



M5.UF2:

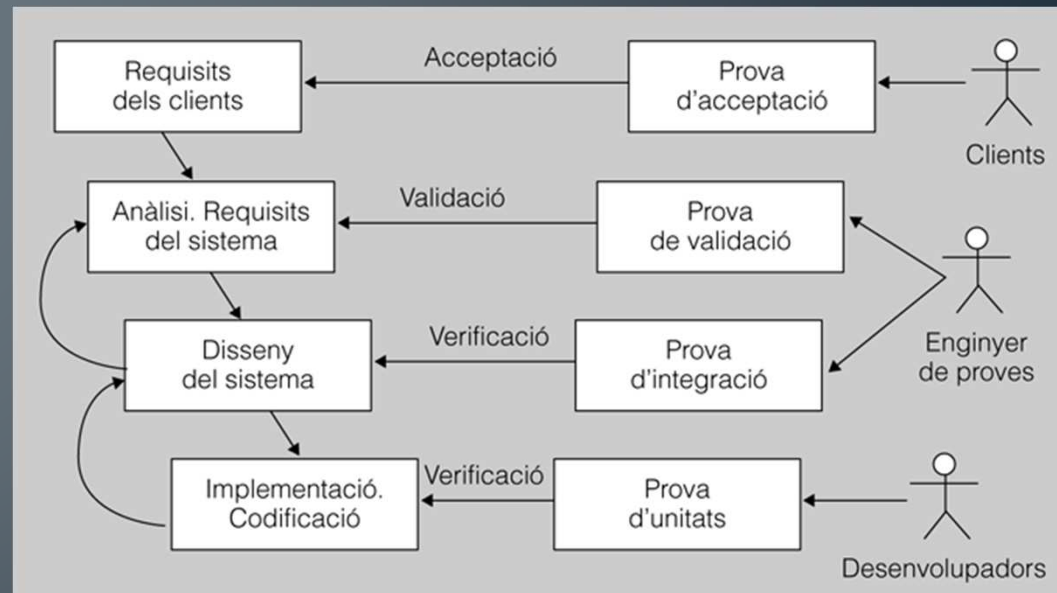
Proves de codi.

Introducció

- Una aplicació informàtica no pot arribar a les mans d'un usuari final amb errades, i menys si aquestes són prou visibles i clares com per haver estat detectades pels desenvolupadors. Es donaria una situació de manca de professionalitat i disminuiria la confiança per part dels usuaris.
- En un projecte de desenvolupament de programari està estipulat que es dedica entre un 30% i un 50% del cost de tot el projecte a la fase de proves. Amb aquesta dada ens podem adonar de la importància de les proves dins un projecte.

Fases desenvolupament d'un projecte

- Les fases de desenvolupament d'un projecte són:
 - Presa de requeriments.
 - Anàlisi de requeriments.
 - Disseny.
 - Desenvolupament.
 - Proves.
 - Finalització.
 - Transferència.



Exemple:

- Un cap de projecte, a l'hora de planificar les tasques de l'equip, estipula el disseny de les interfícies en 4 hores de feina per a un únic dissenyador.
- Si, una vegada executada aquesta tasca del projecte, la durada ha estat de 8 hores i s'han necessitat dos dissenyadors, la repercussió en el desenvolupament del projecte serà una desviació en temps i en cost, que potser es podrà compensar utilitzant menys recursos o menys temps en alguna tasca posterior.

Debugger

- Els depuradors (*debuggers*) són una aplicacions o eines permeten l'execució controlada d'un programa o un codi, seguint el comandament executat i localitzant els errors (*bugs*) que puguin contenir.
- Les persones implicades en la seva execució seran els desenvolupadors o programadors i l'enginyer de proves.

