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.

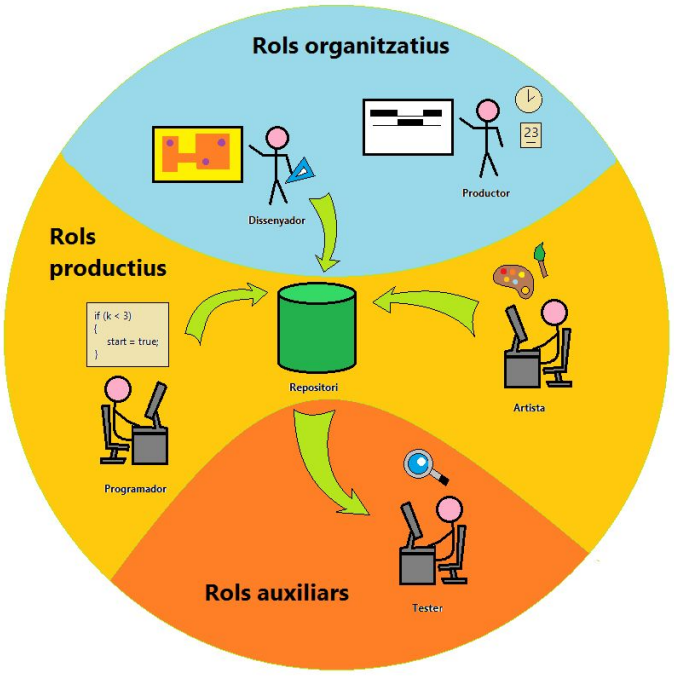
D'altra banda, per poder assegurar que l'equip està desenvolupant el projecte correctament, cal **organitzar el treball en unitats** que puguem caracteritzar i situar en un pla de treball. Aquesta és la feina principal del productor.

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



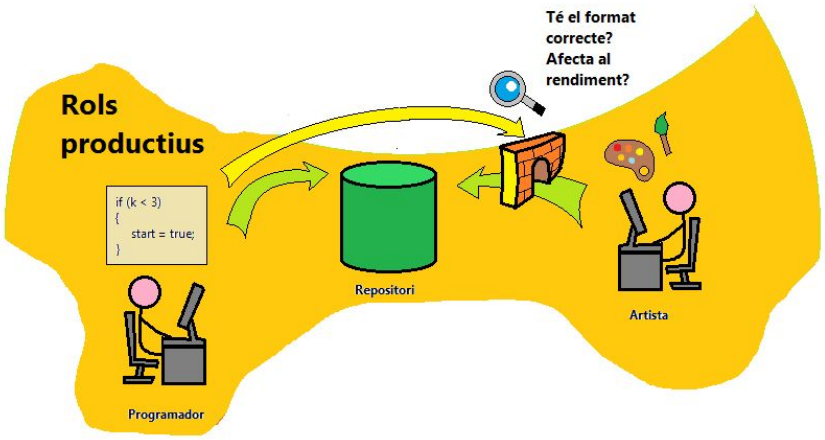
Podem classificar els rols dins d'un equip de desenvolupament en diferents grups. Els rols **productius**, els rols **organitzatius** i els rols **auxiliars**. Cada rol delimita les responsabilitats que té el membre de l'equip i el tipus de tasques que pot dur a terme.

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.

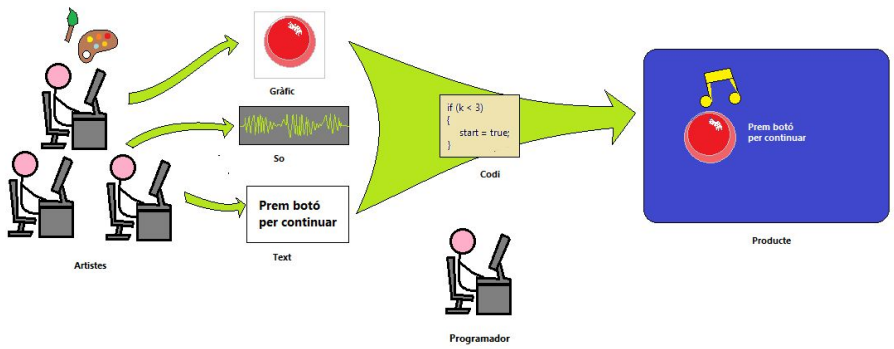
Els **programadors** són els encarregats de dissenyar i produir el codi del projecte. Duen a terme diverses tasques. Per exemple, **integren** els recursos que produeix la resta de rols productius al repositori del projecte, i s'asseguren que tinguin un format adequat per assolir els **paràmetres de 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.

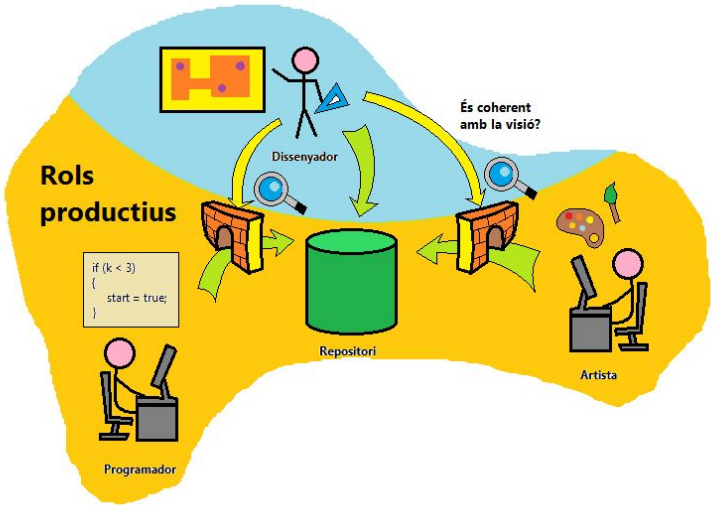
La funció dels dissenyadors és mantenir la visió del resultat que s'està cercant i assegurar-se que els diferents rols productius són coherents amb aquesta visió. Entre les **qualitats desitjables per a un dissenyador** distingim:

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



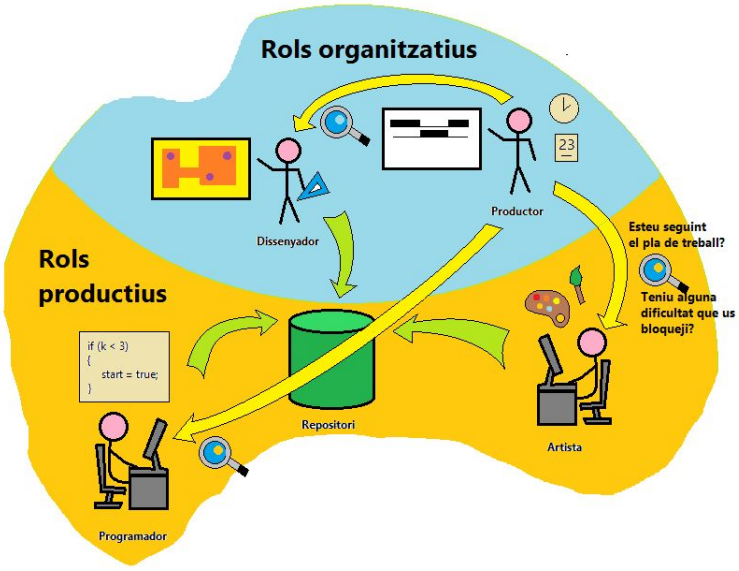
D'altra banda, els productors són les persones encarregades de gestionar el temps, els materials i les persones assignades a les diferents tasques. Entre les **funcions del productor** hi ha:

