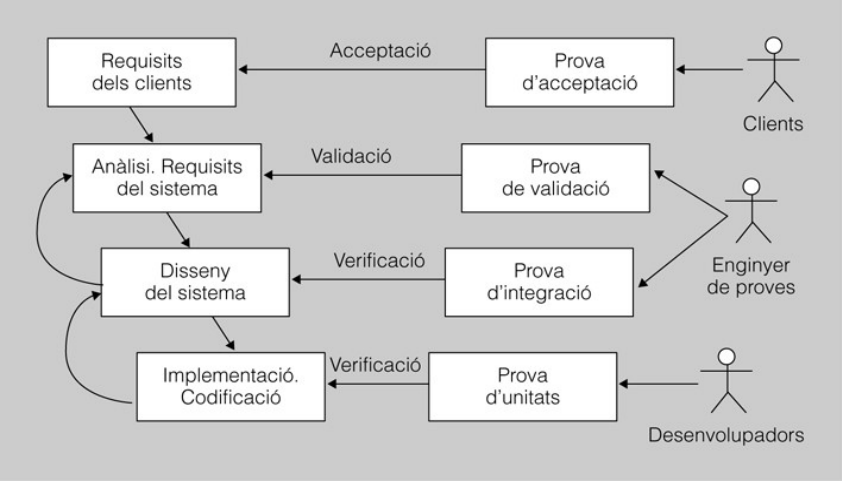
1. Disseny i realització de proves de programari

Les fases de desenvolupament d'un projecte són: presa de requeriments, anàlisi, disseny, desenvolupament, proves, finalització i transferència.

NOTA: Un error no detectat a l'inici del desenvolupament d'un projecte pot arribar a necessitar cinquanta vegades més esforços per ser solucionat que si és detectat a temps.

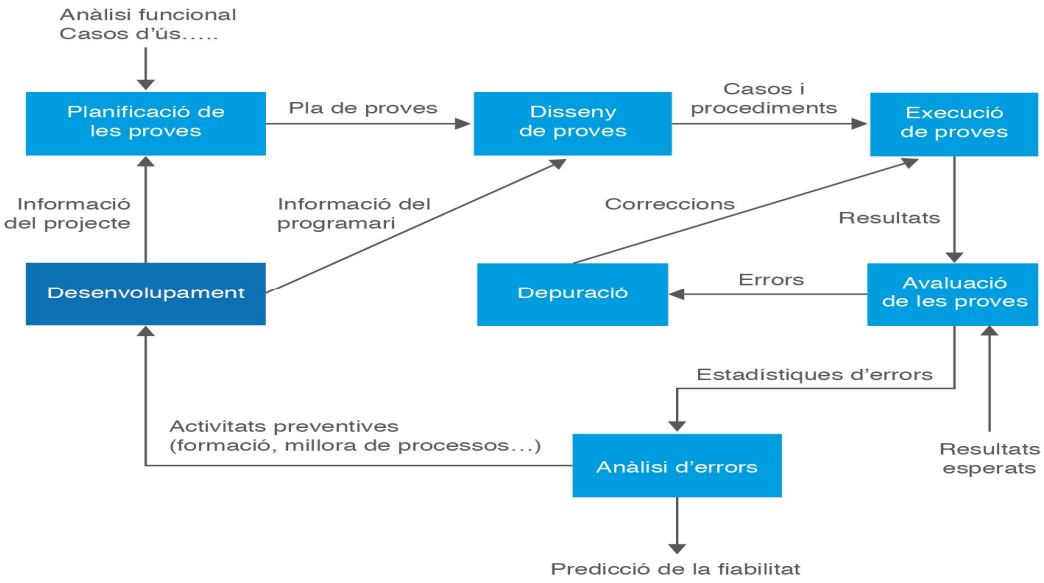
Les proves en el cicle de vida d'un projecte

A cada una de les fases del cicle de vida d'un projecte, caldrà que el treball dut a terme sigui validat i verificat.



Procediments, tipus i casos de proves

Aquest projecte de proves requerirà d'una planificació, un disseny del pla de proves, una execució de les mateixes i una avaluació dels resultats, per tal d'analitzar els errors i poder aplicar les accions necessàries.



Planificació de les proves

La planificació de les proves és una tasca que cal anar desenvolupant al llarg de totes les fases del projecte informàtic. No cal esperar la fase de programació per crear aquest pla de proves; a la fase d'anàlisi i a la fase de disseny ja es tenen prou dades per tal de poder començar a establir les primeres línies del pla de proves.

La **planificació de les proves** té com a objectiu arribar a la creació d'un pla d'actuació que es refereixi a quan i com es duran a terme les proves. Però per a això cal dur a terme una anàlisi minuciosa del sistema i dels seus elements. El pla de proves ha de contenir totes les funcions, les estratègies, les tècniques i els membres de l'equip de treball implicats.

