Programa de Gestió de Matrícula i Expedients de Notes
Guia de Treball
Segona Pràctica

La modularitat ben entesa i posada correctament a la pràctica ens proporciona les eines per un equip de persones treballar conjuntament sobre el mateix programa definint els límits entre les persones segons el problema en concret a tractar. Permet definir l'estructura per treballar el màxim d'independent (sense trepitjar-se ni duplicar feina) però sumant esforços efectivament. Això es tradueix en una forma de treballar. Per tant saber programar no tan sols és aprendre el vocabulari d'un llenguatge nou i els conceptes de programació associats sinó que també és adquirir mètode en com analitzar un problema i en com estructurar i implementar la solució. És a dir, programar bé inclou una forma de treballar competències transversals com planificació i treball en equip. És per això que es proporciona aquest document de guia de treball per inferir en l'actitud, hàbits i manera de pensar i treballar un problema de programació, que si s'aconsegueix es consolida una mentalitat per abordar problemes generals i complexes de qualsevol tipus.

La modularitat no es pot entendre i aplicar si no és en el context d'un problema d'una certa envergadura que necessiti d'organització i estructura. És per això que la pràctica s'emmarca amb un context gran del què hauria de fer un programa complert de matricula i gestió d'expedients, encara que la tasca a realitzar sigui molt més reduïda.

1 Consideracions de disseny

És molt important per construir un programa entenedor i flexible aïllar les funcionalitats que es poden definir autònomament i definir quines dades necessiten per poder-se realitzar i quines dades generen i on cal guardar-les. D'aquest anàlisi en resulta la construcció de les funcions amb els seus paràmetres (d'entrada i sortida) i les seves relacions (el què hem anomenat a teoria el mapa conceptual). Aquest anàlisi s'obté fruit d'estudiar el problema, i per tant es pot construir el mapa conceptual abans de començar la implementació. Això obliga a pensar i entendre millor el problema abans de començar la implementació, cosa que fa que es dissenyi molt millor el programa. El mapa conceptual fet a priori també es converteix en una eina de treball en equip per distribuir la feina entre els membres de l'equip amb instruccions clares per cada membre i amb interrelacions ben definides i acotades. Si la implementació necessita canviar aquest disseny cal actualitzar el mapa conceptual i notificar tot l'equip per decidir quina és la millor manera d'incorporar el canvi.

Segons les funcionalitats que ha de proporcionar aquesta pràctica podria tenir sentit definir les següents funcions:

- Funció d'inicialitzar: per inicialitzar el programa, separadament per inicialitzar i/o reinicialitzar vectors concrets o una posició/element
- Funció de cerca: que identifica un element dins un vector d'estructures (d'alumnes/expedients o assignatures)
- Funció d'ordenació : que ordena un vector d'elements segons un criteri
- Funcions afegir i eliminar element a un vector d'estructures (d'alumnes/expedients o assignatures)
- Funció inicialitzar/crear un element (alumne/assignatura)
- Funcions menú i submenús, i una funció per cada opció.
- Funció que calcula/crea un número nou de NIA o codi d'assignatura...
- Funció que verifica per separat cada dada entrada per l'usuari
- Funció que verifica les dades entrades via fitxer
- Funcions de manipulació d'un expedient: calcular nota promig, calcular estadístiques resum, saber si passa permanència, que verifiqui si un expedient es pot matricular a una assignatura concreta, saber si ha acabat i es pot expedir títol...

Exactament quines funcions es defineix depèn de quina estructura es vol donar al programa. Però certament és important assegurar que el codi no és un seguit infinit de línies de codi sense estructura. Les funcions obliguen a identificar funcionalitats i a definir les relacions entre aquestes funcionalitats. Per tant les funcions defineixen l'estructura del programa i la manera d'entendre i resoldre el problema que aborda. Com més estructurat es fa més fàcil és de fer-ho entendre també als altres i de poder-ho reutilitzar. Com a regla general es pot dir que una funció més gran que una pàgina (de línies de codi) és possiblement massa codi per no separar-lo en funcions. O si una funció fa més d'una cosa és segurament millor identificar les diferents coses que fa, aïllar-les i definir una funció per cada cosa (encara que cada funció sigui només una línia) i relacionar totes aquestes funcionalitats amb les crides o una funció que

les lligui. És difícil que dues funcionalitats sempre vagin juntes i mantenir-les juntes limitarà la reutilització de la funció que les implementi. Pensar i dedicar temps a la definició de funcions i definir el pas de paràmetres i les crides és una part molt important del disseny que permet definir el mapa conceptual sense haver començat encara a implementar.

La modularitat complerta no tan sols es refereix en la separació de codi en funcions sinó també en l'estructuració d'aquestes funcions en fitxers o mòduls. Un mòdul conté totes les funcions i estructures de dades necessàries per implementar una funcionalitat. Per exemple en aquesta pràctica podem tenir el mòdul de cerca-ordenació, d'entrada-sortida, de pla-d'estudis o altres. Es demana que aquest programa tingui un disseny modular (veure enunciat).

Entendre bé el problema també requereix analitzar-lo per saber què cal obtenir en cada moment del programa. Això proporciona un marc per poder comprovar que el programa fa el què necessitem que faci. D'aquest anàlisi se'n pot definir el pla de proves a fer servir per la verificació del programa. El banc de proves definit en la fase de disseny (abans de la implementació) contempla les condicions del problema a resoldre. Un cop la implementació estigui avançada es pot ampliar per incorporar condicions particulars de la implementació (assegurar que totes les proves recorren almenys una vegada totes les línies de codi).

És per això que se us demana que definiu els següents elements abans de començar la implementació i que la contrasteu amb algun professor:

- Estructura de dades definides en el programa
- Variables principals (en el nostre cas restringim que no poden ser globals però són la base de dades del programa)
- Mapa conceptual (del futur) programa
- Banc de proves de la fase de disseny

Es recomana contrastar aquest disseny amb algun professor per anar sobre segur amb els pilars del disseny del programa. Aquesta consulta la podeu fer tan aviat com tingueu el vostre disseny pensat, però hauria d'estar feta abans de la primera sessió de pràctiques d'aquesta pràctica (i com a molt tard en la mateixa sessió). És a dir, la implementació (amb el disseny pensat i contrastat) hauria d'estar començada en aquesta data. Per això es demana una preentrega de la pràctica amb aquests elements de disseny a entregar el dia abans de la primera sessió de pràctiques (veure entrega en l'enunciat).

2 Metodologies en la programació

2.1 Programació progressiva o incremental

Aquesta secció explica com fer un disseny progressiu de la funció que implementi la verificació de grau. Es dona a tall d'exemple però en realitat totes les parts d'un problema gran es poden anar desenvolupant amb aquesta tècnica incremental. Es dóna aquest exemple per mostrar com s'ha de pensar un disseny incremental.

