INTRODUCCIÓ A L'ENGINYERIA DEL SOFTWARE ORIENTADA A OBJECTES

EL SOFTWARE

Un *sistema de software*, també anomenat *aplicació*, *programari* o simplement *software*, és un conjunt integrat de programes, bases de dades, documentació, ... , que permeten a un *sistema informàtic* realitzar les tasques per les quals ha estat dissenyat.

PROPIETATS DESITJABLES DEL SOFTWARE

Un bon software:

1. Ha de poder ser fàcilment mantingut: ha d'estar ben dissenyat i documentat.

2. Ha de ser fiable i robust: ha de fer el que s'espera d'ell, i no fallar més del que estigui especificat.

3. Ha de ser eficient, especialment en el sentit que no ha de malgastar recursos del sistema (memòria, temps de càlcul, disc,…).

4. Ha de tenir una bona interfície. Cal tenir presents els usuaris que hauran d'utilitzar el software, i incorporar-los en el procés de desenvolupament *(user-centered design).*

TRETS DIFERENCIALS DEL SOFTWARE

Tot i que la informàtica necessita components maquinari i components software, entre els dos hi ha diferències importants a l'hora de la seva construcció i manteniment:

1. El software es desenvolupa; no es "fabrica" en el sentit clàssic.

2. El software no es fa malbé. A diferència del hardware, el software no envelleix ni s'espatlla. Una altra qüestió és que canviïn els requeriments al llarg del temps i que calgui adaptar el software.

3. Encara ara molt del software es desenvolupa a mida, a diferència d'altres enginyeries en les que la cultura de la reutilització de components està més estesa.

ETAPES DE L'EVOLUCIÓ DEL SOFTWARE

L'evolució del software es pot representar per mitjà d'una sèrie d'etapes.

Etapa 1: Fer programes és un "art": el software es fa a mida; hi ha poques persones implicades; el disseny és implícit en les aplicacions (o en el cap d'algú); poca documentació i distribució limitada.

Etapa 2: El software comença a ser un producte; apareixen empreses de software; aplicacions estàndard; aplicacions molt grans, però poc documentades i estructurades. Tot això portarà a la crisi del software i a plantejar-se la necessitat de nous enfocaments en el desenvolupament del software.

Etapa 3: Expansió dels ordinadors personals i, en conseqüència, poden començar a vendre's centenars de milers de còpies de programes. El software esdevé un punt important a l'hora de marcar diferències entre equips similars.

Etapa 4: Orientació a objectes, aplicacions distribuïdes, agents, ...

El terme Enginyeria del Software es va utilitzar per primera vegada el 1968, com a nom d'una conferència organitzada per la OTAN, per intentar posar fi al que ja es coneixia com la crisi del software deguda, bàsicament, al desenvolupament de grans aplicacions sense cap metodologia establerta, cosa que feia molt costós el seu manteniment.

L'ENGINYERIA DEL SOFTWARE

L'*Enginyeria del Software* és una disciplina que integra *processos*, *mètodes* i *eines* amb l'objectiu de desenvolupar i mantenir sistemes de software que siguin econòmics, fiables i eficients.

- Els processos defineixen un marc de treball que permeten un desenvolupament racional de la enginyeria del software. Formen la base del control de gestió dels projectes de software i estableixen el context en el qual s'apliquen els mètodes i es produeixen els resultats del treball: models, documents, dades, informes, formularis, etc.

- Els mètodes indiquem com construir el software. Ofereixen tècniques de modelatge per a les diferents etapes del procés: definició de requeriments, anàlisi, disseny, implementació, proves, explotació i manteniment.

- Les eines proporcionen suport automàtic o semiautomàtic per al procés i els mètodes. Les eines **CASE** (Computer assited system enginery) ajuden a l'automatització de tot el procés de construcció del software, ajudant a l'obtenció de resultats d'alta qualitat.

L'enginyeria del software és una activitat de *modelat* per a la solució dels problemes plantejats pel client del sistema. A una primera etapa, d'investigació, els conceptes del mon real es representen per mitjà d'un conjunt de models - construïts fent servir el llenguatge del client - que conformen l'anomenat *domini del problema.* A la segona etapa, de creació, es fan servir els models del domini del problema per a construir – en el llenguatge dels desenvolupadors del sistema - els models (de disseny i d'implementació) que resolen els problemes plantejats pel client i que conformen l'anomenat *domini de la solució*.

