

Documentació del *software*

Introducció: en aquest capítol farem una petita reflexió alguns elements que rodegen el desenvolupament del *software*. Comencem amb una pregunta: us heu preguntat perquè es va canviar la titulació de la FIB de Llicenciatura a enginyeria? En sabeu la diferència?

El diccionari diu que un llicenciat és una “persona que ha obtingut en una facultat universitària el grau que l'habilita per a exercir una professió determinada.” Mentre que un enginyer és una persona que aplica “els coneixements científics a la invenció, al perfeccionament o a la utilització de les tècniques en totes les seves determinacions dins el camp industrial.”

Es defineix tecnologia com la “ciència que tracta de les arts industrials, de tal manera que ve a ésser com la teoria de la indústria pràctica.” I la ciència és un “conjunt de coneixements, i l'activitat destinada a assolir-los, que es caracteritzen formalment per la intersubjectivitat (que vol dir que hi participen dos o més subjectes o consciències) i pràcticament per la capacitat de fer previsions exactes sobre una part de la realitat.” És a dir, que la ciència s'ocupa del coneixement i la tecnologia de la seva aplicació industrial.

La FIB fou la primera facultat de la UPC. Per què a la UPC hi ha facultats i escoles tècniques? Resumint podríem concloure que un llicenciat (que es forma en una facultat) és un científic i un enginyer és un tecnòleg (i se sol formar en una escola tècnica).

Òbviament, i en conseqüència, hi ha elements que incideixen fortament sobre la tecnologia i que no tenen o no poden tenir la mateixa preponderància en el fet científic. Destaquen especialment el fet econòmic i de rendibilitat, i també els de planificació que hi estan molt lligats, especialment en la producció de *software* ja que aquesta activitat es fa fonamentalment a força de mà d'obra (o si ho preferiu de neurones) i el seu cost està molt lligat a les components de temps i, per tant, de planificació.

Possiblement us preguntareu què hi té a veure tot això amb la documentació? Bé, d'una banda us podria parlar ara de planificació, però tornem a estar davant d'un tema que s'escapa de l'àmbit de l'assignatura, tot i que us convé organitzar-vos bé per treure el major rendiment possible del vostre esforç, la planificació del curs ja està tancada i sabeu bé quan heu de fer els lliuraments i què ha d'incloure cada un d'ells.

En canvi convé parlar d'altres elements que tenen una incidència directa sobre la productivitat de les persones que treballen al voltant del *software*, tan dels tècnics que el desenvolupen com dels usuaris del producte. Em refereixo a la documentació.

Necessitat de la documentació: la documentació és una eina facilitadora, ha de servir per reduir l'esforç i millorar la comprensió del producte, augmentant, així, la productivitat de l'usuari davant del programa i la del tècnic, tant en les col·laboracions durant les etapes del desenvolupament com en el manteniment.

El problema és què documentar costa un esforç addicional, un tècnic no en sol tenir facilitat i en general no li agrada fer-ho (potser pels problemes que solem tenir amb el text lliure i que ja em comentat parlant de l'especificació); a més molta de la documentació deixa d'estar al dia de seguida, tan aviat com es facin les primeres operacions de manteniment, fins i tot a vegades abans d'acabar el projecte. És a dir, que es

pot raonar, tot i que sigui amb certa lleugeresa, que encareix el producte i no se li treu rendiment.

A més, resulta fàcil de reduir els costos i temps de desenvolupament (cosa necessària moltes vegades, sobretot quan no s'ha encertat amb la planificació) sacrificant la documentació, especialment la tècnica, però moltes vegades fins i tot la de l'usuari.

Això és una mala pràctica perquè d'una banda pot complicar el propi desenvolupament (això s'intenta pal·liar a base de transmissió oral entre l'equip tècnic), però ho és sobretot de cara al futur manteniment del programa, especialment quan els tècnics que ho faran no són els mateixos que han escrit el *software*.

L'absència o la mala qualitat de la documentació per a l'usuari s'intenta pal·liar a base de donar-li formació addicional sobre l'ús i la funcionalitat del producte, formació on, a més, s'hi implica els tècnics d'informàtica que l'han desenvolupat o que en fan el manteniment; cosa que, com fàcilment podeu imaginar, du a una formació de poca o no gaire qualitat.

