**Enginyeria de Software**

**Sessió 1**

* 1. **Definició**

1. Branca de la ciència dels computadors que gestiona el disseny, la implementació i el manteniment de programes complexos d’ordinador.
2. Procés d’analitzar les necessitats dels usuaris, dissenyar, construir, i verificar les aplicacions que les satisfacin fent servir un llenguatge de programació.
   1. **Software**

Segons la **RAE**, el software és un **conjunt de programes, instruccions i regles informàtiques** que permeten executar diferents tasques en un computador.

Per entendre el concepte, el software és la part de la tecnologia que indica com es relacionen, com es comporten, quines entrades accepten i quines sortides produeixen els components hardware que controla, perquè el control del hardware correspon al software.

Tipus de software: de gestió, de temps real, etc.

* 1. **Complexitat**

Una cosa “complexa” està definida com a quelcom que és “difícil d’entendre” també es pot definir com que “està composta per moltes peces”.

“La complexitat”, en el cas de l’Enginyeria del Software és justament la mesura de la dificultat que ha de resoldre un programa de software. Si el veiem per la part de la quantitat de peces que compondran el resultat final, també és un concepte vàlid, si té més peces, per definició és més complex.

El que **“sempre”** passa és que el conjunt d’eines de l’Enginyeria del Software persegueix la finalitat de “dividir la complexitat”.

* Dinàmica de complexitat
  1. **Procés de Software**

Enfocament disciplinat del software:

* Models + Mètodes + Eines + Processos

Procés que assegura estàndards de qualitat, que permeten dur un alt i rigorós nivell de gestió en recursos, temps i cost.

Permet l’avaluació i la millora del software produït.

Ens cal un procés software si:

* Volem afrontar sistemes complexos.
* Volem escriure software de qualitat (“sempre” cal controlar la qualitat).
* Volem prevenir riscos i solucionar els problemes que sorgiran en el seu moment.
* Volem millorar els nivells de comunicació, control i seguiment dins l’equip i amb els usuaris.
* Volem estandarditzar el procés facilitant l’entrada i la sortida de persones.

Diagrama

Descripción generada automáticamente

Diagrama, Tabla

Descripción generada automáticamente

**Sessió 2**

**2.1 Paradigmes Processos Software**

Seqüencials (clàssics)

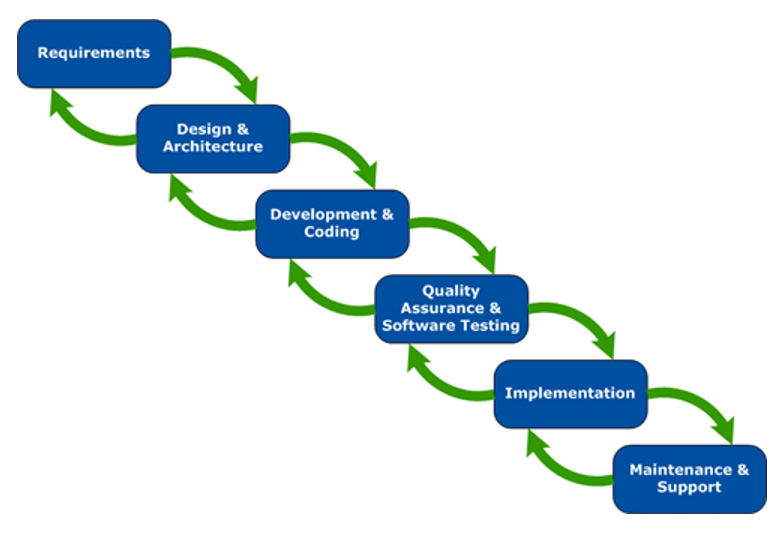
* Metrica 3
* Merise

Incrementals

* Unified Process
* Àgils

**2.2 Model lineal en cascada**

Un dels primers models que es van proposar (finals 60s).



**Característiques:**

* Dividits en fases amb una estructura temporal.
* La seqüència temporal en fases no és realista.
* És molt difícil formular explícitament tots els requeriments al inici.
* Només hi ha resultats executables en fases molt avançades del projecte.
* Les interaccions i referències creuades només es permeten entre fases adjacents.
* No són prou flexibles.
* Massa burocràtics: s’ha de documentar massa.
* Els usuaris o clients i els desenvolupadors treballen de forma separada.
* No proveeixen d’eines pel treball en equip.
* No estan centrats en les persones.