- **Identificador del pla de proves**. És l'identificador que s'assignarà al pla de proves. És important per poder identificar fàcilment quin abast té el pla de proves. Per exemple, si es volen verificar les interfícies i procediments relacionats amb la gestió de clients, el seu pla de proves es podria dir PlaClients.
- **Descripció del pla de proves**. Defineix l'abast del pla de proves, el tipus de prova i les seves propietats, així com els elements del programari que es volen provar.
- **Elements del programari a provar**. Determina els elements del programari que s'han de tenir en compte en el pla de proves, així com les condicions mínimes que s'han de complir per dur-ho terme.
- **Elements del programari que no s'han de provar**. També és important definir els elements que no s'hauran de tenir en compte al pla de proves.
- **Estratègia del pla de proves**. Defineix la tècnica a utilitzar en el disseny dels casos de prova, com per exemple la tècnica de capsa blanca o de capsa negra, així com les eines que s'utilitzaran o, fins i tot, el grau d'automatització de les proves.
- **Definició de la configuració del pla de proves**. Defineix les circumstàncies sota les quals el pla de proves podrà ser alterat, finalitzat, suspès o repetit. Quan s'efectuïn les proves, s'haurà de determinar quin és el punt que provoca que se suspenguin, ja que no tindria gaire sentit continuar provant el programari quan aquest es troba en un estat inestable. Una vegada els errors han estat corregits, es podrà continuar efectuant les proves; és possible que s'iniciïn des del principi del pla o des d'una determinada prova. Finalment, es podrà determinar la finalització de les proves si aquestes han superat un determinat llindar.
- **Documents a lliurar**. Defineix els documents que cal lliurar durant el pla de proves i en finalitzar-lo. Aquesta documentació ha de contenir la informació referent a l'èxit o fracàs de les proves executades amb tot tipus de detall. Alguns d'aquests documents poden ser: resultats dels casos de proves, especificació de les proves, subplans de proves…
- **Tasques especials**. Defineix les taques necessàries per preparar i executar les proves. Però hi ha algunes tasques que tindran un caràcter especial, per la seva importància o per la seva dependència amb d'altres. Per a aquest tipus de tasques, serà necessari efectuar una planificació més detallada i determinar sota quines condicions es duran a terme.
- **Recursos**. Per a cada tasca definida dins el pla de proves, s'haurà d'assignar un o diversos recursos, que seran els encarregats de dur-la a terme.

- **Responsables i Responsabilitats**. Es defineix el responsable de cadascuna de les tasques previstes en el pla.
- **Calendari del pla de proves**. En el calendari queden descrites les tasques que s'hauran d'executar, indicant les seves dependències, els responsables, les dates d'actuació i la durada, així com les fites del pla de proves. Una eina molt utilitzada per representar aquest calendari del pla de proves és el Diagrama de Gantt.
- **Gestionar els canvis**: no és d'estranyar que, al llarg del cicle de vida del projecte, i amb major probabilitat quan més llarga és la seva durada, es presentin canvis en l'abast del projecte. Serà necessari adaptar el pla de proves a les noves especificacions.
- **Gestionar els riscos**: un risc es podria definir com un conjunt de situacions que poden provocar un impediment o retard en el pla de proves. Els riscos s'hauran d'identificar al més aviat possible i analitzar la probabilitat que hi ha que succeeixin, tot aplicant mesures preventives, si es considera oportú, o disposar d'un pla de contingència en cas que el risc sorgís.

Disseny de les proves. Tipus de proves

El disseny de les proves és el pas següent després d'haver dut a terme el pla de proves. Aquest disseny consistirà a establir els casos de prova, identificant, en cada cas, el tipus de prova que s'haurà d'efectuar.

Tipus:

- **Estructurals o de capsa blanca** → Se centren en la implementació dels programes per escollir els casos de prova..
- **Funcionals o de capsa negra** → El seu objectiu és validar que el codi compleix la funcionalitat definida.
- **D'integració**
- **De càrrega i acceptació**
- **De sistema i de seguretat**
- **De regressió i de fum**