De la mateixa manera que s'està tendint a la fabricació amb processos que puguin garantir la qualitat del producte final, és a dir, sense un control posterior ja que la qualitat s'integra en el procés de construcció, es tendeix a que la documentació s'incorpori en el producte final, quan va destinada a l'usuari, o en el propi procés de desenvolupament quan es tracta de la documentació tècnica.

Per anar bé **la documentació tècnica** s'ha d'integrar a la pròpia metodologia, sol formar part de les aportacions de les eines CASE (enginyeria del *software* assistida per ordinador) corresponents, i si es procedeix adequadament i cada cop que es fa un canvi es torna a refer el procés de l'etapa del cicle de vida on es va definir l'aspecte modificat la documentació tècnica estarà sempre actualitzada.

En el cas de l'assignatura us convindrà definir, a nivell de cada *cluster*, un estàndard o normativa de programació que té la missió fonamental de definir com aplicar d'una manera uniforme en els fonts dels programes les tècniques d'autodocumentació. (Cada *cluster* funcionarà com una organització de desenvolupament de *software* independent.) Aquest estàndard ha de ser tal que no es pugui inferir l'autor d'una classe (o d'un fitxer font) si no és perquè ho diu en el propi text del font, mai pel seu estil o aspecte.

Un estàndard de programació ha de facilitar la intercanviabilitat dels tècnics durant el desenvolupament i el manteniment de les aplicacions informàtiques ja que ha de facilitar a comprensió del codi font.

En algunes organitzacions es valida el format del font segons l'estàndard de programació adoptat i no es deixa compilar un font no validat. Només té sentit procedir així quan no es disposa d'una eina CASE adequada ja que s'ha de recórrer a programes fets a mida per i per a la pròpia organització.

No perdeu de vista que aquesta normativa és un element més de la documentació tècnica, però no pas tota, és a dir, que a més dels fonts seran necessaris els diagrames de classes fets amb UML per poder copsar totalment el programa.

Quan amb l'estàndard de programació s'introdueixen elements complementaris als que aporta la metodologia i l'eina CASE corresponent, convé tenir bona cura en adap-

tar la normativa a les eines que s'usin, convindrà evitar que la normativa obligui a modificar allò que es genera automàticament.

Elements d'un estàndard de programació: podríem classificar els aspectes més rellevant que ha de cobrir un estàndard en:

- Aspectes de nomenclatura, és a dir, la formació dels identificadors o noms dels elements.
- Política de comentaris, on es defineix els criteris a tenir en compte, els llocs on van i el nivell d'obligatorietat, és a dir, quins comentaris són obligatoris i quins només, recomanables.
- Estructuració del programa, que són en general qüestions d'aspecte, com els sagnats i la distribució de les estructures de control del llenguatge.

Com podeu veure tots aquests aspectes estan força lligats al llenguatge de programació, especialment les qüestions d'estructuració del programa.

Nomenclatura dels identificadors: avui en dia s'exigeix que siguin mnemònics, és a dir, que el nom ajudi a reconèixer (o recordar) l'element identificat.

Com que moltes vegades consten de varies paraules caldrà definir si s'indica la separació les paraules o no, i en cas de fer-ho definir com. Per marcar aquestes separacions actualment hi ha dues tendències:

1. Usant les majúscules, per exemple, aixoesunidentificador
2. Usant el guió baix (*underscore*) "_", per exemple, aixoes_un_identificador.

Un identificador se sol compondre amb lletres (es limita a les 26 de l'alfabet anglès) números del 0 al 9 i alguns altres caràcters que formen part dels 128 del codi ASCII sense estendre (uns pocs, se sol limitar força, per exemple: "_", "\$"...)

Es recomana limitar-se a fer ús de la diferència entre les majúscules i minúscules quan el llenguatge de programació hi pot donar suport, si no és així aquesta diferència només tindrà un sentit estètic, no real.