El resultat final del procés de desenvolupament d'un sistema de software ha de ser un producte que ha d'ajustar-se als requeriments i s'ha d'haver obtingut en els terminis i amb els costos previstos. Tot i els grans avenços que en el darrers anys hi ha hagut en el camp de l'enginyeria del software, tant la qualitat dels programes elaborats com la productivitat durant el seu procés d'elaboració no han assolit nivells comparables als d'altres enginyeries més antigues.

Els enginyers de software han d'adoptar una metodologia sistemàtica i organitzada per fer el seu treball i utilitzar les eines apropiades segons el problema a solucionar, les limitacions del desenvolupament i els recursos disponibles.

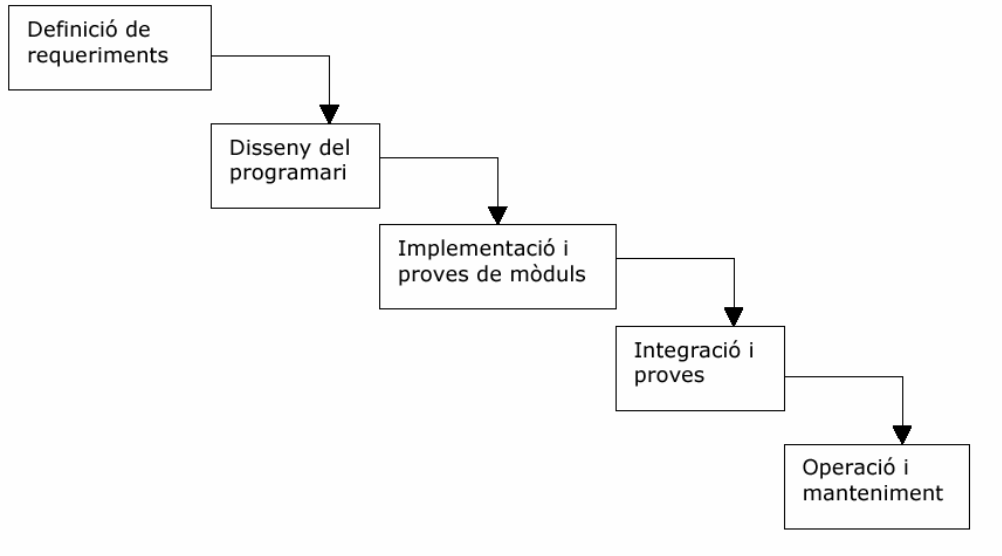
MODELS DEL PROCÉS DE DESENVOLUPAMENT DEL SOFTWARE

S'han plantejat diversos models pel desenvolupament (cicle de vida) del software, que comentarem breument tot seguit.

Model seqüencial

També conegut com a *cicle de vida clàssic* o *model del salt d'aigu*a, és el primer model que es va establir. Tot i que és molt simple (i no reflexa adientment la realitat), estableix clarament quines són les fases que apareixen en la producció del software.

Esquemàticament, el model seqüencial consta de les següents fases:



Els problemes d'aquest enfocament del cicle de vida clàssic són els següents:

1. És massa seqüencial i no respon a la vida real: la construcció del software acostuma a ser un procés iteratiu, amb refinaments progressius.