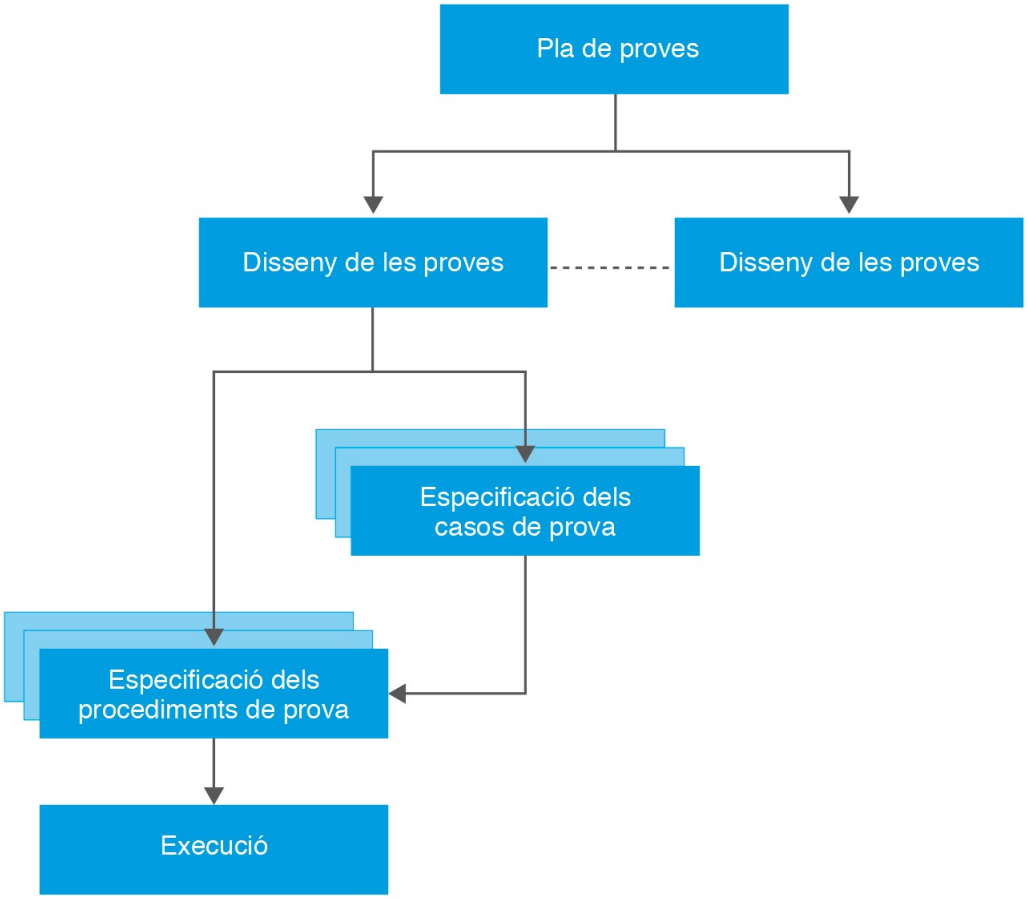
NOTA: Un **cas de prova** defineix com es portaran a terme les proves, especificant, entre d'altres: el tipus de proves, les entrades de les proves, els resultats esperats o les condicions sota les quals s'hauran de desenvolupar.

Els casos de prova segueixen un cicle de vida clàssic**:**

- **Definició dels casos de prova.**
- **Creació dels casos de prova.**
- **Selecció dels valors per als tests.**
- **Execució dels casos de prova.**
- **Comparació dels resultats obtinguts amb els resultats esperats.**

NOTA: Els procediments de prova especifiquen com es podran dur a terme els casos de prova o part d'aquests de forma independent o de forma conjunta, establint les relacions entre ells i l'ordre en què s'hauran d'atendre.

Cada **cas de prova** haurà de ser independent dels altres, tindrà un començament i un final molt marcat i haurà d'emmagatzemar tota la informació referent a la seva definició, creació, execució i validació final.



Tipus de proves

Tipus de proves unitàries:

- Són el tipus de proves de més baix nivell.
- Es duen a terme a mesura que es va desenvolupant el projecte.
- Les efectuen els mateixos programadors.
- Tenen com a objectiu la detecció d'errors en les dades, en els algorismes i en la lògica d'aquests.
- Les proves unitàries es podran dur a terme segons un enfocament estructural o segons un enfocament funcional.
- El mètode utilitzat en aquest tipus de proves és el de la capsa blanca o el de capsa negra.

Tipus de proves funcionals:

- Són les encarregades de detectar els errors en la implementació dels requeriments d'usuari.
- Les duran a terme els verificadors i els analistes, és a dir, persones diferents a aquelles que han programat el codi.
- S'efectuen durant el desenvolupament del projecte.
- El tipus de mètode utilitzat és el funcional.

Tipus de proves d'integració:

- Es duran a terme posteriorment a les proves unitàries.
- També les efectuen els mateixos programadors.
- Es duen a terme durant el desenvolupament del projecte.
- S'encarreguen de detectar errors de les interfícies i en les relacions entre els components.
- El mètode utilitzat és el de capsa blanca, el de disseny descendent i el de bottom-up.

Tipus de proves de sistemes:

- La seva finalitat és detectar errors en l'assoliment dels requeriments.
- Les duran a terme els verificadors i els analistes, és a dir, persones diferents a aquelles que han programat el codi.
- S'efectuen en una fase de desenvolupament del programari.
- El tipus de mètode utilitzat és el funcional.