Si volem implantar un pla docent en el nostre programa de matricula cal incorporar les polítiques de grau i d'escola al procés de matricula i al entrar notes. És a dir, la matricula no pot ser només assignatures que existeixen en el fitxer d'assignatures sinó que han de ser les que toquen segons totes aquestes polítiques. Així implantar la matricula voldria dir tenir una funció que donat un expedient ens retorni la llista d'assignatures que es pot matricular en la següent convocatòria. Potser al pensar aquesta funció s'arriba a la conclusió que té sentit fer una funció que per cada expedient i assignatura del grau que cursa confirma si es pot matricular o no en la següent convocatòria.

Una visió incremental del programa podria ser que inicialment no es filtra la llista d'assignatures que es pot matricular un alumne i per tant quan se li mostra la llista se li permet totes. En previsió de la definició d'aquest filtratge podríem com a primer pas fer la funció i definir els paràmetres però no implantar cap política dins, per tant que retornés igualment tota la llista d'assignatures del grau en qüestió. La següent fase incremental seria omplir de contingut aquesta funció i realment aplicar una política que filtrés les assignatures que es pot matricular. Es pot començar amb un filtre simple i anar complicant fins tenir tota la política dissenyada/implementada.

Al entrar notes, també ens hauria de notificar si un estudiant ha complert tots els requisits per obtenir el grau, o si ha incomplert mínims per continuar el grau. Així podríem definir/pensar una funció que donat un expedient acadèmic ens indiqués si ha complert els requisits per expedir-li el títol de grau. Aquesta funció retornaria si/no o 0/1. Podríem ampliar-la perquè retornés -1 en cas de que no es pogués tornar a matricular degut a no superar la regla de permanència o no superar alguna assignatura amb el nombre màxim permès de convocatòries. El fet de determinar si es passa o no, implica saber per cada assignatura si s'ha complert o és on falla la política. Per tant és un pas incremental més fer que la funció no tan sols retorni el -1/0/1 sinó que retorni la llista d'assignatures que fan que no es pugui matricular o estiguin pendents de superar per acabar.

Si mirem la definició de les dues funcions anteriors veiem que són molt semblants i és un petit pas incremental fer que la mateixa funció ens permeti decidir la llista d'assignatures que s'ha de mostrar a l'usuari per la matricula, i determinar quants anys li falta a l'alumne per acabar quan s'entri les notes.

Aquest disseny progressiu va concentrant ordenadament funcionalitats dintre la mateixa funció fent que cada vegada sigui més complexa verificar el seu correcte funcionament. Aquesta concentració estructura el problema i evita duplicitat de codi, perquè fer dues coses similars separades produirà parts de codi similars.

Al construir funcions denses, es fa cada vegada més imprescindible un mètode per identificar i comprovar tots els casos a considerar. És per això que és important dissenyar el pla de proves conjuntament amb el disseny per concretar aquest pla amb un conjunt d'execucions i tests a realitzar amb la implementació i verificació. O sigui el disseny del pla de proves documenta la comprensió del disseny en forma de comprovacions de la implementació.

En aquest cas el banc de proves de la funció del pla d'estudis ha de contemplar per exemple les següents situacions: 1) un expedient que no compleix la regla de permanència 2) un expedient que té completat el grau 3) un expedient que se li bloqueja el matricular-se a assignatures del següent any perquè no té completat l'any anterior 4) i així successivament. Cal un mètode rigorós i exhaustiu per detallar totes les condicions. En la següent secció abordem el disseny del pla de proves, però de la funció de lectura de fitxer.

2.2 Pla de proves pel processat de fitxers

Com hem dit verificar una funció densa i compacta requereix de rigorositat i aplicació d'un procés metòdic que en el nostre cas ho canalitzem amb el banc de proves. Mostrem el procés de com pensar i construir el banc de proves del procés (funció) de carregar fitxer encara que no el dissenyem completament. Podeu consultar [CP-v1] per veure un exemple de banc de proves per la funció notes.

El format dels fitxers de dades incorporen varis elements que flexibilitzen el seu ús, però a la vegada introdueixen variants en el format que fa que el format en sí sigui una família de possibles opcions. És aconsellable dissenyar i implementar aquestes funcions incrementalment, començant per un format simple i rígid i anar introduint un a un els elements que ho flexibilitzen, aplicant així el mètode incremental explicat en l'apartat anterior (secció 2.1). L'exemple del disseny progressiu ja es dona en aquella secció i aquí explorem el disseny del pla de proves.

El tàndem carregar-gravar són dues operacions de la mateixa moneda que comparteixen un format (comú) de fitxer que està representat en l'estructura de dades que l'emmagatzema a memòria. L'operació de carregar és el procés d'interpretació del fitxer per extreure els elements d'informació a guardar a memòria dins l'estructura de dades. L'operació de gravar és el procés invers de traducció de les dades emmagatzemades a memòria al format text concret de fitxer. La funció carregar és inherentment més complexa perquè s'han de contemplar més situacions imprevistes. En canvi l'operació de gravar és més simple perquè ens podem restringir a un format de sortida rígid i simple sense perdre ni informació ni funcionalitats en l'aplicació resultant.

Així doncs un pla de treball incremental pot centrar-se primer en la funció carregar i del fitxer assignatures que té un format més simple que el d'expedients. Assumir inicialment que només es té les columnes de dades sense comentaris ni capçalera de columnes. Preparar una versió reduïda de fitxer assignatures amb un llistat petit d'assignatures que contingui els diferents casos a contemplar entre les assignatures. Considerar inicialment que no s'aplica cap verificació de les dades. En aquest cas es pot preparar un únic fitxer de proves que inclogui poques assignatures perquè si en llegeix una sabrà llegir les següents. Potser l'única condició a considerar sigui el cas de que la llista d'assignatures en el fitxer sigui més llarga que el nombre de posicions del vector d'assignatures considerada en l'estructura de dades. Caldria doncs fer dues proves una amb un nombre més petit d'assignatures totals en el fitxer i una altra amb un nombre més gran per assegurar que detecta el problema i actua en conseqüència segons definició.

