Universitat Oberta de Catalunya
Enginyeria Informàtica

Projecte de Fi de Carrera

Mètriques de productivitat de desenvolupament de programari per a la gestió de projectes

Autor: David Bosch Geli
Consultor: Jesús Bustínduy Basterrechea

Juny de 2004

El que no és mesurable feu-ho mesurable
Galileo Galilei (1564-1642)

Resum

Malauradament, el sector informàtic destaca per l'elevat nivell de fracàs dels seus projectes. Una part important d'aquest mèrit es degut a una mala planificació dels mateixos, i més concretament, a una mala estimació dels costos associats al desenvolupament de programari.

L'estimació d'aquests costos esdevé una part molt important de la gestió del projecte en el sentit que acostumen a determinar el termini de lliurament i el pressupost del mateix, dos aspectes que repercuteixen directament sobre el resultat. Tot i aquesta importància, moltes organitzacions continuen sense emetre unes estimacions objectives i fonamentades, realitzant-les sovint en base a la intuïció i a l'experiència.

El treball, presenta com dur a terme un procés d'estimació que permeti obtenir estimacions de qualitat, i analitza els principals mètodes d'estimació de programari. Mètodes que persegueixen l'obtenció de mesures objectives i fonamentades sobre a l'amidament de programari i l'estimació de temps i costos d'un projecte.

Finalment, presenta un seguit d'aspectes a tenir en compte a la hora d'avaluar estimacions. Aquests s'enfoquen en tres direccions: l'avaluació de les estimacions obtingudes, l'avaluació de l'organització que les realitza i l'avaluació dels mètodes seguits per a obtenir-les.

Contingut

Llista de Figures ..... **ix**

Llista de Taules ..... **xi**

Llista de Equacions ..... **xiii**

Glossari ..... **xv**

Introducció **1**
1.1 Estructura del document 2

Procés d'estimació de costos de programari **1**
2.1 Determinar els objectius 1
2.2 Planificar els requeriments de dades i recursos 1
2.3 Analitzar i revisar els requeriments 2
2.4 Desenvolupar les estimacions fins al nivell desitjat 2
2.5 Utilitzar diferents tècniques independents 3
2.6 Comparar estimacions 3
2.7 Continuïtat 3
2.8 Recollida d'informació 4

Línies de Codi **5**
3.1 COCOMO II 6
3.1.1 Història i evolució 6
3.1.2 Línies de codi font (SLOC) 7
3.1.3 Equació d'esforç 7
3.1.4 Paràmetres d'escala 8
3.1.5 Paràmetres de cost 10
3.2 Probe 13
3.2.1 Detall del mètode 13
3.2.2 Estimació de la mida 13
3.2.3 Ajustament de la mida 17
1.1.4 Consideracions finals 19

Punts Funció **21**
4.1 Tendències actuals 22
4.2 COSMIC-FFP 22
4.2.1 Descripció del mètode 23
4.2.2 Extensions locals del mètode 27

4.3 IFPUG-FPA 28
4.4 MKII-FPA 29

Punts Característica 31
5.1 Definició d'algorisme 32
5.1.1 Regles per a avaluar algorismes 33
5.2 Escollir entre punts funció i punts característica 33

Recompte Indirecte (*Backfiring*) 35

Punts Cas d'Ús 37
7.1 Història dels casos d'ús 37
7.2 Definició de cas d'ús 38
7.3 Funcionament del mètode 39
7.3.1 Avaluació del model de casos d'ús 39
Avaluació dels actors 40
Avaluació dels casos d'ús 41
7.3.2 Avaluació de l'entorn 42
Factors de complexitat tècnica 43
Factors d'entorn 46
7.3.3 Obtenció dels punts cas d'ús ajustats 47
7.3.4 Determinar l'esforç 48
7.4 Avantatges 49

Model de Rayleigh 51
8.1 Conseqüències de la corba de Rayleigh 52
8.2 Utilitat del mètode 54

Una tècnica senzilla 57

Avaluació d'estimacions 61
10.1 Avaluació del resultat d'una estimació 61
10.2 Avaluació d'organitzacions 62
10.3 Avaluació de mètodes d'estimació 63

Consideracions finals 65

Referències 67

Llista de Figures

Figura 1 Diagrama de funcionament del mètode Probe 14
Figura 2 Estimació de la mida amb Probe.....16
Figura 3 Dades d'exemple sobre projectes històrics.....18
Figura 4 Exemple de recta de regressió.....18
Figura 5 Fase de Representació COSMIC-FFP.....25
Figura 6 Fase de medició COSMIC-FFP 27
Figura 7 Exemple de diagrama de casos d'ús.....39
Figura 8 Corba de Rayleigh 52

Llista de Taules

Taula 1 Valoració dels factors d'escala 9
Taula 2 Avaluació dels factors de cost.....12
Taula 3 Classificació de les dades històriques per a determinar la mida 14
Taula 4 Exemple de taula per a estimació de mida 16
Taula 5 Classificació dels punts funció.....29
Taula 6 Mètode Punts Característica.....33
Taula 7 Ratis de conversió entre Punts funció i Punts característica 34
Taula 8 Recompte Indirecte. Relacions d'equivalencia.....36
Taula 9 Factors de ponderació per als actors 41
Taula 10 Exemple d'obtenció del UAW.....41
Taula 11 Factors de ponderació per als casos d'ús 42
Taula 12 Exemple d'obtenció del UUCW 42
Taula 13 Factors de complexitat tècnica 44
Taula 14 Exemple d'avaluació de factors de complexitat tècnica 45
Taula 15 Factors d'Entorn 46
Taula 16 Exemple d'avaluació dels factors d'entorn 47
Taula 17 Guia per a l'avaluació d'estimacions 61
Taula 18 Sis requeriments per a obtenir estimacions de confiança.....62
Taula 19 Set indicadors sobre la capacitat d'estimació d'una organització 63
Taula 20 Avaluació de mètodes d'estimació 64

Llista de Equacions

Equació 1 Equació d'esforç del mètode COCOMO II.....7
Equació 2 Obtenció de l'exponent E 9
Equació 3 Obtenció del EAF.....10
Equació 4 Equacions de regressió lineal per a la mida del programari.....17
Equació 5 Equacions de regressió lineal per al Temps de Desenvolupament.....19
Equació 6 Punts cas d'ús sense ajustar 42
Equació 7 Punts cas d'ús ajustats.....43
Equació 8 Obtenció del factor de complexitat tècnica TCF.....45
Equació 9 Obtenció del factor d'entorn EF 47
Equació 10 Equació d'esforç 48
Equació 11 Equació del programari 53
Equació 12 Productivitat convencional.....53
Equació 13 Temps de Desenvolupament mínim.....54
Equació 14 Equacions d'Esforç.....55
Equació 15 Equació del cost 55
Equació 16 Equacions d'estimació 57
Equació 17 Equacions de desviació estàndard.....58
Equació 18 Equacions de %Error.....58

Glossari

API: de l'anglès *Application Program Interface*. Interfície d'una aplicació.

CAD: de l'anglès *Computer Aided Design*. Disseny assistit per ordinador. Conjunt d'eines gràfiques que s'utilitzen en entorns de disseny i desenvolupament.

Cfsu: *Cosmic Functional Size Unit*. Unitat de mesura utilitzada en el mètode de medició COSMIC-FFP.

CMM: *Capability Maturity Model*. Model estàndard per a determinar la maduresa d'una orgranització i dels seus processos de programari, i identificar les pràctiques clau requerides per a la millora i l'increment de la millora dels processos. Desenvolupada i coordinada per el SEI.

COSMIC: *Common Software Measurement International Consortium*. Organització internacional voluntària que ha desenvolupat un mètode d'estimació basat en punts funció anomenat COSMIC-FFP.

ISO: *International Organization for Standardization*. Organització internacional dedicada a elaborar i publicar normes que esdevenen estàndards. Està formada per organismes d'estandardització de 148 països diferents.

FFP: *Full Function Points.* Nom que reben els punts funció en el mètode anomenar COSMIC-FFP.

OMG: *Object Management Group*. Organització no lucrativa creada el 1989 en la participen més de vuit-centes grans empreses de programari, maquinari, usuàries i consultores. Té per objectiu fomentar l'ús de la tecnologia d'objectes. És dedica a l'elaboració d'estàndards. Entre els estàndards més coneguts hi ha el CORBA (per a objectes distribuïts) i l'UML.

PBX: *Private Branch Exchange*. Centraleta privada. Sistema telefònic que ofereix serveis de forma localitzada.

RUP: Rational Unified Process. Model de desenvolupament de programari orientat a objectes proposat per l'empresa Rational Software.

SEI: *Software Engineering Institute*. Organització de recerca i desenvolupament de programari, finançada pel Departament de Defensa dels Estats Units i gestionada per la Universitat Carnegie Mellon de Pittsburgh. Més informació a www.sei.cmu.edu.