- 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



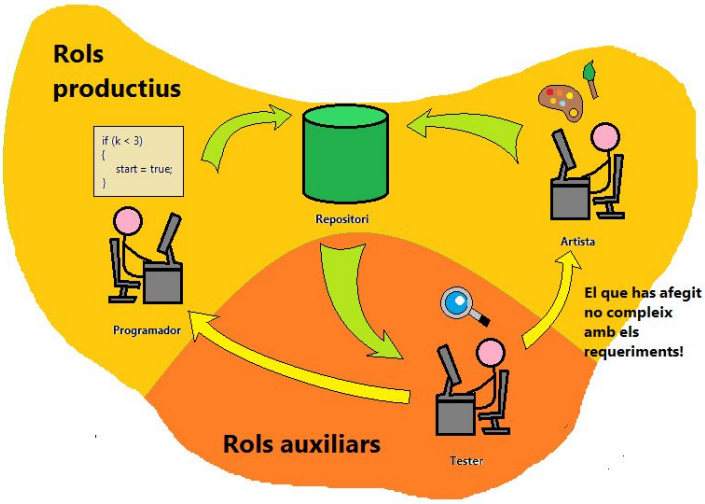
1.1.3 Rols auxiliars

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

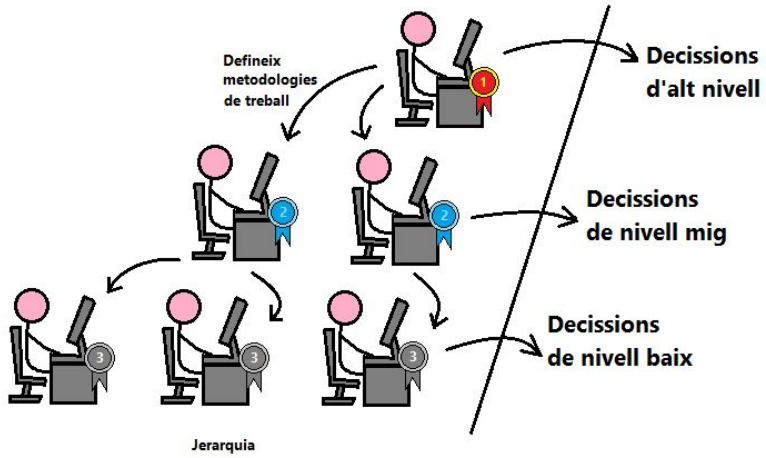
Un equip de desenvolupament pot estar format per un nombre de membres que pot o no coincidir amb els rols requerits per fer les tasques. Això pot donar lloc a l'aparició de jerarquies i especialitats. Poden donar-se **diferents situacions**:

- 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.

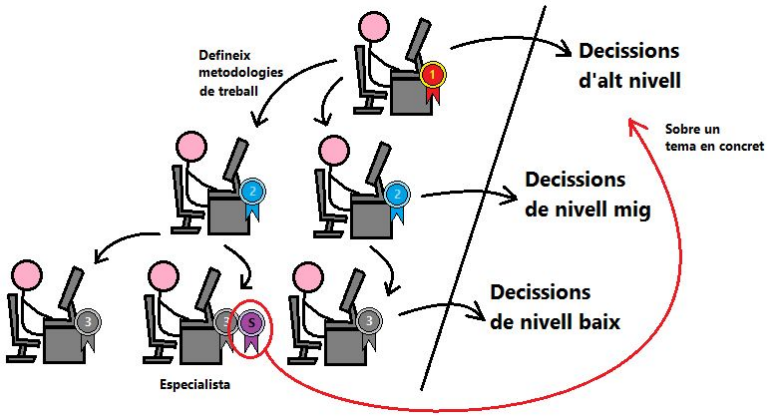
Una **jerarquia** defineix **qui té més pes** a l'hora de definir aspectes comuns com metodologies de treball i decisions importants sobre les tasques. Exemples d'això serien la distinció entre grafistes júnior i sènior o entre artistes i directors d'art. El criteri per establir la jerarquia hauria de 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



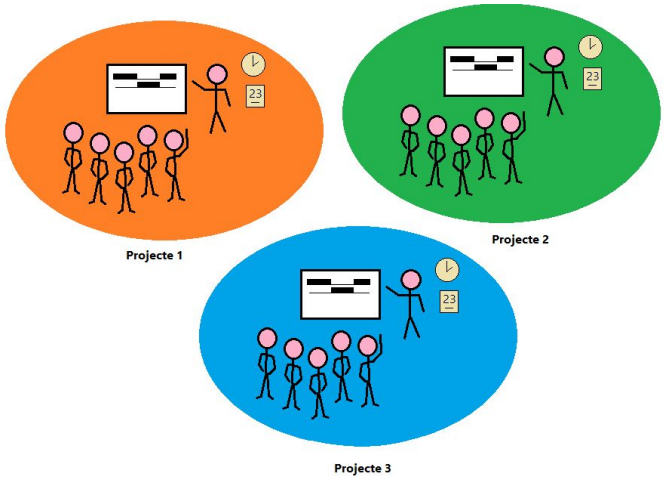
1.1.5 Estils organitzatius

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**,

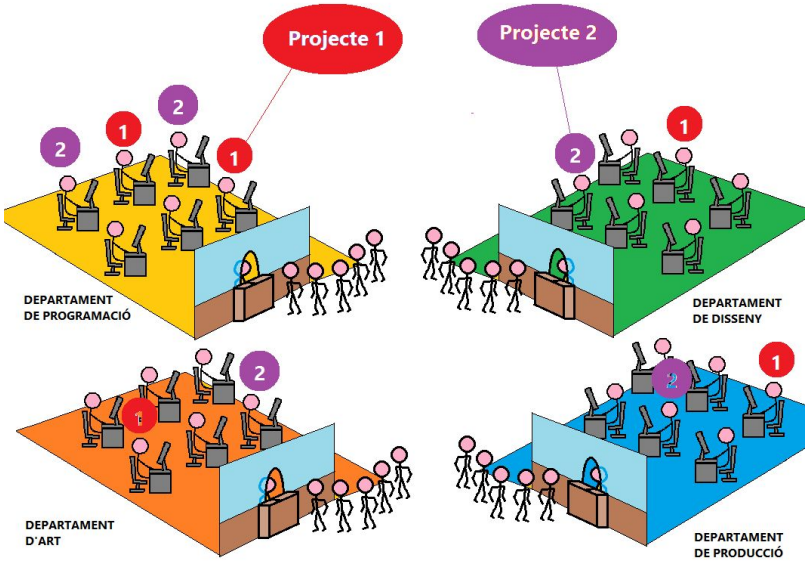
l'equip es reuneix amb freqüència i és relativament senzill canviar les prioritats de les tasques i fer-ne el seguiment (vegeu la figura 1.9).