Interfaz de usuario gráfica, Diagrama, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

**2.3 Models iteratius**

Unified Process (UP), Metodologies àgils, basats en iteracions ràpides i constants.

**2.3.1 UP: Model iteratiu i incremental**

* Cada iteració genera codi executable.
* En cada iteració s’incrementen les capacitats del sistema.
* Cada iteració és provada i validada pel client.

Diagrama

Descripción generada automáticamente

**Fases:**

* *Inception*: es defineix l’abast del projecte
* *Elaboració*: es planifica el projecte, s’especifiquen els casos d’ús i es comença a construir l’arquitectura.
* *Construction*: s’implementa el sistema.
* *Transition*: es posa en funcionament el sistema.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Diagrama

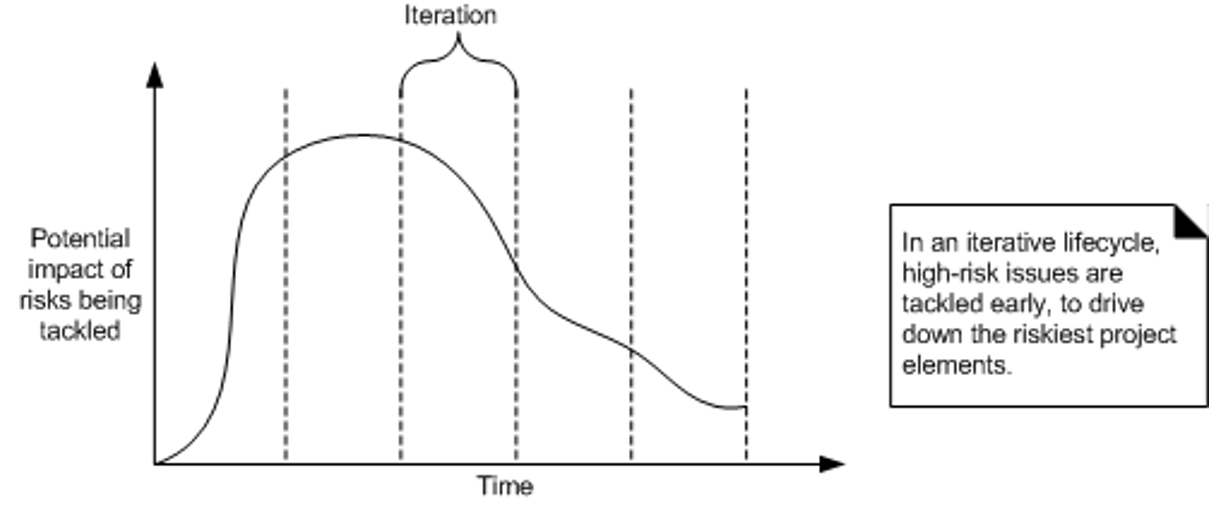
Descripción generada automáticamente

**Gestió de riscos**

A cada iteració es fan les proves (test) del sistema.

En les primeres iteracions s’aborden aquelles parts del sistema que tenen més risc i/o que aporten més valor a l’usuari (client).

El client veu el sistema en funcionament des del començament i es manté involucrat en tot el procés.



**2.3.2 Metodologies Àgils**

Scrum, Crystal Clear, Extreme Programming.

* Individuals and interactions over processes and tools.
* Working software over comprehensive documentation.
* Customer collaboration over contract negotiation.
* Responding to change over following a plan.

That is, while there is value in the items on the right, we value the items on the left more.

* La codificació és l’activitat clau d’un projecte software, XP la posa al centre de la gestió del projecte, és un *programmer-first process*.
* És una metodologia lleugera (*light-weight*) amb la gestió i documentació estrictament necessària.
* Assumeix que els requeriments canvien sempre, també al final del projecte, llavors s’adhereix al canvi continu.
* La millor forma de documentació és el codi; cal programar contínuament, radical, ràpida, amb un feedback continu per part del client.
* Es planifica només a curt termini (una iteració).
* Considera millor que dedicar temps al disseny, buscar un procés software que faci fàcil canviar el codi i redueixi el temps de fer-ho.
* El codi s’ha de millorar contínuament (Refactoring). Hem d’estar segurs de que no introduïm errors.
* Testing és una part molt important del desenvolupament.
* Les últimes tendències apunten cap al Test Driven Development.
* Flexibilitat a canvis.
* Reducció del Time to Market.
* Millor qualitat del software.
* Millor productivitat.
* Maximitzar el retorn de la inversió.
* Prediccions de temps.
* Reduccions de riscos.