Contingut Pla de proves

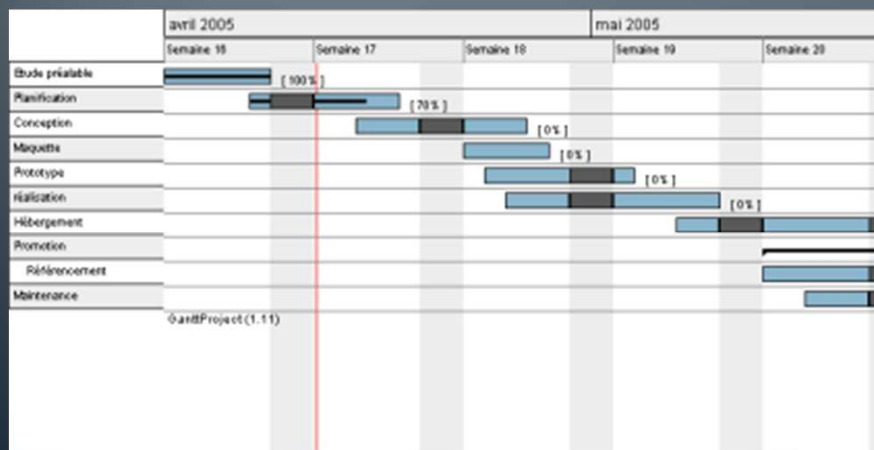
- Alguns dels continguts del pla de proves són:
 - **Identificador del pla de proves.** És l'identificador que s'assignarà al pla de proves. És important per poder identificar fàcilment quin abast té el pla de proves.
 - Per exemple, si es volen verificar les interfícies i procediments relacionats amb 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 caps blanca o de caps negra, així com les eines que s'utilitzaran o, fins i tot, el grau d'automatització de les proves.

Contingut Pla de 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 lílindar.
- **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 tasques 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.

Contingut Pla de proves

- **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.

[illegible]

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.
- Existeixen molts tipus de proves:
 - Estructurals o de capsa blanca
 - Funcionals o de capsa negra
 - D'integració
 - De càrrega i acceptació
 - De sistema i de seguretat
 - De regressió i de fum

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.

Tipus de proves:

- **Proves de capsa negra:** es tracta d'anar provant tot el projecte de forma independent, funció a funció. L'objectiu és validar que el codi compleix la funcionalitat definida.
- **Proves de capsa blanca:** es tracta de comprovar que totes les funcions encaixen sense problemes per complir els objectius del projecte, és a dir, han de garantir que totes les peces del projecte encaixen ajustant-se a les seves funcionalitats i als requeriments establerts.

Exemple:

- S'ha implementat en Java un cas de prova que valida el cost d'una matricula.
 - El mètode CasProva_CostMatricula calcularà el preu que haurà de pagar un alumne per matricular-se en diverses assignatures. La prova valida que el càcul de l'operació coincideixi amb el resultat esperat, fent ús de la instrucció assertTrue.

```
CONST PREU_CREDIT = 100€
public void CasProva_CostMatricula(){
    try {
        int credits = 0;
        float preu = 0;
        crèdits = CreditsAssignatura ("Sistemes informàtics");
        crèdits += CreditsAssignatura ("Programació");
        crèdits += CreditsAssigantura ("Accés a dades");
        preu = crèdits * PREU_CREDIT;
        assertTrue (preu==00€);
    }catch (Exception e) {Fail ("S'ha produït un error");}
}
```

Tipus de proves

- Tot seguit es presenta un resum d'aquests tipus de proves:
 - Proves unitàries.
 - Proves funcionals.
 - Proves d'integració
 - Proves de sistemes.
 - proves d'acceptació

Tipus de proves: Proves funcionals

- Són les encarregades de detectar els errors en la implementació dels requeriments d'usuari.
- Les duren 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: 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: Proves d'integració

- Es duren 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: Proves de sistemes

- La seva finalitat és detectar errors en l'assoliment dels requeriments.
- Les duren 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: Proves d'acceptació

- **Tipus de proves d'acceptació.** Els aspectes més importants d'aquestes proves són els següents:
- El seu objectiu és la validació o acceptació de l'aplicació per part dels usuaris.
- És per això que les duren a terme els verificadors i els analistes, però també els clients, que seran els usuaris finals de les aplicacions.
- Aquestes proves es duren a terme una vegada finalitzada la fase de desenvolupament, i 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.