És aconsellable per un programador novell, dissenyar, implementar i verificar cada pas progressiu i no estendre la implementació fins que el pas actual funcioni correctament i s'hagi verificat. Un cop funciona podem incrementar la funcionalitat per exemple introduint la verificació de les dades, i per tant verificant si hi ha dues assignatures amb el mateix codi, o verificant si l'assignatura ja carregada existeix a memòria actual o no. Això implica, determinar els casos a contemplar i dissenyar i preparar els tests addicional per la verificació d'aquests casos. En aquest cas seria tenir un fitxer assignatures de prova més que inclogués dues assignatures amb el mateix codi. També caldria comprovar per exemple què passa si s'intenta carregar el mateix fitxer dues vegades (duplicitat d'assignatures però amb la informació coherent). Afegir el cas d'un fitxer amb dues assignatures diferents però amb mateix codi, i el cas d'afegir un fitxer amb una assignatura amb un codi que ja s'ha entrat manualment i és diferent, i al revés, el cas d'entrar manualment una assignatura amb un codi ja existent a memòria. La llista d'aquestes execucions i corresponents fitxers d'entrada i sortida seria el banc de proves per aquesta segona versió incremental. El següent pas podria ser permetre tenir una línia de comentari de capçalera al fitxer i fer l'ampliació del disseny i banc de proves i implementació corresponent. El següents passos incrementals podrien ser: permetre una capçalera de fitxer amb vàries línies de comentari, i afegir una per una totes els elements d'especificació del format del fitxer.

Seria possible fer el disseny progressiu i anar avançant en el disseny abans d'iniciar la programació. Però en aquest cas es començaria a programar les funcions quan ja tenen una concentració de funcionalitat elevada i per tan una complexitat significativa. Això és molt eficient en programadors experts, però és contraproduent en programadors novells ja que la complexitat és deseguida inabordable. Es recomana doncs desgranar l'objectiu d'una funció amb elements atòmics incrementals molt fàcils d'implementar. És molt més eficient pensar com definir aquests fites intermèdies tan petites que intentar fer funcionar una cosa complexa sense mètode ni experiència per fer-ho. Dit d'una altra manera és perd molt més temps estan bloquejat en una situació en que el programa no funciona i no saber trobar l'error que en la lentitud de la metodologia de pensar increments tant petits que simplifiquen el problema. Això és cert només si

cada passet que es fa és ferm i no tirem mai enrere. Això només s'assegura si la verificació és de qualitat i tot els errors de verificació en una fase són deguts exclusivament a la nova funcionalitat afegida i no en funcionalitats implementades en fases anteriors. Per això l'èmfasi en la robustesa i verificació de codi per assegurar que quan es doni un codi per bo, realment ho sigui.

Quan detectem que l'error no prové de l'increment actual, hem perdut el context que ens acota les possibles fonts d'error i cal començar de zero tota la verificació per la cerca de l'error. És desitjable minimitzar les vegades que això passa, però és quasi bé impossible evitar-ho completament. Per aquests casos, és important mantenir la progressió del banc de proves i tests a realitzar per poder-lo revisar i tornar-lo a aplicar sense tenir-lo que redissenyar i reimplementar des de zero. En aquest punt és quan s'amortitza àmpliament el temps dedicat a definir un sistema rigorós i metòdic de verificació de codi. I això marca la diferència en la seguretat i actitud del programador davant la situació ja que se sent segur en què pot abordar i superar aquesta fase. Per una banda disposar d'un mètode ens dona la tranquil·litat de tenir el control i la seguretat de tenir una alta probabilitat d'èxit. Mentre que per l'altra banda la falta de mètode ens porta a la incertesa i desesperació davant el problema.

L'eina per una verificació de qualitat no és tan sols el definir i implantar el banc de proves complert per totes les funcionalitats i casos que ens permeten detectar el problema. Sinó també incorporar elements en el programa que ens ajudin a entendre l'error. Això s'aconsegueix exterioritzant (via missatges ben definits) el què realment fa el programa per poder revisar molt detalladament si és el què cal que faci. Dins d'aquest conjunt d'eines tenim les verificacions internes en el programa i el disseny de traces que es puguin activar i desactivar.

Si volem que un programa significativament gran (el tamany és relatiu a l'experiència del programador) tingui una llarga vida d'ús i reusos (extensions) és **important preveure tots els imprevistos possibles**. I implementar com a mínim la notificació d'aquests imprevistos dins el programa encara que només sigui per informar l'usuari quan es donin. Així el programa no peta quan aquests imprevistos passen a ser situacions reals (en futures extensions per exemple) i per tant el propi programa ens avisa que es donen. Per exemple, aquesta aplicació suposa l'existència dels fitxers de dades al directori dades. Un usuari pot tenir disponible aquests fitxers però no en el directori adequat. Si el programa no comprova el resultat de la funció obrir per confirmar que s'ha obert correctament el fitxer, el programa pot donar problemes més endavant en l'execució i el programador pot tardar molt de temps a relacionar aquests afectes en l'error d'obertura. Altres verificacions possibles per evitar mals menors en aquest programa podrien ser: confirmar sempre que al accedir una posició d'un vector aquesta posició es vàlida segons la dimensió del vector (assegurar que la comprovació inclou la mateixa expressió que l'utilitzada en l'accés del vector).....

Les traces no són més que printfs que ens permeten imprimir informació del detall de l'execució per fer-ne un seguiment manual. Com detallen l'operativa d'execució, la quantitat de línies pot ser important i per tant la presentació d'aquesta informació (encara que sigui per pantalla o per fitxer) cal que sigui ordenada i tenir una estructura pensada per poder distingir i identificar els elements que es vulguin analitzar. Així doncs en el disseny de les traces es construeix un representació de la informació útil i fàcil de seguir del funcionament del programa. No és tasca simple però si molt útil. I moltes vegades permet anar més enllà de la verificació del programa per convertir-se en una eina d'anàlisi del comportament del problema que implementa. Així doncs és necessari poder mantenir aquesta eina integrada amb el programa sense afectar l'execució definitiva del programa però disponible pel seu desenvolupament. L'activació/desactivació dels missatges via directives de preprocessor permet fer-les desaparèixer a efectes d'execució en l'executable final però mantenint-les disponibles en el codi per quan convingui en la programació. Un exemple per aquest programa de com utilitzar les traces podria ser en el seguiment de l'execució del programa escrivint cada vegada que entra en una funció el nom de la funció. La traça complerta seria el flux d'execució i per exemple podria permetre començar amb una estructura de programa amb funcions buides i anar omplint cada funció segons les funcionalitats esperades. Inicialment el programa no faria res i visualitzaria el flux però a la llarga mostraria el detall de cada part del programa.

És rellevant ressaltar que la implementació de traces pot incorporar un filtre per només activar part del missatges pel seguiment parcial i concret d'una part del programa. Per exemple ens pot interessar comprovar la informació de tots els missatges rellevants per saber si el fitxer es carrega bé, però desactivar els missatges que mostren el detall d'operació de l'entrada manual de dades i de funcionalitats de plans d'estudi o càlcul d'expedients.

2.3 Robustesa

Un pla de proves exhaustiu determina les diferents condicions d'operació del programa. Indirectament, ajuda a definir els límits d'operació del programa. Un **programa robust és aquell que té identificat exactament els seus límits** i els té integrats dins el programa perquè no peti i notifiqui.