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



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

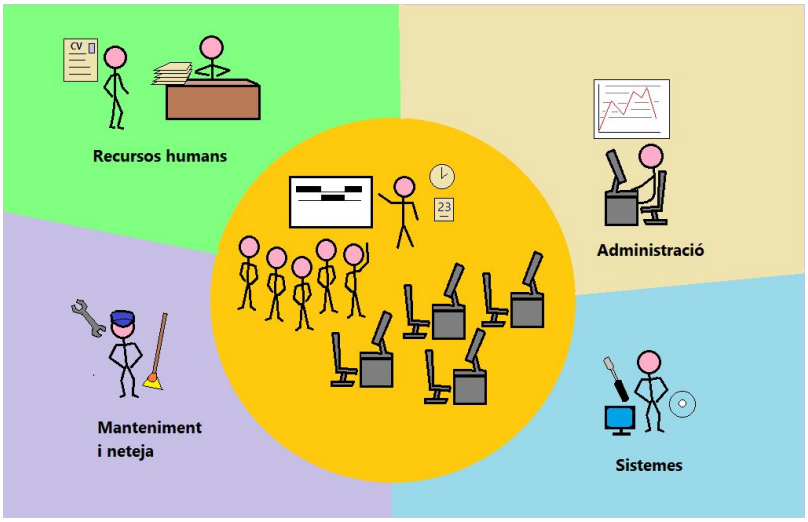
Cada estil és adequat per a un tipus de projecte. El treball per departaments és propi de sectors on els projectes són molt semblants entre si i les tasques que ha de fer cada un d'ells són fixes i conegudes, per exemple el sector editorial, mentre que el treball per equips és més adequat per a projectes 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

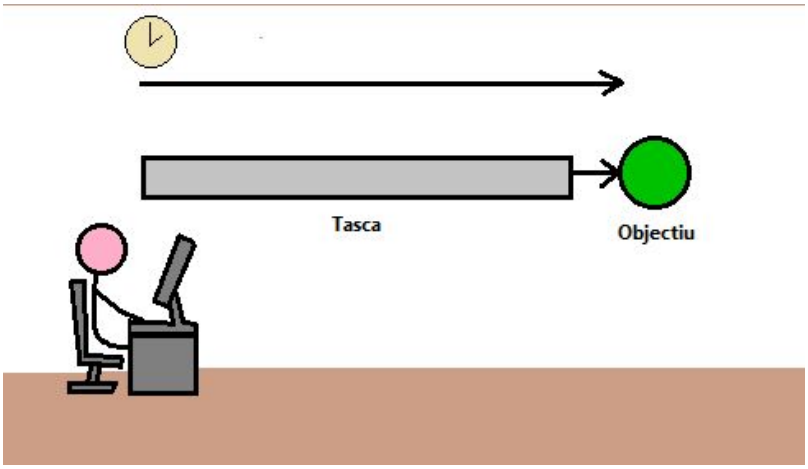
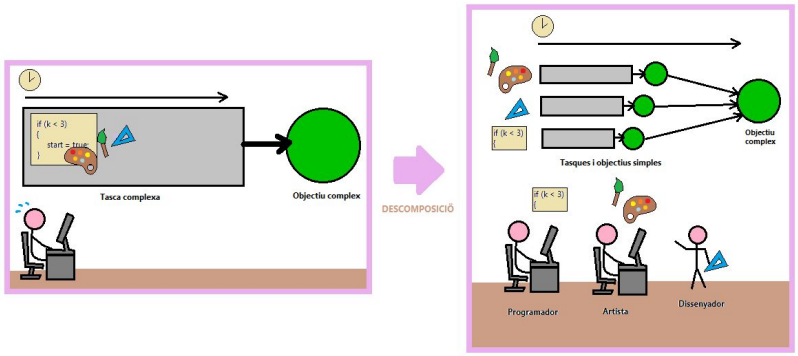


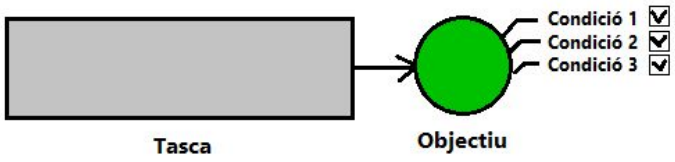
Figura 1.13. Descomposició d'una tasca complexa



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

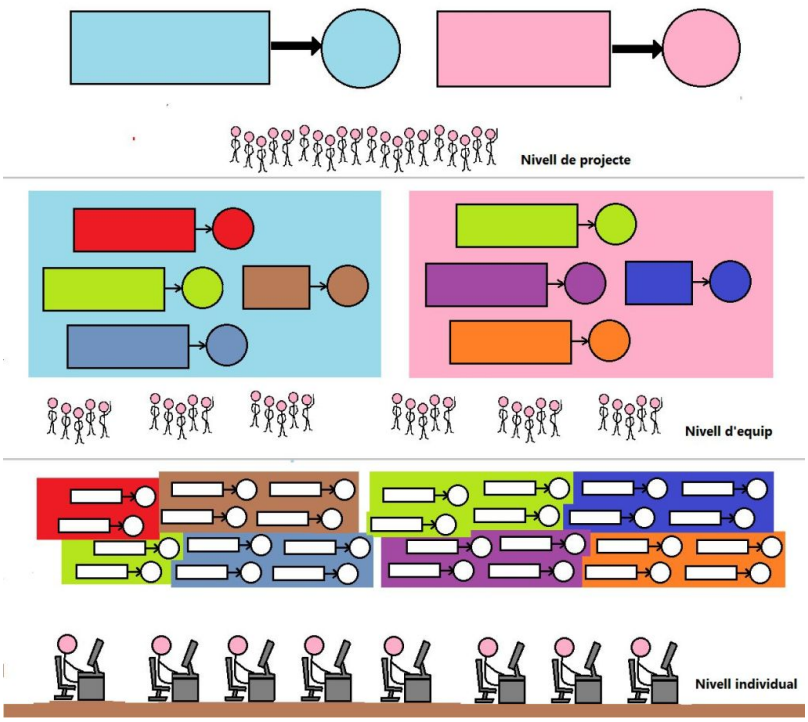


Moltes vegades podem fer servir **tasca i objectiu** de forma intercanviable, ja que tota tasca implica un objectiu i tot objectiu implica una tasca; però no els podem unificar perquè són conceptes diferents.

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.