2. Acostuma a ser molt difícil poder establir tots els requeriments en la fase d'anàlisi inicial.

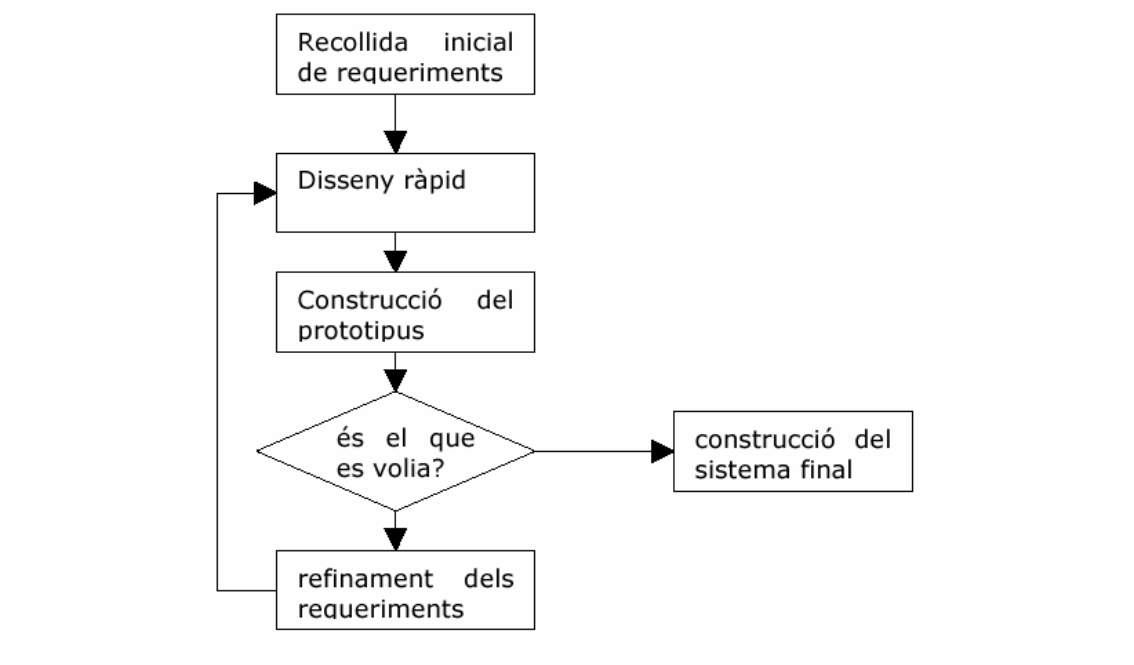
3. Es triga molt a tenir una versió operativa que pugui ser validada pel client.

Una manera de solucionar parcialment els dos primers problemes, passa per establir vies per retornar a etapes anteriors.

Construcció de prototipus

Quan no és possible establir d'entrada els requeriments de l'aplicació de forma complerta, és aconsellable abordar el problema construint prototipus.

Un esquema d'aquesta metodologia s'il·lustra tot seguit:



De fet, la construcció de prototipus no s'ha de veure com un model de construcció de software per si mateix, sinó com una metodologia per ajudar a establir els requeriments de l'aplicació (que correspondria al primer pas del model seqüencial).

Els dos principals problemes que poden aparèixer amb aquest enfocament són:

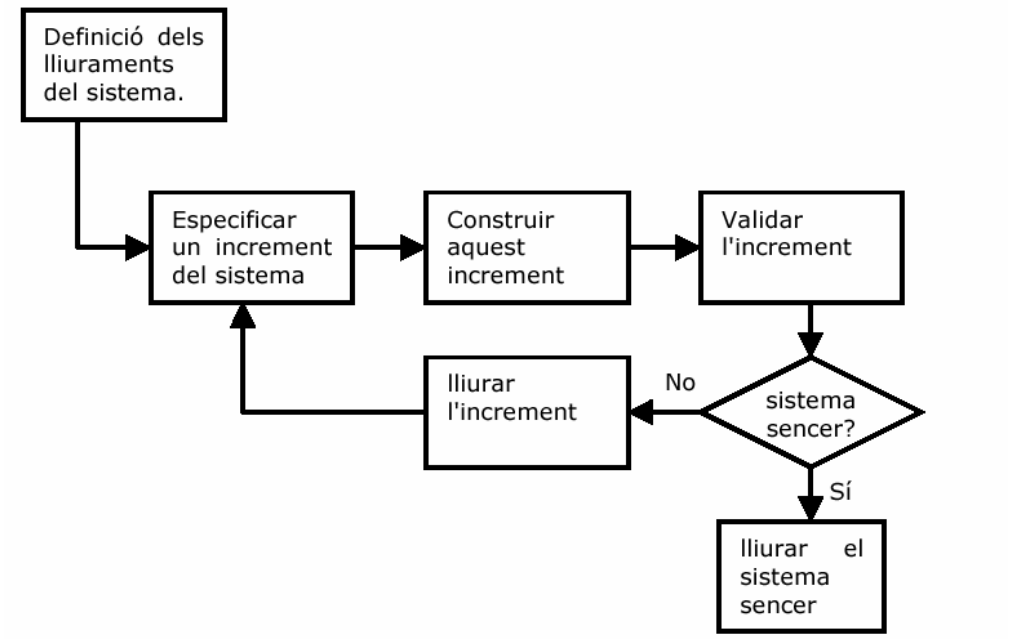
1. La possibilitat que es perdi de vista l'objectiu inicial del prototipatge (obtenir una bona especificació de requeriments), i que es faci servir el prototipus obtingut com a aplicació final.

