aplicació per a un museu

En el cas d'una aplicació que presenti els objectes d'un museu, aquestes podrien ser:

- Escoltar una audiodescripció de cada objecte.
- Poder canviar d'idioma en qualsevol moment.
- Poder girar l'objecte.

Observem com la dificultat és assegurar que aquestes característiques funcionen **en qualsevol circumstància**. Per exemple, si tenim una característica de "canviar idioma en qualsevol moment", això implicarà **haver de comprovar-la** a totes les pantalles, en tots els menús que pot fer servir l'usuari; i en **combinació** amb la resta d'accions que pot fer. Canviar d'idioma abans d'examinar un objecte, després de sortir del menú d'opcions i així successivament.

Quan parlem d'**aspectes** volem dir elements que són **transversals** al projecte, és a dir, que sempre hi són presents però que normalment l'usuari no observa de forma separada, cosa que sí que s'ha de fer durant el procés de verificació per assegurar que compleixen els requisits de qualitat. Podem pensar-hi com a diferents sistemes que funcionen a la vegada i resulten en l'aplicació interactiva. Exemples d'aquests aspectes poden ser:

- Gràfics
- Música, efectes de so i veu
- Interacció
- Textos/idiomes

Quan es verifica un aspecte concret de l'aplicació interactiva, el *tester* ha de fer com si la resta d'aspectes **no existissin**, per exemple, si està comprovant els efectes de so, tota la seva atenció ha d'estar enfocada a comprovar si s'inicien en el moment precís i amb un volum adequat, i a provar diferents combinacions per tal de verificar que tampoc no interfereixen entre si.

Exemple de combinacions d'elements

Els aspectes o característiques i les àrees es poden **combinar**, de manera que si hem de verificar, per exemple, l'aspecte gràfic, podem dividir aquesta tasca per **àrees**, i fer tests separats per exemple per cada pantalla de l'aplicació. Un altre avantatge de fer-ho així és que així els *bugs* que es trobin seran molt més fàcils de **localitzar** per l'equip de desenvolupament, ja que inclouran l'àrea on es produeixen.

2.2.3 Estratègies de verificació

Idealment, voldríem que després d'un procés de verificació una aplicació interactiva estigués **lliure d'errors**, però això en general és impossible, no només per la gran quantitat d'accions que pot fer l'usuari, sinó per la gran quantitat de **combinacions** d'accions i circumstàncies diferents en què poden fer aquestes accions.

Si fem els càlculs, veurem que cadascun d'aquests elements **multiplica** els casos que cal comprovar, de manera que molt ràpidament la xifra de possibilitats sobrepassa els recursos materials i humans de qualsevol equip.

Això fa que sigui necessari establir **estratègies de verificació** que ens permetin assegurar un funcionament correcte en un ampli ventall de casos amb uns recursos humans i materials **limitats**; les estratègies de verificació comprenen els següents **tipus de test**: exhaustiu, repetitiu o estocàstic, de límits, de saturació i d'accions imprevistes.

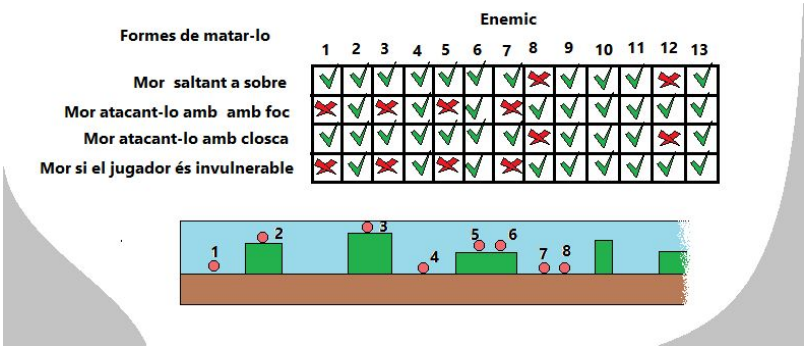
De fet, des de fa temps sabem que no és possible decidir de forma automàtica aspectes aparentment senzills, com per exemple saber si un programa informàtic, una vegada iniciat, s'aturarà en algun moment.

Test exhaustiu

Si el nombre de casos que hem de verificar és suficientment petit, podem plantejar un test que els **inclogui tots**. Sempre que puguem és la forma preferible de fer-ho.

Per plantejar un test exhaustiu hem de fer una llista d'elements com ara accions, circumstàncies i àrees que volem que inclogui, i fer l'**expansió** de totes les **combinacions** possibles d'aquests elements en forma de taula, incloent-hi un espai per indicar si el test l'hem fet o no (vegeu la figura 2.9).

FIGURA 2.9. Test exhaustiu

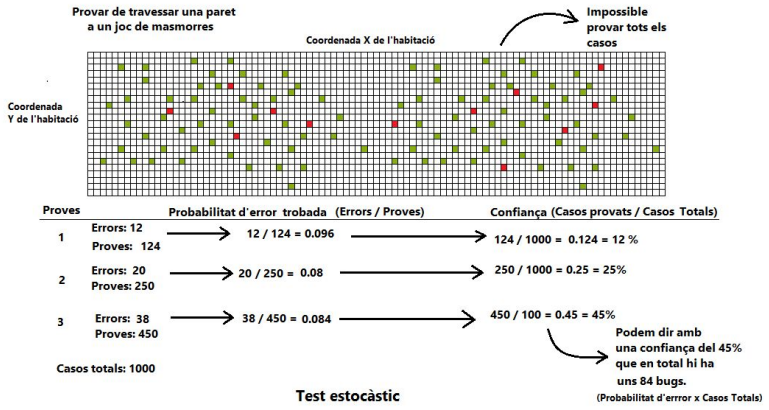


Test repetitiu o estocàstic

Com hem dit moltes vegades, és impossible provar tots els casos possibles en què es pot trobar una aplicació interactiva, però podem obtenir una **confiança estadística** en què hem detectat una bona proporció d'errors si verifiquem **repetidament** una mateixa característica o aspecte, sempre que fem cada prova tan aleatòriament com sigui possible, és a dir, fent variacions **a l'atzar**.

Si procedim així, en cada test on no trobem cap error, la probabilitat que quedi algun error per detectar **disminueix**, ja que, assumint un comportament estadísticament normal del sistema, cada vegada és més improbable que si queda algun error aquest no s'hagi produït i, tot i que no podem assegurar que no existeixi, sí que podem dir amb una certa confiança que l'aplicació bé està lliure d'errors o, si en té, és molt improbable que es produeixin (vegeu la figura 2.10).