Capítol 1

Introducció

El cicle de vida dels projectes de programari acostuma a incorporar una fase de planificació. Aquesta etapa té per objectiu, entre altres, estimar el temps que es trigarà en desenvolupar la solució, els recursos humans que caldrà destinar-hi i preveure el cost associat al procés.

El resultat d'aquestes estimacions determinaran la data de lliurament del producte i el pressupost associat. Per aquest motiu, aquestes estimacions prenent una gran rellevància, ja que el pressupost i la data de lliurament marquen en gran part l'assoliment dels objectius perseguits per el projecte.

En aquest sentit, la bondat de les estimacions és crucial. Fins al punt d'arribar a determinar la viabilitat del projecte. Unes estimacions poc encertades poden derivar, si no es prenen mesures correctives, en una important desviació entre les previsions inicials i el resultat final obtingut. Aquestes desviacions poden derivar en diversos problemes, com per exemple, l'esgotament dels recursos pressupostats abans de finalitzar, ocasionant una possible sobredespesa o bé l'incompliment total o parcial dels requeriments inicials. Poden ocasionar retard en el lliurament, baixa qualitat del producte desenvolupat i una important pressió sobre l'equip de desenvolupament, que es veu obligat a realitzar grans esforços per a poder complir amb terminis pocs realistes.

Desafortunadament, el sector ha demostrat ser notòriament ineficaç realitzant estimacions. Una de les possibles causes d'aquesta ineficàcia és la manca d'aplicació de mètodes que permetin obtenir estimacions objectives sobre el cost, mida i temps de desenvolupament. En moltes organitzacions, les estimacions es realitzen sense un anàlisi rigorós dels requeriments i basant-se en intuïcions subjectives. Un anàlisi rigorós dels requeriments i l'aplicació d'un mètode d'estimació permetria millorar la qualitat de les estimacions.

La major part dels costos d'un projecte van destinats als recursos humans. Són els recursos destinats al desenvolupament, que equival a l'*esforç* que requereix el projecte. L'esforç és un dels aspectes més complicats d'estimar i controlar. S'acostuma a mesurar en hores-persona (o en mesos-persona o anys-persona depenent de l'escala utilitzada). Una unitat d'hores-persona representa el treball que realitzarà una persona dedicada al projecte en una hora.

Com es pot veure, les estimacions proporcionen una relació vital entre les tècniques i conceptes generals de l'anàlisi econòmic i el món particular de l'enginyeria del programari.

Les tècniques d'estimació de costos formen part dels fonaments d'una bona gestió del programari. No acostumen a ser utilitzades si no és juntament amb tot un conjunt de mesures destinades a millorar la qualitat del programari, que en l'enginyeria del programari és anomenat *gestió de la qualitat*.

Cal tenir molt present que l'estimació cal realitzar-la al llarg de tota la vida del projecte. És una activitat que s'inicia a l'etapa d'anàlisi de viabilitat del projecte, però que cal anar refinant i avaluant al llarg de tota la vida del mateix per a controlar que les despeses estan en línia amb el pressupost.

1.1 Estructura del document

El document està estructurat seguint, en un primer nivell, una línia lògica dins el procediment d'obtenció de estimacions. Primer s'expliquen els passos a seguir al realitzar un procés d'estimació. A continuació s'analitzen diferents mètodes i tècniques per a obtenir estimacions. Aquestes ha estat agrupades segons la seva tipologia.

Finalment es donen unes guies per a poder avaluar la qualitats de les estimacions.

Dins d'aquest primer nivell d'ordenació lògica, també s'ha procurat seguir, dins el possible, una segona línia temporal. Sobretot pel que respecta als diferents mètodes i la seva aparició.

L'estructura del document és la següent:

- El capítol 2 defineix un possible procés que permeti assegurar el rigor en la realització d'estimacions. Descriu un conjunt de passos bàsics a seguir per a l'obtenció d'estimacions sobre projectes de desenvolupament de programari.

- El capítol 3 explica les directrius del mètodes basats en línies de codi i presenta dos mètodes: el mètode COCOMO II i el mètode Probe.

- El capítol 4 presenta els mètodes basats en punts funció. Dins d'aquesta categoria s'avaluen quatre dels mètodes més importants.

- El capítol 5 presenta el mètode dels punts característica, un mètode basat en els punts funció i originalment dissenyat per a poder amidar programari més orientat a control de sistemes.

- El capítol 6 exposa la tècnica anomenada *recompte indirecte* (*backfiring*), la qual permetria obtenir la mida en punts funció a partir d'amidament realitzat en línies de codi.

- El capítol 7 analitza un dels mètodes més recents: el mètode basat en punts cas d'us. Aquest utilitza els conceptes bàsics dels punts funció per a aplicar-los als models de casos d'ús.

- El capítol 8 presenta el model de Rayleigh. Aquest és un mètode que permet realitzar estimacions en base a la productivitat, considerada aquesta des d'un punt de vista global de la organització.

- El capítol 9 tanca la descripció de mètodes. Presenta una tècnica molt senzilla que no té res a veure amb les anteriors. Està basada en conceptes estadístics i permet estimar qualsevol tipus de quantitat.

- El capítol 10 tracta l'avaluació d'estimacions. Té en compte tres punts de vista: avaluació del resultat d'una estimació, avaluació de la capacitat d'estimació d'una organització i avaluació del mètode utilitzat en les estimacions.

Capítol 2

Procés d'estimació de costos de programari

L'estimació de costos de programari no consisteix tant sols en l'aplicació d'un determinat mètode. És aconsellable seguir tot un procés que permeti obtenir les màximes garanties de la bondat de les estimacions.

En aquest capítol és presenten un seguit de passos bàsics a tenir en compte a la hora de realitzar estimacions.

2.1 Determinar els objectius

És molt important tenir clars els objectius que es volen assolir amb l'estimació. La transcendència de la decisió que necessiti ser presa ha d'incidir en la qualitat de les estimacions. Cal valorar el nivell d'eficàcia que es requereix de les estimacions.

També caldrà revisar els objectius de les estimacions un cop el procés hagi començat i modificar-los quan sigui necessari per adaptar-los a les necessitats de cada moment.

2.2 Planificar els requeriments de dades i recursos

És important planificar les dades que faran falta per a realitzar l'estimació i els recursos que es requeriran per a la seva obtenció. En aquest sentit, no és mala idea considerar el procés d'estimació com un sub-projecte per ell mateix, amb les seves corresponents fases de planificació, disseny i execució.

La mini-fase de planificació del procés d'estimació produirà un pla de projecte que hauria d'incloure un primer detall de totes les preguntes que caldrà respondre amb les activitats d'estimació: com, què, qui, quan, quant i perquè.

2.3 Analitzar i revisar els requeriments

És important disposar d'unes especificacions el menys ambigües possible (tot i que molt probablement estaran subjectes a canvis en funció del resultat de l'estimació). En aquest sentit, la millor manera d'avaluar la bondat d'uns requeriments és analitzar com de fàcil és verificar el seu compliment.

El nivell de concreció d'un requeriment vindrà donat per com de fàcil serà desenvolupar un mètode que permeti decidir si l'aplicació construïda compleix o no el determinat requeriment. Si no resulta fàcil descobrir aquest mètode, caldrà treballar més amb el requeriment per a obtenir unes especificacions més concretes.

Per a que un requeriment sigui fàcilment verificable ha de ser específic, sense ambigüitats, i quantificable en la mesura del possible.

2.4 Desenvolupar les estimacions fins al nivell desitjat

El nivell desitjat de fiabilitat per a una estimació serà el nivell indicat en la declaració dels objectius per a les estimacions (definits a la secció 2.1).

En general, com més detallades siguin les activitats d'estimació, més acurades seran les estimacions aconseguides. Això s'explica per les següents raons:

1. Estudiar els requeriments a un nivell més detallat permet entendre més bé els aspectes tècnics del programari a desenvolupar.

2. Quan es realitzen estimacions a més baix nivell, es disposa de més informació, de més dades parcials, i això redueix la variància de les estimacions totals.

3. Com més s'analitza el sistema a través de les funcions que ha de realitzar, més difícil és obviar un possible augment de cost en determinades funcions més amagades. Sense un coneixement profund de les funcionalitats es tendeix a subestimar el seu cost.

2.5 Utilitzar diferents tècniques independents

Entre els diferents mètodes per a realitzar estimacions, no n'hi ha cap que sigui millor que els altres en tots els aspectes. Les debilitats d'un mètode són complementaris amb els punts favorables d'un altre.

És recomanable utilitzar més d'un mètode. Permetrà esquivar les debilitats d'un mètode concret i ressaltarà els punts favorables del conjunt mètodes.

2.6 Comparar estimacions

El gran avantatge d'utilitzar més d'un mètode és la possibilitat que brinda d'investigar perquè difereixen els seus resultats. El resultat d'aquest anàlisi permetrà obtenir unes conclusions que ajudaran a realitzar noves iteracions dels mètodes.

L'objectiu serà aconseguir que els resultats convergeixin, permetent l'obtenció d'unes estimacions més realistes.