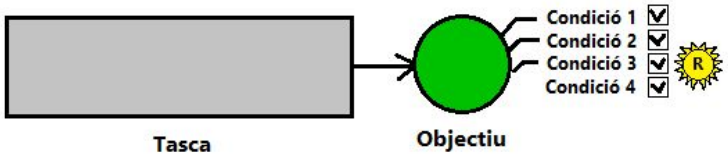
Requisits i tasques a nivell de projecte

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



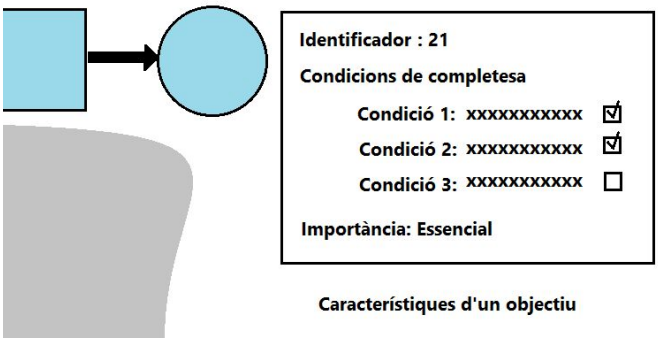
Quant a les tasques necessàries per complir amb aquests objectius, aquestes esdevenen **tasques de nivell de projecte**, que s'han de descompondre en tasques d'equip i organitzar-se també en un pla de treball, i a aquest nivell sol prendre la forma d'un **full de ruta**. Aquests tipus de tasques es duen 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.

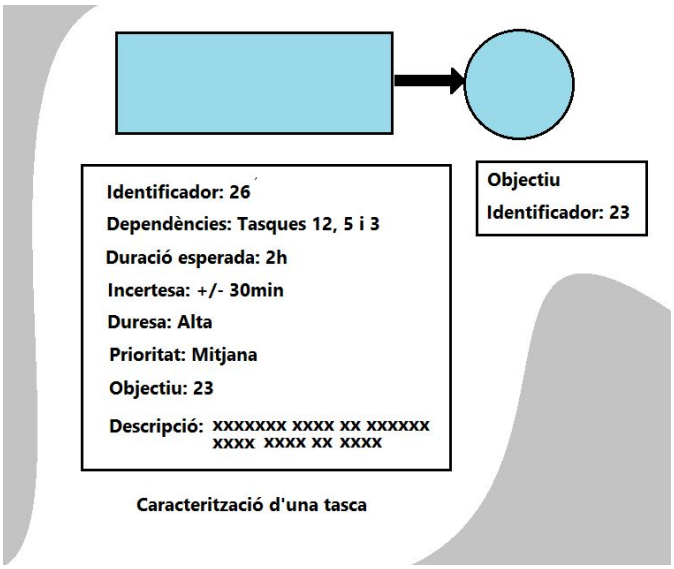
Si caracteritzem els objectius per importància, com a productors, podrem decidir entre **diferents estratègies** que tindran un impacte al producte final:

- 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

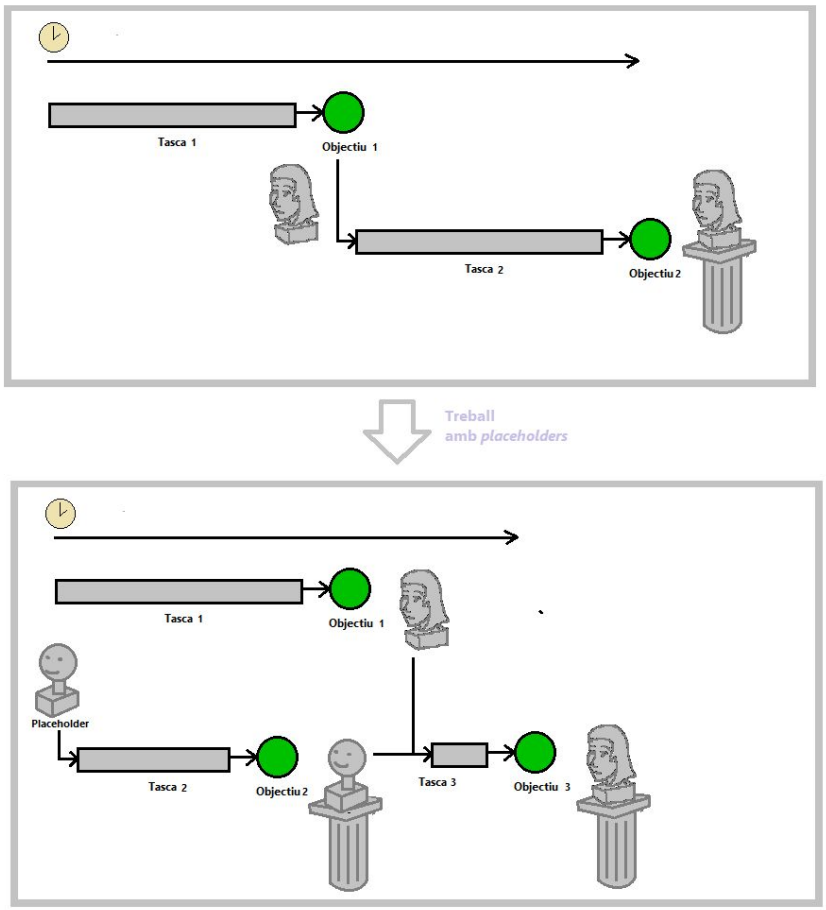
Una tasca pot incloure dependències, és a dir, **altres tasques** que s'hagin d'haver assolit per tal que es pugui dur a terme. Això pot ser degut al fet que necessita alguns dels recursos produïts per aquestes tasques o bé perquè requereix que es compleixi alguna condició que forma part dels objectius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

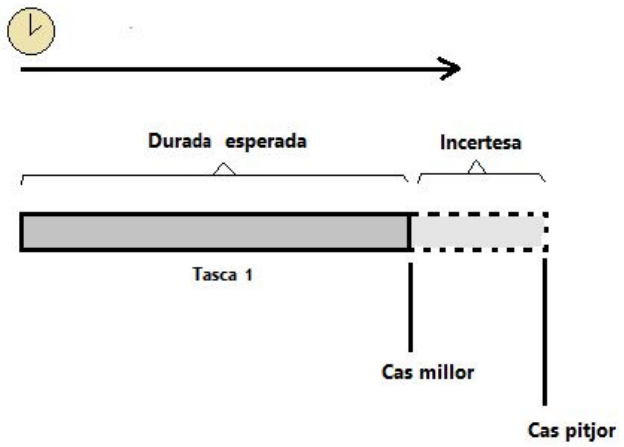
Tot i que amb l'experiència els productors i membres de l'equip poden donar unes mesures força confiables, hem de tenir en compte que totes les durades de les tasques en un pla de treball **sempre són estimacions**. En concret, la durada esperada representa una estimació de la durada en el millor cas, 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

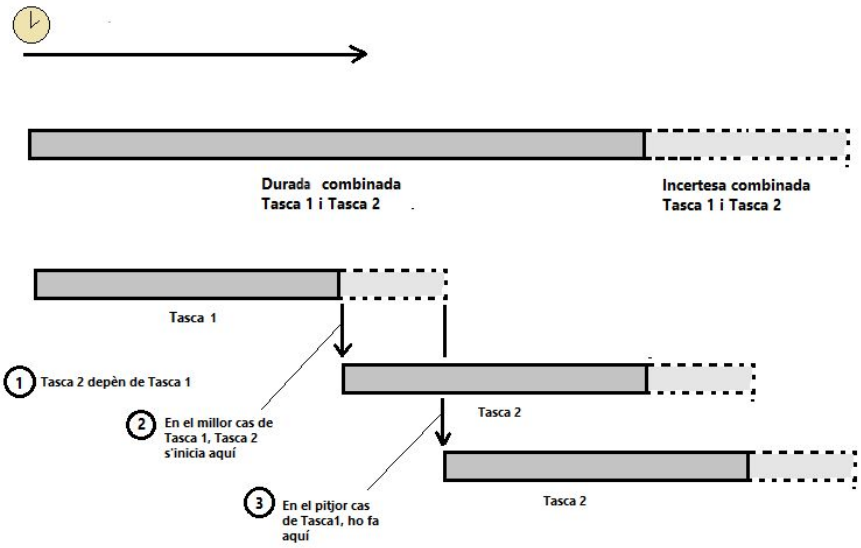
Vegeu la secció "Objectius" i "Caracterització d'un objectiu" d'aquest apartat per tenir una explicació més detallada del que és un objectiu.