FIGURA 2.10. Test repetitiu o estocàstic

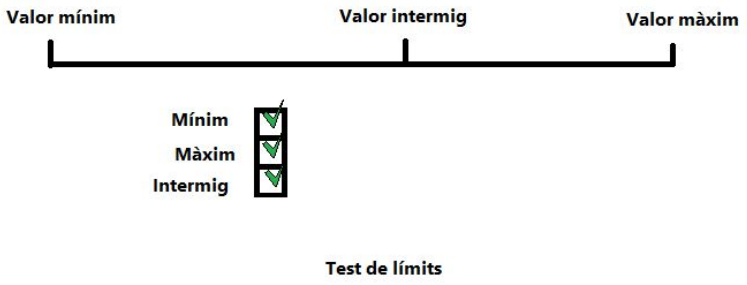


Un problema que trobem aquí és que les persones no som bones per generar comportaments aleatoris i quasi sempre seguim **algun patró** més o menys conscientment. Això fa que perquè un test aleatori generi una bona confiança estadística, s'hagi de fer **automàticament** o bé el facin persones diferents per tal d'evitar que les proves sempre segueixin el mateix patró.

Test de límits

Aquest test és molt adequat quan tenim una magnitud **contínua**, és a dir, una propietat que pot tenir molts valors reals diferents o bé una magnitud discreta, és a dir, que es pot representar com un nombre sencer, però amb un **rang molt ampli**, o sigui, que el valor mínim i el màxim que pot tenir estan tan separats que no es pot plantejar un test cas per cas. Consisteix a fer una prova amb el valor **mínim**, una altra amb el valor **màxim** i un altre amb un valor **mitjà** o típic (vegeu la figura 2.11).

FIGURA 2.11. Test de límits



L'assumpció que estem fent en aquest cas és que si l'aplicació interactiva **funciona bé** amb el valor mitjà i amb els valors mínims i màxims, també funcionarà correctament amb els valors **intermedis**, cosa que en general podem assumir que serà així. Dit d'una altra manera, normalment els errors als programes es produeixen quan alguna propietat pren els valors **extrems**, tot i que cal també comprovar el valor central, encara que sigui menys probable.

Exemple de test de límits: provar el volum de so de l'aplicació

Un exemple de test de límits seria provar el volum de so de l'aplicació. Si aquest pot anar de 0 a 100, normalment podem assumir que si provem amb volum 0, amb volum 100 i amb un valor intermedi i el so funciona correctament, no cal verificar els cent nivells de volum possibles.

Test de saturació

Aquest test consisteix a intentar **saturar** alguns dels sistemes o característiques del joc a força d'acumular instàncies d'un cert tipus d'element i comprovar si l'aplicació continua funcionant correctament.

En cert sentit el test de saturació és un test de límits, però quan aquests no són explícits sinó que corresponen a propietats de l'aplicació que no són accessibles **directament**, de manera que el *tester* ha de trobar una forma indirecta de portar-les al màxim.

Exemple de test de saturació

Per exemple, en un sistema de finestres que permeti obrir múltiples documents simultàniament, un *tester* podria començar a obrir tants documents com pugui, cercant el límit de capacitat del sistema.

Test d'accions imprevistes

Aquesta tècnica de verificació podríem dir que és fer **el contrari** del que faria un usuari normal de l'aplicació, i és molt característica d'un *tester* experimentat. Consisteix a fer accions que tot i que són possibles, en un ús normal de l'aplicació són molt **improbables**, per exemple obrir un menú i tancar-lo immediatament, o obrir el menú d'opcions just quan s'acaba de canviar l'idioma de l'aplicació.

Amb aquest tipus de tests s'està comprovant que els programadors gestionen correctament els esdeveniments i els canvis d'estat de l'aplicació i que no confien que l'usuari els donarà un marge de temps entre una acció i una altra, i que a més han tingut en compte tots els casos possibles.

2.2.4 Els 'bugs'

Com a resultat de l'activitat dels *testers*, es detectaran *bugs*, que són errors de l'aplicació o situacions en què el seu comportament no sigui l'esperat. Els *bugs*

també poden provenir del fet que algun element de l'aplicació no compleixi amb els **requisits de qualitat** que s'hagin establert i en general de qualsevol aspecte que sobti el *tester* i que cregui necessari comunicar a l'equip de desenvolupament.

Cada vegada que es detecta un *bug* s'ha de fer un **informe** que especifiqui molt clarament **on** i **en quines circumstàncies** es produeix, **sense** suggerir cap tipus de solució, ja que això correspon a la persona que ho ha d'arreglar (vegeu la figura 2.12). Aquest informe pot incloure materials que puguin ajudar a caracteritzar-lo, per exemple captures de pantalles o vídeos i, en el cas de les aplicacions multidispositiu, quin és el model de dispositiu en què s'està fent la prova. Un altre element important és la versió de l'aplicació, per tal d'evitar situacions on un *bug* sembla que torna a produir-se però perquè el *tester* està fent servir una versió antiga de l'aplicació.

FIGURA 2.12. Llista de 'bugs'

Llistat de bugs

Identificador	Títol	Descripció	Reproductibilitat	Àrea	Categoria	Severitat	Prioritat	Tester	Encarregat	Estat
1	[illegible]	[illegible]	Mitja	Nivell 1	Gràfics	Major	Normal	Pep	Albert	Obert
2	[illegible]	[illegible]	Difícil	Nivell 1	Programació	Menor	Alta	Marta	Maria	Tancat
3	[illegible]	[illegible]	Mitja	Nivell 3	Programació	Menor	Normal	Marta	Maria	No un bug
4	[illegible]	[illegible]	Difícil	Nivell 4	So	Major	Alta	Joan	Albert	Reobert
5	[illegible]	[illegible]	Fàcil	Nivell 1	Gràfics	[illegible]	Normal	Laura	Maria	Tancat

Una vegada omplert d'informe d'un error, aquest es classifica i es prioritza segons la seva gravetat per tal que el pugui atendre l'equip de desenvolupament i s'integra a una **llista o base de dades de *bugs***, compartida entre l'equip de desenvolupament i el de verificació.

> **La llista o base de *bugs*** conté els reports de tots els que eventualment s'hagin detectat, classificats per àrees i categories, avaluats i prioritzats, per tal de poder atendre'ls en funció de la seva severitat.