2. Es prenen decisions d'implementació pensant en obtenir ràpidament un prototipus, però no es fa un replantejament a l'hora de desenvolupar l'aplicació final.

Una metodologia similar a la del prototipatge és la coneguda com a ***programació exploratòri***a, en la que partint d'un esbós dels requeriments es construeix una versió inicial del sistema, i aquest es va modificant o ampliant a mesura que va essent utilitzat. La diferència fonamental està en que aquí, els passos intermedis corresponen realment a aplicacions funcionant.

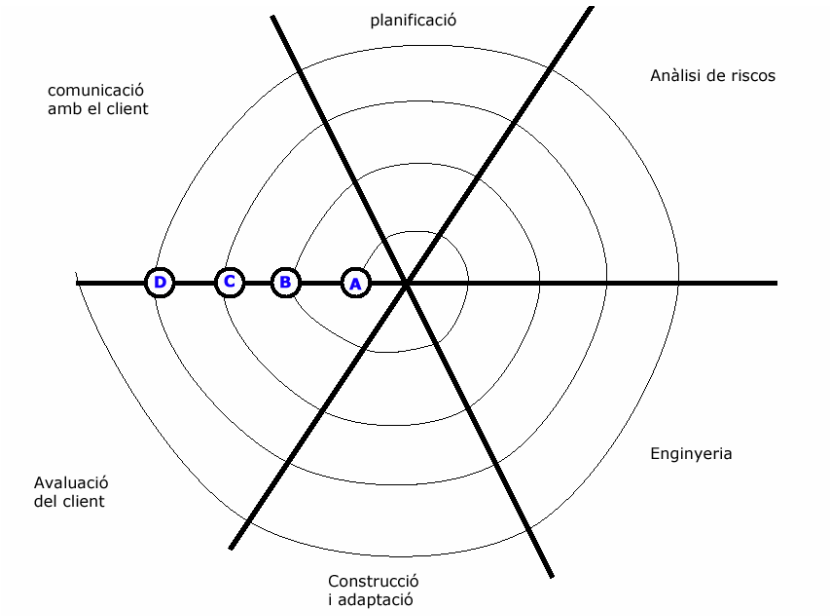
Programació incremental

La idea és establir una metodologia més precisa que en la programació exploratòria: en aquest cas no és tota l'aplicació que va evolucionant amb el temps, sinó que es planifica en quin ordre s'aniran lliurant parts senceres del sistema, i les parts lliurades ja no es modifiquen (exceptuant, evidentment, que hi hagi errors). Els comentaris dels usuaris sobre les parts que ja s'han lliurat s'incorporen en l'anàlisi i disseny dels components que encara no s'han lliurat.



Model d'espiral

Proposat per Boehm, el 1988, inclou la idea que el desenvolupament del software és un procés iteratiu:



*Model d'espiral. **A**, inici desenvolupament conceptes. **B**, inici desenvolupament nous productes. **C**, inici millores de productes. **D**, inici manteniment de productes.*

METODES DE DESENVOLUPAMENT DEL SOFTWARE

Un mètode de desenvolupament del software determina la forma d'organitzar les activitats relacionades amb la creació, utilització i manteniment dels sistemes de software. Els mètodes de desenvolupament més utilitzats fins ara són els mètodes estructurats i els mètodes orientats a objectes.

Mètodes estructurats

Provenen de la programació estructurada. Per a modelar les dades s'acostuma a fer servir el *model entitat-relació*. Per a modelar els processos es fan servir *diagrames de flux* a diferents nivells de detall. L'especificació de les dades i dels processos queden generalment força diferenciades. Acostumen a Ser tècniques que passen del general al particular (*top-down*) o a l'inrevés (*bottom-up*). Entre els mètodes estructurats més coneguts hi trobem: SSADM, Merise, ...