**Sessió 3**

**3.1 Requeriments**

**3.1.1 Definició**

1. Una condició o necessitat d’un usuari per resoldre un problema o assolir un objectiu.
2. Una condició o capacitat que ha d’estar present en un sistema o components de sistema per satisfer un contracte, norma, especificació o altre document formal.
3. Una representació documentada d’una condició o capacitat com en els punts anteriors.

**3.1.2 Característiques**

**Necessari**: És fàcil saber si un requeriment és necessari, perquè quan no hi és apareix un defecte al sistema que estem tractant de construir. A més no podem “rodejar” la falta d’aquest part del programa.

**Concret**: Aquesta característica parla de què ha d’estar expressat de manera que no hi hagi cap ambigüitat que pugui confondre a qui el llegeixi.

**Sense ambigüitat**: Vol dir que hi ha una única manera d’interpretar-lo dintre del negoci en el qual estem treballant. Si tots els que comparteixen el negoci no entenen el mateix quan llegeixen un requeriment, cal resoldre les ambigüitats.

**Complet**: Un requeriment és complet si no necessita més detalls ni aclariments per tal d’entendre’l.

**Consistent**: És quan no es contradiu amb un altre requeriment.

**Verificable**: A vegades és possible, d’una manera no molt costosa validar que el programa que desenvoluparem satisfà el requeriment, aquest és un requeriment verificable. A vegades es fan “prototipus” per tal que l’usuari pugui “verificar” que hem entès quin “comportament” ha de tenir el sistema tot i que encara no estigui fet.

**3.1.3 Tipus**

Requeriments funcionals:

* Són els que descriuen de quina manera “ha de funcionar” (s’ha de comportar) el programa davant un “esdeveniment”. Per regla general construïm un programa perquè faci determinades “funcions” quan li proporcionem determinades “entrades” o exercim determinades “accions”. Un requeriment funcional, per regla general descriu “què ha de fer el sistema si introduïm determinat conjunt de dades” o “quina informació ens ha de donar quan li demanem de determinada manera”. En sistemes de temps real o jocs, els requeriments funcionals es comporten de la mateixa manera, però la seva redacció fa referència a parts més concretes del conjunt de programes que volem crear.
* Dintre d’UML (el nostre conjunt d’eines), una eina molt bona per especificar requeriments funcionals són els “Casos d’Ús”.

Requeriments no funcionals:

* Els requeriments no funcionals d’un programa, fan referència a tot el conjunt de restriccions tècniques que aquest ha d’assolir independentment de la funcionalitat a la qual estigui donant resposta.
* Actualment, els requeriments no funcionals d’un programa fan referència als estàndards mínims de qualitat que ha de respectar.

**3.1.4 Redacció**

Com dividim la complexitat?

* La redacció de requeriments no és més que la expressió de la nostra complexitat “dividida”.
* Redactar requeriments és el primer pas per poder dividir la complexitat.
* Expressar correctament les frases que detallen els nostres requeriments ens ajudarà a obtenir “tota” la complexitat del sistema, expressada de manera que es pugui analitzar des del punt de vista de la creació de software.

**Sessió 4**

**4.1 UML “Unified Modeling Language”**

Perquè:

* Ens permet **modelar** a tots els nivells d’un procés de software, des de l’Anàlisi fins a la codificació.
* Què és modelar?
* Ens permet parlar a diferents “nivells d’abstracció”
* Què és l’abstracció? *Agafar un element separador i analitzar-lo en detall. Si tenim un sistema complex, agafem les peces que el formen i anar-les analitzant part a part*.
* Ens permet aclarir si existeixen ambigüitats.
* Els models són entenedors fins i tot per persones de l’equip que no són informàtics.

**4.1.1 Parlem de UML**

És un llenguatge, és un estàndard, és gràfic i visual. Permet documentar un sistema.  
No és programació.

Quines etapes del “Procés de software” cobreix UML?

**4.2 Models UML**

**Diagrames de “ comportament” del sistema (aspectes “dinàmics”)**

* Diagrama de Casos d’Ús
* Diagrama de Seqüència
* Diagrama de Col·laboració
* Diagrama d’Estats
* Diagrama d’Activitats

**Diagrames “d’estructures” del sistema (aspectes “estàtics”)**

* Diagrama de Classes
* Diagrama d’Objectes
* Diagrama de Components
* Diagrama de Desplegament