Introducció a l'enginyeria del programari OO

Benet Campderrich Falgueras

P00/05007/00299

UOC
www.uoc.edu

Índex

Introducció

En aquest mòdul es comença a introduir el concepte d'*enginyeria de programari* i a presentar-ne la problemàtica. Després, es descriu en detall el cicle de vida clàssic o en cascada, i es presenten també models de cicle de vida alternatius, en especial els coneguts com a *models iteratius i incrementals*, entre els quals es descriu el cicle de vida del Rational Unified Process.

A continuació, es defineixen les dues grans línies tecnològiques actuals en el desenvolupament de programari: el desenvolupament estructurat i el desenvolupament orientat a objectes, i s'introdueix el concepte d'eina CASE. El mòdul acaba amb la presentació de l'origen i *status* del model estàndard UML, que serà el model orientat a objectes que es fa servir en aquesta assignatura, i de l'OMG, l'organització que se'n fa responsable.

Objectius

A partir dels coneixements previs que l'estudiant té sobre programació en general i sobre programació orientada a objectes, l'objectiu principal d'aquest mòdul és que aquest aprengui què s'entén per *enginyeria de programari orientat a objectes* i quines en són les bases. Aquest objectiu general es descompon en els objectius parcials següents:

1. Entendre què és l'enginyeria de programari i saber què la diferencia d'altres enginyeries.

2. Fer-se una primera idea dels problemes principals de l'enginyeria del programari actualment: la qualitat i la productivitat.

3. Conèixer el model de cicle de vida que ha tingut més impacte fins a l'arribada de l'orientació a objectes i també els principals models posteriors.

4. Adquirir el concepte d'eina CASE en general i, especialment, en entorns orientats a objectes.

5. Conèixer una mica l'origen d'UML, el model orientat a objectes esdevingut un estàndard mundial.

1. Què és l'enginyeria del programari

1.1. Què és el programari

Un **sistema de programari***, anomenat també **aplicació** o simplement **programari**, és un conjunt integrat de programes que en la seva forma definitiva es poden executar, però comprèn també aquelles definicions d'estructures de dades (per exemple, definicions de bases de dades) que fan servir els programes en qüestió i també la documentació referent a tot plegat (tant la documentació d'ajuda en l'ús del programari pels seus usuaris com la documentació generada durant la seva construcció, part de la qual també servirà per al seu manteniment posterior).

* *Software* és un sinònim habitual de *programari*.

1.2. El programari com a producte industrial

Un programari no és una obra d'art, sinó un producte de consum utilitari i massiu; per a una empresa o treballador autònom, el programari és un mitjà auxiliar que intervé de manera més o menys indirecta, però sovint imprescindible, en la seva gestió i cada vegada més en el seu procés productiu; també hi ha, com tothom sap, un consum privat de programari. Per tant, se'l pot considerar plenament un producte industrial.

Per exemple,...

… els bancs, en les indústries de fabricació en sèrie, en les empreses de comerç electrònic, etc., actualment no podrien funcionar sense programari.

Tanmateix, és un producte industrial amb algunes característiques especials. D'antuvi, és molt més un producte singular que un producte que es fabriqui en sèrie (tot i que hi ha programaris que tenen molts milers d'usuaris i, àdhuc, milions), ja que, si bé hi ha –i no pas sempre– producció en sèrie de còpies del programari, aquesta és una activitat molt poc important dins el conjunt del procés productiu del programari i relativament senzilla.

La producció de programari s'assembla a la construcció!

Des d'un punt de vista, la producció de programari s'assembla a la construcció d'habitatges o edificis industrials, per exemple, en el fet que cada producte és diferent i la seva elaboració es basa en un projecte específic (en el cas de producció en sèrie, el que es projecta és un prototip del producte, no pas cada unitat que es produeix).

```
function GifAnim(layer,imgName,imgSeries,end,speed,starFrame)
{this.layer = layer
this.imgName = imgName
this.frame = new Array()
for (var i=0; i<=end; i++) this.frame[i] = imgSeries+i
this.end = end
this.speed = speed
this.active = false
this.count = (startFrame)? startFrame : 0
this.obj = imgName + "GifAnim"
eval(this.obj + "=this")
this.play = GifAnimPlay
this.run = GifAnimRun
this.stop = GifAnimStop
this.goToFrame = GifAnimGoToFrame
```

Altres característiques del programari són, com assenyala Pressman, que no es fa malbé per l'ús ni pel pas del temps –si finalment s'ha de substituir és perquè ha esdevingut tecnològicament antiquat o inadaptat a noves necessitats o perquè ha arribat a ser massa car de mantenir.

1.3. L'enginyeria del programari

En general, a cada tipus de producte industrial correspon un tipus d'enginyeria, entesa com el conjunt de mètodes, tècniques i eines que es fan servir tant per a desenvolupar el producte (és a dir, elaborar el projecte o prototip) com per a fabricar-lo (afinant més es pot dir que hi ha, doncs, dues enginyeries per a cada tipus de productes: la de producte i la de procés).

Una **tècnica** és la manera preestablerta en què es duu a terme un pas en l'elaboració del producte, un **mètode*** és una manera determinada d'aplicar diverses tècniques successivament i una **eina** és un instrument de qualsevol tipus que es fa servir en l'aplicació d'una tècnica.

En aquesta assignatura...

... preferim *mètode* en lloc de *metodologia*, encara que només sigui per simplicitat.

El programari no és cap excepció a aquesta regla, i, per tant, hi ha una enginyeria del programari.

L'**enginyeria del programari** comprèn les tècniques, mètodes i eines que es fan servir per a produir-lo.