2.7 Continuïtat

Un cop un projecte ha començat és essencial recollir dades sobre els progressos i costos que es van assolint i comparar-los amb les estimacions realitzades.

Per una banda, cal tenir present que les estimacions es realitzen sobre informació incompleta i imperfecte: estimacions sobre la mida, factors de l'entorn de desenvolupament, factors tecnològics, etc.

Per aquest motiu, és important recollir informació sobre el rendiment del nostre equip de treball a mesura que aquesta estigui disponible. Caldrà actualitzar les estimacions realitzades a l'inici en conseqüència amb aquesta informació. Això permetrà obtenir mesures més realistes per a una millor gestió del projecte.

A més, es aconsellable tenir en compte els següents aspectes:

- Hi ha projectes que no s'ajusten bé als models d'estimació. En aquestes situacions, les estimacions realitzades per al curt termini han de ser comparades amb les dades reals de costos per a aconseguir ajustar el funcionament del mètode i obtenir estimacions més realistes.

- Els projectes de programari acostumen a ser força volàtils: s'hi afegeixen mòduls, funcionalitats, es retallen requeriments, etc. a mesura que el projecte avança. El cap de projecte haurà d'identificar aquests canvis i generar noves estimacions que siguin coherents amb el projecte actual i les fases pendents d'implementar.

- El programari és un camp en constant evolució. Les tècniques d'estimació estan calibrades segons experiències en projectes anteriors i és possible que no s'adaptin del tot bé a entorns actuals o futurs de desenvolupament. És important notar aquestes diferències i incorporar-les a les estimacions i tècniques per tal de millorar-les.

2.8 Recollida d'informació

Des de l'inici del projecte serà molt important recollir el màxim d'informació sobre la seva evolució. Aquesta informació podrà ser utilitzada més tard en futurs projectes.

Caldrà recollir, entre altres, els següents aspectes:

- Mida del mòdul desenvolupat
- Temps de desenvolupament requerit
- Desviació respecte a la estimació original
- Comentaris sobre el perquè de la desviació

Tota aquesta informació permetrà aprendre dels errors comesos en l'estimació i millorar futures aplicació del procés.

Capítol 3

Línies de Codi

Les línies de codi han estat utilitzades com a unitat de referència de la mida d'un programa des de l'inici de la informàtica. Tot i que amb les tècniques de desenvolupament de programari actuals (generació automàtica de codi, entorns visuals, etc.) ha anat perdent importància, segueix utilitzant-se.

Aquesta mesura presenta l'inconvenient de que depèn completament del llenguatge de programació utilitzat en el desenvolupament. Aquest fet ocasiona la dificultat de comparar estimacions realitzades en diferents llenguatges.

Per altra banda, els detractors indiquen que en el moment de realitzar les estimacions no és coneix amb tota certesa quina serà la mida del programari (aquesta només es pot assegurar un cop ha estat desenvolupat el sistema). Si bé aquest argument és cert, també és cert que l'estimació aproximada del número de línies de codi pot donar bons resultats si es correctament.

En aquest capítol es presenten dos mètodes. El primer és un dels mètodes que més transcendència a tingut en la història de l'amidament de programari, el mètode COCOMO. Concretament la versió anomenada COCOMO II.

El segon mètode, el Probe, no és tant conegut però aporta un punt de vista interessant per la seva manera de procedir. Presenta una gran dependència i confiança en les dades històriques de la organització. Destaca per la seva senzillesa i la manera com té en compte els desviaments ocorreguts en anteriors projectes.

Aquest mètode, però, tot i presentar-se dins l'apartat de les línies de codi (així és també com el presenta el seu autor), podria funcionar amb moltes altres unitats de mesura (per exemple, punts funció) sense cap canvi en el seu raonament.

3.1 COCOMO II

3.1.1 Història i evolució

El mètode COCOMO va ser originàriament publicat el 1981 per Barry Boehm [Boehm 1981]. El seu nom prové de l'abreviació de *Constructive Cost Model*. Aquest model permet estimar el cost, esforç i temps que requereix el desenvolupament de programari d'acord amb les tècniques que s'utilitzaven durant els anys en que va ser desenvolupat: els anys 70 i 80. Va esdevenir un dels models d'estimació més utilitzat. Actualment, aquell model original es conegut amb el nom de COCOMO 81, i han anat realitzant-se diferents versions del mètode.

L'any 1987 van aparèixer dos versions del model original: Ada COCOMO [Boehm 1987] i Incremental COCOMO. El primer assumeix que s'utilitza el llenguatge de programació ADA per al desenvolupament. En aquest model s'afegeixen dos paràmetres de cost i es revisa el significat d'alguns dels paràmetres del model original. També es té en compte que es dedica més temps a les etapes d'anàlisis i disseny que a les de programació, ja que el llenguatge Ada facilitava el desenvolupament i control d'aplicacions complexes.

El segon model, Incremental COCOMO, era una alternativa a la utilització del model clàssic de desenvolupament de programari en cascada (*waterfall*). Barry Boehm va publicar un article l'any 1988 [Boehm 1988] en el qual definia el model de desenvolupament de programari en espiral. Aquest model de desenvolupament permetia reduir el risc en projectes considerats d'"alt risc", en els quals no es sabia molt bé el tipus de programari que calia desenvolupar, i per tant, era difícil disposar de uns requeriments complets i fiables. Posteriorment, aquest model ha estat modificat per altres organitzacions, com la *Software Productivity Consortium*, i el mateix Boehm en treballs posteriors [Boehm 1996]. Incremental COCOMO té en compte aquesta mètode de desenvolupament.

L'any 1995 es revisa el model original i apareix la versió COCOMO 2.0. Aquesta versió posterior va passar a anomenar-se COCOMO II que es el nom que manté la versió actual del model. Barry Boehm i els seus col·laboradors continuen desenvolupant el model a la *University of Southern California*[1].

La versió actual té en compte els canvis en el desenvolupament de programari que s'han anat produint des de l'època del model inicial, actualitzant el model al tipus de desenvolupament que es produeixen actualment. Té en compte, per exemple, aspectes com la reutilització de programari, la utilització de llibreries i components comercials (*Commercial-off-the-shelf software*), el desenvolupament de programari distribuït, el desenvolupament orientat a objectes, etc.

[1] Per a més informació consultar l'apartat Referències.

3.1.2 Línies de codi font (SLOC)

Els models COCOMO utilitzen com a unitat mesura línies de codi. En concret, es basen en el concepte de *línies de codi font*, abreujat com SLOC (*Source Line of Code*). Les SLOC tenen les següents característiques:

- Només les línies de codi font que són lliurades al client són comptabilitzades (no es tenen en compte les proves de test ni altres tipus de programari de suport).
- Només es comptabilitzen les línies de codi creades per el personal de desenvolupament. No es té en compte codi creat automàticament per aplicacions de generació de codi.
- Una SLOC és una línia lògica de codi font. Per exemple, una instrucció "if-then-else" serà comptada com una sola SLOC.
- Les declaracions es tenen en compte com a SLOC.
- Els comentaris no són comptabilitzats.

Alternativament a l'ús de línies de codi per a mesurar la mida del programari, el model COCOMO II també admet l'ús de punts funció, si bé utilitza les taules definides per Caper Jones [Jones 95] per a traduir els punts funció a línies de codi. Per a més informació sobre aquestes taules consultar el capítol dedicat al Recompte Indirecte (*Backfiring*).

3.1.3 Equació d'esforç

L'equació principal del model és la utilitzada per a calcular l'esforç necessari per a desenvolupar un projecte. Aquesta equació està expressada en unitats de mesos-persona. La resta de resultats que es poden obtenir amb el mètode són derivats d'aquesta equació.

L'equació d'Esforç té el següent aspecte:

$$\boxed{Esforç = 2.45 * EAF * (KSLOC)^{E} \quad Mesos - persona}$$

Equació 1 Equació d'esforç del mètode COCOMO II

on:

- KSLOC: és el número de SLOC (línies de codi font) expressat en milers. Representa la mida del programari a desenvolupar.
- E: Exponent derivat de cinc paràmetres anomenats *paràmetres d'escala*.
- EAF (*Effort Adjustment Factor*): Factor d'ajustament d'esforç. Valor que intenta representar l'esforç addicional que pot requerir un

determinat entorn de desenvolupament. Es calcula en funció d'uns determinats paràmetres anomenats *paràmetres de cost*.

La constant 2.45 és un valor obtingut empíricament que permet repercutir en l'equació d'esforç la mida del desenvolupament a realitzar. Com més gran és un projecte, més esforços requerirà. Però aquest augment d'esforç no és proporcional a la mida del programari. En les mateixes condicions d'entorn i equip de desenvolupament, un projecte de 10.000 línies de codi no representarà el doble d'esforç que un de 5.000, sinó que requerirà molt més esforç. La gestió del projecte, la comunicació entre els membres de l'equip i altres factors augmentaran de manera no-lineal. Aquesta constant intenta reflectir aquesta situació.