Un altre aspecte important és el del tipus. El concepte modern de tipus que ens fa considerar a més del valor que pot tenir un objecte, quines operacions es podran fer amb ell (els mètodes), ens força a l'existència d'una tipologia d'objectes força més extensa. Per això hi ha propostes d'incloure un recordatori del tipus en els identificadors dels elements del programa que en tenen, una d'aquestes és la notació hongaresa. Aquestes propostes avui en dia tenen força detractors perquè aquesta informació no deixa de ser redundant en certa mesura.

La **notació hongaresa** no és feta a Hongria, de fet la va proposar un americà d'origen hongarès (Charles Simonyi que treballa a Microsoft, quan treballava per a Xerox) i com que d'entrada els identificadors resulten xocants, li digueren que aquella notació recordava l'idioma hongarès.

Es tracta de què cada identificador tingui un prefix que indiqui de quin tipus és. És a dir, que l'identificador de tot element que tingui tipus comença una primera part que indica el seu tipus, per exemple, `indPersonesInscrites` podria ser el subíndex de la taula de persones inscrites, sempre que `ind` fos el prefix del tipus subíndex.

Caldrà, doncs, definir els criteris per a la formació dels prefixos. Bàsicament cal tenir present els aspectes següents:

- Es defineixen els prefix de les classes bàsiques (enter, real, booleà, caràcter, taula, cadena, cua, etc.)
- Se sol treballar per juxtaposició, per exemple si "e" és el prefix dels enters i "p" el dels punters, "pe" serà el prefix d'un punter a un enter.
- Se sol limitar la longitud màxima de cada prefix a 3 o 4 caràcters, quan per juxtaposició s'arriba a una longitud major, se'n defineix un de nou. Així, quan la longitud màxima admesa és de tres i volem un punter a una taula de noms, que té el prefix "tnm", en lloc de "ptnm" haurem de definir-ne un de nou, per exemple "pn" (si està lliure, és clar). No cal dir que no es poden repetir, que hi ha d'haver biunivocitat, un prefix és només d'una classe i una classe només tindrà un prefix.
- Cada cop que es crea un nou tipus (una classe no abstracta) es defineix el seu prefix seguint els criteris que s'hagin definit, aquest prefix ha de formar part de la informació de la capçalera de la classe en el fitxer font.

Els **comentaris** són l'aspecte més literari de la documentació del font del programa, no tenen cap efecte sobre l'executable tot i que són bàsics de cara a la llegibilitat. Es recomana de posar només els imprescindibles per fer el font llegible.

Un comentari textual ha de ser curt i concís, ha d'aportar informació addicional, i ha de ser entenedor. Per exemple, "do // això és un bucle" no aporta cap informació (és una obvietat). (Recordeu que són força més entenedores les frases directes amb una sintaxi simple i ordenada.)

Caldrà definir quins han de ser els comentaris obligatoris, com els de capçalera de classe (data, breu descripció, autor (A PP l'autor d'una classe ha de ser únic, cada classe estarà escrita per un únic alumne), història de les modificacions, etc.), dels atributs (una breu descripció, etc.) o dels mètodes o operacions (precondició, postcondició, breu descripció, diàleg resumit, etc.)

També s'ha de definir quan, perquè i en quines circumstàncies es posaran comentaris addicionals, obligatoris o no. Per exemple, alguns llenguatges com Eiffel permeten introduir els predicats de l'especificació formal (precondició, postcondició, invariant, fitament, etc.) amb elements del propi llenguatge, en d'altres llenguatges això no és possible, llavors es pot optar per introduir-los com a comentaris addicionals. També quan s'usa una truculència, per exemple, aquesta:

```
int a, b;  
...  
a = a + b;  
b = a - b;  
a = a - b;  
...
```

Aquesta "trampeta" només és vàlida per als enters, si l'analitzeu veureu que intercanvia els valors d'a i b estalviant l'ús d'una variable auxiliar, però no és obvi, per tant, convé algun aclariment en un comentari addicional. No puc estar-me de comentar que avui en dia no té gaire sentit plantejar-se de fer això per estalviar-se una variable entera.

Una bona estructuració dels comentaris permet que es pugui plantejar de processar-los de cara a documentar el programa i les biblioteques de classes. Alguns llenguatges disposen de diferents opcions de comentari, concebudes en aquest sentit o possibilitats addicionals com la clàusula **indexing** de l'Eiffel i també d'eines addicionals de documentació com *javadoc*.