Tipus de proves de càrrega:

- S'efectuen un cop acabat el desenvolupament, però abans de les proves d'acceptació.
- També les realitzen **analistes** i **verificadors**.
- Es comprova el rendiment i la integritat de l'aplicació ja acabada amb dades reals i en un entorn que també simula l'entorn real.
- Es realitzen amb un enfocament funcional.

Tipus de proves d'acceptació:

- El seu objectiu és la validació o acceptació de l'aplicació per part dels usuaris.
- És per això que les duran a terme els clients o els usuaris finals de l'aplicació.
- Aquestes proves es duran a terme una vegada finalitzada la fase de desenvolupament. És possible fer-ho en la fase prèvia a la finalització i a la transferència o en la fase de producció, mentre els usuaris ja fan servir l'aplicació.
- El tipus de mètode utilitzat també és el funcional.
- Inclouen diferents tipus de prova. Entre altres, les proves alfa i les proves beta. En aquestes, el client realitza les proves a l'entorn del desenvolupador, al primer cas, i, al segon cas, en el propi entorn del client.

Tipus de proves de sistema:

- Es realitzen després de les proves d'acceptació i amb el sistema ja integrat a l'entorn de treball.
- La seva finalitat, precisament, és comprovar que aquesta integració és correcta.
- L'enfocament utilitzat, lògicament, és el de **capsa negra**.
- Les realitzen verificadors i analistes.
- Inclou diferents tipus de prova. Entre altres, proves de rendiment, de resistència, de robustesa (davant entrades incorrectes), de seguretat, d'usabilitat i d'instal·lació.

Tipus de proves de regressió:

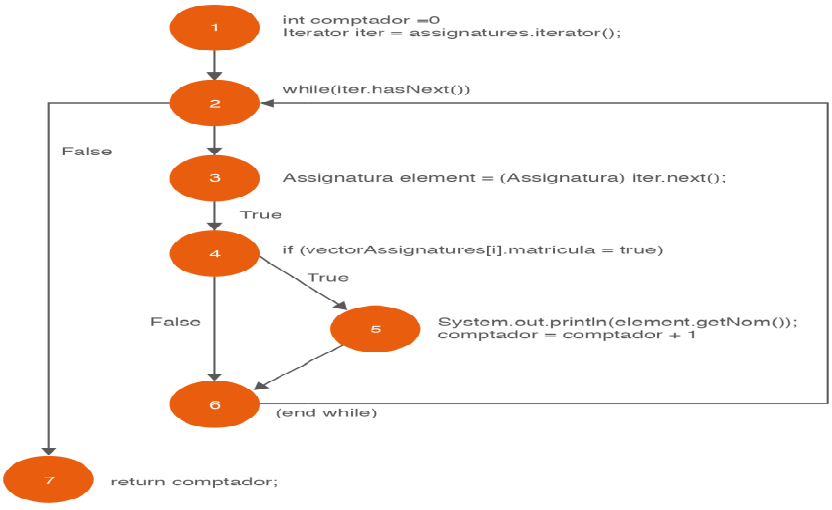
- La seva finalitat és detectar possibles errors introduits en haver realitzat canvis al sistema, bé per millorar-lo, bé per corregir altres errors.
- Consisteixen bàsicament en repetir proves ja realitzades amb èxit abans de realitzar el canvi. Per tant, inclourà tant proves de capsa blanca com de capsa negra.

Tipus de proves "de fum":

Són proves ràpides de les funcions bàsiques d'un programari que normalment es realitzen després d'un canvi en el codi abans de registrar aquest codi modificat en la documentació del projecte.

NOTA: Les **proves de capsa blanca** se centren en la implementació dels programes per escollir els casos de prova. L'ideal seria cercar casos de prova que recorreguessin tots els camins possibles del flux de control del programa. Aquestes proves se centren en l'estructura interna del programa, tot analitzant els camins d'execució. Permetran recórrer tots els possibles camins del codi i veure què succeeix en cada cas possible. Els mètodes que es veuran dintre de les proves de capsa blanca són el de **cobertura de flux de control** i el de **complexitat ciclomàtica**.