3.1.4 Paràmetres d'escala

Els paràmetres d'escala representen els factors més importants a la hora de determinar el cost i durada d'un projecte. Determinen l'exponent de l'equació d'esforç, la qual cosa implica que tenen un efecte exponencial sobre l'esforç en funció de la mida del programari a desenvolupar.

Cada un dels paràmetres mesura un determinat aspecte de la organització. Els paràmetres d'escala són cinc, i són els següents[2]:

- PREC (*Precedentness*): Antecedents. Mesura l'experiència de la organització en projectes similars a l'actual.
- FLEX (*Development Flexibility*): Flexibilitat de desenvolupament. Mesura com de flexible és la organització pel que fa al desenvolupament.
- RESL (*Architecture/Risk Resolution*): Arquitectura / Resolució de riscos. Determina com es tracten els temes relatius a la resolució de riscos i a l'arquitectura de programari que es desenvoluparà.
- TEAM (*Team Cohesion*): Cohesió de l'equip.
- PMAT (*Process Maturity*): Maduresa del procés. Mesura la maduresa del procés segons les indicacions del model CMM (*Capability Maturity Model*).

En la següent taula, extreta del manual de COCOMO II[3], veiem una manera d'avaluar-los:

[2] La identificació dels paràmetres manté la nomenclatura en anglès per a guardar la uniformitat amb el mètode original. Es proporciona la traducció corresponent en català.
[3] El manual del mètode COCOMO II es pot trobar a la pàgina web de la USC (*University of Southern California*). Veure l'apartat de referències per a més informació.

Scale Factors (W_i)	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	throughly familiar
FLEX	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
RESL[3]	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
TEAM	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
PMAT	Weighted average of "Yes" answers to CMM Maturity Questionnaire					

Taula 1 Valoració dels factors d'escala

Cada paràmetre prendrà un valor entre 0 (*Extra High*) i 5 (*Very low*). Assignarem una puntuació de 0 a cada paràmetre quant la situació de la organització es correspongui amb la descripció indicada a la columna "Extra High". I assignarem una puntuació de 5 quan el paràmetre entri dins la categoria "Very Low". El valor per defecte de cada paràmetre és 3. Es correspon amb la columna anomenada "*Nominal*". Els valors per defecte també són anomenats valors nominals. El fet de disposar de valors nominals és útil quan no es vol calcular un determinat paràmetre o quan, pel que sigui, aquest no es pot valorar.

Els paràmetres d'escala permeten obtenir el valor *E* de l'equació d'esforç del model COCOMO (Equació 1). Per a obtenir el valor d'aquest exponent E cal aplicar la següent fórmula:

$$\boxed{E = 0.91 + 0.01 * \sum w_i}$$

Equació 2 Obtenció de l'exponent E

on w_i representa el valor de cada un dels paràmetres d'escala.

A tall d'exemple, si tenim un projecte que hem estimat en 8000 línies de codi SLOC, i tots els paràmetres d'escala entren dins la categoria "Extra High", tindrem que el valor de l'exponent E de la equació d'esforç tindrà un valor de

$$E = 0.91 + 0.01 * \sum w_i = 0.91 + 0.01 * (0 + 0 + 0 + 0 + 0) = 0.91$$

Si prenem l'equació d'esforç, i prenem la resta de paràmetres amb valor nominal, tindrem que el valor del paràmetre EAF (*Effort Adjustment Factor*) és 1. Amb aquesta consideració, obtindrem el següent resultat per a l'equació d'esforç

$$Esforç = 1 * (8)^{0.91} = 6.63 \ \textit{Mesos-persona}$$

Modificant l'exemple, si tenim que la nostra organització, enlloc d'avaluar tots els paràmetres amb la puntuació "Extra High", ara els avalua amb la puntuació "Very low", el resultat seria el següent:

$$E = 0.91 + 0.01 * \sum w_i = 0.91 + 0.01 * (5 + 5 + 5 + 5 + 5) = 1.16$$

i per tant, l'equació d'esforç, en les mateixes condicions que en el darrer càlcul, ens retornaria el següent resultat:

$$Esforç = 1 * (8)^{1.16} = 11.16 \ \textit{Mesos-persona}$$

És a dir, gairebé el doble d'esforç per a resoldre el mateix problema.

3.1.5 Paràmetres de cost

Els paràmetres de cost són factors multiplicadors que repercuteixen sobre l'esforç necessari per a realitzar el projecte. Són un conjunt de 17 paràmetres que permeten obtenir un valor per al paràmetre EAF de l'equació d'esforç. El valor de EAF prové de la següent equació:

$$\boxed{EAF = \prod_{i=1}^{17} Ci}$$

Equació 3 Obtenció del EAF

on C_i és cada un dels paràmetres de cost.

Els paràmetres de cost són els següents:

- RELY (*Software Required Reliability*): Confiança requerida per el programari
- DATA (*Data Base Size*): Mida de la Base de dades
- CPLX (*Product Complexity*): Complexitat del producte
- RUSE (*Required Reusability*): Reusabilitat requerida
- DOCU (*Documentation match to life-cycle needs*): Documentació d'acord amb les necessitats del cicle de vida

- TIME (*Execution Time Constraint*): Necessitats de temps d'execució
- STOR (*Main Storage Constraint*): Necessitats de emmagatzematge
- PVOL (*Platform volatility*): Volatilitat de la plataforma
- ACAP (*Analyst Capability*): Capacitació d'anàlisi
- PCAP (*Programmer Capability*): Capacitació de programació
- AEXP (*Aplications Experience*): Experiència en les aplicacions
- PEXP (*Platform Experience*): Experiència en la plataforma
- LTEX (*Language and Tool Experience*): Experiència en el llenguatge i les eines
- PCON (*Personnel Continuity*): Taxa de continuitat del personal
- TOOL (*Use of Software Tools*): Ús d'eines de programari
- SITE (*Multisite Development*): Desenvolupament des de més d'una ubicació
- SCED (*Required Developement Schedule*): Dedicació al desenvolupament necessària

A la Taula 2 tenim una guia per a avaluar cada un dels paràmetres dins un rang. Aquest rang, com en la taula presentada per als paràmetres d'escala, varia entre les puntuacions "Very Low" i "Extra High". Puntuarem cada paràmetre de cost entre 0 i 5, assignant zero quan un paràmetre entri dins la categoria "Extra High" i 5 quan entri dins la categoria "Very Low".

COCOMO està basat en l'estudi de mètriques produides per centenars de projectes de programari. La University of Southern California (USC) porta a terme des de fa uns anys un programa de recollida de dades sobre projectes i les seves mètriques. El programa consisteix en una enquesta que es pot omplir anonimament amb les dades de un o més projectes que s'estiguin portant a terme i que hagin utilitzat COCOMO II com a mètode d'estimació. A cada projecte li serà associat un identificador. Aquest idenfiticador permetrà a l'usuari anonim indicar informació sobre l'evolució dels seus projectes. L'objectiu d'aquestes enquestes es l'evaluació de dades de projectes reals que permetin calibrar més acuradament el mètode.

	Very Low	Low	Nominal	High	Very High	Extra High
RELY	slight inconve-nience	low, easily recoverable losses	Moderate, eas-ily recoverable losses	high financial loss	risk to human life	
DATA		DB bytes/Pgm SLOC < 10	10 ≤ D/P < 100	100 ≤ D/P < 1000	D/P ≥ 1000	
CPLX	see Table II-15					
RUSE		none	Across project	across program	across product line	across multi-ple product lines
DOCU	Many life-cycle needs uncovered	Some life-cycle needs uncovered.	Right-sized to life-cycle needs	Excessive for life-cycle needs	Very exces sive for life-cycle needs	
TIME			50% use of available exe cution time	70%	85%	95%
STOR			50% use of available stor age	70%	85%	95%
PVOL		major change every 12 mo.; minor change every 1 mo.	major: 6 mo.; minor: 2 wk.	major: 2 mo.; minor: 1 wk.	major: 2 wk.; minor: 2 days	
ACAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
PCAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
PCON	48% / year	24% / year	12% / year	6% / year	3% / year	
AEXP	≤ 2 months	6 months	1 year	3 years	6 years	
PEXP	≤ 2 months	6 months	1 year	3 years	6 year	
LTEX	≤ 2 months	6 months	1 year	3 years	6 year	
TOOL	edit, code, debug	simple, fron tend, backend CASE, little integration	basic lifecycle tools, moder ately inte grated	strong, mature lifecycle tools, moderately integrated	strong, mature, proactive life cycle tools, well inte grated with processes, methods, reuse	
SITE: Colloc ation	International	Multi-city and Multi-com pany	Multi-city or Multi-com pany	Same city or metro. area	Same building or complex	Fully collo cated
SITE: Comm unicati ons	Some phone, mail	Individual phone, FAX	Narrowband email	Wideband electronic communica tion.	Wideband elect. comm, occasional video conf.	Interactive multimedia
SCED	75% of nomi nal	85%	100%	130%	160%	

Taula 2 Avaluació dels factors de cost

3.2 Probe

Probe és un acrònim de *Proxy-Based Estimation*, que vindria a significar *estimació basada en semblants*. És una tècnica descrita dins el mètode anomenat *Personal Software Process* (*PSP*), que fou dissenyat per Watts S. Humphrey [Humphrey 1995].