En el cas de l'enginyeria del programari no se sol parlar d'enginyeria de procés; potser es podria pensar que és aquella que fa referència a la programació en sentit estricte, però cada vegada és menys neta la distinció entre la programació i les fases anteriors en el desenvolupament de programari.

1.4. Els grans problemes de l'enginyeria del programari: la qualitat i la productivitat

Tot i els grans avenços que hi ha hagut en les tècniques de desenvolupament de programari durant els darrers trenta anys, tant la qualitat del programari com la productivitat del seu procés d'elaboració encara no han assolit nivells plenament comparables amb els d'altres tecnologies més antigues. Tot això, combinat amb un augment realment espectacular de la demanda de programari, ha provocat el que s'ha anomenat la *crisi del programari*.

Quant a la **qualitat**, la causa principal de dificultats és la gran complexitat del programari comparat amb altres tipus de productes, que fa, per exemple, que no sigui possible, ni de bon tros, de provar el funcionament d'un programari en totes les combinacions de condicions que es poden donar. I això passa en una època en què es dóna cada vegada més importància a la qualitat en tots els àmbits, en considerar-la un factor de competitivitat dins uns mercats cada cop més saturats i, per tant, més exigents. No és estrany, doncs, que el tema de la qualitat (i dintre aquest, qüestions com la garantia de qualitat i les certi-

ficacions oficials de qualitat) prengui una importància creixent dins l'enginyeria del programari.

Pel que fa a la **productivitat**, cal dir per començar que qualsevol fabricació en sèrie té necessàriament una productivitat molt més elevada que la fabricació d'un producte singular; però, fins i tot, si la comparem amb altres enginyeries de producte singular la productivitat és clarament més baixa. La complexitat del producte també pot ser una causa d'aquest fet, però certament no és l'única; un factor que té un pes realment important en la baixa productivitat és el fet que, a diferència de les altres tecnologies, en un projecte de programari el desenvolupament comença tradicionalment de zero (difícilment s'aprofiten trossos de programari de sistemes anteriors).

L'aprofitament d'elements en la fabricació en sèrie

Quan es dissenya un nou model de cotxe s'hi inclouen des del començament moltíssims elements que ja existien i que, per tant, no cal dissenyar; no tan sols elements normalitzats com cargols i femelles, sinó també elements més complexos, com bateries i, fins i tot, motors o caixes de canvi complets. També en la construcció d'edificis –una activitat que té més semblances amb la producció de programari, com s'ha indicat– es fan servir molts elements estàndards o prefabricats: teules, bigues, finestres, persianes, aixetes, etc.

Per tant, no és estrany que un dels grans reptes, ara com ara, de l'enginyeria del programari sigui aconseguir de desenvolupar fragments de programari (anomenats *components*) que siguin reutilitzables, d'una banda, i d'una altra desenvolupar programari i reutilitzar fragments d'aquests (que segurament estaran més ben provats que si es fessin de nou, cosa que de retop milloraria la qualitat del programari produït).

Una de les vies mitjançant les quals es pretén d'aconseguir una certa reutilització és el desenvolupament orientat a objectes; d'altres són els patrons de disseny (reutilització, si no de fragments de programari, almenys d'idees o "receptes" per a fer-los) i els bastiments o *frameworks*, que són estructures formades per sistemes de programari als quals es poden acoblar d'altres sistemes de programari, substituïbles, per a fer funcions concretes.

Arribats en aquest punt, convé que responguеu els exercicis d'autoavaluació 1 a 3.

2. El cicle de vida del programari

La producció de programari és quelcom més que la programació; hi ha etapes que la precedeixen i d'altres que la segueixen.

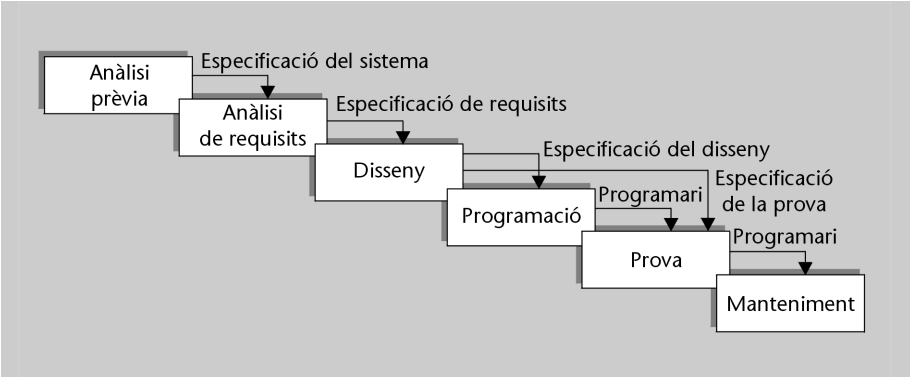
El **cicle de vida del programari** està constituït pel conjunt de totes aquestes etapes. Els mètodes i tècniques de l'enginyeria del programari s'inscriuen dins el marc delimitat pel cicle de vida del programari, i més concretament per les diferents etapes que s'hi distingeixen.

La mateixa existència de diversos models del cicle de vida del programari ja fa comprendre que no n'hi ha cap que sigui ideal o que no tingui fortes limitacions. Tanmateix, és indispensable que tot projecte es desenvolupi dins el marc d'un cicle de vida clarament definit, si es vol tenir un mínim de garantia de complir amb els terminis, bo i respectant els límits dels recursos assignats; i també la garantia de qualitat i les certificacions de qualitat* pressuposen que el procés de producció de programari es desenvolupi segons un cicle de vida amb etapes ben definides.