Mètodes orientats a objectes

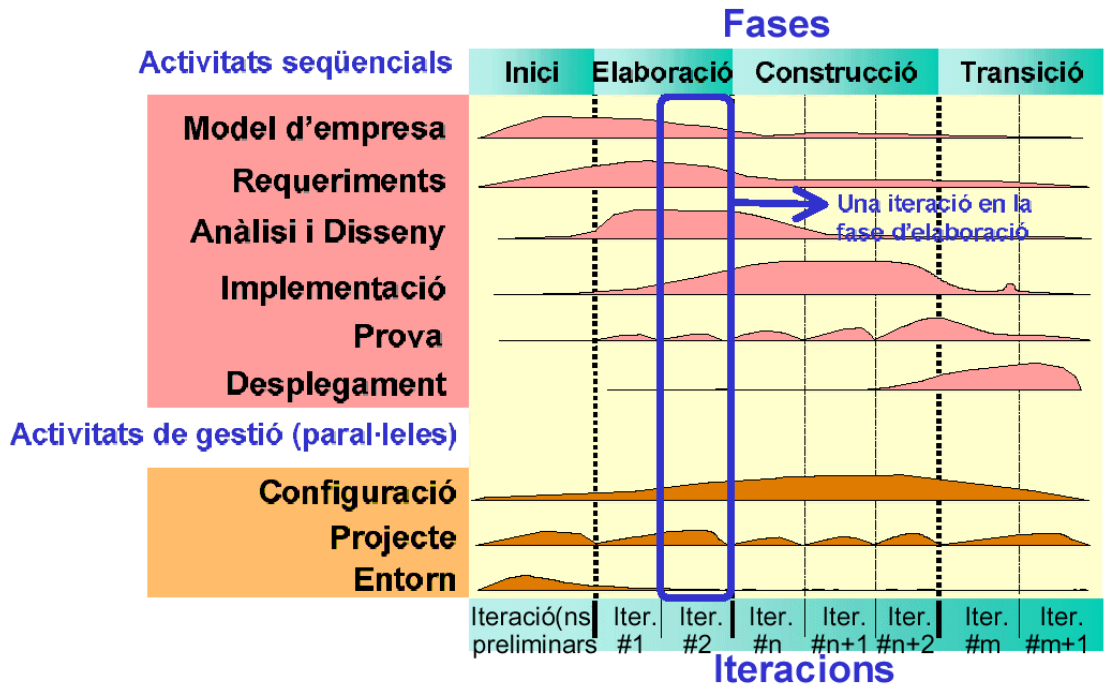
Tenen el seu origen en la programació orientada a objectes. Els diagrames bàsics d'aquests mètodes són els diagrames de classe. Moltes de les classes descrites a l'anàlisi es fan servir al disseny i finalment s'implementen, cosa que fa que durant tot el procés de desenvolupament es faci servir la mateixa representació. El procés no és ni top-down ni bottom-up, ja que allò que es fa és construir grups de classes interrelacionades per mitjà d'un procés incremental i interactiu. Entre els mètodes orientats a objectes hi trobem: UP, Iconix, FDD, ...

El Procés Unificat

El *Procés Unificat* (Unified process-UP) de desenvolupament del software és un mètode d'enginyeria de software proposat per Rational Rose. Ha estat desenvolupat amb l'objectiu de guiar als enginyers del software en les activitats necessàries per a construir software usant el *Llenguatge Unificat de Modelat* (UML). El mètode defineix les components que es faran servir per a construir el software i les interfícies que connectaran les components. Usant una combinació de desenvolupament incremental i iteratiu, el procés unificat defineix la funció del software des del punt de vista de l'usuari, aplicant un enfocament basat en escenaris. Després acobla la funció amb un marc de treball arquitectònic que identifica la forma que finalment tindrà el software.

El cicle vida, iteratiu i incremental, proposat per UP consta de 4 fases:

1. Inici (inception)
2. Elaboració,
3. Construcció,
4. Transició.



Totes les fases tenen iteracions i a cadascuna d'aquestes fases es duen a terme, en diferents proporcions, tasques (incrementals) de:

- recollida de requeriments del sistema,
- anàlisi i disseny,
- implementació,
- prova.

Iconix

Iconix és una simplificació del procés que cal seguir amb UP que, a més a més, completa alguns dels aspectes que amb UP no acaben de quedar prou clars.