Cobertura de Flux de Control:



Pot ser impossible cobrir-ne el 100% si el programa és molt complex, però podem tenir un mínim de garanties d'eficàcia si seguim els suggeriments per dissenyar els casos de prova tenint en compte el següent:

- **Conjunt bàsic de camins independents**: és el conjunt de camins independents que cal cobrir amb el joc de proves.
- **Camí independent**: camí simple amb alguna branca no inclosa encara a cap camí del conjunt bàsic.
- **Camí simple**: camí que no té cap branca repetida.
- **Casos de prova**: un cop determinat el conjunt bàsic, cal dissenyar un cas de prova per a cadascun dels seus camins de manera que, entre tots s'executi almenys una vegada cada sentència.
- **Condicions**: cal assegurar-se que els casos de prova elaborats d'aquesta manera cobreixen totes les condicions del programa que s'avaluen a cert/fals. Cal tenir en compte, però, que, les condicions múltiples, s'han de dividir en expressions simples (una per a cada operand lògic o comparació), de manera que s'ha de provar que es compleixi o no cada part de cada condició. Per tant, en realitzar el gràfic, a una condició múltiple li correspondrà un node per cada condició simple que formi part d'ella i caldrà afegir també al gràfic les branques necessaris per representar correctament el funcionament d'aquestes condicions.
- **Bucles**: s'han de dissenyar els casos de prova de manera que s'intenti executar un bucle en diferents situacions límit.

Complexitat ciclomàtica:

Complexitat ciclomàtica = nombre de branques – nombre de nodes + 2

Si tenemos en cuenta el gráfico anterior, los nodos que intervienen son 1, 2, 3, 4, 5, 6, 7. Si hacemos el cálculo de Complejidad ciclomática quedaría de la siguiente manera:

- Número de nodos: 7
- Número de ramas: 8
- Complexitat ciclomàtica CC = 8 – 7 + 2 = **3**
- Esto significa que habrá 3 caminos posibles:
  - **Camino 1**: 1-2-7
  - **Camino 2**: 1-2-3-4-6-2-7
  - **Camino 3**: 1-2-3-4-5-6-2-7

Proves unitàries: enfocament funcional o proves de capsa negra:
Les proves de capsa negra proven la funcionalitat del programa, per al qual es dissenyen casos de prova que comprovin les especificacions del programa.
Les tècniques de prova de capsa negra pretenen trobar errors en funcions incorrectes o absents, errors d'interfície, errors de rendiment, inicialització i finalització. Es centra en les funcions i en les seves entrades i sortides.

Classes d'equivalència:

Pasos:

1. **Identificar les condicions**, restriccions o continguts de les entrades i les sortides.
2. **Identificar, a partir de les condicions, les classes d'equivalència de les entrades i les sortides**. Per identificar-ne les classes, el mètode proposa algunes recomanacions:
    - Cada **element de classe** ha de ser tractat de la mateixa manera pel programa, però cada classe ha de ser tractada de manera diferent en relació amb una altra classe. Això assegura que n'hi ha prou de provar algun element d'una classe per comprovar que el programa funciona correctament per a aquesta classe, i també garanteix que cobrim diferents tipus de dades d'entrada amb cadascuna de les classes.
    - Les classes han de recollir tant **dades vàlides com errònies**, ja que el programa ha d'estar preparat i no bloquejar-se sota cap circumstància.
    - Si s'especifica un **rang de valors** per a les dades d'entrada, per exemple, si s'admet del 10 al 50, es crearà una classe vàlida ($10 \leq X \leq 50$) i dues classes no vàlides, una per als valors superiors ($X > 50$) i l'altra per als inferiors ($X < 10$).
    - Si s'especifica un **valor vàlid** d'entrada i d'altres de no vàlids, per exemple, si l'entrada comença amb majúscula, es crea una classe vàlida (amb la primera lletra majúscula) i una altra de no vàlida (amb la primera lletra minúscula).
    - Si s'especifica un **nombre de valors** d'entrada, per exemple, si s'han d'introduir tres nombres seguits, es crearà una classe vàlida (amb tres valors) i dues de no vàlides (una amb menys de dos valors i l'altra amb més de tres valors).
    - Si hi ha un conjunt de **dades d'entrada concretes** vàlides, es generarà una classe per cada valor vàlid (per exemple, si l'entrada ha de ser vermell, taronja, verd, es generaran tres classes) i una altra per un valor no vàlid (per exemple, blau).
    - Si no s'han recollit ja amb les classes anteriors, s'ha de seleccionar una classe per cada possible classe de **resultat**.

3. **Crear els casos de prova a partir de les classes d'equivalència detectades**. Per a això s'han de seguir els passos següents:
    - Escollir un valor que representi cada classe d'equivalència.
    - Dissenyar casos de prova que incloguin els valors de totes les classes d'equivalència identificades.

Anàlisi de valors límit i errors típics:

Rangos:

- En els rangs de valors, agafar els extrems del rang i el valor intermedi.
- Si s'especifiquen una sèrie de valors, agafar el superior, l'inferior, l'anterior a l'inferior i el posterior al superior.
- Si el resultat es mou en un determinat rang, hem d'escollir dades a l'entrada per provocar les sortides mínima, màxima i un valor intermedi.
- Si el programa tria una llista o taula, agafar l'element primer, l'últim i l'intermedi.