Probe permet predir la mida d'un producte i el temps necessari per a construir-lo. Per a fer-ho utilitza coneixement derivat de desenvolupaments de programari anteriors. Estima la mida dels diferents mòduls a desenvolupar a partir d' informació històrica de mòduls semblants. Un cop estimada la mida, estimarà l'esforç necessari per a desenvolupar-lo utilitzant també dades històriques sobre el rendiment de l'organització. Els resultats obtinguts seran ajustats utilitzant regressió lineal per a que tinguin en compte que és possible que es produeixi una desviació.

3.2.1 Detall del mètode

Probe busca la similitud entre els mòduls a desenvolupar i mòduls desenvolupats en projectes anteriors per a obtenir una previsió de la mida que assolirà el nou desenvolupament.

Un cop determinada la mida dels diferents mòduls, el següent pas serà realitzar la previsió per a tot el projecte. Es sumaran les estimacions parcials dels diferents mòduls obtenint l'estimació per a tot el projecte. El procediment per a l'estimació de la mida es veure més en detall en la següent secció (3.2.2).

Amb la mida estimada i utilitzant dades històriques sobre la productivitat de la organització s'obtindrà una estimació de l'esforç necessari per a desenvolupar-lo (en hores-persona).

Finalment, es desenvoluparà el nou producte. De l'execució del projecte se n'obtindrà la mida i els recursos dedicats a cada tasca de la planificació. Aquestes informarà s'afegirà a la base de dades històrica de la organització per a tenir-les disponibles per a posteriors projectes. La informació històrica és crucial per a l'èxit del procés.

A la Figura 1 es pot observar un diagrama que indica el procés d'estimació del mètode.

3.2.2 Estimació de la mida

Probe es basa en informació història obtinguda d'anteriors desenvolupaments.

És necessari disposar d'una bona classificació d'aquestes dades històriques. El mètode assenyala que una manera de fer-ho és utilitzant una taula com la que es pot veure a la Taula 3. En aquesta es classifiquen les dades històriques segons la seva categoria i la seva mida relativa.

	Mida relativa				
	Molt petita	Petitat	Mitjana	Gran	Molt gran
Categoria					
Càlcul	8,3	42,1	75,6	99,6	136,5
Dades	7,5	44,3	82,1	102,4	155,2
E/S	6,9	39,6	71,5	107,5	144,5
Logica	11,2	45,2	69,7	95,2	157,6
Instal·lació	7,7	33,6	83,2	91,1	160,2
Text	9,5	51,2	77,9	89,6	141,6

Taula 3 Classificació de les dades històriques per a determinar la mida

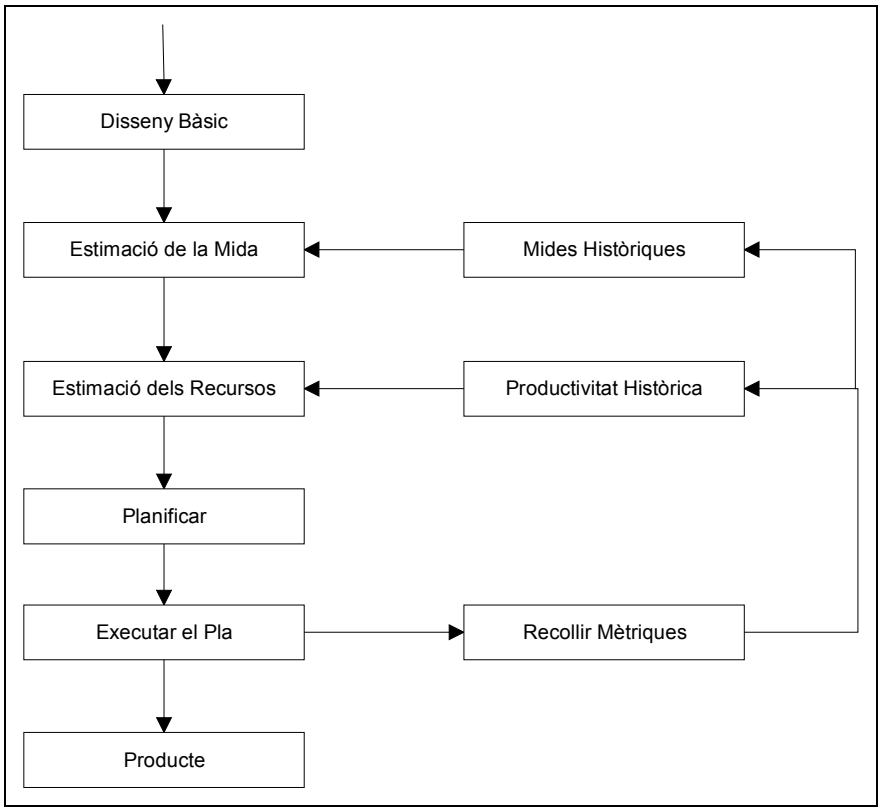

Figura 1 Diagrama de funcionament del mètode Probe

La tècnica utilitzada per a estimar la mida dels nous mòduls de programari es basa en la idea de dividir un problema que no es sap com resoldre en parts més petites fins que sigui possible resoldre'l.

El diagrama de la Figura 2 exposa amb més detall l'etapa d'Estimació de la Mida.

Per a obtenir una estimació de la mida total del projecte, caldrà sumar les estimacions dels diferents mòduls en que s'ha dividit l'aplicació. Per a treballar amb comoditat amb la informació d'aquests mòduls el mètode proposa la utilització d'una taula com de l'exemple de la Taula 4. Com es pot veure, en aquesta es relaciona la categoria associada a cada mòdul i l'estimació obtinguda. Sumant la mida estimada de cada mòdul s'obté la mida total estimada per al projecte a desenvolupar.

Com ja ha estat comentat, el PSP (el procés on es defineix el Probe), treballa en línies de codi per a expressar la mida del programari. Tot i això, aquesta tècnica, el Probe, funcionaria de la mateixa manera utilitzant qualsevol altre tipus d'unitat de mesura, i fins i tot, es podria aplicar a qualsevol tipus d'estimació, encara que no estigui relacionada amb el programari. Per exemple: Línies de Codi, Punts funció, número de formularis d'entrada, pàgines de documentació, diapositives en una presentació, etc.

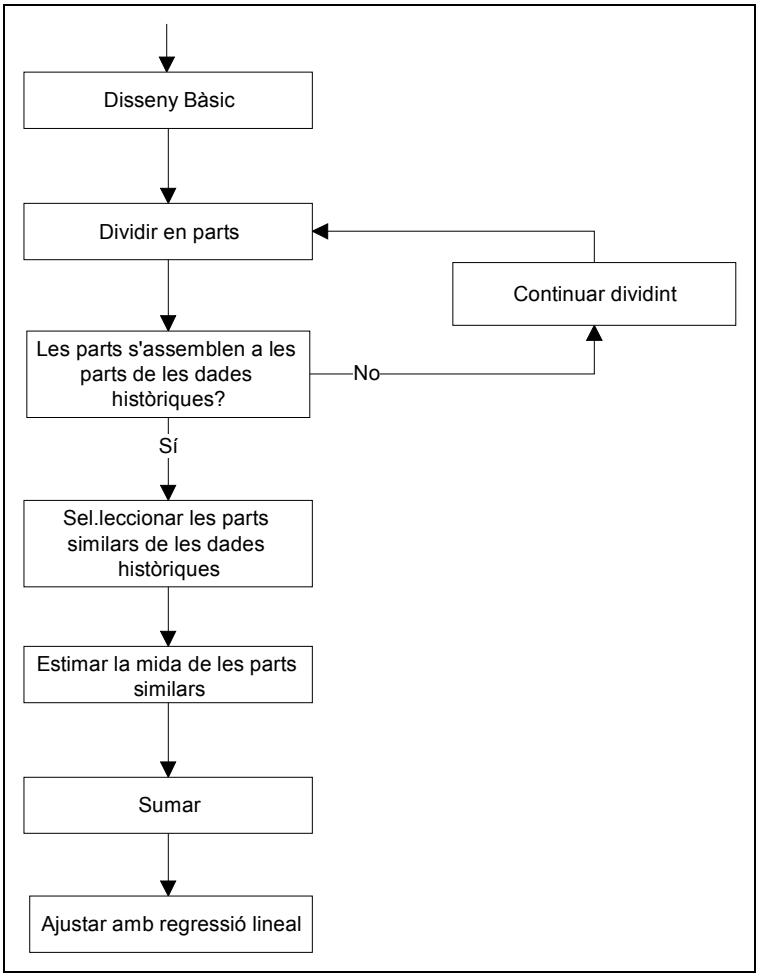

Figura 2 Estimació de la mida amb Probe

Nom	Categoria	Mida relativa	Linies de codi segons dades històriques
Rutina_principal	Instal·lació	Mitjana	83,2
Calcul_recàrrec	Càlcul	Mitjana	75,6
Alta_client	E/S	Petita	39,6
,,,	,,,	,,,	,,,
Alta_producte	E/S	Molt Petita	6,9

Mida Inicial Total Estimada **2684,9 Línies de codi**

Taula 4 Exemple de taula per a estimació de mida

3.2.3 Ajustament de la mida

La Taula 4 permet obtenir una estimació *sense ajustar* de línies de codi per a tot el producte a desenvolupar. El següent pas consistirà en ajustar aquest valor. El Probe utilitza la regressió lineal per a ajustar l'estimació de la mida del programari, contrastant informació de les prediccions que es van fer en el passat amb les mides que finalment els productes van assolir.