S'engega en aquest moment un procés en què el *bug* pot passar per diversos estats i ser assignat a diferents persones i departaments fins que se soluciona, en el que es pot anomenar **cicle de vida** del *bug*. Aquest cicle de vida es pot veure com una mena de dinàmica en què el *bug* passa de l'equip de desenvolupament al de verificació, fins que o bé se soluciona o bé es pren alguna decisió per part dels administradors que el dona per tancat.

En funció de l'escala del projecte, aquesta llista pot incloure's en un simple full de càlcul o base de dades o bé podem fer servir un **sistema específic per gestió de *bugs*** d'entre els que existeixen al mercat. El nombre de *bugs* que pugui incloure aquesta llista depèn de la duresa dels tests a què s'hagi sotmès l'aplicació, però per exemple en el cas d'un projecte petit, de tres a cinc persones i uns pocs mesos de durada, típicament pot arribar a uns pocs milers.

Dins de la llista de *bugs*, tindrem **diferents propietats** que recullen tota la informació associada a cada *bug*. Aquestes propietats poden variar segons el

sistema que fem servir, però finalment proveeixen una informació semblant. Les més importants són: identificador, títol, descripció, àrea, categoria, reproductibilitat, passos per reproduir-lo, severitat, prioritat, estat, arxius adjunts, comentaris, versió de l'aplicació i dispositiu.

Identificador

L'identificador és un **codi únic** per a cada *bug*. Pot ser un número o una combinació de números i lletres però sempre ha d'identificar un únic *bug*. Una altra raó que justifica la necessitat d'identificar-los és que un mateix *bug* pot ser detectat diverses vegades, i és important poder distingir quin és l'original i quins els duplicats.

Importància de la identificació

Insistim en aquest punt perquè els *bugs* poden ser molt nombrosos i és molt fàcil que es perdi la referència si els identifiquem per la seva descripció, per exemple "problema amb el menú d'opcions".

Títol

El títol és una frase curta que informa de l'**efecte** que s'està observant.

És molt important que sempre que parlem d'un *bug* en el context d'un procés de verificació **ens hi referim per l'efecte observat**; és a dir, els *bugs* sempre s'han de descriure des del punt de vista del que percep un usuari, no s'han de formular mai com una tasca, és a dir, com alguna cosa que cal fer, perquè fins que no es revisa el *bug* no se sap quina és la solució.

Exemple de com posar un títol correcte a un 'bug'

Per exemple, si detectem que a una aplicació interactiva per a un museu, de vegades volem girar un objecte arrossegant-lo amb el dit i no podem, encara que tinguem un cert coneixement del sistema d'entrada i sospitem que el problema pot ser que no està funcionant bé el codi que processa l'entrada tàctil, hem d'anotar "No es pot girar l'objecte" i no que "Cal revisar el codi que processa l'entrada tàctil", ja que no tenim la seguretat que aquesta sigui la causa i, pitjor encara, si fem un canvi en aquest codi i cometem algun error, potser estarem introduint sense voler un nou *bug* i, a més, possiblement no solucionarem el *bug* anterior.

Descripció

La **descripció** ha de seguir sempre un mateix patró, on podem distingir diversos elements. Com el títol, no ha de contenir mai les nostres hipòtesis sobre la possible solució. Si tenim experiència en reportar *bugs*, podem fer servir només un camp de descripció a la base de *bugs* i incloure aquests elements com un petit text; si no, podem separar aquests elements en camps diferents per tal d'assegurar que no quedaran mai buits. Aquests elements són:

- La circumstància, és a dir, en quin context (nivell o situació) es produeix el *bug*.
- Les accions o passes que cal fer per reproduir el *bug*.
- L'efecte observat.

Exemple de descripció d'un 'bug'

Un exemple de descripció d'un *bug*, amb tots els elements integrats, podria ser:

"En el menú principal, quan arrenquem l'aplicació per primer cop, si entrem al menú d'opcions i tornem a sortir al menú principal tres vegades i seleccionem l'opció de veure un objecte, aquest no es mostra".

Les descripcions que fem dels diferents elements, han de ser **clares** i **explícites**; és a dir, hem d'evitar que la persona que ho llegeixi hagi de fer **suposicions** o interpretar el que escrivim, fins i tot si aquesta persona som nosaltres mateixos.

Àrea

La persona que arregla un *bug* normalment n'haurà d'arreglar molts més, així que hem de procurar que perdi la menor quantitat de temps possible en **localitzar** on s'està donant el problema. Per aquesta raó, hem d'incloure un camp **àrea** que permeti identificar ràpidament on s'està produint.

Una altra raó per introduir aquest camp és que d'aquesta manera la persona que arregla el *bug* pot aprofitar que està treballant en una àrea concreta per solucionar la resta de *bugs* que hi hagi en aquesta àrea, i així estalviar-se la feina que li pot suposar **canviar de context**.

D'altra banda, si dins d'un equip de desenvolupament cada persona ha treballat en àrees diferents, per exemple una persona s'ha encarregat del menú principal i una altra de la pantalla de presentació, classificar els *bugs* per àrees és una forma també de tenir una **guia de com assignar-los**.

Categoria

La categoria té a veure amb el sistema o departament al qual pertanyi el *bug*, per exemple "Programació" o "Gràfics", i ens dona una primera indicació de com assignar-lo.

Com que inicialment només sabem què està passant però no quina és la solució, la classificació que posi un *tester* en aquest camp només serà una **suposició**, i serà la persona **encarregada** qui determinarà si el *bug* pertany realment a aquesta categoria i si s'hauria de canviar o reassignar.

També pot passar que un *bug* sigui complex i requereixi l'acció de diversos departaments o rols, fet que també provocarà que aquest es recategoritzi i es reassigni diverses vegades.

Exemple de 'bug' reclassificat

Per exemple, podem tenir un *bug* en què l'efecte observat sigui que una imatge no es mostri en una galeria. El *tester* pot classificar inicialment el *bug* amb categoria "Gràfics", però quan el grafista l'examini pot determinar que el gràfic està present, de manera que la responsabilitat és del codi que gestiona la galeria i el *bug* s'ha de reclassificar amb la categoria "Programació".

Reproductibilitat

La reproductibilitat és la facilitat que hi ha en reproduir un *bug*, és a dir, amb quina facilitat es pot provocar l'efecte negatiu observat.

Donada la naturalesa atzarosa d'alguns *bugs*, pot passar perfectament que, tot i situar-nos al punt concret on hauria de produir-se, i seguint escrupolosament les accions o passes que indica el *tester*, siguin necessàries **moltes repeticions** per tal d'observar l'efecte.