Fer un programa robust ajuda a que no es pengi el programa si els usuaris el manipulen malament. Però també serveix al programador en el seu procés de verificació que el programa li diu quan es dona les condicions que estan previstes no ser correctes. L'objectiu sobretot en aquesta assignatura és aquest segon en el sentit que invertir temps en bones pràctiques ens ajuda a millorar en l'aprenentatge dels conceptes i en la velocitat en què s'adquireix fluïdesa i auto-control. Això és important perquè es retroalimenta.

L'objectiu és doncs, que adquiriu els hàbits d'anar afegint les verificacions a mesura que van sorgint al fer créixer al codi (pel vostre propi ús), i no afegir-ho (al final) només per complir un requeriment de la pràctica.

2.3.1 Robustesa de la Configuració

La robustesa de configuració s'aconsegueix inclouen les verificacions de les constants que es defineixin dins el programa, acotant i documentant clarament els valors que cada una pot prendre, i implementant la detecció, notificació i parada controlada del programa sempre que el programa arrenquí amb algú valor incorrecte.

Aquesta revisió com és de configuració cal fer-la al iniciar el programa i ja no continuar si no és correcte. És més útil permetre que revisi totes les constants i mostri totes les inconsistències de configuració abans de parar, que no pas parar a la primera inconsistència. La notificació en cas d'error ha de ser clara indicant tots els casos incoherents i aportant informació de què cal fer per configurar-ho correctament.

Si aquesta verificació de configuració es fa amb una funció que es pugui cridar en qualsevol lloc del programa, podrem cridar-la a diferents llocs del programa quan vulguem, i ens pot servir per detectar modificacions errònies durant l'execució que pot ser molt útil quan el programa no funciona i no sabem trobar l'error.

També és útil que hi hagi la possibilitat d'escriure tota la configuració i que es pugui activar o no aquesta possibilitat (tipus una etiqueta de DEBUG d'imprimir configuració o no). Això serveix per tenir amb la sortida del programa les dades de configuració i tenir tota la informació per reproduir una execució en cas de que falli, o doni resultats erronis i ho revisem molt més tard que ja podríem haver fet altres execucions o modificacions de codi amb canvis de configuració.

En el programa de notes del cas pràctic, tenim un conjunt de constants definides al fitxer notes.h. Per exemple tenim els percentatges del pesos de les notes per calcular la nota final, o la nota mínima i màxima. Amb això tenim, els exemples següents de regles de robustesa de configuració pel programa de notes del cas pràctic (sense ànim de ser exhaustiu):

1. La suma dels pesos percentuals han de sumar 100%
2. La nota mínima ha de ser més petita que la nota màxima en la definició del rang de valors de notes

Notar que l'existència de la configuració ve donada per la definició dels números com a constants, que a la vegada això permet la flexibilitat del programa i l'extensió a altres contextos i aplicacions. Si tenim les constants de nota mínim i màxim com valors a configurar vol dir que el nostre programa pot funcionar amb rang de notes diferents només canviant els valors d'aquestes constants. Això és cert sempre i quan el codi estigui pensat en què realment els valors són identificadors i com a tal poden canviar encara que no ho facin. I per tant els problemes de configuració venen només pel fet de tenir flexibilitat de canviar aquests valors. Per tant la flexibilització és bona per la reutilització de codi, però la generalització que comporta pot ser contraproduent si no se sap utilitzar i controlar correctament.

2.3.2 Robustesa d'Entrada Manual de Dades

Aquest cas estem parlant de possibles incoherències en l'entrada de valors donats per l'usuari en relació al què s'espera. Com l'usuari interactua amb el programa, la verificació s'ha de fer per cada dada, per notificar a l'usuari de l'error i donant instruccions clares de quin valor s'espera i demanar-li un nou valor. Es important donar l'opció de sortir sense realment completar l'opció pels casos en què l'usuari no pugui donar un valor correcte, ja que llavors el programa es podria quedar bloquejat en una situació que l'usuari no vol o pot continuar.

Com exemple podem citar algunes de les regles de robustesa de l'entrada de dades (sense ànim de ser exhaustiu):

3. Les notes han de ser valors numèrics entre un valor mínim (0) i màxim (10) ambdós inclosos (no poden ser alfanumèrics)
4. Els codis d'assignatures acceptables al matricular un alumne només poden ser codis d'assignatures reconegudes
5. Els codis d'assignatures acceptables al entrar notes en un expedient només són els codis de les assignatures en que l'alumne s'ha matriculat

Una manera de minimitzar els errors d'usuari es oferir una interfície que només es pugui seleccionar opcions acceptables. En aquesta pràctica no contemplem les interfícies gràfiques per fer aquesta selecció, però podeu pensar com assimilar aquest plantejament en l'entorn text per minimitzar la possible quantitat d'errors d'usuari.

2.3.3 Robustesa d'Entrada de Dades Via Fitxers

La possible modificació de fitxers via editor de text imposa que cal contemplar inconsistència dels fitxers d'entrada en el contingut del fitxer però també en el format del fitxer. Aquí hi ha moltes possibilitats a contemplar, i seria imprescindible en una aplicació real. En el cas d'aquesta pràctica suposem que ens restringim a contemplar només inconsistències dels valors de paràmetres que estan dins el fitxer però assumim que el fitxer compleix estrictament el format acceptable (dins les possibles variants que accepta).

Però cal que tingueu molt present, i no deixar de banda, la possibilitat de que el programa no us funcioni degut a que el format del fitxer que se li proporciona no sigui del tot correcte.

Cal tenir en compte que les comprovacions d'entrada manual de dades apliquen exactament a les dades entrades via fitxer també. Per tant les comprovacions anteriors han de ser implementades de manera general perquè les mateixes funcions es puguin aplicar quan les

dades venen de fitxer. Les dades que aporta el fitxer, però que no entra manualment l'usuari són regles addicionals que cal contemplar en aquest apartat.

Exemple de regles d'aquest tipus tenim (sense ànim de ser exhaustiu):

1. El número de convocatòria ha de ser un valor enter entre 0 i un màxim (no pot ser decimal ni alfanumèric)

Exemple de comprovacions que no s'espera contemplar relacionades amb el format del fitxer (sense ànim de ser exhaustiu):

1. Totes les columnes d'assignatura del fitxer expedient tenen que tenir un valor per poder llegir sempre el mateix nombre de valors. Si una columna no té valor encara (exemple sense nota) cal que indiqui el valor identificat com "sense valor" per aquella columna.

2.3.4 Robustesa de mal funcionament

Si ens acostumem a aplicar la robustesa anterior, ja agafem l'hàbit de construir el programa perquè ens parli del què no va bé. Per tant de manera natural introduïm les comprovacions de funcionament que sabem reconèixer, i sabem que ja no hi hem de pensar més perquè està integrat en el programa.