*** ISO, per exemple.**

2.1. El cicle de vida clàssic

La figura següent ens mostra les etapes previstes en una certa versió del cicle de vida clàssic.



De vegades, el cicle de la vida clàssic també s'anomena **cicle de vida en cascada**, cosa que vol dir que a cada etapa s'obtenen uns documents (en anglès, *deliverables*) que són les bases de partida de l'etapa següent –que, per tant, no pot començar abans que hagi acabat l'anterior– i mai no es torna a etapes passades.

2.1.1. Etapes

La primera etapa s'anomena **anàlisi prèvia** i també **anàlisi de sistemes** o **enginyeria de sistemes**. En aquesta etapa es defineixen els grans trets del sistema de programari que haurà de donar suport informàtic a unes activitats determinades d'uns certs usuaris dins el marc més general de l'activitat de l'empresa o organització.

A més, aquest sistema haurà de funcionar en un entorn de maquinari i xarxa determinat, que caldrà indicar, i potser també haurà d'intercanviar informació amb altre programari o compartir una base de dades. Aquests fets constitueixen altres aspectes de l'entorn del futur programari dels quals també caldrà deixar constància.

Cal tenir en compte els recursos necessaris per al desenvolupament del programari i els condicionaments temporals, especialment els terminis imposats des de fora del projecte, que sovint estan determinats pels fets que han causat les necessitats d'informació que ha de satisfer el programari en qüestió, i també restriccions eventuals i condicions addicionals que calgui respectar; i en funció de tot això s'avalua la viabilitat tècnica, econòmica i legal del projecte de desenvolupament del programari en qüestió.

El document resultant d'aquesta etapa s'anomena "Especificació del sistema", i serveix de base per a prendre la decisió definitiva sobre la continuació del projecte.

La segona etapa és la de l'**anàlisi de requisits** o simplement **anàlisi**. El seu objectiu és definir detalladament les necessitats d'informació que haurà de resoldre el programari, sense tenir en compte, de moment, els mitjans tècnics* amb què s'haurà de dur a terme el desenvolupament del programari.

*** Com ara el llenguatge de programació, el gestor de bases de dades, els components que es poden reutilitzar, etc.**

En aquesta etapa detallem els requisits de l'etapa anterior; ara només pensem en el programari que cal desenvolupar i les seves interfícies amb l'entorn.

La figura responsable de l'anàlisi –l'**analista**, que pot ser un informàtic o un usuari– ha de tenir o adquirir coneixements generals sobre el domini de l'aplicació i obtenir informació dels usuaris i d'altres fonts* que li permeti de fer-se una idea precisa de les funcions, i dels requisits en general, del futur programari. Amb aquesta informació es redacta el document que anomenarem "Especificació de requisits", que té una doble funció: especificar què ha de fer el programari, amb prou precisió perquè es pugui desenvolupar, i servir de base

*** Per exemple, programari que s'ha de substituir, programari que fa una funció semblant, etc.**

per a un contracte, explícit o no, entre l'equip de desenvolupament del programari i els seus futurs usuaris.

El **disseny** és l'etapa següent. Si l'anàlisi especifica el problema o "què ha de fer el programari", el disseny especifica una solució a aquest problema o "com el programari ha de fer la seva funció".

Del programari cal dissenyar diversos aspectes diferenciats: la seva arquitectura general, les estructures de dades (base de dades, etc.), l'especificació de cada programa i les interfícies amb l'usuari*, i s'ha de fer de manera que a partir de tot això es pugui codificar el programari, d'una manera semblant a com es pot construir un edifici o una màquina a partir d'uns plànols.

* Per exemple, menús, dissenys de pantalla, llistats, etc.

El document resultant és l'"Especificació del disseny". Dins l'etapa de disseny és el millor moment per a elaborar l'"Especificació de la prova", que descriu amb quines dades s'ha de provar cada programa o grup de programes i quins són els resultats esperats en cada cas.

La **programació** o **codificació**, que és la quarta etapa, consisteix a traduir el disseny a codi processable per l'ordinador.

La prova és l'última etapa del desenvolupament del programari i la penúltima del model de cicle de vida del programari que hem considerat.

L'**etapa de prova** consisteix a provar el programari des de diversos punts de vista d'una manera planificada i, naturalment, localitzar i corregir dins el programari i la seva documentació els errors que s'hi detectin.

La prova es duu a terme en les dues fases següents:

- En la primera es fan proves, primer per cadascun dels programes separadament i després per grups de programes directament relacionats, i s'aplica l'especificació de la prova que hem esmentat abans.

- En la segona fase es comprova que el conjunt de programes doni els resultats que s'esperen i que ho faci amb el rendiment desitjat.

El primer equip de desenvolupament fa la darrera fase de la prova, i si els resultats són satisfactoris es lliura el programari al client, el qual pot fer una prova semblant pel seu compte i amb les seves dades, amb vista a decidir si accepta el programari. Amb l'acceptació pel client es dóna per acabat el desenvolupament.

La darrera etapa del cicle de vida és el **manteniment** o, si es prefereix, **explotació**, del programari, ja que tant de temps com es faci servir el programari caldrà mantenir-lo, és a dir, fer canvis –petits o grans– per a corregir errors, millorar les funcions o l'eficiència, o adaptar-lo a un nou maquinari o a canvis en les necessitats d'informació.

Com que un programari pot estar en explotació deu anys o més, sovint el cost total del manteniment durant la vida del programari és de dues a cinc vegades més gran que el cost de desenvolupament.