Tipus de proves: Proves unitàries

- Les proves unitàries, també conegudes com a proves de components, són les proves que es faran a més baix nivell, sobre els mòduls o components més petits del codi font del projecte informàtic.
- Aquestes proves poden desenvolupar-se sota dos enfocaments:
- L'enfocament estructural és la part de les proves unitàries encarregades de l'estructura interna del codi font, des de qual s'analitzen tots els possibles camins.
- L'enfocament funcional (o proves de **capsa negra**) és la part de les proves unitàries encarregades del funcionament correcte de les funcionalitats del programari.

Proves Caixa Blanca

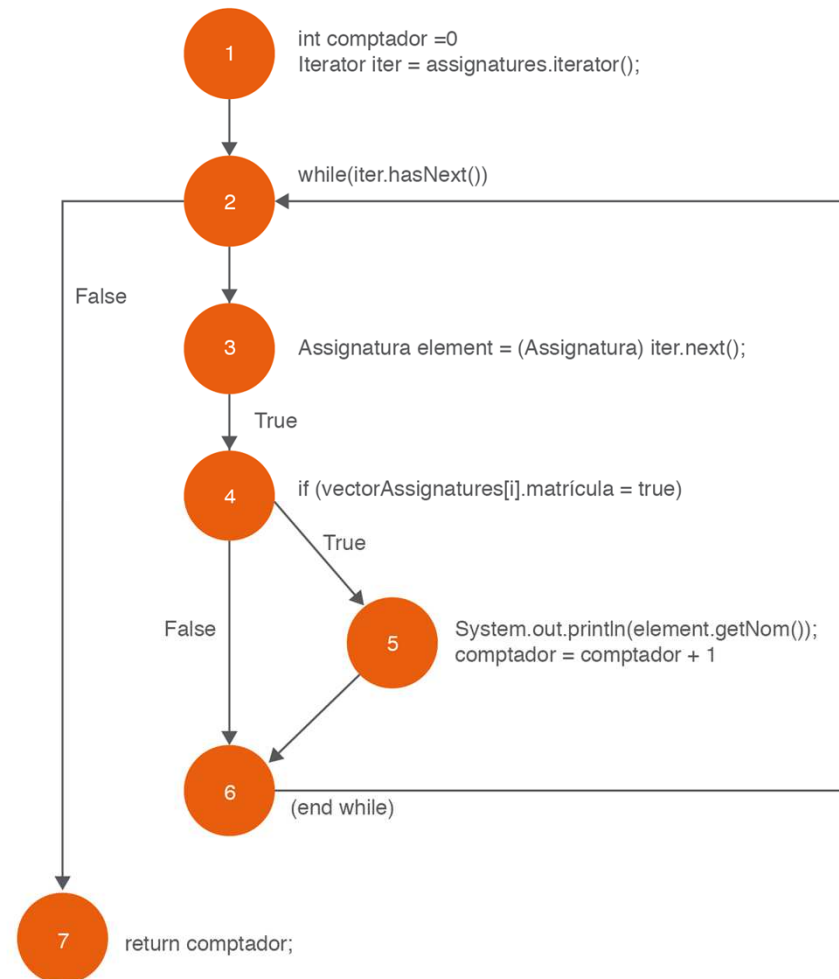
- **Les proves de capsa blanca** se centren en la implementació dels programes per escollir els casos de prova. L'ideal seria cercar casos de prova que recorreguessin tots els camins possibles del flux de control del programa. Aquestes proves se centren en l'estructura interna del programa, tot analitzant els camins d'execució.

Les proves de capsa blanca permetran recórrer tots els possibles camins del codi i veure què succeeix en cada cas possible. Es provarà què ocorre amb les condicions i els bucles que s'executen. Les proves es duren a terme amb dades que garanteixin que han tingut lloc totes les combinacions possibles. Per decidir quins valors hauran de prendre aquestes dades és necessari saber com s'ha desenvolupat el codi, tot cercant que no quedi cap racó sense revisar.