Els valors que pot prendre la reproductibilitat d'un *bug* són, doncs, una escala de dificultat, per exemple "fàcil", "normal" i "difícil". Típicament als *bugs* amb reproductibilitat més alta se'ls posa el sobrenom de *Hard to repro*.

Passos per reproduir-lo

Molt lligada a la reproductibilitat d'un *bug*, trobem els passos per reproduir-lo. Aquests són una **llista ordenada** d'accions que ha de fer l'usuari, en aquest cas l'encarregat, per poder observar l'efecte.

En cas que la reproductibilitat sigui molt fàcil, aquestes passes seran molt senzilles, i pràcticament només serà necessari fer l'acció indicada al lloc indicat o repetir-la pocs cops. En canvi, si un *bug* té una reproductibilitat més alta o és *Hard to repro*, normalment el *tester* haurà de treballar per **elaborar** aquesta llista de passes, ja que normalment és indicatiu que el procés que desencadena el *bug* és més complex del que inicialment s'havia considerat.

Exemple de llista de passes

Una llista de passes elaborada podria tenir aquest aspecte:

1. Anar al menú de selecció d'objecte
2. Girar el model del cotxe durant dos segons.
3. Tornar al menú principal.
4. Abans que el menú es mostri completament, prémer la tecla 'ESC'
5. L'aplicació es bloqueja.

Tot i que, inicialment, les passes podrien ser:

1. Anar al menú de selecció d'objecte.
2. Prémer la tecla 'ESC'.
3. L'aplicació es bloqueja.

Severitat

La severitat és una valoració de la **importància** d'un *bug*, és a dir, de si és més o menys greu des del punt de vista de l'experiència de l'usuari. Aquesta

és una mesura subjectiva i correspon jutjar-la, en última instància, al supervisor o les persones que administrin l'empresa, però podem establir alguns **criteris objectius**.

Primer de tot, hi ha dos tipus de *bugs* que són els més greus de tots, els *freezes* i els *crashes*. Un *freeze* és una situació d'error que es dona quan l'aplicació interactiva es **congela**, de manera que a partir d'aquell moment **no respon** a cap acció de l'usuari, mentre que un *crash* és una finalització abrupta de l'aplicació, en què es tanca i retorna el control al sistema operatiu, normalment per un error de sistema.

Tant els *crashes* com els *freezes* són exemples de *bugs* **bloquejadors**, és a dir, que no deixen que l'usuari continuï fent servir l'aplicació interactiva, però no els únics, ja que un usuari pot anar a una pantalla i, per exemple, no tenir un botó per tornar a la pantalla anterior.

D'altra banda, tenim el concepte de **ruta típica**, és a dir, les accions o el recorregut que fa normalment un usuari de l'aplicació. En el cas d'una aplicació amb finestres, aquesta podria ser obrir un document, editar-lo, salvar els continguts i tancar l'aplicació, i en el cas d'una aplicació interactiva podria ser seleccionar un objecte, examinar-lo fent algunes rotacions, i escoltar l'audiodescripció.

Aquest camí pot estar dissenyat però també podem deixar llibertat als usuaris i observar el seu comportament, per exemple fent servir eines de *tracking* que registrin quines àrees visiten i quines accions fan.

En qualsevol cas, la ruta típica representa l'experiència de la **majoria** dels usuaris i per tant si un *bug* succeeix en algun punt d'aquesta ruta, serà **molt més greu** que si succeeix en una ruta secundària, és a dir un itinerari que per a l'aplicació és **opcional**.

El pitjor *bug* que podem tenir, és un de bloquejador a la ruta típica.

Deixant de una banda els *bugs* bloquejadors, la resta de *bugs* podríem dir que no impedeixen que l'usuari faci servir l'aplicació interactiva però sí que poden fer que la seva experiència d'ús sigui deficient, de manera que també hauríem d'adreçar-lo, tot i que això ja és una decisió particular de cada empresa.

Per a aquests *bugs*, podem fer servir un criteri de **visibilitat**, és a dir, un *bug* serà més greu com més noticiable sigui per a l'usuari. Per exemple en una aplicació narrativa serà més greu un *bug* que faci que la roba d'un personatge que ocupa una gran proporció de la pantalla canviï de color entre una vinyeta i una altra que ho faci petit arbre situat en el fons.

Un altre factor que podem considerar és la **probabilitat** que un usuari pateixi el *bug*, és a dir, com més improbable o difícil de reproduir sigui un *bug*, o més allunyat estigui de la ruta típica, podem considerar que la seva severitat també és menor. Així i tot, com que aquest valor ja el recollim a la reproductibilitat, és millor jutjar la severitat només per l'efecte per a l'usuari.

Una vegada definit el criteri amb què mesurarem la severitat hem de constituir una **escala** on en el punt més baix tindrem els *bugs* amb severitat "baixa" o *minor*, els

bugs amb severitat "alta" o *major* i sovint també un valor especial per als *bugs* "bloquejadors" o *block*.

Prioritat

Els *bugs* no deixen de ser petites tasques pendents que també es poden **prioritzar**, és a dir, donar una indicació a l'equip de desenvolupament de quins són els que cal fixar com més aviat millor i els que poden esperar, encara que també puguin ser severs. Donar valor a aquest camp correspondrà normalment al supervisor i per valorar-lo haurà de tenir en compte aspectes com la reproductibilitat, la severitat del *bug* i si aquest està situat a la ruta típica o allunyada d'ella.

Els valors que podem donar a aquesta prioritat han de reflectir una escala creixent d'urgència, per exemple, "baixa", "mitjana" i "alta". Sovint també és útil tenir un valor de prioritat "màxima" o "crítica" per indicar que un *bug* s'ha de solucionar abans que tots els altres.

Estat

Trobareu exposats amb detall els possibles estats d'un *bug* al punt "Estats d'un *bug*", d'aquest mateix apartat.

L'estat ens dona una indicació de la **situació** en què es troba un *bug*. Si un *bug* fos una tasca normal, aquest camp podria tenir valors com "pendent", "en progrés" i "completat", però els *bugs* tenen una dinàmica pròpia que fa que aquests estats siguin una mica més nombrosos.

En termes generals, però, podem dir que els estats d'un *bug* són diversos però es poden agrupar en "obert", mentre el *bug* no s'hagi solucionat, i "tancat", una vegada s'hagi solucionat.