Les equacions de regressió lineal que permetran obtenir una estimació *final* de línies de codi són les següents:

$$EstimacióFinalLOC = \beta_0 + (\beta_1 * EstimacióInicialLOC)$$

on:

$$\beta_1 = \frac{\sum (LOC_i Estimades * LOC_i Actuals) - n * LOC_{AVG} Estimades * LOC_{AVG} Actuals}{\sum (LOC_i Estimades)^2 - n * (LOC_{AVG} Estimades)^2}$$

per a *n* projectes anteriors

$$\beta_0 = LOC_{AVG} Actuals - \beta_1 * LOC_{AVG} Estimades$$

Equació 4 Equacions de regressió lineal per a la mida del programari

En aquestes equacions s'ha utilitzat l'acrònim LOC per a fer referència a línies de codi (*Lines of Code*, LOC). També s'ha utilitzat l'acrònim AVG (de l'anglès *Average*) per a fer referència a la mitjana aritmètica.

La regressió lineal traça la línia recta que dista menys de tots els punts (d'un conjunt de punts donat) en un sistema de dos dimensions. S'utilitza la informació de projectes realitzats anteriorment per a realitzar l'estimació. D'aquesta informació històrica, és pren, per a cada projecte, l'estimació inicial que es va realitzar sobre el mateix (en línies de codi), i la mida que va assolir un cop finalitzat (també en línies de codi). En les equacions presentades LOCActuals fa referència a la mida que va assolir un projecte un cop acabat, mentre que LOCEstimades fa referència a la estimació que es va realitzar del projecte.

Si prenem com a exemple les dades dels següents projectes

	Estimada	Actual
Projecte 1	550	675
Projecte 2	1100	795
Projecte 3	1500	1450
Projecte 4	1750	1200
Projecte 5	2100	2315

Figura 3 Dades d'exemple sobre projectes històrics

obtindrem els següents valors

$\beta_0 = -43$
$\beta_1 = 0.96$

Amb els quals s'obtindria la recta de regressió de la Figura 4.

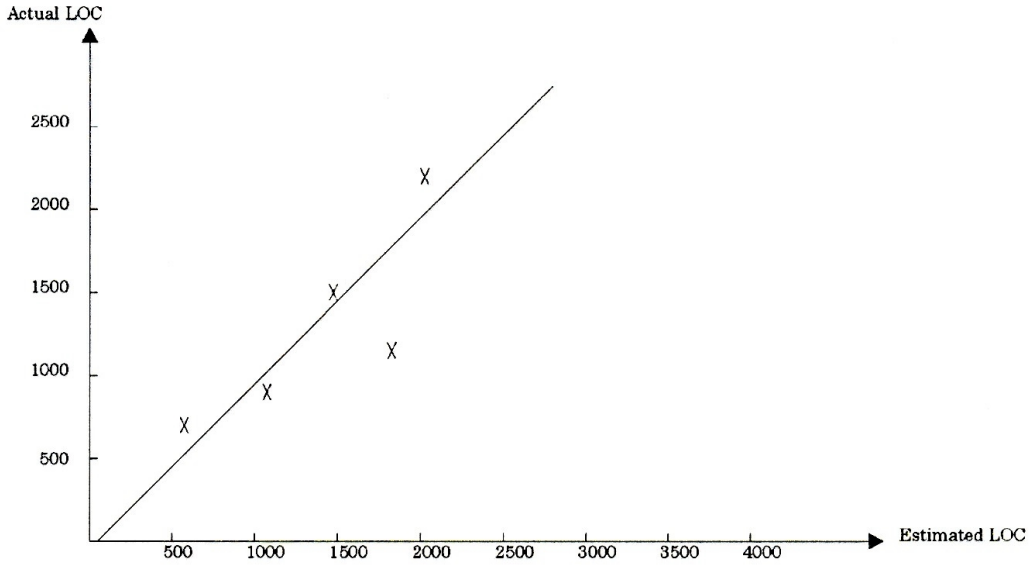

Figura 4 Exemple de recta de regressió

Si suposem que estem intentant mesurar la mida d'un projecte i hem obtingut una estimació inicial total de 1000 línies de codi, el valor final estimat seria

$$EstimacióFinalLOC = \beta_0 + (\beta_1 * EstimacióInicialLOC)$$

$$EstimacióFinalLOC = -43 + (0.96 * 1000) = 917$$

El següent pas seria tornar a utilitzar la regressió lineal per a obtenir una mesura del temps de desenvolupament calculada a partir de l'estimació de la mida acabada de realitzar.

En aquest cas les equacions utilitzen dades històriques sobre el temps que ha implicat el desenvolupament de projectes passat i la mida que es va estimar inicialment per a aquests projectes passats.

Les equacions per a aquesta segona regressió lineal serien les següents:

$$TempsEstimat = \beta_0 + (\beta_1 * LOCEstimades)$$

on:

$$\beta_1 = \frac{\sum (LOC_i Estimades * Temps_i Actual) - n * LOC_{AVG} Estimades * Temps_{AVG} Actual}{\sum (LOC_i Estimades)^2 - n * (LOC_{AVG} Estimades)^2}$$

per a *n* projectes anteriors

$$\beta_0 = Temps_{AVG} Actual - \beta_1 * LOC_{AVG} Estimades$$

Equació 5 Equacions de regressió lineal per al Temps de Desenvolupament

3.2.4 Consideracions finals

Probe transforma el que semblava un problema d'estimació difícil de resoldre en un seguit de petits passos fàcils de dur a terme. Tanmateix, és necessari disposar de bona informació històrica d'anteriors projectes realitzats. Per a aquest motiu, Humphrey, en la descripció del mètode PSP (*Personal Software Process*), posposa l'ensenyament de *Probe* fins que l'estudiant no ha aplicat el mètode PSP en diversos projectes i no ha recollit informació sobre l'execució d'aquests.

Tot i que el mètode Probe va ser concebut per a obtenir estimacions personals (sobre el treball d'una sola persona) aquest és directament adaptable al seu ús en projectes grans que impliquin un grup de persones.

Probe pot estimar la mida i el temps necessari per a realitzar qualsevol tipus de tasca (ja sigui el número de dispositives en una presentació, el temps per a preparar-la, el temps dedicat a realitzar documentació, la mida d'aquesta, etc.).

Capítol 4

Punts Funció

La tècnica dels punts funció va ser concebuda l'any 1977 per Allen Albrecht, que llavors treballava per IBM, com una via per a mesurar la mida del programari i la productivitat.

La idea principal dels punts funció és que la unitats de mesura han de ser independents de les tècniques de desenvolupament i de les decisions d'implementació. Si es compleixen aquestes característiques els resultats poden ser utilitzats per a comparar la productivitat de diferents tècniques i tecnologies.

Els punts funció es basen en la teoria de que les funcionalitats que ha de permetre realitzar una aplicació són la millor mesura per a l'amidament de l'aplicació. Els mètodes basats en punts funció compten operacions funcionals des del punt de vista de l'usuari, com per exemple entrades de dades al sistema, sortides de dades que aquest presenta a l'usuari, i consultes.

Els partidaris dels punts funció apunten que la utilització de línies de codi com a unitat de mesura presenta el problema de que la mida només és coneix amb tota certesa un cop el producte ha estat construït. En canvi, les funcionalitats que ha d'oferir el programari es coneixen des de la definició dels requeriments.

També argumenten que les línies de codi, al ser una mesura que depèn directament de la tecnologia utilitzada, presenta dificultats a la hora de realitzar comparacions de rendiment, especialment entre diferents llenguatges.

El mètode original desenvolupat per Albrecht, conegut com *Function Point Analysis* (FPA) va evolucionar en el mètode IFPUG-FPA. Sobre les seves idees originals s'han desenvolupat altres mètodes, alguns dels quals han assolit certa importància, com veurem en el següent apartat.

4.1 Tendències actuals

Actualment existeixen tres grans projectes que marquen la tendència dels mètodes basats en punts funció.