Exemple: Caixa Blanca

- La funció LlistatAssignatures mostra, per pantalla, les assignatures que estan disponibles per tal que els alumnes es puguin matricular, i retorna un valor numèric corresponent al nombre d'assignatures disponibles.

```
public float LlistatAssignatures(ArrayList assignatures)
{
    int comptador= 0;
    Iterator iter = assignatures.iterator();
    while (iter.hasNext())
    {
        Assignatura element = (Assignatura) iter.next();
        if (element.getDisponible() == true)
        {
            System.out.println(element.getNom());
            comptador = comptador + 1
        }
    }
    return comptador;
}
```



Complexitat ciclomàtica

- L'estratègia de cobertura de flux de control requereix dissenyar casos de prova suficients per recórrer tota la lògica del programa.
- Es pot saber quants casos de prova cal crear i executar? Com es calcula?
 - El matemàtic Thomas J. McCabe va anomenar complexitat ciclomàtica (CC) al nombre de camins independents d'un diagrama de flux, i va proposar la fórmula següent per calcular-la:

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

- La complexitat ciclomàtica del graf de l'exemple anterior, proporciona el nombre màxim de camins linealment independents.
- Els nodes que intervenen són 1, 2, 3, 4, 5, 6 i 7, i les branques són les línies que uneixen els nodes, que són un total de 8.

$$\text{complexitat ciclomàtica CC} = 8 - 7 + 2 = 3$$

Complexitat ciclomàtica

- Això significa que s'hauran de dissenyar tres casos de prova. Així, els tres recorreguts que s'haurien de tenir en compte són:
 - Camí 1: 1 – 2 – 3 – 4 – 5 – 6 – 2 - 7
 - Camí 2: 1 – 2 – 3 – 4 – 6 – 2 - 7
 - Camí 3: 1 – 2 – 7
- Hi ha tres camins a recórrer. Per a cadascun d'ells serà necessari un vector amb una característiques concretes:
- Per recórrer el **camí 1** serà necessari un vector d'assignatures que contingui com a mínim una assignatura que estigui disponible.
- Per recórrer el **camí 2** serà necessari un vector d'assignatures que contingui com a mínim una assignatura que no estigui disponible.
- Per recórrer el **camí 3** serà necessari un vector d'assignatures buit.

Complexitat ciclomàtica

Camí 1

Assignatura	
Id	123
nom	Bases de dades
hores	231
crèdits	12
disponible	true

Camí 2

Assignatura	
Id	123
nom	Formació en centres de treball
hores	317
crèdits	18
disponible	false

Camí 3

Assignatura	
Id	
nom	
hores	
crèdits	
disponible	

D'aquesta manera, el resultat de les proves serien:

Entrada (Assignatures1)

Sortida esperada: 1

Sortida real: 1

Camí seguit: 1.

Entrada (Assignatures2)

Sortida esperada: 1

Sortida real: 1

Camí seguit: 2.

Entrada (Assignatures3)