Desenvolupament guiat per elements (FDD)

Els desenvolupament guiat per elements (*Feature Driven Development-FDD*) és un model basat en petites iteracions. Comença establint un model global del sistema a desenvolupar i a continuació es genera una llista d'*elements* (features). Els elements són petites funcionalitats – resultats tangibles - útils als ulls del client. Després es segueix amb una sèrie d'iteracions curtes (d'unes dues setmanes) de disseny i construcció. L'objectiu del FDD és aconseguir productes de software amb cicles de vida molt curts i de qualitat controlada.

L'ORIENTACIÓ A OBJECTES

Un dels grans reptes de l'enginyeria del software es aconseguir desenvolupar fragments del software (*components*) que assegurin la qualitat (estiguin ben provats) i que siguin reutilitzables.

Una de les formes d'aconseguir una certa reutilització del software és fer servir el paradigma orientat a objectes. En un sistema orientat a objectes el mon real es representa en termes d'objectes que interactuen enviant-se missatges.

El paradigma orientat a objectes es fonamenta en cinc conceptes bàsics: objecte, classe, encapsulació, herència i polimorfisme.
A continuació donem, de forma resumida, algunes idees bàsiques relacionades amb el paradigma orientat a objectes:

- Forma natural d'estructurar el domini d'un problema:

  - Objectes
  - Mètodes
  - Missatges

- Assegurament de la qualitat:

  - Ocultació de la informació
  - Abstracció
  - Modularització
  - Reutilització

- Objecte = estructura de dades + comportament (alt acoblament)

- Software = col·lecció d'objectes que s'intercanvien missatges

- Els objectes estan definits per:

  - Variables que emmagatzemen les dades
  - Mètodes que manipulen les dades

- Enviament de missatges:

  - Els objectes s'envien missatges entre ells

- L'objecte receptor determina el codi a executar-se:

nom missatge + objecte receptor $\Rightarrow$ codi

- Característiques dels objectes:

  - Identitat
  - Classificació
  - Herència
  - Polimorfisme
  - Encapsulació

- Identitat:

  - Les dades estan associades amb entitats diferenciables
  - Poden existir dos objectes amb dades idèntiques
  - S'accedeix als objectes per mitjà del seu identificador

- Classificació:

  - Agrupació d'objectes similars: classe
    * mateixos atributs
    * mateixes operacions
  - Classe: abstracció de característiques importants (determinades per l'aplicació)
    * Una classe descriu un conjunt (infinit) d'objectes = instàncies de la classe

- Herència:

  - Atributs i operacions comunes a diferents classes
  - Les subclasses hereten tots els atributs i mètodes de les superclasses
  - Les subclasses defineixen els seus propis atributs i mètodes

- Avantatges de l'herència:

  - Evita redundàncies: el codi idèntic només s'ha d'escriure una vegada
  - Reducció de la grandària del codi
  - Reutilització del codi

- Polimorfisme:

  - Sobrecàrrega d'operacions = mateix nom de funció, diferents implementacions

- Encapsulació: empaquetat de mètodes i atributs dins d'un objecte mitjançant una interfície de missatges:

  - Ocultació de detalls no necessaris o no importants
  - Accés utilitzant només operacions predefinides

- OO llenguatges:

  - Modelat: UML
  - Implementació: Java, C++, …

AVANTATGES DE L'ORIENTACIÓ A OBJECTES

- simplifica el desenvolupament de software, perquè els models que utilitza són molt pròxims al mon real que es modela,

- permet fer servir la mateixa representació durant tot el procés de desenvolupament del software, des de la fase d'anàlisi fins a la d'implementació,

- l'encapsulació permet generar sistemes que s'adapten be als canvis,

- l'herència i el polimorfisme incrementen la flexibilitat i potencialitat dels sistemes.

EL LLENGUATGE DE MODELAT UNIFICAT

El llenguatge de modelat unificat (UML) és un llenguatge de propòsit general per a la construcció de software orientat a objectes que ha estat proposat com a estàndard per L'Object Management Grup (OMG). L'OMG, creat el 1989, és una organització no lucrativa participada per grans empreses de software, hardware, usuaris i consultors que té per objectiu la elaboració de estàndards relacionats amb el paradigma orientat a objectes.

UML permet descriure un model d'anàlisi i disseny d'un sistema mitjançant diagrames construïts utilitzant símbols que tenen regles semàntiques, sintàctiques i pràctiques.