Regles útils en aquest àmbit depèn molt del programa i implementació, però podem citar:

2. Verificar sempre l'índex d'un vector (sobretot dins una funció si es passa per paràmetre) per assegurar que està dins el rang de posicions del vector.

3 Pla de treball

El problema plantejat en l'enunciat resulta un programa llarg i meticulós ple de detalls a considerar. Però no és un problema difícil d'abordar. Es pot construir una solució casi complerta i ben definida amb els continguts del segon trimestre ben aplicats. Si ens reduïm al segon trimestre podem tenir l'aplicació ben dissenyada amb l'estructura de dades necessària per abordar el problema complert, un conjunt de funcions ben definides abordant totes les funcionalitats de manipulació de dades del problema (excepte l'ordenació) però entrades manualment (sense fitxers). Aquesta solució estaria construïda amb un únic fitxer de codi ja que les llibreries es donen el tercer trimestre. Tot i això la programació del segon trimestre representa una part molt important de tota la feina que s'ha de fer. Per tant la pràctica és pot començar i avançar molt des del primer dia de la seva publicació (sense les classe del tercer trimestre).

Es pot començar amb el disseny pensant que s'entraran totes les dades manualment (o es crea una funció que assigni valors per anar treballant), i afegir carregar i gravar de/a fitxer és una funcionalitat incremental (el disseny principal no varia) que es pot afegir més endavant. Igualment, podem suposar que l'ordre del llistat d'assignatures no és un requeriment en la definició de l'expedient acadèmic i deixar de banda l'ordenació inicialment. Altre cop, afegir-la més tard només és un element incremental que implica afegir un procés (crida a una funció ben definida) entre les línies de codi adequades.

La primera setmana del tercer trimestre treballarem la modularitat i llibreries que permeten estructurar i separar un programa per avançar-lo en equip de manera efectiva. Això és cabdal per poder abordar un problema gran i poder-lo completar a temps amb un equip. Així que la primera setmana tindrem les eines per executar l'implementació de manera efectiva en equip. S'espera que aquests conceptes es posin en pràctica en la implementació de la solució si no serà molt difícil, si no impossible, poder acabar a temps el programa pel dia esperat d'entrega. És per això que es demana un disseny del programa via mapa conceptual en la preentrega que obliga a pensar el problema general i estructurar-lo sense implementar-lo encara. Aquesta eina és fonamental a l'hora de distribuir la feina entre els membres de l'equip.

Els primers dies, també treballem conceptes i metodologia de verificació de codi per progressiva i incrementalment fer créixer un programa amb garanties de qualitat. La identificació d'errors, mal funcionaments o resultats inesperats centren una part molt important de la feina del programador sobretot quan és inexpert. Aplicar una metodologia perquè sigui el propi programa que detecti i identifiqui els errors i inconsistència canvia completament l'enfocament i esforç del programador. Convertir-se en un bon programador no és només saber els llenguatges de programació sinó també adquirir uns hàbits de bones pràctiques (metodologia de programació) més intangibles però imprescindibles per abordar programes d'una certa embargadora i complexitat. Disposareu per tant dels conceptes de traces per verificació de codi i banc de proves molt a l'inici per aplicar-ho a la pràctica en aquest programa. Es demana el disseny del pla de proves en la preentrega per assegurar que ho apliqueu aviat per ajudar-vos a abordar el problema proposat.

Així doncs s'ha d'entendre la preentrega com una fita d'equip en el procés d'adquirir un mètode de treball concretant elements de disseny i fites de treball en el procés d'elaboració de la solució. Ho heu de veure important per la vostra progressió de programadors i no com un element d'avaluació. És per això que no s'avalua i és la vostra responsabilitat consultar els professors els vostres dubtes referents a aquests elements elaborats. Tot i així la data de la preentrega és molt àmplia i estableix un termini temporal molt de màxims. És a dir, és molt aconsellable, i s'espera, que aquests elements de disseny estiguin elaborats molt abans a la data de termini de la preentrega i es recomana entregar-la tan aviat com sigui possible per utilitzar-ho com a eina de diàleg entre l'equip i també amb el professorat (per consultes) a l'hora de resoldre dubtes i avançar en el disseny i la programació a abordar.

Aquesta preentrega està pensada entregar-la just abans de la primera sessió d'aquesta pràctica per tal d'encarar la sessió cap a debatre i resoldre dubtes sobre el disseny de la pràctica amb feina ja avançada i dubtes ja tangibles. Per tot aquest treball disposeu del professorat amb horari de consulta. Hi ha un horari publicat a l'aula global d'hores de consulta a una aula d'ordinadors cada dia durant tot el trimestre [HC].

A partir de la segona setmana, ja es treballarà fitxers a teoria i a seminaris i per tant en aquell punt s'estarà preparat per avançar el detall de fitxers en la pràctica. Cal remarcar que per la definició de les funcions (i disseny dels mapes) només cal decidir què cal que faci la funció encara que no sapiguem com fer-ho. Per tant el disseny el podem pensar bastant complert i un cop fet a teoria el concepte corresponent es pot perfilar el disseny i abordar implementació i verificació. En qualsevol cas, aquest calendari proporciona la teoria de fitxers amb el temps més que suficient per tractar-ho complertament en la preentrega que té el termini en la tercera setmana.

A la quarta setmana treballem els algorismes de cerca i ordenació. Aquest continguts representen funcions molt concretes i aïllades dins la funcionalitat del programa. L'ordenació sobretot pot ser completament prescindible i per tant no hi ha problema, i de fet pot ser aconsellable per la implementació incremental, incorporar-ho al final quan tota l'aplicació ja funciona. La funció de cerca és més necessària perquè identifica en quina posició una possible assignatura o expedient està dintre del vector. A teoria veurem diferents algorismes per optimizar aquest procés, i estudiarem la complexitat d'aquests algorismes. De totes maneres tots podem construir una funció intuïtiva de cerca per identificar la posició d'un element donat. Si us cal implementar-la abans podeu fer la funció que us sembli per anar tirant i després repensar l'eficiència de la que heu fet i dels algorismes estudiats. L'entrega no obliga a cap algorisme en concret però sí es demana que calculi el nombre d'operacions que realitza. El més segur és que la funció que definiu sigui el primer algorisme que treballem a teoria.