Errores:

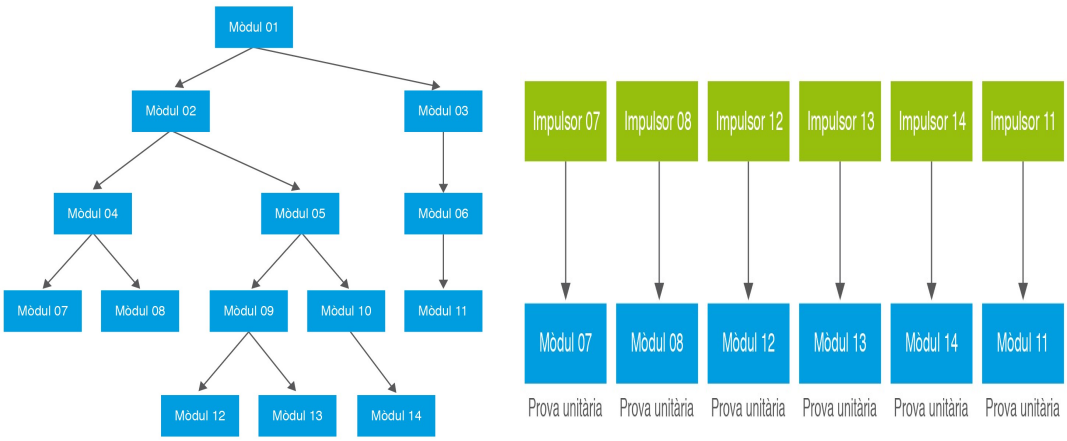
- El valor zero sol provocar errors, per exemple, una divisió per zero bloqueja el programa.
- Quan s'ha d'introduir una llista de valors, caldrà centrar-se en la possibilitat de no introduir cap valor, o introduir-ne un.
- S'ha de pensar que l'usuari pot introduir entrades que no són normals, per això és recomanable posar-se en el pitjor cas.
- Els desbordaments de memòria són habituals, per això s'ha de provar d'introduir valors tan grans com sigui possible.

Ús d'interfície gràfica:
Les proves d'interfície gràfica d'usuari han d'incloure:

- Proves sobre finestres: icones de tancar, minimitzar…
- Proves sobre menús i ús de ratolí.
- Proves d'entrada de dades: quadre de textos, llistes desplegables…
- Proves de documentació i ajuda del programa.
- Altres.

Proves d'integració:
Un objectiu important de les proves d'integració és localitzar errors en les interfícies entre les diferents unitats. A més, les proves d'integració serveixen per validar que les parts de codi que ja han estat provades de forma independent continuïn funcionant correctament en ser integrades.

Tipus:
Prova d'integració ascendent:
Aquesta estratègia de desenvolupament de les proves d'integració començarà pels mòduls finals, els mòduls de més baix nivell, agrupant-los per les seves funcionalitats. Es crearà un mòdul impulsor que anirà efectuant crides als diferents mòduls a partir de les precondicions indicades i recollint els resultats de cada crida a cada funció.



Ventajas:

- **Ordre adequat**: primer s'avaluen els mòduls inferiors, que són els que acostumen a tenir el processament més complex, se'n solucionen els errors, i després es nodreix de dades la resta del sistema.
- **Més senzillesa**: les entrades per a les proves són més fàcils de crear, ja que els mòduls inferiors solen tenir funcions més específiques.
- **Millor observació dels resultats de les proves**: com que es comença pels mòduls inferiors, és més fàcil l'observació dels resultats de les proves.

Desventajas:

- **Anàlisi parcial**: fins que no es fa la crida al darrer mòdul no es valida el sistema com a tal.
- **Alt temps de dedicació**: caldrà dedicar molt de temps a implementar cada mòdul impulsor, que poden arribar a ser molts.

Prova d'integració incremental descendent:
Aquesta estratègia de desenvolupament de les proves d'integració començarà pel mòdul de control principal (el més important, el de més nivell). Una vegada validat, s'aniran integrant els altres mòduls que en depenen de forma progressiva, sense seguir una estratègia concreta, només tenint en compte que el nou mòdul incorporat a les proves tindrà ja validats tots els mòduls que el referencien. En funció del tipus de mòduls i del tipus de projecte, s'escollirà una seqüència o una altra a l'hora d'anar integrant mòduls, analitzant el problema concret. Les etapes de la integració descendent són:

- **Se selecciona el mòdul més important**, el de major nivell. Aquest mòdul farà d'impulsor. Caldrà escriure altres mòduls ficticis que simulin els mòduls que cridarà el principal.
- **A mesura que es van integrant mòduls, caldrà provar-los independentment** i de forma conjunta amb els altres mòduls ja provats. Una vegada s'ha finalitzat la prova, se substitueix el mòdul fictici creat pel real que s'ha integrat.
- Llavors caldrà escriure els mòduls ficticis subordinats que es necessitin per a la prova del nou

Ventajas:

- **Identificació de l'estructura**: permet veure l'estructura del sistema des d'un principi, facilitant l'elaboració de demostracions del seu funcionament.
- **Disseny descendent**: primer es defineixen les interfícies dels diferents subsistemes per després seguir amb les funcions específiques de cada un per separat.
- **Detecció més ràpida dels errors** que es trobin als mòduls superiors pel fet de detectar-se en una etapa inicial.