- Les regles semàntiques ens diuen que significa cada símbol i com interpretar-lo, tant quan es troba aïllat com quan es combina amb altres símbols en un diagrama.

- La sintaxis ens diu com mostrar i combinar els símbols per a obtenir els diagrames del model.

- Les regles pràctiques defineixen com utilitzar els símbols per a obtenir els diagrames del model de manera que siguin comprensible per altres persones.

DIAGRAMES QUE FA SERVIR UML

Diagrama de Casos d'Us
Diagrames de Classe
Diagrames d'Objectes
Diagrames d'Estats
Diagrames d'Activitat
Diagrames de Seqüència
Diagrames de Col·laboració
Diagrames de Components
Diagrames de Desplegament

Un pla de estudi dels diagrames bàsics d'UML pot començar amb els *casos d'ús* i continuar amb l'anàlisi dels fluxos de treball d'un grup de casos d'ús mitjançant els *diagrames d'activitat.* A partir d'aquest punt es disposarà del bagatge suficient com per a introduir-se en l'abstracció dels objectes i comprendre la importància de separar les tres perspectives bàsiques de la representació de les classes: concepció, especificació i implementació i estar en condicions de fer un primer *diagrama de classes*. A continuació, separar els escenaris que agrupen una sèrie de activitats i fer aflorar, a través dels *diagrames d'interacció* (*diagrames de seqüència* i *diagrames de col·laboració*), els objectes que intercanvien una sèrie de missatges. El pas següent és identificar alguna classe amb un comportament complex que la faci candidata a revisar tots els seus possibles estats i esbrinar quins esdeveniments són capaços de provocar un canvi d'estat. El *diagrama de estats* és l'adient per a representar aquesta dinàmica d'estats. Finalment, es pot abordar la configuració de components i el seu desplegament en una arquitectura utilitzant els *diagrames de components* i els *diagrames de desplegament*.

VISTES

Un sistema es pot representar per cinc vistes diferents, representades mitjançant un conjunt de diagrames, que el descriuen des de diferents perspectives:

- *Vista de l'usuari*: representació del sistema des del punt de vista dels *actors*. Es modela fent servir els *diagrames de casos d'us*.

- *Vista estructural*: les dades i funcionalitats es mostren des de dins del sistema. Modela l'estructura estàtica: *diagrames de classes, objectes i associacions*.

- *Vista del comportament*: representa els aspectes dinàmics del sistema. Mostra les interaccions entre els diferents elements estructurals del sistema: *diagrames d'activitat, seqüència, col·laboració i estats*.

- *Vista d'implementació*: representa els elements estructurals i els aspectes de comportament tal com seran implementats: *diagrames de components*.

- *Vista de l'entorn*: elements estructurals i aspectes de comportament relatius a l'entorn organitzacional en el qual residirà el sistema a construir: *diagrames de desplegament*.

UML I EL PROCÉS DE DESENVOLUPAMENT

L'UML no defineix un procés concret de desenvolupament. En general els diferents processos definits a partir de UML (Unified Process, Iconix, FDD, ...) fan servir un model incremental i interactiu. A continuació donem una descripció genèrica i breu – fent servir diverses fonts - de les fases de definició de requeriments, anàlisi i disseny en un context de desenvolupament de software fent servir UML.

DEFINICIÓ DE REQUERIMENTS

Abans que el software es pugui construir s'han d'entendre els elements de l'entorn organitzacional en el qual residirà el sistema informàtic: les persones, els procediments operacionals, el hardware i el propi software (*domini del problema*).

Els *Requeriments del software* recullen, a grans trets, els objectius d'un sistema informàtic i les seves funcionalitats desitjades. El document de Requeriments serveix de base per a la posterior construcció del software.

Un document de requeriments del software, redactat utilitzant el paradigma d'orientació a objectes, ha d'identificar: guions, actors, activitats, casos d'us i proporcionar un model de classes preliminar (objectes, classes, interaccions entre classes, ...) . També ha de descriure les interfícies d'usuari que haurà de tenir el sistema amb l'ajuda, si cal, de diagrames d'estat. La definició de requeriments orientada a objectes (ROO) es centra en la de vista d'usuari.