Arxius adjunts

Els arxius adjunts són tots els **documents** que acompanyen l'informe del *bug* i que poden ajudar a localitzar-lo i reproduir-lo. Tenen força importància, especialment les captures de pantalla i els vídeos en el cas d'una aplicació interactiva, perquè permeten que l'encarregat observi **exactament** quines passes s'han donat sense cap tipus d'ambigüitat. És bona idea, doncs, gravar les sessions de verificació per poder produir després aquests clips.

Un altre detall important és incloure a la captura de pantalla o al vídeo el **número de versió** de l'aplicació, per tal que la persona encarregada sàpiga a quina versió es produïa.

Comentaris

Un *bug* pot tenir un camp de comentaris on es registrin els missatges que s'envien el *tester* i les persones encarregades que l'estiguin arreglant. Totes les **conjectures** que fem sobre l'origen del *bug* i que no s'han d'incloure mai a la descripció, sí que es poden incloure en aquest camp, ja que el que posem aquí s'interpreta que

és una observació **personal** i no un fet comprovat. El comentari es deixa perquè la persona encarregada sàpiga a quina versió es produïa.

Versió de l'aplicació

És molt important incloure també, entre les propietats d'un *bug*, quin és el **número de versió** de l'aplicació en què s'ha detectat.

> **Exemple de 'bug' que no es pot reproduir**
>
> Pot passar que diferents *bugs* siguin causats per un problema comú, i un programador pot estar intentant arreglar un *bug* a la versió 2, però no pot reproduir-lo perquè el problema que el causava es va arreglar amb motiu d'un altre *bug* a la versió 1. En canvi, si veu que el *bug* que està intentant arreglar pertany a la versió 1, pot donar-lo per arreglat perquè sap que el problema ja no està present.

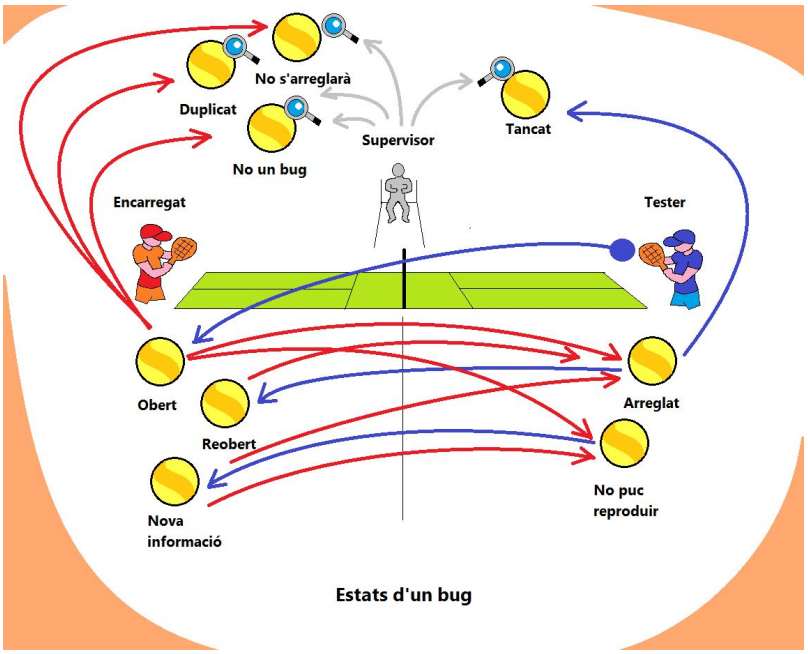
Dispositiu

De manera similar a la versió, i especialment important en el cas d'una aplicació multidispositiu, s'ha d'indicar també quin és el dispositiu **en què es produeix el *bug***. Això és especialment important en *bugs* que tinguin una relació estreta amb el *hardware*, per exemple, un *bug* provocat perquè s'esgoti la memòria principal o un *bug* on un so en particular no funcioni perquè té un format que la targeta d'àudio no pot reproduir. Si no ho fem així, la persona que ho hagi d'arreglar no podrà reproduir mai el *bug*, ja que no comptarà amb el dispositiu apropiat.

2.2.5 Dinàmica de resolució d'un 'bug'

El procés de resolució d'un *bug* és una mica més complicat que el d'una tasca normal, i implica diferents rols com el de *tester*, encarregat i el supervisor. Consta d'una sèrie d'**estats** en què en cada moment la responsabilitat la té **un rol concret**, i ha de realitzar una sèrie d'**accions** abans que el *bug* passi a un altre estat. Això crea una dinàmica, especialment entre el *tester* i l'encarregat, que podríem assimilar a un joc en què el *bug* cada vegada està al terreny d'un de l'altre (vegeu la figura 2.13).

FIGURA 2.13. Estats d'un 'bug'



Aquest "joc" té un component psicològic força important, ja que el *tester* en el fons el que està indicant és que alguna cosa que ha fet l'equip de desenvolupament està malament. Això pot fer que l'encarregat es torni defensiu, li costi admetre que alguna cosa falla o directament negui que s'estigui produint un *bug* encara que el *tester* n'hi doni proves. Això fa que sigui extremadament important:

1. Que l'encarregat i el *tester* siguin **persones diferents**.
2. Que qui determini si un *bug* està resolt **sigui sempre el *tester*** que n'ha informat.

Estats d'un 'bug'

Durant el procés de resolució d'un *bug*, aquest passa per diferents estats o situacions; són els següents:

- **Obert**. És l'estat inicial quan el *tester* introdueix el *bug* a la llista i vol dir que el que sigui que s'hagi detectat encara no s'ha solucionat. En aquesta situació algun membre de l'equip de desenvolupament s'assignarà com a encarregat del *bug* o serà el supervisor o el cap de l'equip de desenvolupament qui ho faci, segons la categoria i àrea d'aquest. Mentre un *bug* estigui a l'estat obert la responsabilitat és, doncs, de l'**encarregat**, i les passes que ha de fer són:
  1. Reproduir el *bug*.
  2. Trobar què l'està causant.
  3. Arreglar-lo.
  4. Comprovar que ja no succeeix.

- **Arreglat**. "Arreglat" o *fixed* vol dir que la persona que té assignat el *bug* l'ha pogut reproduir, ha identificat una possible causa de l'error, ha aplicat els canvis necessaris i des de la seva perspectiva, afirma que està arreglat. En aquest punt el *bug* passa a ser responsabilitat del ***tester***, que ha de verificar que efectivament el *bug* ja no es dona, independentment que ho afirmi la persona que ho ha arreglat.