Desventajas:

- **Cost molt elevat**: caldrà implementar molts mòduls addicionals per oferir els mòduls ficticis a fi d'anar efectuant les proves.
- **Alta dificultat**: en voler fer una distribució de les proves del més genèric al més detallat, les dades que s'hauran d'utilitzar són difícils d'aconseguir, ja que són els mòduls de nivell més baix els que tindran els detalls.

Proves de càrrega i acceptació:
Les **proves de càrrega** són proves que tenen com a objectiu comprovar el rendiment i la integritat de l'aplicació ja acabada amb dades reals. Es tracta de simular l'entorn d'explotació de l'aplicació.
L'objectiu de la **prova d'acceptació** és obtenir l'aprovació del client sobre la qualitat de funcionament del sistema desenvolupat i provat.
Els desenvolupadors exerciten unes tècniques denominades **proves alfa** i **proves beta**.
Les **proves alfa** consisteixen a convidar el client que vingui a l'entorn de desenvolupament a provar el sistema. Es treballa en un entorn controlat i el client sempre té un expert a mà per ajudar-lo a usar el sistema i per analitzar els resultats.
Les **proves beta** vénen després de les proves alfa, i es desenvolupen en l'entorn del client, un entorn que és fora de control per al desenvolupador i l'equip de treball. Aquí el client es queda tot sol amb el producte i tracta de trobar els errors, dels quals informarà el desenvolupador.

Proves de sistema i de seguretat:
Les proves de sistema serviran per validar l'aplicació una vegada aquesta hagi estat integrada amb la resta del sistema de l'usuari. Encara que l'aplicació ja hagi estat validada de forma independent, a les proves de sistema es durà a terme una segona validació amb l'aplicació ja integrada en el seu entorn de treball real.
Proves:

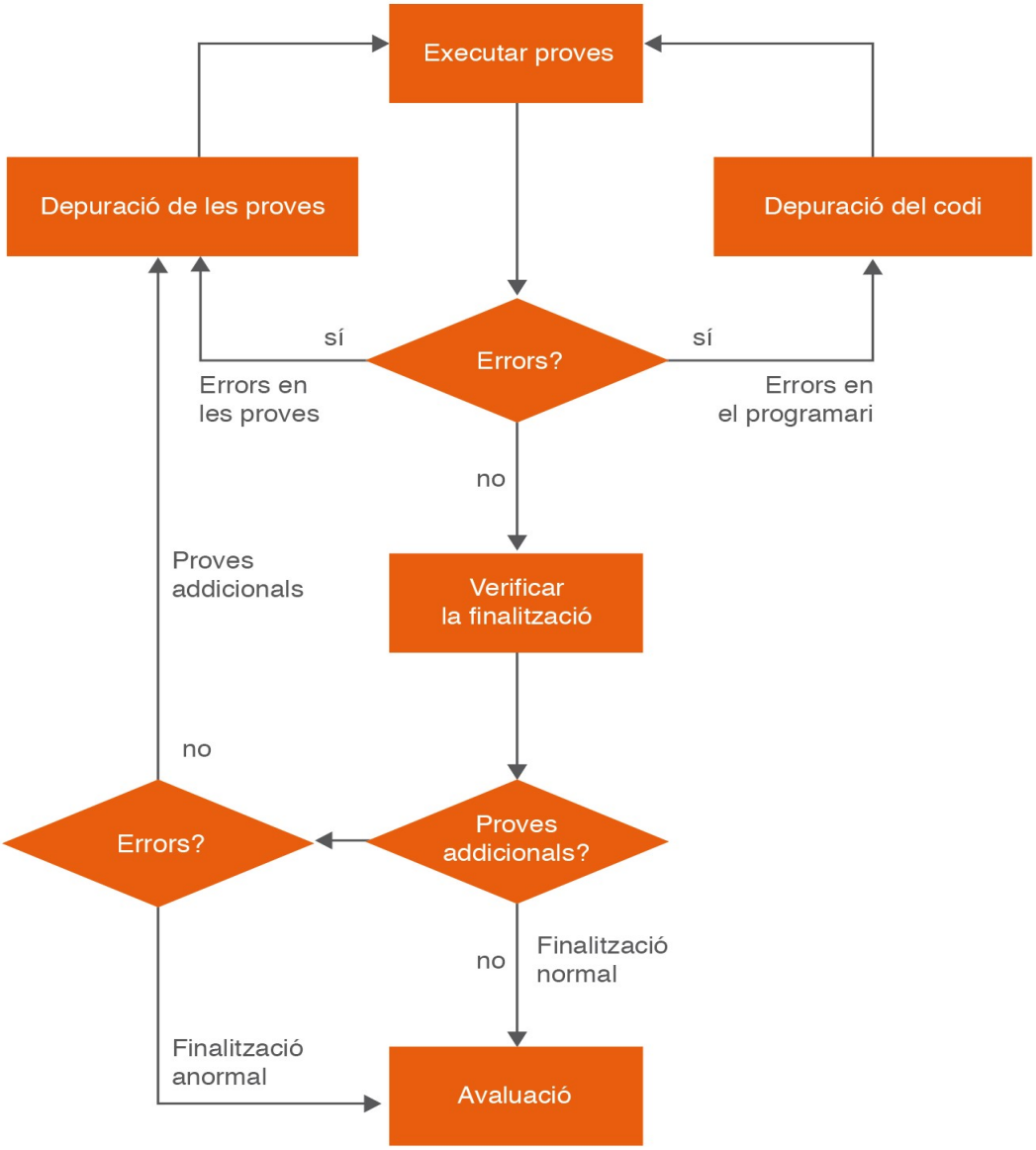
- **Proves de rendiment**: valoraran els temps de resposta de la nova aplicació, l'espai que ocuparà en disc, el flux de dades que generarà a través d'un canal de comunicació.
- **Proves de resistència**: valoraran la resistència de l'aplicació per a determinades situacions del sistema.
- **Proves de robustesa**: valoraran la capacitat de l'aplicació per suportar diverses entrades no correctes.
- **Proves de seguretat**: ajudaran a determinar els nivells de permisos dels usuaris, les operacions que podran dur a terme i les d'accés al sistema i a les dades.
- **Proves d'usabilitat**: determinaran la qualitat de l'experiència d'un usuari en la manera d'interactuar amb el sistema.
- **Proves d'instal·lació**: indicaran les operacions d'arrencada i d'actualització dels programaris.