ANÀLISI

L'objectiu de l'anàlisi és obtenir una comprensió precisa de les necessitats del sistema. L'anàlisi s'ocupa de traduir els requeriments a un llenguatge més formal. El procés d'anàlisi no comença preocupant-se pels objectes, sinó intentant comprendre com es farà servir el sistema per persones, per màquines o per altres sistemes. Després es procura definir, identificar i descriure els objectes dins del domini del problema.

El propòsit de l'anàlisi orientat a objectes (AOO) és definir totes les classes que són rellevants pel problema a resoldre, els atributs i operacions associats a les classes, les associacions i comportaments lligats a les classes. Les característiques estàtiques i dinàmiques de les classes es modelen amb UML. S'obté un model que refina el casos d'us (elimina redundàncies i afegeix casos d'us no demanats pels usuaris però necessaris) i defineix els atributs de cada classe, les associacions i comportaments i les comunicacions entre classes, així com el comportament de les classes en el temps. Es generen diagrames de classe (d'anàlisi), i diagrames de seqüència i de col·laboració d'alt nivell. L'AOO es centra en les vistes d'usuari i estructural.

Els models creats durant les fases de definició de requeriments i anàlisi formen part del domini del problema.

DISSENY

El disseny orientat a objectes (DOO) transforma el model d'anàlisis obtingut fent servir AOO en un model de disseny que serveix d'avantprojecte per a la construcció del software. Durant el disseny es procuren definir els objectes lògics del software que finalment seran implementats en un llenguatge de programació orientat a objectes.

Es dissenyen els objectes a partir de diagrames de seqüència i de col·laboració de baix nivell (assignant responsabilitats a les classes i fent servir patrons de disseny), centrant-se en els detalls interns de cada classe (de disseny), la definició d'atributs (estructures de dades), de les operacions (algorismes) i en detalls relatius als missatges. Es fa el disseny arquitectònic, identificant les classes que són encapsulades pels diferents components i subsistemes, per mitjà dels diagrames de components i de desplegament. S'especifiquen les interfícies d'usuari (amb l'ajuda de diagrames d'estat), la gestió de dades i els mecanismes d'administració de tasques. El DOO es centra en les vistes del comportament i de l'entorn.

Els models creats durant la fase de disseny formen part del domini de la solució.

DISSENY D'INTERFÍCIES D'USUARI

Les interfícies d'usuari (IU) més habituals són entrades mitjançant teclat, pantalla i sortides per pantalla i impressora, entrades i sortides sonores i entrades per scanner. Els usuaris interactuen amb el software a través de les interfícies d'usuari. Les interfícies d'usuari tenen una influència molt gran en la comoditat i productivitat dels usuaris. Moltes vegades els usuaris jutgen la qualitat del software per les interfícies amb les que interactuen amb més freqüència.

El disseny d'interfícies gràfiques d'usuari (IGU), implementades per mitjà de pantalles gràfiques i generalment fent servir el ratolí, considera tres aspectes: contingut, el format i la interacció. És pràctic dissenyar les IGU a partir de diagrames d'estat.

DISSENY DE LA PERSISTÈNCIA

Entre les bases de dades utilitzades per a emmagatzemar dades persistents, actualment (i previsiblement en el futur immediat) les bases de dades relacionals són les més àmpliament utilitzades. La orientació a objectes es centra en la construcció d'aplicacions basant-se en objectes, que tenen atributs (dades) i mètodes (comportament), mentre que el model relacional es centra tan sols en el fet d'emmagatzemar dades. En el paradigma orientat a objectes s'accedeix als objectes via relacions entre objectes, mentre que en el model relacional es dupliquen dades que possibiliten lligar files de diferents taules. Aquestes dues fonamentacions teòriques tan diferents fan difícil la combinació dels dos paradigmes. No obstant, a l'igual que hi ha regles de transformació dels diagrames del model entitat-relació al model relacional, també hi ha regles per a transformar de forma raonable els diagrames de classes al model relacional.

PATRONS DE DISSENY

A l'enfrontar-nos a un problema nou (per a nosaltres), no cal dissenyar una nova solució. Els *patrons de disseny* permeten reutilitzar les solucions d'altres dissenyadors. Els patrons permeten al dissenyador crear l'arquitectura de disseny integrant components provats i reutilitzables. Un patró soluciona problemes de disseny específics.