- **Reobert**. Si el *tester* determina que un *bug* que estava en estat "arreglat" o *fixed* no s'ha solucionat realment, és a dir, aconsegueix reproduir-lo a una nova versió de l'aplicació, li assignarà l'estat de reobert. Aquest estat té el mateix significat que l'estat obert, tot i que indica que el *bug* s'ha intentat solucionar. Cal dir que també pot passar que s'hagi solucionat **parcialment**, és a dir, que encara que continuï passant ara sigui més difícil de reproduir.

- **Tancat**. Si el *tester* verifica que un *bug* en estat "arreglat" o *fixed* està solucionat, aquest passa a estat tancat i no cal que ni el *tester* ni l'encarregat facin cap acció addicional. El *bug*, però, roman a la llista per tal de no perdre'n la informació.

- **No puc reproduir**. Podem dir que el curs típic d'un *bug* és "obert, arreglat i tancat", possiblement amb cicles de "reobert arreglat" abans de "tancat". Poden donar-se, però, altres situacions per les quals existeixen estats especials com "No puc reproduir" (*Cannot Repro*). Aquesta situació consisteix en que la persona encarregada d'un *bug* no pot reproduir-lo, és a dir, no pot observar l'efecte del qual s'ha informat, tot i posicionar-se al mateix punt dins l'aplicació i seguir les passes indicades pel *tester*. En aquesta situació i una vegada comprova que està fent servir la versió correcta de l'aplicació i un dispositiu on el *bug* s'hi hauria de produir, el *bug* ha de passar a estat de "No puc reproduir". En aquest estat la responsabilitat passa a ser del ***tester***, que ha de proveir **passos addicionals** que permetin determinar millor el procés que s'ha de seguir per reproduir el *bug*.

- **Nova informació**. Quan el *tester* aporta informació addicional que permeti caracteritzar millor el *bug*, és a dir, informació que faciliti localitzar-lo i reproduir-lo, com per exemple, passes addicionals, posa el *bug* en estat "Nova informació" (*New Info*). En aquesta situació el *bug* torna a estar en una situació equivalent a l'estat "obert", i passa a ser responsabilitat de la persona assignada per solucionar-lo, que ha de revisar la informació addicional aportada i tractar de reproduir altra vegada el *bug*.

- **"No un *bug*"**. Aquesta és una situació que pot generar conflicte en el costat del *tester*, ja que el desenvolupador li està comunicant que el que s'està informant **no és un *bug*** ('Not A Bug'), sinó una característica de l'aplicació interactiva. En aquest punt, és molt important que el *tester* tingui una informació actualitzada dels requisits i normes de qualitat de l'aplicació per tal que se li pugui indicar quina és la característica que està confonent amb un *bug*. En cas de conflicte entre ambdues parts, correspon al supervisor donar el *bug* per tancat.

Exemple de 'Not A Bug' amb l'ús de les barres de desplaçament

Un exemple d'aquesta situació seria una aplicació d'un museu on es permeti que l'usuari examini textos fent servir unes barres de desplaçament, però si algun text és molt curt, aquestes barres no es mostren. Un *tester* podria pensar que en alguns dels textos curts aquestes barres haurien de sortir i informar-ne com a *bug*. En aquesta situació la persona encarregada canviarà l'estat del *bug* a "No un *bug*" (*Not A Bug*) tot indicant al *tester* la secció del document de disseny on es parla d'aquesta característica.

- **No s'arreglarà**. Aquesta situació és diferent d'un "No un *bug*", en el sentit que la persona encarregada reconeix que el comportament és un *bug*, però indica que aquest no s'arreglarà canviant l'estat a "No s'arreglarà" o *Will Not Fix*. Lògicament cal alguna raó especial per prendre aquesta decisió i normalment la responsabilitat d'aquesta situació recau en el **supervisor**. Tot i que cada situació és diferent, en general podem dir que en aquests casos s'aplica un criteri de **practicitat**, és a dir, es valora si donat el punt del desenvolupament en què s'està, solucionar aquest *bug* pot causar més problemes que avantatges.

Exemple d'absència d'un element a una aplicació

Per exemple, podem trobar un *bug* que diu, correctament, que falta un quadre a una aplicació que permet fer una visita virtual d'un museu, però si en aquell punt hem fet totes les proves necessàries per assegurar que l'aplicació funciona correctament a tots els dispositius sense esgotar la memòria i no tenim temps per repetir les proves, podem decidir que és massa arriscat incloure aquesta nova imatge, ja que augmentarà la quantitat de memòria requerida i pot provocar errors greus en alguns dispositius. En aquesta situació estarà justificat respondre al *tester* amb l'estat "No s'arreglarà" o *Will Not Fix*.

- **Duplicat**. Aquesta situació es dona quan la persona assignada considera que el *bug* del qual s'està informant ja s'havia detectat abans. En aquestes circumstàncies, a més de posar el *bug* en l'estat "duplicat", el desenvolupador ha d'indicar **l'identificador del *bug* original**. El *tester* hi pot estar d'acord o no, i en cas de conflicte serà el **supervisor** el que determinarà si el *bug* és un duplicat o no. A vegades la causa d'un *bug* duplicat pot ser que efectivament s'hagi detectat dues vegades el mateix problema, però també pot passar que l'encarregat jutgi que dos *bugs* tenen una causa comuna quan no la tenen. Per aquesta raó, si el *tester* no està convençut, ho farà saber a l'encarregat i, en cas de conflicte, es requerirà l'arbitratge del supervisor.

2.3 Suport al procés de verificació

El procés de verificació és molt costós en termes de temps i recursos, ja que requereix moltes hores de treball per part dels *testers* i, a més, disposar d'uns recursos **materials** i de *software* adequats. Hi ha una sèrie de mesures que es poden prendre, des del punt de vista de l'empresa o l'equip de desenvolupament, per tal de facilitar aquesta tasca; ens referim a: les proves internes, els modes de depuració o *debug*, les trampes, les dreceres, els tests automàtics, els informes d'error greu i els sistemes de gestió de *bugs*.

2.3.1 Proves internes

El procés de verificació no pot completar-se en un temps raonable si la quantitat de *bugs* és excessiva, ja que, tot i que la verificació té un paper de xarxa que assegura que qualsevol error present a l'aplicació interactiva amb temps suficient s'acabarà detectant, això **no pot substituir** que l'equip hagi incorporat les diferents característiques a l'aplicació amb una mínima seguretat que no contenen errors greus.