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:

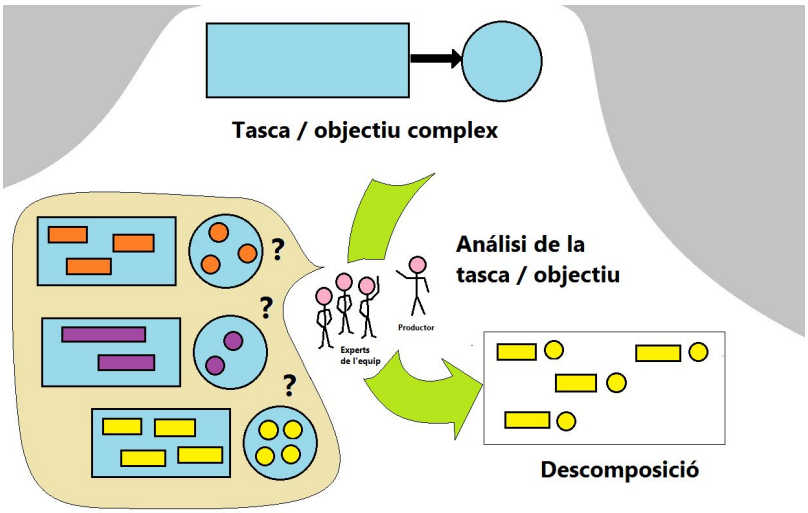
1. Anàlisi i descomposició
2. Reunió de l'equip
3. Elaboració del pla
4. Comunicació del pla
5. Supervisió del pla

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

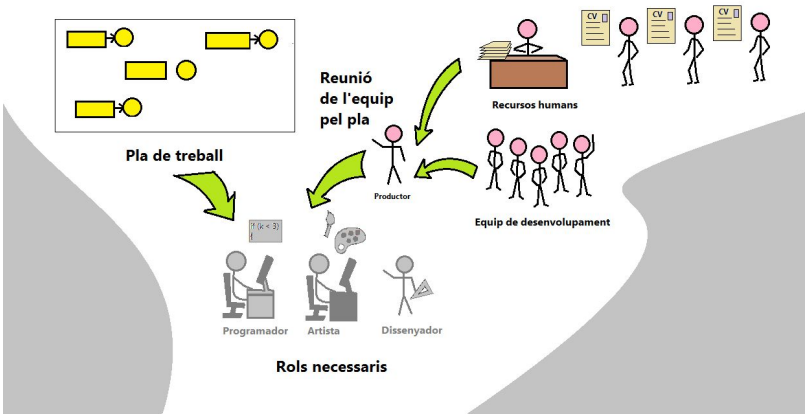


Reunió de l'equip

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



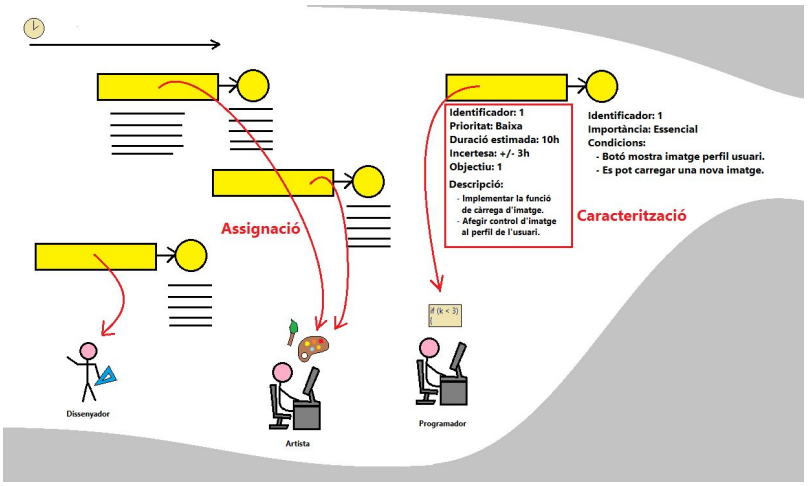
Elaboració del pla

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.

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.

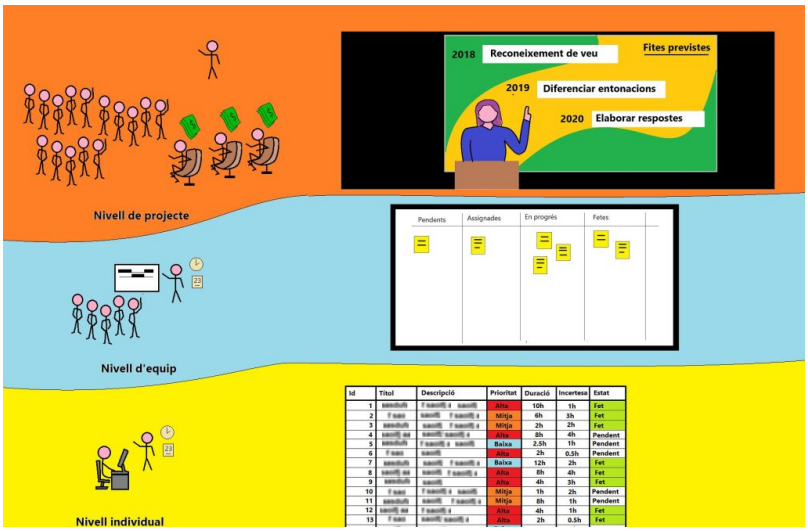
Un productor estil brúixola, en canvi, farà un pla mínim per tal d'assolir els objectius inicials i no planificarà res més per avançat, sinó que esperarà que aquests objectius s'assoleixin i només llavors estendrà el pla de treball a nous objectius. Aquest estil és adequat quan el projecte implica un 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

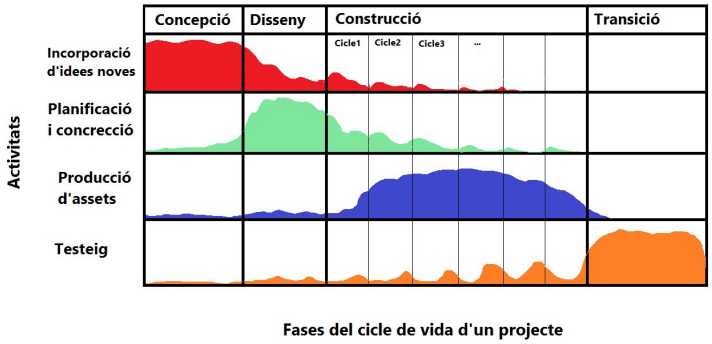
a terme les tasques, ja que dona la impressió d'una manca de confiança per part del productor i a algunes persones els pot crear inseguretat.