2.1.2. El cas de llenguatges de quarta generació

Els llenguatges –o més aviat entorns de programació– de quarta generació, són de molt alt nivell (en el sentit que sovint una sola de les seves instruccions equival a moltes instruccions del llenguatge assemblador) i en gran part són no procedimentals i estan integrats a un gestor de bases de dades relacionals; inclouen eines de dibuix de pantalles, generació de llistats i de vegades sortides gràfiques i full de càlcul. Alguns poden generar codi en un llenguatge de tercera generació.

Per a aplicacions senzilles, es pot passar directament dels requisits a la codificació, però en projectes complexos cal dur a terme una etapa de disseny, si bé simplificada.

2.2. Els cicles de vida iteratius i incrementals

El cicle de la vida en cascada ha estat molt criticat i s'han proposat alguns models alternatius.

2.2.1. Inconvenients del model de cicle de vida en cascada

L'inconvenient del model de cicle de vida en cascada és que no és realista.

Com s'ha vist, el model de cicle de vida en cascada comporta que les successives etapes del desenvolupament es fan de manera lineal, de manera que una fase no comença mentre no s'hagi acabat l'anterior i no es torna mai enrere. També queda implícit en el model que quan s'acaba una fase se sap almenys aproximadament quin percentatge del projecte resta per a fer, ja que si l'anàlisi s'ha completat i el seu resultat és cent per cent fix certament es pot estimar amb certa precisió la durada del disseny i, fins i tot, de la programació.

Vegeu el cicle de vida en cascada en l'apartat 2.1 d'aquest mòdul didàctic.

Ara bé, a la realitat és possible que l'especificació del sistema sigui fiable pel que fa a les funcions, ja que no s'espera que es descriguin fil per randa; tanma-

teix, precisament per això darrer, el cost i la durada del projecte s'han calculat sobre una base molt poc sòlida i tenen un gran marge d'error.

Però el problema més greu es presenta en l'anàlisi de requisits, pel fet que els requisits gairebé sempre o bé són incomplets al principi o bé canvien abans que s'hagi acabat de construir el programari, i sovint passen totes dues coses alhora. I si l'especificació de requisits és incompleta i insegura, és obvi que el disseny i la programació tindran problemes i, sobretot, retards i augments de cost importants per la feina no prevista que caldrà fer i també per la que caldrà refer.

Hi ha dues raons per les quals és pràcticament impossible d'elaborar uns requisits complets i estables en el primer intent:

a) En primer lloc, és difícil de trobar un conjunt de futurs usuaris que coneguin prou l'entorn en què s'ha de fer servir el programari, que hagin reflexionat prou sobre allò que volen aconseguir i, a més, que es posin d'acord.

b) En segon lloc, perquè el treball de consolidació de les peticions d'aquests usuaris mai no serà perfecte.

En qualsevol cas hem de comptar amb el fet que un cop acabada oficialment l'etapa d'anàlisi i començada la de disseny encara sorgiran requisits nous i canvis en els ja existents.

Què es pot fer, doncs? Sembla que l'opció més raonable sigui estudiar a fons una petita part dels requisits que tingui una certa autonomia, i dissenyar-la, programar-la i provar-la, i un cop el client l'hagi donada per bona, fer el mateix amb una altra part, i una altra; si partim d'un programari ja construït en part, es pot esperar que la idea que ens fem dels requisits restants pugui ser cada vegada més precisa i que també obtinguem una estimació cada cop més segura del cost i la durada del projecte complet; això és el que anomenem **cicle de vida iteratiu i incremental** (iteratiu perquè es repeteix dins un mateix projecte i incremental perquè procedeix per parts). I caldrà considerar normal que de vegades quan se'n construeixi una part es vegi que cal modificar-ne una de feta anteriorment.

La necessitat d'estimar el cost i els terminis

Els inconvenients del model del cicle de vida clàssic esmentats en aquest subapartat no volen pas dir que no hi pugui haver termini o límit de cost per a un projecte de desenvolupament de programari; simplement, s'ha de reconèixer que no és realista de creure que es pot fixar exactament la funcionalitat, el cost i la durada del projecte, tot alhora. Si el programari ha de funcionar en una data determinada i no es pot augmentar la despesa en personal, caldrà estar disposat a acceptar que el programari no faci totes les funcions desitjades; si la funcionalitat i la data de lliurament del programa són innegociables, caldrà augmentar el nombre de programadors o analistes. I això no és cap renúncia en relació amb els resultats que s'aconseguien fins ara, perquè a la pràctica ben pocs eren els projectes en què no es produïen desviacions quant a funcionalitat, pressupost o termini, si no en diverses d'aquestes coses alhora.

Per tant, el model de cicle de vida en cascada pot ser vàlid si s'aplica de manera que cada etapa, de l'anàlisi de requisits a la prova, no prevegi tot el conjunt del programari, sinó sols una part cada vegada; llavors tindríem un cicle de vida iteratiu i incremental basat en el cicle de vida en cascada.

2.2.2. El cicle de vida amb prototipatge

Per tal d'ajudar a concretar els requisits es pot recórrer a construir un prototip del programari.

Un **prototip** és un programari provisional, construït amb eines i tècniques que donen prioritat a la rapidesa i a la facilitat de modificació abans que a l'eficiència en el funcionament, que només ha de servir perquè els usuaris puguin veure com seria el contingut o l'aparença dels resultats d'algunes de les funcions del futur programari.

Un prototip serveix perquè els usuaris puguin confirmar que efectivament allò que se'ls mostra és el que necessiten o bé ho puguin demanar per comparació, i llavors es prepara una nova versió del prototip i es té en compte les indicacions dels usuaris i se'ls ensenya una altra vegada. Un cop el prototip ha permès de concretar i confirmar els requisits, es pot començar un desenvolupament segons el cicle de vida en cascada, en aquest cas, però, partint d'una base molt més sòlida.