Cal remarcar que per l'elaboració d'aquesta pràctica es disposa d'un cas pràctic resolt [CP] molt similar a aquest que pot servir de guia en tot el procés des del disseny fins la implementació i la verificació. El cas pràctic proporciona un programa pel càlcul de notes d'una assignatura. No és exactament el mateix problema a abordar en aquesta pràctica però sí està molt relacionat i es poden aprofitar moltes analogies (començant per l'estructura de dades fins la verificació amb les traces dels algorismes d'ordenació per exemple). La solució s'aporta desgranada en diferents versions per veure l'evolució del programa i l'aplicació de cada concepte individualment més enllà dels conceptes a incorporar en la pràctica (es fa servir de cas pràctic en l'explicació dels conceptes a teoria). Les diferents versions us donen exemple també de la programació incremental. Per cada versió s'aporta els mapes conceptuals i el codi implementat tal (el què demana en aquesta pràctica). I per la primera versió també es mostra el disseny del pla de proves de la funció de càlcul de notes. El codi està documentat per ser un model en com documentar un codi (encara s'espera que adquiriu el vostre estil però que sigui útil a altres i no només per vosaltres). I proporciona un seguit de filtres de traces per poder analitzar el funcionament del programa de funcionalitats concretes. El seguiment manual d'aquestes traces s'ha dissenyat per la verificació del codi. També incorpora tests interns de verificació i comprovacions de les dades entrades per no deixar petar el programa amb els errors d'ús de l'usuari (encara que no és robust completament). Tot plegat és un material que podeu utilitzar de guia en tots els aspectes que se us demana en aquesta entrega.

En la pràctica no s'espera que mantingueu les diferents versions. Però en el cas pràctic es mantenen les versions per guiar el procés d'aprenentatge des d'un programa petit a gran. No és recomana anar directament a entendre la última versió i fer un únic esforç per entendre la solució final. És un salt massa gran a aquest nivell i enlloc d'estalviar-vos temps en la comprensió de la solució final us pot frustrar la incapacitat d'entrendre-ho tot de cop. En contra, s'espera que analitzeu, entengueu i practiqueu en les versions progressivament i no passeu a la següent versió fins haver consolidat els conceptes teòrics i pràctics de la versió anterior. En realitat aquesta progressió serà similar a com el vostre programa anirà creixent a mesura que aneu treballant. Però no és necessari mantenir les etapes intermèdies dels vostres dissenys i programes per l'entrega final.

Tot el material del cas pràctic (mapes conceptuals i codi i transparències que ho utilitzen) està disponible a l'aula global a la secció de teoria perquè el cas s'utilitza de material de suport per teoria. En les transparències de teoria, i en cada capítol, hi ha un apartat on s'explica el concepte amb exemples del cas pràctic. Però cal remarcar, que no es dedica temps exclusiu a les classes a entrar en aquest codi i treballar-ho. Això es deixa a què vosaltres el treballeu independentment, i demaneu tots els dubtes que us sorgeixin a qualsevol professor de l'assignatura.

Així doncs al pla de treball anterior que se us proposa, se us recomana abans de començar tot el treball anterior de la pràctica estudiar el material del cas pràctic i practicar sobre ell, tan amb el disseny com la implementació i execució del programa. Estudieu-lo i jugueu amb ell amb diferents execucions i paràmetres. Us donarà idees com pensar la vostra pràctica enlloc de començar completament en blanc.

Recordar que el professorat ha establert un horari regular de consultes diari durant tot el trimestre per donar suport en l'elaboració de les pràctiques. Cada dia hi haurà un professor de pràctiques per la tarda per donar suport a tots els alumnes de l'assignatura. Podeu consultar l'horari de consultes i aules a l'aula global. Us animem a fer ús d'aquest suport contrastant el vostre disseny amb el professorat el abans possible per assegurar la bona direcció o per demanar suport en qualsevol dubte o dificultat que us pugui sorgir. Ajudar-vos a millorar el funcionament del treball en equip també forma part del suport que els professors us poden subministrar, per tant no espereu a tenir problemes greus i possiblement insalvables abans de comentar o demanar ajuda en aquests aspectes.

Finalment, ressaltar l'enorme importància de l'ús d'un bon entorn de programació en el procés d'aprenentatge. És vital per un programador novell disposar d'un bon debugger i saber-lo utilitzar. És per això que recomanem us instal·leu el programa netbeans (o

un de similar) al vostre ordinador el abans possible. És aconsellable tenir-lo instal·lat abans de la primera sessió de pràctiques (que per alguns és la primera setmana) i seria encara millor tenir-ho pel primer dia de classe de teoria del tercer trimestre.

4 Treball en equip

Agafar experiència en el treball en equip vol dir practicar en la col·laboració d'idees i en la distribució eficient de la feina i en fer que un equip sumi per sobre de la suma de les seves individualitats. Com hem comentat anteriorment, les eines de modularitat van enfocades cap aquesta direcció però s'han de saber aplicar.

El primer pas a realitzar en un equip és tenir les idees clares del què cal fer i per tant dedicar esforços tot l'equip en entendre el problema i assentar els pilars de disseny. **Primerament** s'ha de decidir l'**estructura de dades** i les **variables principals** a utilitzar en tot el programa i per tot l'equip. Simultàniament, i en la intenció d'aprofundir en la comprensió del problema, cal analitzar el problema i partir-lo en parts i aquestes parts en més parts i així successivament fins tenir una estructura suficientment disseccionada del problema principal. Aquest treball pot resultar amb l'elaboració dels mapes conceptuals. El primer nivell d'estructura s'ha de fer conjuntament per debatre les idees conjuntament i construir una perspectiva conjunta i consensuada de l'estructura del programa. D'aquí surt la primera segmentació del problema a molt alt nivell amb possiblement poques caixes (mòduls) amb funcionalitats molt clares i diferenciades, més les estructures de dades inicials i les relacions. Aquest primer pas es pot fer ràpid i en conjunt.

Més difícil pot ser anar aprofundint en els detalls de cada una d'aquestes grans funcionalitats per desgranar tot el què cal fer. Però aquesta tasca es pot repartir de manera eficient responsabilitzant a cada membre de l'equip de pensar i aprofundir en l'estructura d'un (o varis) mòdul del mapa conceptual inicial. Al haver-hi un mapa clar inicial que divideix el problema en funcionalitats independents (caixes o mòduls) entrellaçades apropiadament, es pot avançar en paral·lel i per separat el detall de cada un sense duplicitats de feina ni esforços. Amb això tenim un mapa conceptual per cada mòdul que representa un nivell de detall molt més acurat. L'estudi detallat pot implicar identificar noves necessitats a altres mòduls del programa, o modificacions de les relacions existents en els grans mòduls o necessitats de nous elements de dades a compartir. Tot això afecta l'aplicació com un tot i s'ha de tractar diferent que les necessitats, relacions i dades definides o creades només dins el propi mòdul que s'està definint.

Les implicacions globals cal consensuar-les amb tot l'equip abans d'integrar-les en el disseny per analitzar globalment si aquesta és la millor manera de tractar la necessitat del mòdul en particular, i per actualitzar les implicacions a nivell global immediatament per tal de què els dissenys independents evolucionin coherentment.