L'organització IFPUG (*International Function Point Users Group*), originaria dels Estats Units; el projecte Mark II, de la UKSMA (*United Kingdom Software Metrics Association*), originaria del Regne Unit; i el projecte de la organització COSMIC (*Common Software Measurement International Consortium*), gestionada per la UQÀM (*Université du Quebec à Montreal*).

Aquests mètodes han estat estudiats i aprovats per la ISO com a estàndards internacionals. La raó de que tots tres mètodes hagin estat aprovats és perquè la ISO va considerar que no podia escollir quin dels tres mètodes havia de ser l'estàndard aprovat i per tant va considerar que la solució més equànime era aprovar-los tots tres.

D'entre els tres mètodes indicats, en destaca especialment un: COSMIC-FFP. És el mètode més recent (1998) i soluciona alguns dels problemes del mètode original (actualment IFPUG). Tot sembla indicar que tindrà una bona projecció. Per aquest motiu, s'estudiarà més en detall aquest mètode que la resta.

4.2 COSMIC-FFP

COSMIC (*Common Software Metrics International Consortium*)[4] va ser fundada el 1998 i es descriu com una organització voluntària d'experts en amidament de programari. Els seus membres són de diferents països, entre els que trobem Alemanya, Austràlia, Canadà, Finlàndia, Irlanda, Itàlia, Japó, Holanda i el Regne Unit.

El mètode sorgeix en un intent de resoldre el problema que tenia el mètode IFPUG-FPA a la hora d'enfrontar-se amb l'amidament de programari dedicat a tasques de temps real.

Un cop evolucionat el mètode que van desenvolupar han pogut comprovar que a més de respondre correctament davant de problemes amb programari dedicat a aplicacions de temps real, també funciona correctament en programari d'aplicacions tradicionals (banca, assegurances, producció, distribució, etc.). Tot i així, el mètode no ha estat dissenyat per a mesurar programari amb un elevat nombre d'algorismes matemàtics complexes (sistemes experts, simulacions, sistemes de previsió meteorològica, etc.), ni per a programari que processi

[4] Per a més informació consultar la secció Referències, apartat UAQM.

variables continues de so i vídeo (jocs d'ordinador, aparells de música, etc.). Per aquest motiu el mètode no donaria unes bones previsions per a aquest tipus d'aplicació. Per a poder solucionar aquest inconvenient, i poder mesurar aquesta mena de programari, el mètode proposa la utilització del que anomenen "extensions locals", que veurem més endavant.

COSMIC proposa una nova nomenclatura per als seus punts funció per a diferenciar-los dels del mètode IFPUG. COSMIC els anomena FFP (*Full Function Points*).

El mètode es basa en l'aplicació d'una sèrie de regles i mètodes sobre els requeriments funcionals d'usuari del programari a mesurar. Com a resultat de l'aplicació d'aquests mètodes i regles s'obté un número (mesurat en FFP). Aquest número representa la mida funcional del programari, des de la perspectiva de l'usuari.

El mètode COSMIC-FFP ha estat recentment aprovat com a estàndard internacional per la ISO. La referència d'aquesta norma és ISO/IEC 19761:2003.

4.2.1 Descripció del mètode

El mètode consta de tres fases:

1. Extracció dels requeriments funcionals de l'usuari.
2. Fase de Representació, on els requeriments funcionals obtinguts seran homogeneïtzats per a adaptar-los al funcionament del model.
3. Fase d'Amidament, que mesurarà la representació obtinguda en la fase anterior.

A part d'aquestes tres fases hi ha un apartat final de presentació del resultat de l'amidament realitzat amb el mètode. Aquesta tasca final és més una convenció de com s'haurien de presentar els resultats per a que qualsevol persona, hagi participat en l'amidament o no, sigui capaç d'entendre les característiques concretes que l'organització hagi pogut aplicar en el procés.

L'objectiu de la **primera fase** és determinar les funcionalitats que el programari haurà de proporcionar a l'usuari. És a dir, els requeriments funcionals des del punt de vista de l'usuari.

És possible que aquesta informació ja hagi estat fixada en un document (un document amb les especificacions funcionals requerides, per exemple) o bé que calgui obtenir aquests requeriments a partir d'altres elements d'enginyeria del programari (per exemple, documents de concepció i arquitectura).

En altres ocasions ens podem trobar que no existeix documentació o bé que aquesta no ha estat actualitzada a mesura que s'han anat fent canvis i no està al dia. En aquests casos el sistema sol estar construït, i l'objectiu de l'aplicació del mètode és de cares a obtenir una estimació per a una possible modificació del programari o una ampliació de les seves funcionalitats. En aquesta situació encara és possible derivar els requeriments funcionals a partir de l'anàlisi del programari instal·lat en el sistema informàtic. Aquesta tasca, però, pot representar una mica més d'esforç en comparació amb les primeres situacions comentades.

La **segona fase**, anomenada fase de representació, considera com a entrada els requeriments funcionals recollits a la primera fase. A partir d'aquest conjunt de requeriments, s'utilitzen un conjunt de regles i procediments definits que permetran generar un model de programari (el model de programari COSMIC-FFP) sobre el qual es treballarà amb el mètode. Aquest model de programari serà apropiat per a mesurar la mida funcional del sistema.

El model de programari COSMIC-FFP està format per quatre tipus d'elements:

- Capes de programari. Les capes són particions del programari en el qual tots els mòduls que en formen part tenen el mateix nivell d'abstracció. Una capa inferior ha de proporcionar serveis a la capa immediatament superior. Un exemple pot ser la divisió que hi ha en entorn web sobre la lògica de negoci i l'aspecte.
- Fronteres. Són separacions implícites que permeten separar els elements de formen part de l'interior dels que formen part de l'exterior d'un component.
- Usuaris. Poden ser éssers humans, dispositius d'enginyeria o altres. Les capes també són considerades usuaris: una capa és considerada com un usuari de la capa inferior.
- Requeriments funcionals d'usuari. Descriuen les funcionalitats que ha de donar el sistema a l'usuari, sense tenir en compte requeriments tècnics o de qualitat.

A part, cal que es compleixin dos principis generals:

1. El programari que es vol representar amb el model és alimentat per entrades de dades i genera sortides de dades útils per als usuaris. És a dir, es defineixen fluxos d'informació d'entrada i sortida de dades.

2. El programari que es vol representar amb el model manipula parcel·les d'informació. Aquestes parcel·les d'informació, anomenades en el mètode *grups de dades*, tenen atributs associats. En definitiva: es defineix la manipulació de dades.

Tenint present aquestes definicions i principis, s'avaluaran els requeriments funcionals obtinguts a la primera fase del mètode i es traduiran a un model que basat en aquests criteris. És una operació de traducció. L'objectiu que es persegueix és representar la informació que tenim sobre els requeriments funcionals d'usuari d'una manera ordenada sigui quin sigui l'origen d'aquesta informació i el seu format. A la figura següent es pot veure un esquema del processos que tenen lloc en aquesta fase.

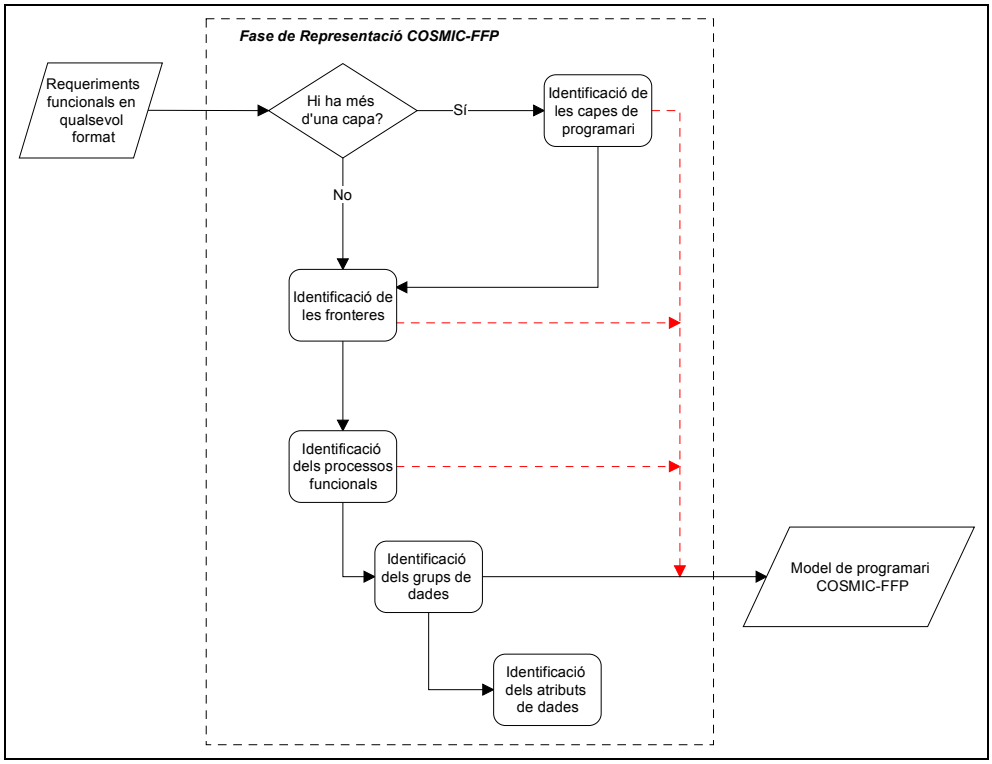

Figura 5 Fase de Representació COSMIC-FFP

La **tercera fase**, anomenada *fase d'amidament*, té per objectiu obtenir un valor que representi la mida del programari que es vol mesurar. Aquesta fase considera com a entrada una instància del model COSMIC-FFP generat a la fase anterior. Sobre aquesta entrada, aplica un conjunt definit de regles i procediments per a acabar obtenint un valor que serà directament proporcional a la mida funcional del model.