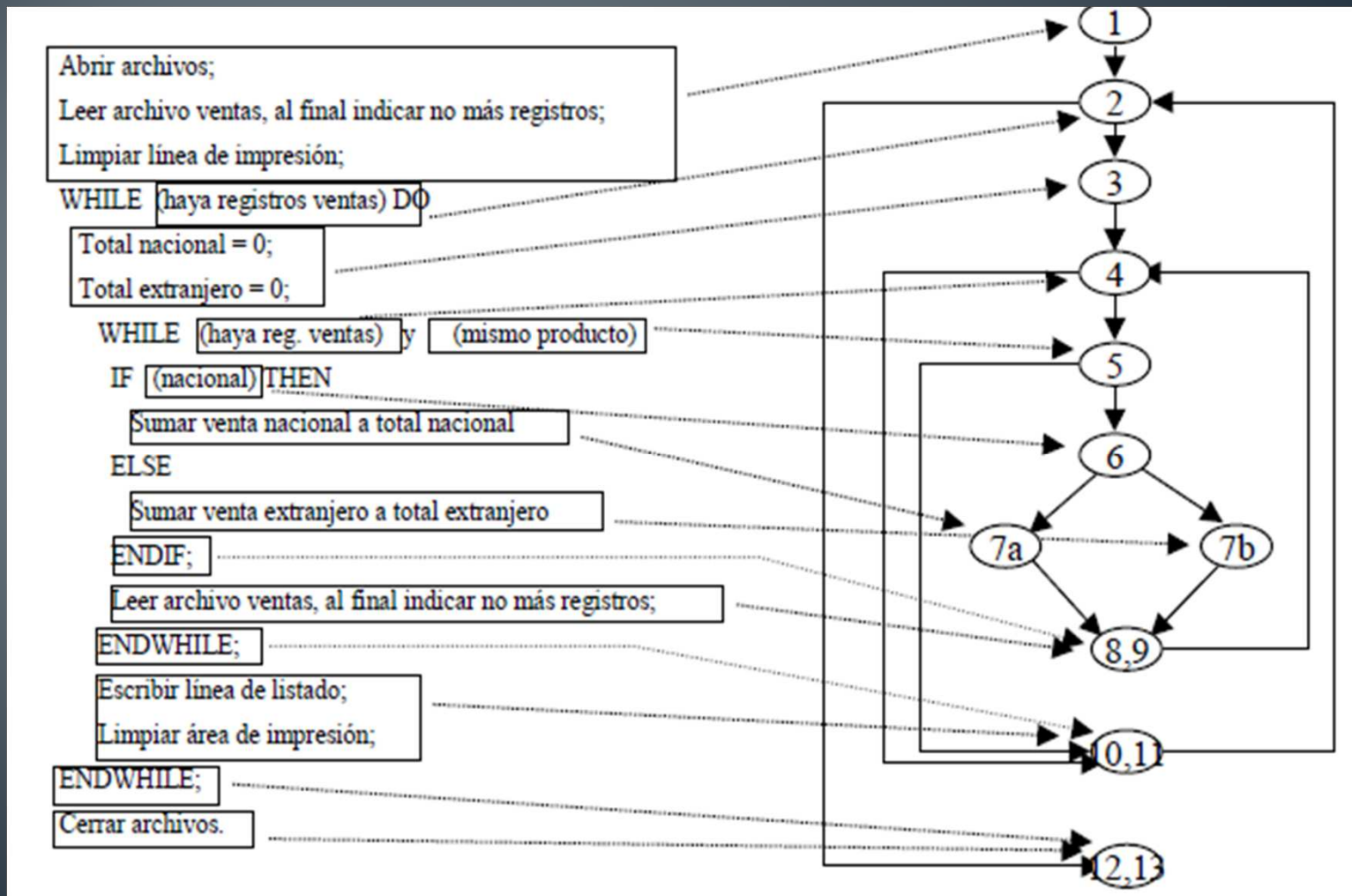
Sortida esperada: 0

Sortida real: 0

Camí seguit: 3.

Una vegada executat el joc de prova de caps blanca, es pot afirmar que han estat superades satisfactòriament

Example:



Proves Caixa 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.
- Cal seguir els passos següents:
- **Identificar les condicions**, restriccions o continguts de les entrades i les sortides.
- **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**.
- **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

Exemple:

- Per exemple, es volen definir les proves de capsa negra per a una funció que retorna el nom del mes a partir del seu valor numèric.

```
String nom;  
nom = NomDelMes(3);  
El valor del nom serà Març.
```

- Caldrà identificar tres classes d'equivalències:

```
..., -02, -01, 00 valors invàlids  
01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12 valors vàlids  
13, 14, 15, ... valors invàlids.
```

Recomanacions

- A l'hora d'escollir els representants de cada classe se seguiran les recomanacions següents:
- 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.
- També es pot aprofitar l'experiència prèvia. Hi ha una sèrie d'errors que es repeteixen molt en els programes, i podria ser una bona estratègia utilitzar casos de prova que se centrin a buscar aquests errors. D'aquesta manera, es millorarà l'elecció dels representants de les classes d'equivalència:
- El valor zero sol provocar errors, per exemple, una divisió per zero bloqueja el programa. Si es té la possibilitat d'introduir zeros a l'entrada, s'ha d'escollir en els casos de prova.
- 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.