Característiques del cicle de vida amb prototipatge

El cicle de vida amb prototipatge no es pot considerar plenament un cicle de vida iteratiu i incremental, ja que només el prototip s'elabora de manera iterativa, i no necessàriament incremental; tanmateix, és un model de cicle de vida que pot ser adequat en alguns casos, especialment quan n'hi ha prou de prototipar un nombre reduït de funcions perquè les altres siguin prou semblants a aquestes perquè les conclusions a què s'arribi amb el prototip també els siguin aplicables.

2.2.3. La programació exploratòria

La programació exploratòria consisteix a elaborar una primera versió del programari, o d'una part d'aquest, ensenyar-la als usuaris perquè la critiquin i tot seguit fer-hi els canvis suggerits per ells, i això tantes vegades com calgui.

La diferència principal amb el prototipatge és que aquí el programari és real des del començament.

Característiques de la programació exploratòria

La programació exploratòria es pot considerar un cicle de vida iteratiu, però no pas incremental, ja que el programari és complet des de la primera versió. A conseqüència de les nombroses modificacions que sofreix, la qualitat del programari desenvolupat així i de la seva documentació tendeix a ser deficient, com la d'un programari que hagi experimentat un manteniment llarg i intens.

2.2.4. El cicle de vida del Rational Unified Process

Aquest cicle de vida ha estat proposat per l'empresa Rational Software com a marc per al desenvolupament de programari que fa servir les seves eines. És clarament un cicle de vida iteratiu i incremental.

S'hi distingeixen aquestes quatre etapes (anomenades *fases*):

1) Inici (***inception***), en què s'estableix la justificació econòmica del programari i es delimita l'abast del projecte.

2) Elaboració, en la qual s'estudia el domini del problema, o simplement domini (part de l'activitat de l'empresa dins la qual es farà servir el programari) i té en compte moltes de les necessitats d'informació i eventuals requisits no funcionals i restriccions, s'estableix l'arquitectura general del programari i es fa una planificació del projecte.

3) Construcció, en què es desenvolupa tot el producte de manera iterativa i incremental, té en compte totes les necessitats d'informació que ha de satisfer i desenvolupa l'arquitectura obtinguda en la fase anterior.

4) Transició, que comprèn el lliurament del producte al client i el començament de la seva utilització; encara pot ser que calgui fer retocs al programari i afegir-hi noves funcions com a conseqüència d'errors detectats o de requisits que s'havien passat per alt fins aleshores.

A cadascuna d'aquestes fases es duen a terme (en diferents proporcions) els components següents de procés:

- recollida de requisits (*requirement capture*),
- anàlisi i disseny,
- realització (*implementation*),
- prova (*test*).

Cada unitat en què s'executa poc o molt dels components de procés és una **iteració**, i s'aplica a un nou fragment de programari. Totes les fases tenen iteracions.

Ara convé que feu els exercicis d'autoavaluació 4 a 10 d'aquest mòdul didàctic.

3. Desenvolupament estructurat i desenvolupament orientat a objectes

Els mètodes de desenvolupament de programari més utilitzats fins ara pertanyen a dos grans grups: els mètodes estructurats i els mètodes orientats a objectes.

3.1. Els mètodes estructurats

Els mètodes estructurats provenen de la programació estructurada i fan servir tècniques no gaire integrades entre si.

Tècniques emprades pels mètodes estructurats

Les tècniques més usades en els mètodes estructurats són segurament els diagrames d'entitat-relació i de flux de dades, amb les seves variants. El primer fa referència a les dades i el segon, als processos.

Els mètodes estructurats tenen, a més, aquestes característiques:

- L'especificació dels processos i la de les estructures de dades generalment queden força diferenciades, i hi ha mètodes que posen més èmfasi en aquells o en aquestes.

- Moltes de les seves tècniques o bé passen d'allò que és general a allò que és particular (tècniques *top-down*) o bé a l'inrevés (tècniques *bottom-up*).

3.2. Els mètodes orientats a objectes

Per bé que els mètodes estructurats continuen essent àmpliament utilitzats, els anomenats *mètodes orientats a objectes* guanyen terreny ràpidament.

Si els mètodes estructurats de desenvolupament de programari tenen el seu origen en la programació estructurada, els mètodes orientats a objectes tenen les seves arrels en la programació orientada a objectes.

És lògic que hagi estat així en tots dos casos, ja que una nova tècnica de programar demana una nova manera de dissenyar els programes adaptada a les característiques de la programació, i un nou mètode de disseny fa desitjable un nou mètode d'anàlisi de requisits que tingui la mateixa orientació que el disseny, per tal d'evitar que el pas de l'anàlisi de requisits al disseny impliqui un canvi de model que inevitablement comportaria un treball addicional i un risc d'errors més gran.

De la mateixa manera que la programació orientada a objectes gira al voltant del concepte de *classe*, també hi giren l'anàlisi de requisits i el disseny. Per aquesta raó, el diagrama bàsic d'aquests mètodes, el **diagrama de classes i objectes**, es fa servir tant en l'anàlisi com en el disseny; a més, moltes de les classes descrites en l'anàlisi de requisits s'implementen en els programes passant pel disseny, cosa que fa que el pas de l'anàlisi de requisits al disseny sigui més suau que en els mètodes estructurats i també més senzill i ràpid.