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.

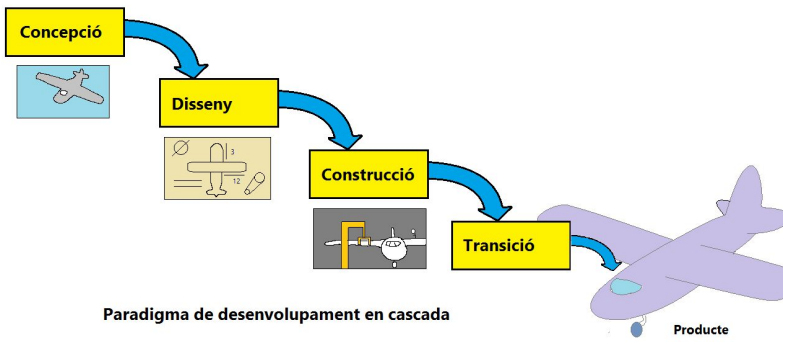
1.4.4 Paradigmes organitzatius

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.

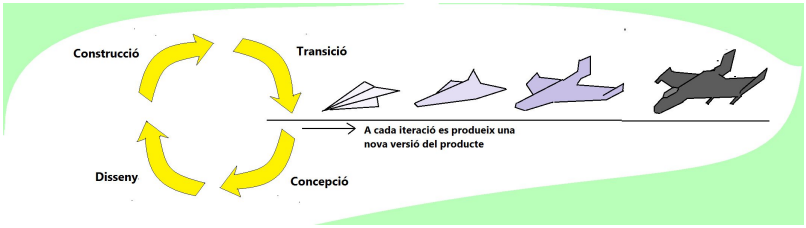
FIGURA 1.27. Desenvolupament en cascada



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.

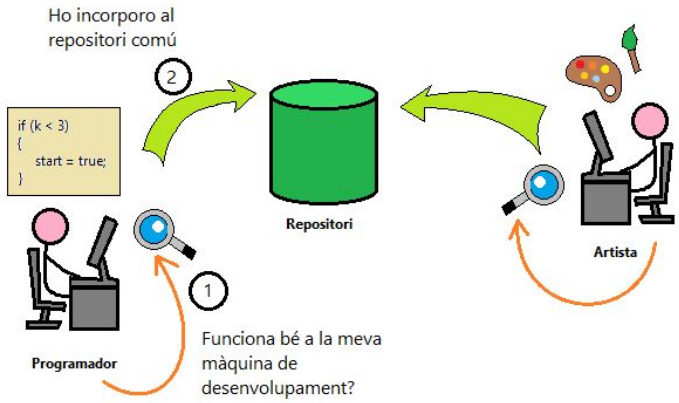
2.1 Proves internes

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:

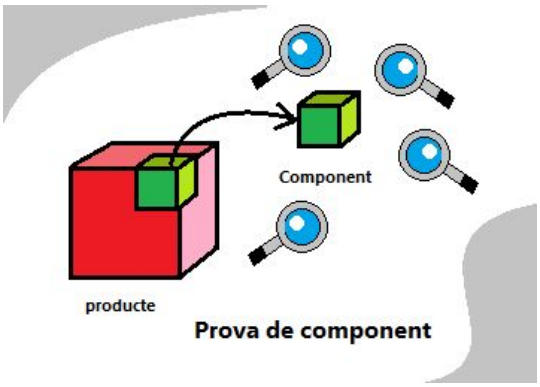
- Lògicament, aquest no pot ser un test complet, ja que durant el dia a dia del desenvolupament d'un projecte no disposem d'una quantitat suficient de temps, i possiblement no puguem tenir accés al dispositiu final, per exemple una *tablet*, de manera que hem de fer aquesta prova sobre un **nombre limitat 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



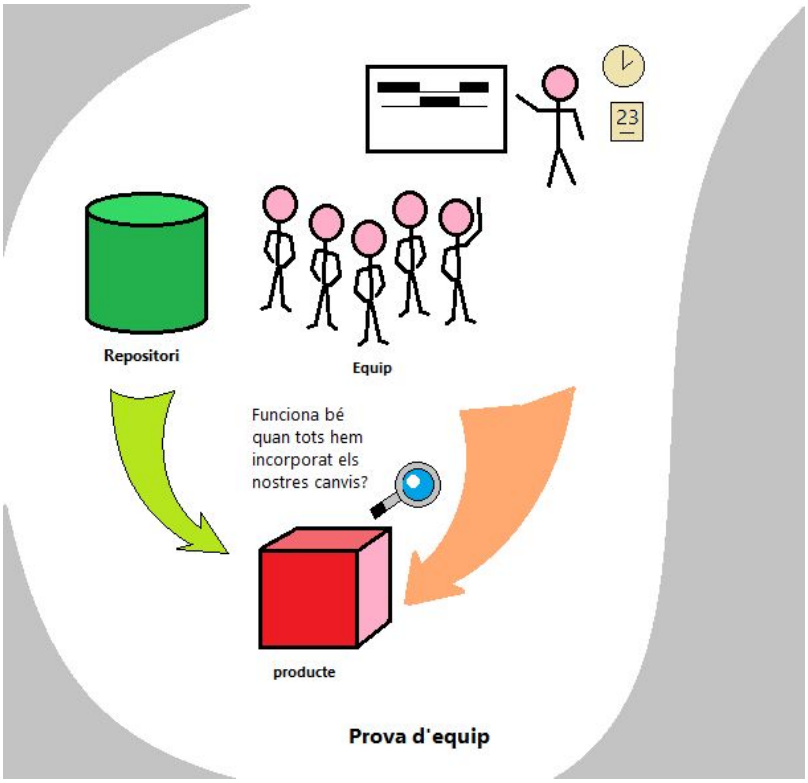
En el cas dels programadors, es poden fer **proves sobre un component o sistema concret**. Com a mínim, hauria de contenir una trucada a cadascun dels seus mètodes i, si és possible, algun tipus de prova automatitzada (vegeu la figura 2.2).

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

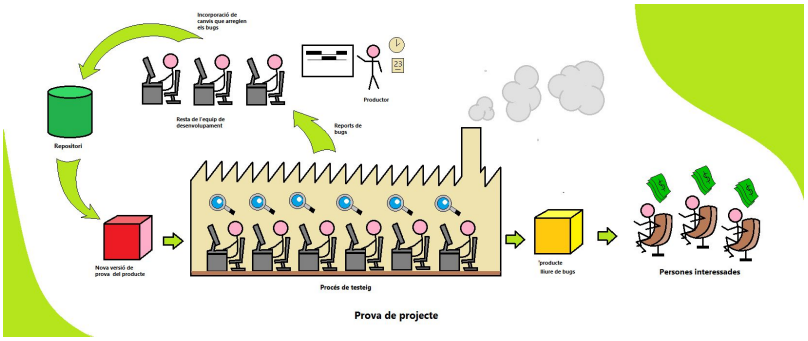
Donar una càrrega de treball molt alta a un sistema de l'aplicació o a un component, per exemple una aplicació que mostri un mapa, fent-li representar una quantitat molt alta d'icones per veure si així i tot el component continua funcionant. El raonament que hi ha darrere d'aquesta prova és que si el 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.