Dit d'una altra manera, no podem confiar que la verificació solucionarà tots els errors i treballar d'una forma descuidada, perquè llavors ens trobarem amb una quantitat de feina **inabastable** al final del projecte.

2.3.2 Modes de depuració o 'debug'

Des del departament de programació es pot facilitar la tasca de verificació tot incloent en l'aplicació interactiva un o més modes de **depuració**. Aquests modes serveixen per poder visualitzar informació addicional a l'aplicació que ens permeti identificar més de pressa quin element està causant un problema.

La seva importància no és tant des del punt de vista dels *testers*, ja que aquests no haurien de fer servir el mode de depuració, sinó que faciliten molt la tasca dels **encarregats**, ja que poden reproduir el *bug* i, amb aquesta informació addicional, formular millors **hipòtesis** del que pot estar passant.

2.3.3 Trampes

Una altra eina que accelera el procés de verificació són les "trampes" (*cheats*). Les trampes són combinacions de tecles o accions complicades que donen accés a característiques ocultes per a l'usuari i que permeten saltar-se les normes de l'aplicació interactiva.

Les trampes no són tan útils des del punt de vista dels *testers* com des del punt de vista dels **encarregats**, ja que permeten accedir més ràpidament a aquella àrea de l'aplicació on es produeix l'error. Per exemple, en una aplicació on hem d'escoltar una narració que dura molts minuts, podríem tenir una trampa que ens permeti saltar-la de manera que puguem arribar abans a la pantalla següent.

2.3.4 Dreceres

Les dreceres són potser l'única eina que té sentit que posem a disposició dels *testers*, ja que els permet **saltar directament** a una pantalla d'una aplicació interactiva, sense haver de passar per les anteriors.

Hem de dir, però, que en el moment en què es fa servir una drecera, automàticament està **invalidant** el test, ja que s'està fent la prova en un estat alterat de l'aplicació. Així i tot, si des de l'equip de desenvolupament podem tenir una seguretat molt forta que la part de l'aplicació que està ometent **no té influència** o té una influència limitada a la part que s'està verificant, podem incloure la drecera per tal que els *testers* puguin dedicar més temps a provar àrees que, d'altra manera, no podrien verificar a fons.

Així i tot, hem de ser conscients que **l'única prova completament vàlida** és aquella que fem **des del començament**, sense cap mena d'ajuda addicional.

2.3.5 Tests automàtics

Des de l'equip de desenvolupament també podem proveir de **tests automàtics** que permetin verificar una gran quantitat de casos, encara que no excloguin que també s'hagi de fer una verificació o *testeig* manual en la mesura que es pugui. Per exemple, en una aplicació on hi hagi diàlegs de veu amb subtítols, podem fer un test que verifiqui que totes les frases se subtitulen correctament mostrant-les una darrere l'altra en seqüència.

Aquests tests també poden ser estocàstics, per exemple, podem fer un test que generi milions de clics aleatoris a la pantalla per tal de comprovar que la interfície d'una aplicació interactiva no es bloqueja en cap cas.

2.3.6 Informes d'error greu

Trobareu més informació sobre els *bugs* greus al punt "Severitat", d'aquest mateix apartat.

A vegades, podem trobar *bugs* que facin que l'aplicació finalitzi **abruptament**, per exemple perquè hagi esgotat la memòria principal disponible o hagi intentat fer un accés il·legal a una zona de memòria.

En aquests casos, normalment el sistema operatiu envia un avís a l'aplicació abans de tancar-la, de manera que si preparem un codi que guardi ràpidament a un fitxer algunes variables clau de l'aplicació, el *tester* podrà **adjuntar-lo** amb l'informe del *bug* i serà més fàcil fer-nos una idea de què pot haver passat. Aquests fitxers s'anomenen informes d'error greu (*crash reports*) i moltes vegades els genera el mateix sistema operatiu.

2.3.7 Sistemes de gestió de 'bugs'

La gestió dels *bugs* pot ser complicada perquè d'una banda són molt nombrosos i, de l'altra, cadascun d'ells passa per una sèrie d'estats que només uns usuaris amb uns rols concrets poden manipular. Això fa que fer aquesta gestió amb eines no específiques com un full de càlcul o una base de dades convencional sigui propens a fer que es produeixin **malentesos** entre l'equip de desenvolupament i els *testers*. Per adreçar això hi ha **eines específiques** que permeten gestionar aquesta informació.

Com que els *bugs* tenen unes característiques particulars, aquestes eines són diferents de les eines estàndard de gestió de tasques o es presenten com un **mòdul diferent** dins d'aquestes. Els seus serveis inclouen:

- Registrar els diferents perfils i usuaris.
- Definir les àrees i categories que distingim a l'aplicació.
- Gestionar fitxers adjunts i comentaris als informes de *bugs*.
- Generar avisos per correu electrònic o algun altre mitjà quan un *bug* canvia d'estat.

Aquests sistemes també permeten generar estadístiques i gràfics que ens serveixen per visualitzar quants *bugs* hi ha a la base de dades, de quin tipus són i com evoluciona la proporció de *bugs* que hem fixat.

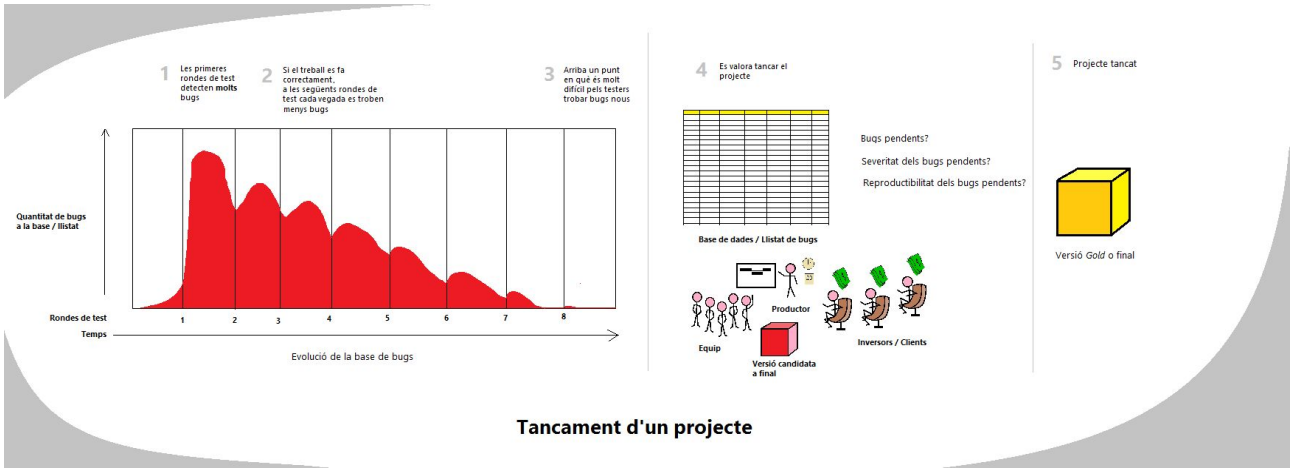
2.4 Tancament del projecte

La verificació (o *testeig*) és un procediment que es porta a terme durant tota la vida d'un projecte, i és la part final de qualsevol cicle de desenvolupament iteratiu, al final del qual es produeix una nova versió.