Com que dins una classe hi ha alhora atributs i operacions, és a dir, dades i processos, en el desenvolupament orientat a objectes a mesura que es defineixen i implementen classes s'avança alhora en aquestes dues dimensions. El desenvolupament no procedeix ni de manera *top-down* ni *bottom-up*, sinó que, més aviat, es construeixen grups de classes interrelacionades, sovint per nivells: classes que gestionen la presentació de la informació, o bé les entrades per pantalla, o bé les lectures i enregistraments de la base de dades, o bé els algorismes principals del programari, o bé segons un cicle de vida incremental, tal com hem vist.

Val a dir que el desenvolupament orientat a objectes, a més d'introduir tècniques noves, també aprofita algunes tècniques i conceptes del desenvolupament estructurat, com el diagrama d'estats i transicions, segons veurem.

Hi ha dues **característiques del desenvolupament orientat a objectes** que probablement han afavorit de manera decisiva la seva expansió fins ara i probablement la continuaran afavorint:

- Sembla que permet –per primera vegada en la història de la tecnologia del programari– la **reutilització de programari** en un grau significatiu, en forma de classes implementades, cosa que podria significar una via per a solucionar, si més no en part, els problemes de productivitat i qualitat descrits en apartats anteriors.
- La seva relativa simplicitat facilita el **desenvolupament d'eines informàtiques d'ajuda al desenvolupament**; aquest factor podria ser potenciat pel fet que aquests últims anys ha aparegut una notació orientada a objectes molt àmpliament acceptada, UML.

3.3. Els mètodes formals

Els anomenats *mètodes formals* parteixen d'una especificació de les necessitats d'informació en termes d'un model matemàtic rigorós, del qual es podria deduir el programa que les satisfaci; també permetrien de demostrar matemàticament que un programa és correcte en el sentit que s'ajusta a aquelles necessitats.

Tot i que haurien de permetre d'eliminar les ambigüitats i mancances dels mètodes no tan rigorosos, la seva utilització directa en el desenvolupament de programari per a usos reals és poc freqüent ara com ara, sens dubte per la gran complexitat que tindria un modelatge tan detallat i formalitzat en casos reals.

Alguns dels llenguatges d'especificació formal més coneguts són Z, VDM, CSP i LARCH.

4. Les eines CASE

"CASE" vol dir *Computer-Aided Software Engineering*. Les eines CASE són programari de suport al desenvolupament, manteniment i documentació informatitzats de programari.

D'aquesta definició generalment s'exclouen aquelles eines que tenen una de les funcions següents:

1) o bé no són només per a aquesta finalitat (eines de tractament de text, de full de càlcul, de dibuix en general, de planificació de projectes de qualsevol enginyeria), ja que pròpiament pertanyen a d'altres àmbits;

2) o bé es fan servir directament per a codificar el programari (compiladors, entorns de quarta generació, editors ordinaris de programes, etc.), ja que sempre hi són presents, fins i tot quan el desenvolupament de programari es fa de la manera més manual possible.

Resten, doncs, principalment les eines que ajuden a aplicar tècniques concretes de desenvolupament i manteniment de programari i per això gestionen informació sobre els elements i conceptes que es fan servir en els mètodes de desenvolupament, com ara les següents:

Eines *UpperCASE* i *LowerCASE*

De vegades es distingeix entre eines *UpperCASE*, que són les d'anàlisi i disseny, i *LowerCASE*, que es fan servir durant la programació i la prova.

- Les eines diagramàtiques, les quals, a diferència de les eines de dibuix, saben que un determinat símbol és una classe i no simplement un rectangle. Aquestes eines també acostumen a acceptar documentació textual sobre aquells elements.

- Les eines de gestió de la prova i de gestió de la qualitat en general.

- Les eines de gestió de canvis, etc.

La importància de la integració de les eines

És convenient que les eines que donen suport a diferents tècniques utilitzades dins el mateix mètode estiguin integrades, en el sentit que si hi ha un tipus d'element que és comú a dues tècniques sigui compartit per les dues eines respectives, de manera que només calgui descriure'l una vegada i que tots els canvis que es facin després en aquesta descripció arribin a totes dues.

L'expansió de l'ús d'eines CASE en el mètode estructurat va ser frenada per la diversitat i manca d'estandardització de les tècniques que es fan servir; en els mètodes orientats a objectes, en canvi, actualment la situació és la contrària: d'una banda, alguns diagrames serveixen tant per a l'anàlisi com per al disseny, i de l'altra, s'ha produït una estandardització de les tècniques i notacions

Vegeu els mètodes orientats a objectes en el subapartat 3.2 d'aquest mòdul didàctic.

en el model conegut com a UML que ha fet que en el poc temps transcorregut d'ençà de la seva publicació hagin aparegut un nombre important de conjunts integrats d'eines CASE basades en aquest; aquest suport informatitzat, ampli i creixent, al desenvolupament de programari orientat a objectes sens dubte reforçarà la millora de la qualitat i la productivitat en el desenvolupament de programari que, segons hem vist, la tecnologia orientada a objectes ha de fomentar. 

5. L'OMG i l'UML

Per al desenvolupament orientat a objectes farem servir el model anomenat UML, del qual actualment és responsable l'organització anomenada OMG.

5.1. L'Object Management Group (OMG)

L'Object Management Group (OMG), creat el 1989, és una organització no lucrativa en què participen més de vuit-centes grans empreses de programari, de maquinari, usuàries i consultores, i té la finalitat de fomentar l'ús de la tecnologia d'objectes i impulsar la introducció de programari orientat a objectes que ofereixi reusabilitat, portabilitat i interoperabilitat en entorns distribuïts heterogenis.

L´altre estàndard de l'OMG

A més d'UML, un altre estàndard que ha elaborat l'OMG és CORBA, sobre objectes distribuïts, les implementacions del qual tenen una expansió ràpida.