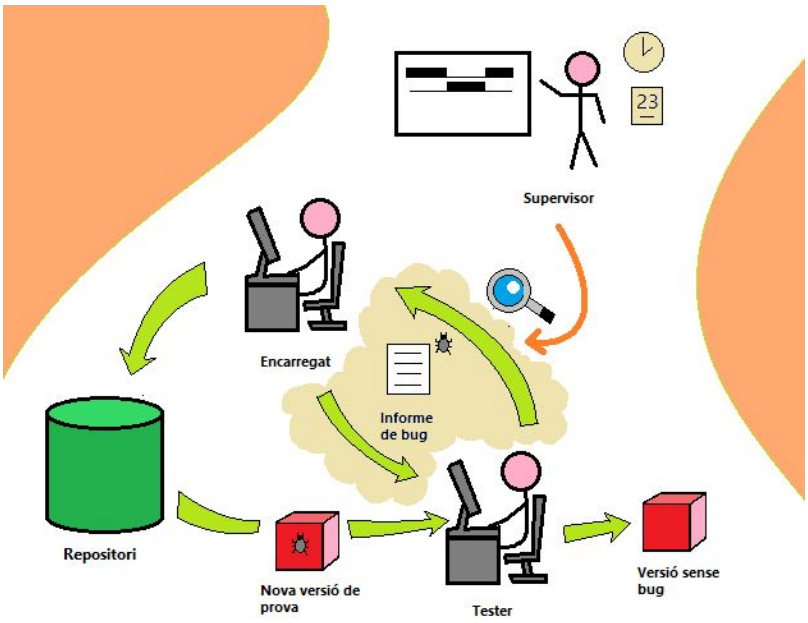
Com que és un procés col·lectiu que té una extensió en el temps i implica diferents persones, des del punt de vista de la producció caldrà **definir rols**, organitzar el treball en **àrees** i crear unes **dinàmiques de treball** que ens permetin assegurar que en finalitzar el procés haurem verificat 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.

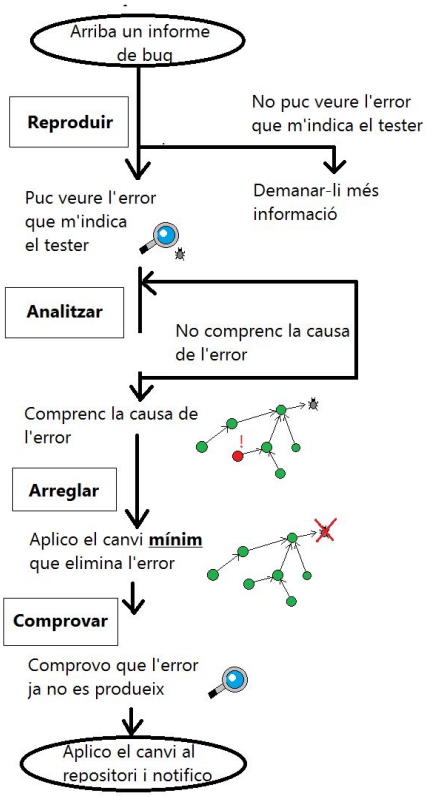
L'encarregat

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

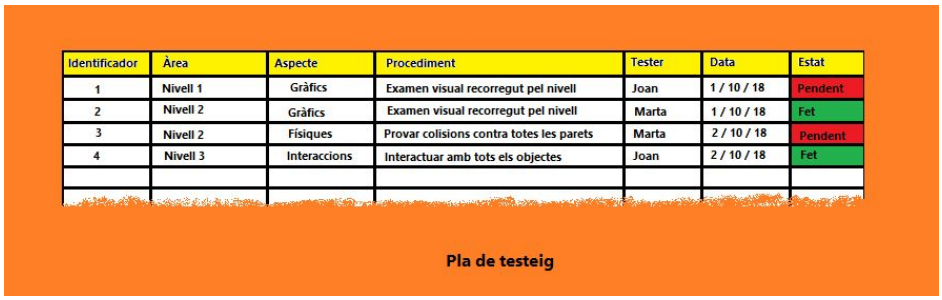
El supervisor és una persona **amb un cert rang** dins el projecte que supervisa l'activitat dels verificadors i els encarregats, i pot prendre accions en aquelles situacions que sobrepassin l'àmbit de responsabilitat d'aquests. Si es dona una d'aquestes situacions, ha de consultar els administradors 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



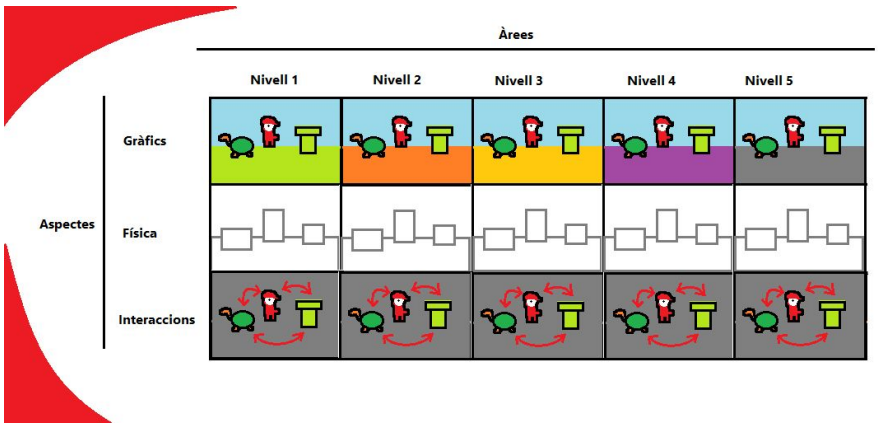
Identificador	Àrea	Aspecte	Procediment	Tester	Data	Estat
1	Nivell 1	Gràfics	Examen visual recorregut pel nivell	Joan	1 / 10 / 18	Pendent
2	Nivell 2	Gràfics	Examen visual recorregut pel nivell	Marta	1 / 10 / 18	Fet
3	Nivell 2	Físiques	Provar colisions contra totes les parets	Marta	2 / 10 / 18	Pendent
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.

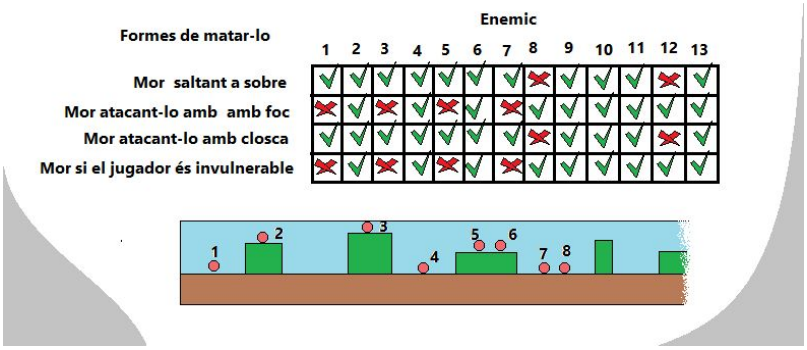
De fet, des de fa temps sabem que no és possible decidir de forma automàtica aspectes aparentment senzills, com per exemple saber si un programa informàtic, una vegada iniciat, s'aturarà en algun moment.