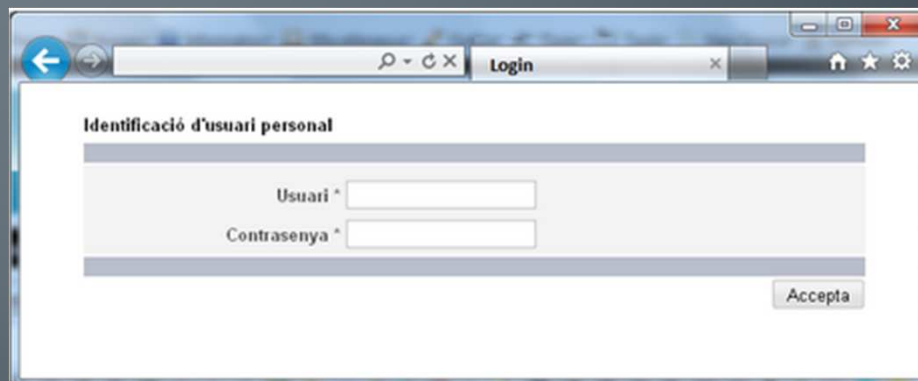
Exemple 2:

Funció Buscar (DNI as string, vectMatricula de Matricules) retorna Matricula

- Per tal de simplificar el joc de proves, el nombre de matrícules que admet la funció Buscar és 10.
- Per a la variable DNI hem de tenir en consideració que està formada de vuit xifres numèriques i una lletra(en aquest exercici no es valida el valor de la lletra):
 - 00000001A Prova vàlida.
 - Null Prova invàlida, el DNI no té valor.
 - 00000001 Prova invàlida, el DNI té un format incorrecte, falta la lletra.
 - 00000001AA Prova invàlida, el DNI té un format incorrecte.
 - AAAAAAAAAA Prova invàlida, el DNI té un format incorrecte.
- Pel vector vectMatricula hem de tenir en consideració que pot contenir de 0 a 10 matrícules:
 - []. Prova vàlida, vector buit.
 - [matrícula1]. Prova vàlida, vector amb una matrícula.
 - [matrícula1, matrícula2, matrícula3, ... matrícula10]. Prova vàlida, vector amb un nombre de matrícules entre 0 i 10.
 - [matrícula1, matrícula2, matrícula3, ... matrícula10, matrícula11]. Prova invàlida, vector amb un nombre de matrícules superior a 10.
- Per a la sortida:
 - Alumne.DNI = DNI. Prova vàlida. Classe amb les dades d'un alumne.
 - Classe buida. Prova vàlida. S'ha buscat el DNI d'una persona que no és alumne.
 - Alumne.DNI <> DNI. Prova invàlida. Classe amb les dades d'un altre alumne.
 - Classe buida. Prova invàlida. S'ha buscat el DNI d'un alumne i no s'ha trobat.

Exemple 3:

- Finestra que controla l'accés al sistema de matriculació dels alumnes mitjançant la introducció d'un nom d'usuari i una clau (*password*). El sistema comprova si hi ha un compte amb el nom i clau especificat i, si és així, es dóna permís per entrar. Si hi ha un compte amb aquest nom i la clau és incorrecta, permet tornar a introduir la clau fins a un màxim de tres vegades.



The screenshot shows a web browser window with a single tab titled 'Login'. The address bar is empty. The main content area displays a form titled 'Identificació d'usuari personal'. The form contains two input fields: 'Usuari *' and 'Contrasenya *', both followed by asterisks indicating they are required. Below the input fields is an 'Accepta' button. The browser's address bar shows a search icon, a refresh icon, and a close icon.

Exemple 3:

- Tot seguit es mostren alguns dels casos de prova que es podrien utilitzar amb aquest programa:
- **Cas de prova 1:**
 - Entrada: usuari correcte i contrasenya correcta. Prémer botó d'accedir al sistema.
 - Condicions d'execució: en la taula existeix aquest usuari amb la contrasenya i amb un intent fallat (nombre inferior a 3).
 - Resultat esperat: donar accés al sistema i reflectir que el nombre d'intents per a l'usuari correcte és zero en la taula USUARI (compte, contrasenya, nre. intents).
- **Cas de prova 2:**
 - Entrada: usuari incorrecte i contrasenya correcta. Prémer botó d'accedir al sistema.
 - Condicions d'execució: en la taula no existeix aquest usuari amb aquesta contrasenya.
 - Resultat esperat: no donar accés al sistema.