En aquesta fase de disseny el resultat del disseny de cada mòdul és només a nivell d'identificar funcionalitats necessàries i ordenar-les i relacionar-les amb l'execució i en la manipulació de les dades. Això bàsicament representa decidir quines funcions cal definir, què ha de fer cada funció i quines dades necessita per fer-ho i quines dades produeix cap a la resta del programa. Com hem vist, això ho podem materialitzar gràficament amb un mapa conceptual per cada mòdul amb les seves caixes i relacions. Però la descripció cal recopilar-la en algun tipus de document. Si hi pensem, aquesta descripció és la informació que es recomana incorporar com comentaris dins el codi: 1) cada funció ha d'anar acompanyada d'una descripció del què fa i quins paràmetres intervenen. 2) cada fitxer ha de tenir una capçalera que indiqui què conté aquest fitxer, qui ho ha fet, ... Així que podem plantejar que en la fase de disseny podem construir la documentació com a plantilles de codi degudament comentades[1]. Així tindrem un fitxer per cada mòdul que tindrà una capçalera que expliqui aquest fitxer què descriu i amb la llista de funcions buides (no implementades) però amb la descripció del què han de fer i quins paràmetres tenen que definir. Aquest fitxer de codi c pot començar sent un seguit de comentaris en c que identifiquen les funcions, després definir per cada comentari de funció el seu prototip formal (mantenint el comentari de descripció de la funció), i una mica més enllà construir les funcions en c amb aquests prototips sense codi (o potser amb un simple printf que informi de que l'execució ha passat per la funció). Si tots els mòduls fan el mateix tenim una descripció del disseny que pot ser un executable sense realment executar res. I aquesta plantilla es el punt de partida per la implementació del programa amb els comentaris necessaris per qualsevol persona (membre de l'equip o no) poder omplir el codi necessari segons la descripció. Ara es pot treballar en paral·lel i desincronitzadament evolucionant la implementació de cada fitxer (mòdul) per separat de manera coherent amb la resta, i sobretot amb una eina executable amb la qual podem anar provant els nostres propis canvis sense que afectin als altres (i vice-versa). Tenim doncs tothom ocupat cap un objectiu comú amb assignacions clares del què s'ha de fer.

La construcció de la plantilla de les funcions també aplica amb les declaracions, i estructures de dades. Però en aquest cas enlloc de ser un fitxer c és un fitxer capçalera (.h). Podem començar tenint un llistat de declaracions definides per una descripció (inicialment un comentari en c) i llavors traduir-ho a noms i identificadors en c dins el mateix fitxer .h. Així tenim un comentari ben articulat per cada declaració i variable. La diferència de les declaracions principals és que són molt globals i són utilitzades en tot el programa. Per tant han d'estar fixades des del primer nivell del mapa conceptual i cada modificació ha de ser molt valorada i consensuada amb tot l'equip.

Amb aquest plantejament tenim tot l'equip treballant segons la seva assignació inicial. Però és possible que cada part del mapa conceptual inicial tingui diferent pes, o que les funcionalitats no estiguin ben estructures segons aquesta perspectiva. És a dir, la separació de funcionalitats i fitxers (modularitat) és un element important a analitzar. Seguint el raonament del treball en equip, volem que cada membre de l'equip pugui treballar de manera independent i no depengui ni li afectin els canvis dels altres. Per això hem de limitar la seva tasca al fitxer (o fitxers) que està treballant i ser l'únic autoritzat per poder-hi fer modificacions. Això serà efectiu

[1] De fet, existeixen entorns de desenvolupament que convinen documentació i programació precisament per tenir que evitar fer documentació separada si es té un bon codi comentat (i vice-versa).

sempre i quan les funcionalitats estiguin ben separades i repartides entre fitxers. Cal concentrar les funcions relaciones al mateix fitxer per poder-les treballar i ampliar conjuntament. I així arribem a una estructuració del programa en possibles llibreries que és tan útil per l'eficiència del treball en equip com per la reutilització del codi per diferents programes. És important doncs treballar la implementació d'un mòdul o fitxer de manera prou general perquè sigui el màxim reutilitzable en altres entorns i esdevenir realment una llibreria reutilitzable. Podem dir que unes funcionalitats separades i independents d'un programa defineixen un mòdul. Un mòdul té sentit i vida dins el seu propi programa. Quan aquest mòdul és freqüentment necessari i utilitzat en altres programes o contexts es pot dir que és una llibreria (útil per si sola). En aquesta pràctica podríem definir els mòduls d'entrada/sortida, de plans docents, cerca i ordenació. Una empresa que elaborés programes de gestió per escoles podria considerar cada un d'aquests mòduls una llibreria ja que serien elements molt reutilitzables en el seu context.

Cal remarcar que amb les diferents fases de refinament i progressió del detall dels mapes conceptuals i fins i tot ja en la fase d'implementació i verificació poden sorgir elements inesperats i identificar en un mòdul algun element que vincula de manera diferent la relació amb els altres mòduls. Quan això passa estem "violant" la propietat dels altres programadors i cal abordar i acordar aquest canvi immediatament amb tot l'equip ja que estem definint el nostre mòdul amb un encaix dins el programa diferent al definit. **Ningú (cap membre de l'equip) pot avançar en la seva implementació** fins que aquest encaix es torna a definir adequadament a les noves necessitats identificades, ja que aquesta nova condició pot afectar l'encaix i funcionalitats de més parts en el programa. Només podem estar segurs que s'avança i que el treball que es realitzi sigui útil si tot encaixa dins el programa global.

La fase d'implementació es pot fer molt progressivament. Es pot començar com hem dit amb les plantilles buides de tots els fitxers i omplir de manera bàsica les funcions d'un mòdul. Si aquest mòdul proporciona una funcionalitat més àmplia i funciona es pot passar còpia del fitxer actualitzat a tots els membres de l'equip i només recompilant tindran les noves funcionalitats incorporades. Per tant és important fer passos petits per poder anar compartint aquestes millores en la implementació amb tot l'equip regular i progressivament[2]. Crea distorsions importants compartir codi amb problemes amb tot l'equip perquè efecte la possibilitat d'avançar la resta del programa. D'aquí la importància de ser metòdic en la comprovació del programa i de tenir un mètode efectiu per identificar els errors i verificar correctament el què fa el programa.