Les **proves de validació** permeten comprovar si, efectivament, es compleixen els requisits proposats pel nostre sistema.
Les **proves de regressió** cerquen detectar possibles nous errors o problemes que puguin sortir en haver introduït canvis o millores en el programari.
Les **proves** "**de fum**" es fan servir per descriure la validació dels canvis de codi en el programari, abans que els canvis en el codi es registrin en la documentació del projecte.
Són proves d'execució ràpida i comproven les funcions bàsiques del programari.

Execució de les proves:
Pasos:

1. Execució de les proves.
2. Comprovació de si s'ha produït algun error en l'execució de les proves.
3. Si no hi ha hagut cap error:
    - Es verifica la finalització de les proves.
    - Es valida si calen proves addicionals.
        - Si es necessiten proves addicionals, cal validar que no existeixin condicions anormals. Si hi ha condicions anormals, es finalitza el procés de proves fent una avaluació del mateix procés; si no, caldrà depurar les proves.
        - Si no es necessiten proves addicionals, es durà a terme una finalització del procés de proves fent una avaluació del mateix procés.

4. En el cas d'haver trobat errors en l'execució de les proves, s'haurà de veure si aquests errors han estat deguts a:
    - Un defecte del programari. En aquest cas, l'execució de les proves ha complert el seu objectiu i caldrà depurar el codi de programació, localitzar el o els errors i solucionar-ho per tornar al punt inicial, en què es tornaran a executar les proves i es tornarà a validar si el canvi efectuat ha estat exitós.
    - Un defecte del disseny de les proves. En aquest cas, caldrà revisar les proves que s'han executat, depurant-les, localitzant el o els errors i solucionar-ho, per tenir unes proves correctes, sense errors, llestes per tornar al punt inicial i tornar a executar-les.



Finalització: avaluació i anàlisi d'errors:
El darrer pas dels procediments de proves serà la finalització del procés. Per poder donar-lo per tancat de forma exitosa, caldrà efectuar una avaluació i una anàlisi dels errors localitzats, tractats, corregits i reavaluats.

Depuració del codi Font:
La **depuració del codi font** consisteix a anar executant pas a pas el codi font, observant els estats intermedis de les variables i les dades implicades per facilitar la correcció d'errors.
Procedimientos:

- Identificar la casuística per poder reproduir l'error.
- Diagnosticar el problema.
- Solucionar l'error atacant el problema.
- Verificar la correcció de l'error i verificar que no s'han introduït nous errors a la resta del programari.

La **depuració del codi** és força útil també en l'execució de les proves d'integració, de sistema i d'acceptació.