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

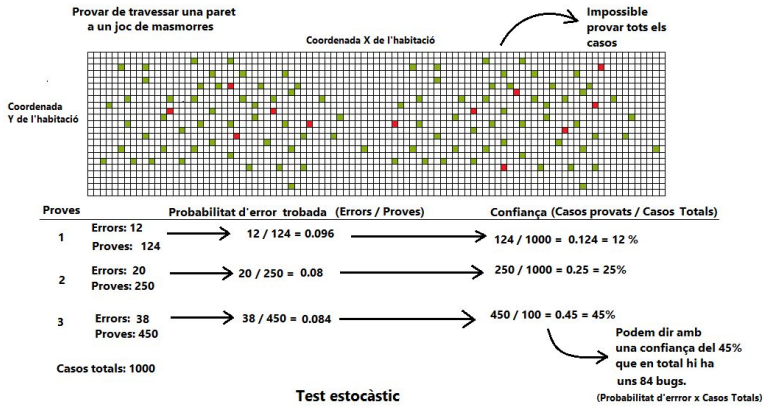


Test repetitiu o estocàstic

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

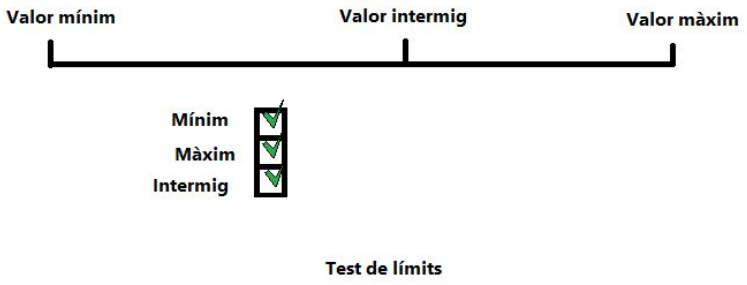


Un problema que trobem aquí és que les persones no som bones per generar comportaments aleatoris i quasi sempre seguim **algun patró** més o menys conscientment. Això fa que perquè un test aleatori generi una bona confiança estadística, s'hagi de fer **automàticament** o bé el facin persones diferents 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.

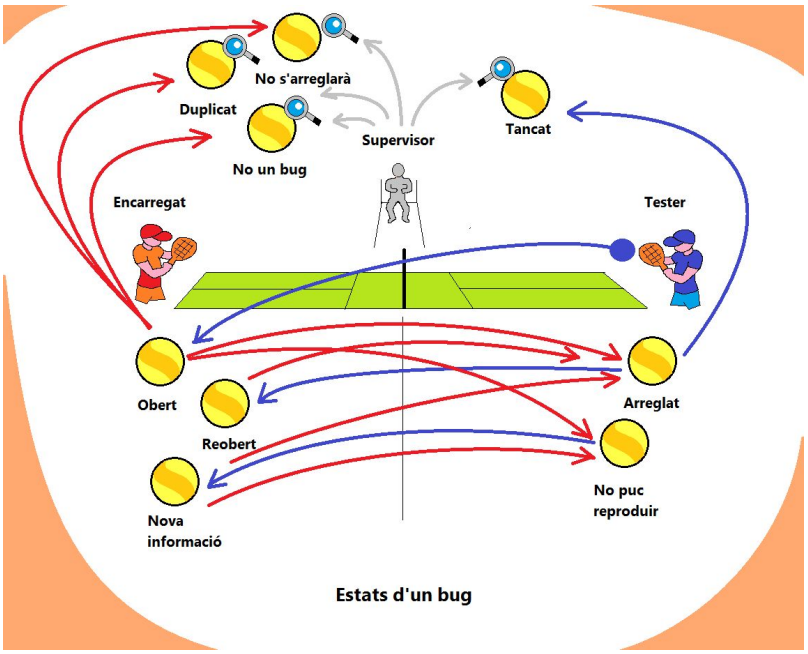
Dispositiu

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.

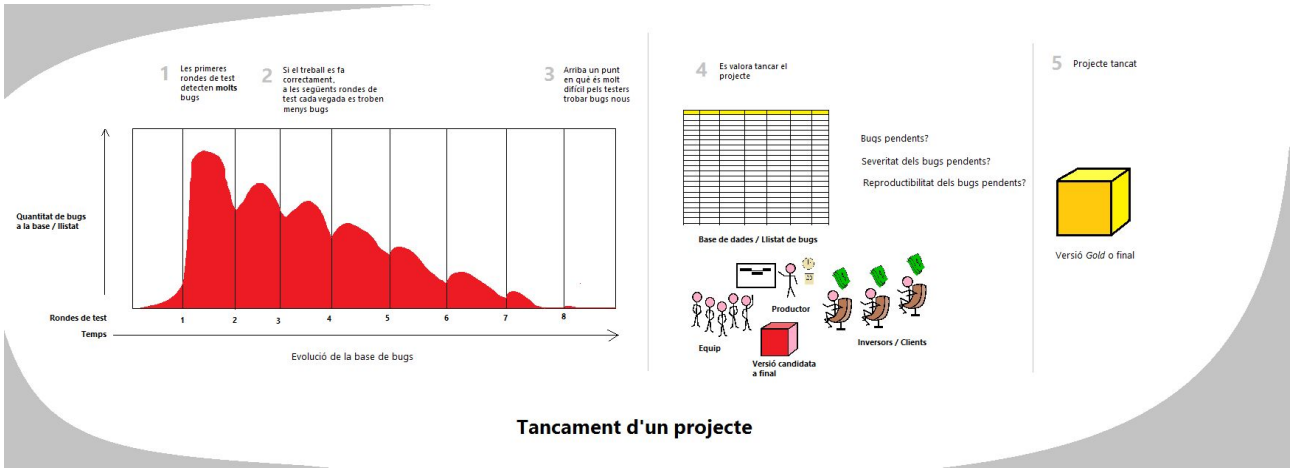
2.4 Tancament del projecte

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.

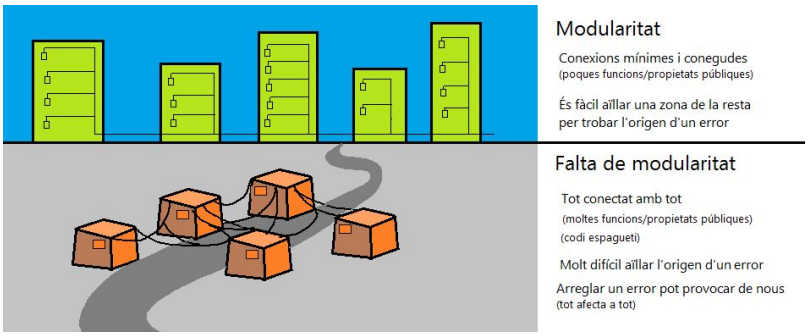
Hi ha alguns **factors que afavoreixen el tancament** d'un projecte interactiu, en un temps raonable; parlem de la modularitat, la congelació de característiques i els continguts completats.

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.