El mitjà amb què l'OMG intenta d'aconseguir els seus objectius és l'elaboració d'estàndards, per als quals accepta propostes; en canvi, no produeix programari ni elabora especificacions d'implementació o funcionalitat.

5.2. Unified Modeling Language (UML)

L'Unified Modeling Language (UML) és un model per a la construcció de programari orientat a objectes que ha estat proposat com a estàndard d'ISO per l'OMG. Consta d'un conjunt de tipus de diagrames interrelacionats, dins els quals s'empren elements del model, que serveixen per a descriure diversos aspectes de l'estructura i la dinàmica del programari.

UML és el resultat d'una certa unificació dels models utilitzats en tres mètodes preexistents de desenvolupament de programari orientat a objectes fets per llurs autors en col·laboració. Aquests mètodes són els següents:

- el mètode de Grady Booch;
- l'OMT, de Jim Rumbaugh i altres;
- l'OOSE, d'Ivar Jacobson.

A més, hom hi troba conceptes aportats per força altres autors, entre ells Peter Coad, Edward Yourdon, James Odell i Bertrand Meyer.

Evolució del model UML

Les primeres passes cap al model unificat es van donar l'any 1994 en què Booch i Rumbaugh, treballant a Rational Software Corporation, van començar la unificació dels models respectius, i l'octubre de 1995 es va publicar la versió provisional 0.8 del llavors anomenat *Unified Method*.

El mateix any, Jacobson es va incorporar amb la seva empresa a l'equip esmentat i a Rational, i com a resultat del treball de tots tres autors el 1996 van sortir les versions 0.9 i 0.91 d'UML. L'OMG va emetre en aquella època una *Request For Proposal*, per a un model d'aquesta mena, i llavors Rational, per a respondre-hi, va constituir un consorci amb altres organitzacions, amb el resultat que el gener de 1997 es va presentar a l'OMG la versió 1.0 d'UML.

D'altres empreses que havien presentat també respostes independentment es van afegir al consorci i es va publicar la versió 1.1, que va ser acceptada per l'OMG el novembre de 1997 (hi va haver una altra proposta, la del model OML, que tenia i encara té un nombre important de partidaris). L'OMG va encarregar-ne una revisió, com a resultat de la qual hi va haver una versió 1.2, no publicada, i la versió 1.3, ja publicada com a estàndard.

Amb UML s'ha arribat a un model orientat a objectes únic com a model oficial, però això no vol dir que s'hagi arribat a un mètode únic de desenvolupament orientat a objectes; la veritat és que ara com ara sembla que falta bastant per a arribar-hi, si és que mai s'hi arriba. És a dir, que el que s'ha aconseguit és que hi hagi uns diagrames que tots els desenvolupadors de programari orientat a objectes entendran i faran de la mateixa manera, cosa que és un avenç realment important sobre la situació anterior en què cada mètode tenia la seva notació gràfica; però, tot i això, continua essent possible que hi hagi mètodes diferents que facin ús d'UML i que, per exemple, facin servir els mateixos diagrames en ordre diferent o dins models de cicle de vida diversos.

Vegeu el subapartat 1.4 d'aquest mòdul didàctic.

Arribats en aquest punt, convé que respongueu els exercicis d'autoavaluació 11 a 13 d'aquest mòdul didàctic.

Resum

S'ha concretat què s'entén per *programari* en aquesta assignatura, s'ha indicat per què es pot considerar un producte industrial i, consegüentment, per què té sentit parlar d'una enginyeria del programari, i hem entrat en contacte amb els dos grans problemes que han afectat tradicionalment el desenvolupament de programari: les mancances quant a productivitat i qualitat.

Si ja entrem dins el procés d'elaboració del programari, hem vist el concepte de cicle de vida i els seus dos grans models –el cicle de vida en cascada o clàssic i els cicles de vida iteratius i incrementals–, més dues modalitats intermèdies –el desenvolupament amb prototipatge i la programació exploratòria. Com a cicles concrets hem estudiat d'una manera més o menys detallada el cicle de vida del Rational Unified Process, com a representant de les tendències actuals vers cicles iteratius i incrementals, i el cicle de vida clàssic, l'interès del qual ve, a més de la seva importància històrica, del fet que alguns dels seus conceptes continuen essent vàlids encara.

A continuació, s'han presentat les tres grans famílies de mètodes de desenvolupament de programari: les dues més utilitzades en entorns reals, que són els mètodes estructurats –més tradicionals– i els mètodes orientats a objectes –en plena expansió–, sense menystenir els mètodes formals, interessants per llur gran rigor teòric.

Després de parlar de tècniques és lògic parlar de les eines que els donen suport; per això hem vist el concepte d'eines CASE i ens hem assabentat una mica de la seva situació actual i de l'evolució previsible del seu paper.

Com que per a estudiar l'anàlisi i disseny orientats a objectes –tema essencial d'aquesta assignatura– farem servir els conceptes i la notació d'UML, s'ha exposat l'origen d'aquest model i la comesa de l'organització responsable d'aquest, l'OMG.

Exercicis d'autoavaluació

1. Té sentit distingir entre enginyeria del producte i enginyeria del procés en el cas de l'enginyeria de programari?

2. Per què altres enginyeries no passen, actualment, una crisi anàloga a la crisi del programari?

3. Comenteu l'afirmació "programari és igual a conjunt de programes".

4. Quina relació hi ha entre el cicle de vida del programari i l'enginyeria del programari?

5. Per què el cicle de vida clàssic també s'anomena *cicle de vida en cascada*?

6. Quines són les fases del cicle de vida clàssic?

7. Per què hi ha un model de cicle de vida especial quan les eines de desenvolupament del programari són de quarta generació?