Estructuració del programa: com que volem que tots els programes tinguin el mateix aspecte s'haurà de definir totes les característiques que configuren aquest aspecte. Això vol dir, definir la posició relativa de cada un dels elements del fitxer font. Fonamentalment són:

- Posició i format dels comentaris, tant dels obligatoris com dels opcionals, tan a la capçalera de la classe, com en cada un dels atributs i dels mètodes.
- Distribució del elements del llenguatge, especialment en les estructures de control, tant alternatives (com els **if**) com iteratives. Per exemple, en les estructures alternatives **if** podria fer-se:

```
if ( c )                o també      if ( c ) {
    {                    s1            s1
    }                    }
else                    }
{                        else {
    {                    s2
    }                    } //if
} //if
```

Òbviament hi ha altres possibilitats, això són només un parell d'exemples.

- Un bon costum és aprofitar aquesta proposta que fa Eiffel de marcar amb comentari la naturalesa de cada una de les clàusules **end** del programa (Eiffel tanca totes les estructures amb la clàusula **end**) i exportar-la a altres llenguatges on es pugui produir ambigüitat en aquest punt on es tanquen conjunts de sentències. Per exemple en el cas de C i Java passa això mateix al tancar una clau "}".
- Cal definir el sagnat (*sangrado* o *indentation*) en espais o tabuladors. La barreja d'espais i tabuladors sol donar problemes, especialment si es canvia d'eina d'edició i no es defineixen els tabuladors adequadament.
- Fixar la longitud màxima de la línia –s'hi inclou els sagnats.
- Limitar nombre de línies d'un mètode. Serveix per limitar la complexitat dels mètodes. A vegades es fixen dos valors, el màxim recomanat i el límit. Per exemple, es podria dir: es recomana que no es passi de 25 línies per mètode, però mai un mètode tindrà més de 40 línies.
- Criteris sobre les separacions amb línies en blanc, quan se'n deixa i quan no; o de línies de guions, etc.
- Documentació de les constants definides.
- Etc.

Sol ser més fàcil usar exemples o esquemes per definir les normes d'aspecte que intentar descriure-les textualment.

Altres elements: hi algun altre aspecte a tenir present com:

- Gestió de les versions i revisions de les classes.

- Elements del llenguatge a recomanar, a considerar poc recomanables o que es vulguin prohibir.
- Versions dels productes de desenvolupament, compilador, eines CASE, etc.
- I d'altres.

La documentació de l'usuari: en aquest cas es tracta de fer els textos necessaris perquè l'usuari pugui usar el *software* amb el rendiment més alt possible. L'objectiu és coordinar els objectius de l'usuari amb les capacitats del producte.

Avui en dia aquesta documentació se sol integrar en els programes en forma d'hipertext en una ajuda (*help* en format `hlp`) o bé en format HTML. Recordeu que el text ha de ser clar i fàcil de seguir, per tant, ben redactat, és a dir, de sintaxi simple, ben estructurada, ordenada i directe.

Tipus de documentació per a l'usuari:

1. Manual o guia de referència: una relació i descripció exhaustiva de les funcionalitats del programa (totes les opcions una per una: com usar-les i què fan).
2. Guia de l'usuari: una descripció del comportament que hauria de tenir l'usuari davant del programa per assolir els seus objectius. És a dir, quines opcions ha de triar i en quina seqüència per fer determinada operació. Dit d'una altra manera explica quina és la seqüència lògica d'operació per a cada una de les tasques de l'usuari.
3. Manual d'aprenentatge o *tutorial*: un curs interactiu d'aprenentatge o autoformació.
4. Targetes de referència ràpida: avui han caigut bastant en desús, són un resum o extracte del manual de referència.
5. Diccionaris i glossaris: descripció del vocabulari especial, sobretot l'informàtic, que ha calgut usar al programari i als manuals.

El manual de l'usuari que us demanem per al 4t lliurament ha de cobrir, com a mínim, els aspectes dels punts 1 i 2 d'aquesta llista, és a dir, de manual de referència i de guia de l'usuari.