Per últim, recordar que perquè un equip funcioni cal que tots els seus membres funcionin. És molt important parlar de les complexitats i dificultats que cada membre troba al seu camí per tal de què una part del projecte no es quedi enrere. No és culpa de la persona que se li ha assignat una tasca si aquesta surt més difícil o més llarga de l'esperat. Però si que és culpa de la mateixa persona no notificar-ho al grup en temps per l'equip tenir temps de prendre mesures suficients per pal·liar-ho i evitar problemes més grans. Així és important posar la dedicació suficient i espaiada en el temps per tot l'equip rotllar (no només un membre en solitari) i no esperar a última hora per anar progressant en el projecte conjuntament i adequadament. També és important notificar quan un es bloqueja en un error per veure si algú pot ajudar a resoldre-ho i/o notificar que hi ha problemes. Així doncs, cal marcar terminis i tasques curtes i establir comunicació continua per anar avançant i modificant la planificació i assignació en funció del progrés. Tota la coordinació no és fàcil i intervenen situacions personals també. Cal tenir respecte als companys i informar si la dedicació que s'ha acordat (o s'espera) no és possible per qüestions personals. És inevitable que tard o d'hora el problema emergeix i val més aviat que hi ha temps de prendre decisions que tard que arrastra a tot l'equip.

El professorat està per ajudar-vos a gestionar el pla de treball i d'equip i ajudar-vos a què vosaltres resoleu els problemes que puguin sorgir adquirint així l'experiència en el treball d'equip. No espereu a consultar el vostre procés a ser irreversible, esteu aprenent i heu d'utilitzar els recursos al vostre abast per aprendre en totes les vessants possibles. I precisament les competències transversals és experiència intangible que us servirà per tirar endavant el projecte i per més endavant en un futur.

5 Eines de treball

Pel disseny i desenvolupament de programes hi ha moltes aplicacions que es fan servir en equips de desenvolupaments grans. En aquesta assignatura ens centrem només en un entorn de desenvolupament per la programació "individual" perquè tot just estem començant a programar i necessitem consolidar els conceptes de programació primer per treballar en entorns més sofisticats.

Es proposa treballar amb l'entorn integrat de programació (IDE – integrated development environment) NetBeans com a eina de desenvolupament. Aquesta ens proporciona la gestió dels diferents fitxers d'un programa (anomenat projecte dins a netbeans) i construeix el makefile automàticament per la seva integració. Té integrat el compilador, debugger i gestió de projecte conjuntament pel desenvolupament de programes modulars. A l'aula global hi ha una guia d'instal·lació i ús del NetBeans [NB] per donar-vos les pautes per instal·lar-vos-el al vostre ordinador i com iniciar-vos amb aquesta aplicació. El NetBeans es pot relacionar amb eines de gestió de versions i altres funcionalitats que no farem servir en aquesta assignatura (però podeu fer servir si us interessa o si ja teniu experiència prèvia en programació).

No és obligatori fer servir el NetBeans, ja que el què realment s'entrega és el codi. Podeu fer servir qualsevol altre que ja sapigueu fer servir, si és el cas. Però és imprescindible que aprengueu a utilitzar un IDE que proporcioni un debugger, perquè el debugger és imprescindible per detectar segons quins tipus d'errors. És important doncs que l'entorn que utilitzeu tingui debugger i el sapigueu fer servir.

[2] Hi ha aplicacions de gestió de versions que serveixen per compartir fitxers en equip pel desenvolupament progressiu de programació. Però nosaltres no les farem servir en aquesta assignatura. En aquesta assignatura ens centrem a aprendre les bases de programació i a adquirir els hàbits i bones pràctiques adequades.

6 Gestió del temps

Per últim, un recordatori de la importància cabdal de la gestió del temps. Heu de ser capaços de detectar quan no avanceu i perquè i assignar el temps adequat a les coses i reaccionar amb temps raonable a nivell personal cada un, i a nivell de grup col·lectivament.

Si esteu bloquejats en un problema, estar-hi més hores a sobre havent esgotat les idees de què més fer no és la millor opció.

Cal aportar una visió nova, que pot venir després d'un descans propi (que pot ser fent una altra assignatura o dormint, o el què sigui diferent), o parlant amb una altra persona o consultant fonts addicionals. Recordar que a vegades només explicant el problema un mateix identifica la solució perquè s'adona del què no està fent bé. No espereu a esgotar tot el temps que teniu intentant trobar la solució. Assigneu dedicació a les tasques i si us passeu del temps, reviseu conscientment què fer.

Limiteu el temps que utilitzeu a buscar recursos a la web, ja que la web té infinits recursos i no s'acaba mai. S'estipula un temps per trobar informació que us pugui ajudar i si no es troba, es canvia d'estratègia (consultar professorat, o altres fonts).

Per tot això no és una bona estratègia assignar blocs de temps sencers a entregues. Cal donar temps al nostre inconscient (i conscient) a assentar idees. Canvi d'activitat ajuda a mirar la mateixa cosa amb una perspectiva diferent. Si esteu bloquejats a la pràctica reviseu teoria o altres coses de la mateixa matèria encara que no estigui directament relacionada. Pot ajudar a desbloquejar la manera de pensar d'un moment donat. Donar marge de temps i tenir classes entre mig ajuda a injectar perspectiva nova i poder consultar.

Feu una planificació global de la feina incloent l'assistència a les classes. No anar a classe vol dir que comenceu completament de zero a l'hora d'abordar un concepte (sigui de l'assignatura que sigui). No podeu començar directament a implementar-ho. Us cal primer conèixer i entendre els conceptes no tan sols per començar a crear la solució però també per entendre els enunciats. Si no s'ha anat a classe això vol dir que cal estudiar de zero abans de començar les entregues. Cal realment valorar si us val la pena fer-ho sols sense guia enlloc d'aprofitar el moment que el professor ho explica donant-li el context que us cal per aplicar-ho a les tasques que se us demana. Fer tot això pel vostre compte requereix més temps però a més us podeu estar dirigint cap una direcció que no és l'esperada. A més la presencialitat us permet resoldre els dubtes que us puguin sorgir durant l'explicació si realment esteu concentrats a la classe. Per tant assistir a classe és útil sempre i quan s'hagi près la decisió conscient de dedicar una estona als continguts de la classe. I com hem dit abans el canvi de feines ajuda a descansar d'una i tornar-hi més fresc més tard.

Així doncs, recordeu que és molt i molt important que penseu com utilitzeu el vostre temps sempre, però en programació més!

7 Conclusió

Com a conclusió, una recopilació de frases celebres ....

Albert Einstein:

- No tot el que compta pot ser quantificat, i no tot el què pot ser quantificat compta
- Bogeria és fer la mateixa cosa una i una altra vegada i esperar obtenir resultats diferents
- La imaginació és més important que el coneixement
- Tot s'ha de simplificar el màxim possible, però no més
- L'educació és lo que queda després d'oblidar el què s'ha après a l'escola

Digues i ho oblido, ensenya'm i ho recordo, involucra'm i ho aprenc – Benjamin Franklin