Aquest procés de verificació inicialment no ha de ser gaire intens per tal de no bloquejar el desenvolupament de les característiques del projecte, però amb el temps i les successives versions ha de ser cada vegada més sever, fins a arribar a les últimes fases del projecte, on el procés de verificació ha de ser exhaustiu.

D'aquesta manera, en les fases finals el focus d'activitat ha de consistir **només en verificar i corregir errors**, primer poblant la base de *bugs* amb els resultats dels tests i després treballant per corregir-los, de manera que a cada cicle de verificació i correcció es **redueixi progressivament** el nombre de *bugs* pendents, fins a arribar a un punt on en quedin pocs o no siguin gaire greus, moment en què parlarem de la versió "Candidata a Publicació" o *Release Candidate* i es començarà a plantejar la publicació (vegeu la figura 2.14, part esquerra i mitjana).

FIGURA 2.14. Evolució de la base de dades de 'bugs' i tancament del projecte



La publicació s'aprovarà quan tots els *bugs* estiguin corregits o les persones responsables i administradors de l'empresa considerin que els *bugs* pendents no són greus, moment en què es declara la versió "publicable", *Release* o *Gold* i el desenvolupament del projecte es donarà per finalitzat o tancat.

> El criteri mínim que hauríem de complir per publicar una aplicació interactiva és que sigui possible per a un usuari o jugador recórrer la **ruta típica** sense trobar *bugs* bloquejadors o errors greus.

Trobareu la definició de "ruta típica" al punt "Severitat" d'aquest mateix apartat.

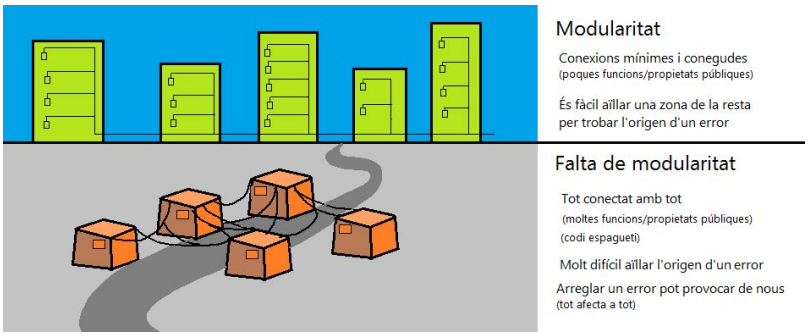
Hi ha alguns **factors que afavoreixen el tancament** d'un projecte interactiu, en un temps raonable; parlem de la modularitat, la congelació de característiques i els continguts completats.

2.4.1 Modularitat

La **modularitat** és un aspecte principalment de programació, o més pròpiament d'enginyeria de *software*, però que té una influència molt forta en el procés de resolució de *bugs*, ja que si no s'ha tingut en compte pot fer que sigui **molt difícil** o directament **impossible** corregir tots els *bugs* d'un projecte.

Vol dir que l'aplicació interactiva ha d'estar composta per **blocs independents** amb responsabilitats ben definides i connectats entre si per **interfícies** que impedeixin que un mòdul **accedeixi directament** a les variables d'un altre (vegeu la figura 2.15).

Figura 2.15. Modularitat



Si ho fem així, d'una banda podrem localitzar més fàcilment els *bugs*, ja que només haurem d'activar i desactivar blocs fins a trobar aquell que estigui donant problemes, i també tindrem la garantia que si el comportament que no funciona és responsabilitat d'un mòdul concret, la causa del *bug* estarà en aquell mòdul.

Aquest no és, però, l'aspecte més important, sinó també que quan corregim un *bug* el corregim en un punt concret d'un mòdul **sense afectar la resta**, de manera que tenim la garantia que a l'arreglar un *bug* no n'estem introduint un altre.

Tot connectat amb tot

Potser és més fàcil d'entendre si ho plantegem a la inversa, és a dir, què passa si tots els mòduls estan connectats entre si i accedeixen directament a totes les variables de la resta de mòduls? Que tot queda connectat amb tot, i un petit canvi a un variable de la interfície pot provocar que un procediment de so, que en principi no té res a veure, ja que és una àrea completament diferent, comenci a fallar. En aquestes circumstàncies qualsevol canvi que fem per arreglar un *bug*, pot provocar indirectament una pila de nous *bugs*, de manera que pot arribar a ser impossible reduir la mida de la base de *bugs* i que sigui una versió publicable.

Congelació de característiques i continguts completats

Dintre del procés de desenvolupament, cada característica o contingut de l'aplicació interactiva pateix una evolució en les successives etapes del desenvolupament, i forçosament les primeres versions d'una característica en concret, per exemple un menú d'opcions, contenen molts *bugs*. A més, cada vegada que introduïm una modificació a la característica, introduïm *bugs* addicionals.

Això fa que es necessitin un **nombre mínim de versions** perquè una característica es desenvolupi completament i a més, en algun punt, **deixar d'introduir modificacions** per tal de poder tancar-la, és a dir, concentrar-nos només en corregir els *bugs* que presenti.

Això, que és cert per a una característica aïllada, també ho és per a un projecte; de manera que en les últimes fases d'un projecte, i per tal de poder tancar-lo correctament, s'han de **definir dues fites**, relacionades amb les característiques i els continguts respectivament anomenades:

- "Congelació de característiques" o *Feature Freeze*.
- "Continguts completats" o *Content Complete*.

La primera marca el punt a partir del qual **no s'han d'incorporar noves característiques** al projecte, per exemple, un mode de control amb pantalla tàctil i, la segona, el punt on **ja no afegirem més contingut**, per exemple més recursos gràfics o sons.