8. Per què s'afirma que el cicle en cascada no és realista?

9. El cicle de vida amb prototipatge, és iteratiu? I incremental? Justifiqueu la resposta.

10. Quines són les fases del Rational Unified Process?

11. Un compilador, és una eina CASE?

12. Per què són més viables les eines CASE per al desenvolupament orientat a objectes que per al desenvolupament estructurat?

13. Resumiu la finalitat de l'OMG.

Solucionari

Exercicis d'autoavaluació

1. Com que l'enginyeria de procés considera la producció en sèrie de còpies idèntiques (o tones, litres, etc.) del producte un cop desenvolupat, no hi ha pròpiament enginyeria de procés del programari.

Vegeu el subapartat 1.3.

2. Perquè les enginyeries més antigues han arribat a aconseguir, en termes generals, una tecnologia força satisfactòria en termes de qualitat i productivitat; la utilització d'elements reusables o estandarditzats és un factor important.

3. Un programari és més que un simple conjunt de programes: cal que els programes treballin interrelacionats i, a més, també inclou les estructures de dades i la documentació relatives a aquests programes.

Vegeu el subapartat 1.1.

4. L'enginyeria del programari comprèn els mètodes per al desenvolupament del programari, i un mètode necessàriament es compon d'etapes, és a dir, té un cicle de vida.

Vegeu el començament de l'apartat 2.

5. Per dues raons: perquè cada etapa comença allà on va acabar l'anterior, i perquè teòricament no es repeteix mai cap etapa.

Vegeu el subapartat 2.1.

6. Anàlisi prèvia, anàlisi de requisits, disseny, programació, prova i manteniment.

Vegeu el subapartat 2.1.1.

7. Perquè en projectes d'informàtica de gestió petits i mitjans el cicle de vida clàssic es pot simplificar si es fan servir eines de quarta generació, ja que es pot saltar l'etapa de disseny.

Vegeu el subapartat 2.1.2.

8. Perquè pressuposa que un cop acabada una etapa ja no s'hi torna més, i a la pràctica generalment es continuen descobrint o modificant requisits quan ja s'està a les etapes de disseny i programació.

Vegeu el subapartat 2.2.1.

9. Es pot considerar iteratiu perquè el prototip es pot modificar tantes vegades com calgui fins que correspongui a allò que els usuaris volen. No és incremental perquè ni el prototip ni el programari definitiu es construeixen per parts.

Vegeu el subapartat 2.2.2.

10. Inici, elaboració, construcció i transició.

Vegeu el subapartat 2.2.4.

11. No, perquè es considera que les eines CASE serveixen per a aconseguir un desenvolupament més o menys informatitzat, i un compilador hi ha de ser sempre que es faci servir el llenguatge de programació en qüestió, encara que el desenvolupament sigui totalment manual.

Vegeu l'apartat 4.

12. Perquè en el desenvolupament orientat a objectes hi ha molta menys diversitat de conceptes i eines que cal suportar que en el desenvolupament estructurat; a més, d'ençà que el model UML ha esdevingut estàndard per a aquests conceptes hi ha una sola representació gràfica.

Vegeu l'apartat 4.

13. L'OMG té com a finalitat el foment de la tecnologia orientada a objectes per mitjà de l'establiment d'estàndards.

Vegeu l'apartat 5.

Glossari

CASE
Sigla de Computer-Aided Software Engineering.

cicle de vida
Conjunt d'etapes del desenvolupament de programari per les quals es passa en l'ordre establert prèviament.

cicle de vida clàssic
Cicle de vida en el qual se suposa que cada etapa es basa en els productes de l'anterior i no es torna mai a etapes anteriors.

cicle de vida en cascada
Sinònim de *cicle de vida clàssic*.

cicle de vida iteratiu i incremental
Tot cicle de vida en el qual es passa de manera repetida per les mateixes fases desenvolupant cada vegada un tros més de programari.

desenvolupament de programari orientat a objectes
Desenvolupament de programari amb mètodes centrats en el concepte d'objecte derivats de la programació orientada a objectes.

desenvolupament estructurat de programari
Desenvolupament de programari amb mètodes derivats de la programació estructurada i històricament anteriors als orientats a objectes.

eines CASE
Programari de suport al desenvolupament, manteniment i documentació informatitzats de programari.

enginyeria del programari
Conjunt de les tècniques, mètodes i eines que es fan servir per a produir programari.

mètodes formals de desenvolupament de programari
Mètodes de desenvolupament que es basen en l'especificació dels requisits en terme d'un formalisme matemàtic rigorós.

OMG
Sigla d'*Object Management Group*. Organització no lucrativa d'empreses productores i consumidores de béns i serveis informàtics que té la finalitat de fomentar l'ús de la tecnologia d'objectes, i el mitjà amb què intenta d'aconseguir-ho és l'elaboració d'estàndards.

prototip
El prototip d'un sistema de programari és una versió provisional del sistema, que només té l'imprescindible perquè l'usuari pugui comprovar si s'han entès bé els seus requisits.

UML
Sigla d'*Unified Modeling Language*. Model estàndard per a la construcció de programari orientat a objectes.

Bibliografia

Pressman, R.S. (1997). *Ingeniería del software. Un enfoque práctico* (4a. ed.). Madrid: McGraw-Hill.

Sommerville, I. (1995). *Software Engineering* (5a. ed.). Harlow: Addison-Wesley.

Kruchten, P. (2000). *The Rational Unified Process. An Introduction* (2a. ed.). Addison-Wesley.