El procediment es basa en el que el model anomena *Principi d'Amidament*, que diu el següent:

> Principi d'Amidament: La mida funcional del programari és directament proporcional al número dels seus sub-processos de moviments de dades.

Per a entendre el principi, cal definir el concepte de **sub-procés**. Segons COSMIC, un sub-procés és un moviment de dades que té lloc durant l'execució d'un procés funcional. Existeixen quatre tipus de sub-processos: entrada, sortida, lectura i escriptura. Cada un d'aquests sub-processos pot incloure altres sub-processos específics de manipulació de dades, però aquests no seran tinguts en compte per el mètode COSMIC-FFP. Un sub-procés COSMIC només té en compte els requeriments funcionals des de la perspectiva de l'usuari i exclou els requeriments derivats de motius tècnics o de qualitat.

Sobre el model COSMIC-FFP obtingut en la segona fase del mètode, s'hi aplicaran un conjunt de regles. Totes les regles i procediments que s'aplicaran en aquesta fase d'amidament tenen en comú quatre característiques que regeixen tot el procés. Aquestes característiques són les següents:

- Funció d'amidament: A cada instància d'un sub-procés li és assignada una quantitat numèrica a través d'una funció d'amidament.

- Unitat de mesura: Es defineix la unitat de mesura 1 Cfsu (*Cosmic Functional Size Unit*), la qual equival a un moviment de dades a nivell de sub-procés.

- Additivitat: la mida funcional d'un procés funcional és definida com la suma aritmètica de les mides funcionals dels sub-processos que el conformen. Per altra banda, la mida funcional de una modificació en un mòdul de programari és la suma aritmètica de les mides funcionals de tots els sub-processos afegits, modificats i eliminats del mòdul modificat.

- Sub-unitat de mesura: Quan sigui necessari un grau de precisió addicional per a mesurar el moviment de dades, podrà ser definida una sub-unitat de mesura. Per analogia, un metre pot ser subdividit en 100 centímetres o 1000 mil·límetres, i el mateix criteri es seguirà amb la unitat de mesura Cfsu.

La primera de les característiques indicades fa referència a una ***funció d'amidament***. Aquesta funció consisteix en assignar una unitat Cfsu a cada instància d'un sub-procés. És a dir, per a cada sub-procés d'un procés funcional, li assignarà un valor de 1 Cfsu.

Els processos que tenen lloc a la fase d'amidament són tres: a) identificació de sub-processos, b) aplicació de la funció d'amidament i c) agregació dels resultats.

L'esquema de la Figura 6 representa el funcionament de la fase d'amidament i permet veure on tenen lloc aquests processos. La idea és identificar tots els moviments de dades (sub-processos: entrades, sortides, lectures i escriptures) que tinguin lloc en els diferents processos que formen part del programari a mesurar. Un cop identificats, se'ls hi aplicarà la funció d'amidament. Aquest procés es repetirà fins que tots els sub-processos hagin estat tinguts en compte. En aquest moment, es sumaran tots els resultats parcials per a obtenir un valor que representarà la mida funcional del programari.

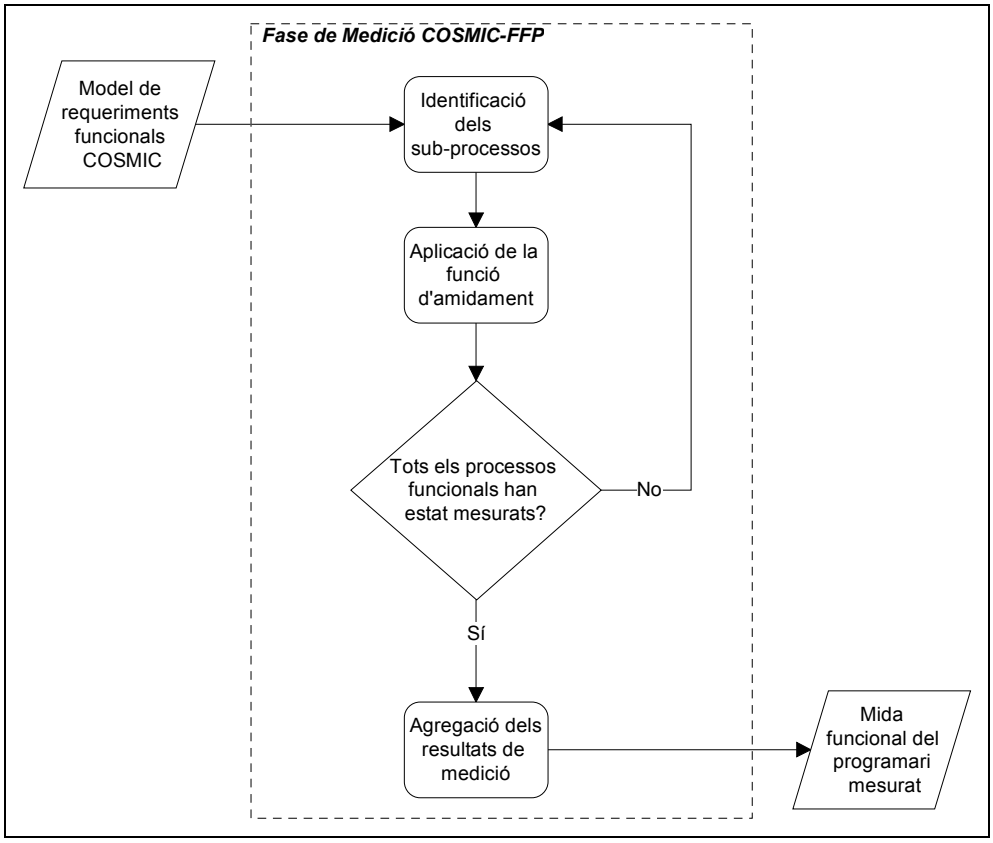

Figura 6 Fase de medició COSMIC-FFP

4.2.2 Extensions locals del mètode

Tot i que el mètode COSMIC-FFP en un principi va ser dissenyat per a poder aplicar les bases teòriques del punts funció al programari dedicat a tasques de temps real, s'ha pogut comprovar que dóna bons resultats en molts altres camps de

la informàtica. Malgrat això, hi ha alguns camps en què el mètode no proporciona resultats congruents amb el problema que es vol mesurar. És el cas, per exemple, d'aplicacions amb un elevat component d'algorísmica complexa o de sistemes experts que requereixen grans seqüències de regles complexes.

Per a mirar de solucionar aquest inconvenient, el mètode proposa la utilització del que anomenen *extensions locals del mètode*. Aquesta solució consisteix en que sigui la pròpia organització la que assigni les seves pròpies unitats de mida funcional a aquells sub-processos més complexes del normal. La organització haurà de tenir curar de redactar les regles estàndards que formaran part de les extensions locals del mètode. Un exemple d'una d'aquestes regles podria ser el següent:

> *En la nostra organització, valorarem amb tres unitats de mida funcional els algorismes que realitzin operacions d'extrapolació d'informació sobre les imatges obtingudes per les nostres càmeres.*

En utilitzar extensions locals cal tenir en compte de reflexar-ho adequadament en la fase de presentació de la informació d'amidament.

4.3 IFPUG-FPA

IFPUG (*International Function Points User Group*)[5] va ser fundada el 1986. Aquesta organització ha continuat desenvolupant i adaptant el treball original d'Allen Albrecht amb el nom de IFPUG-FPA (de les sigles *Function Point Analysis*). Recentment aquest mètode ha estat aprovat com a estàndard internacional per la ISO. La referència d'aquesta norma és ISO/IEC 20926:2003.

El mètode és àmpliament utilitzat. Més de 1200 empreses són afiliades de la IFPUG.

Segons el mètode, els punts funció estan classificats en cinc categories. El mètode associat a cada categoria un pes concret sobre el total. La Taula 5 presenta les 5 categories i el pes associat a cada categoria:

El mètode consisteix comptar el número de punts funció que tindrà l'aplicació a partir dels seus requeriments. Cada punt funció localitzat serà valorat en funció de la seva complexitat en tres categories (simple, mitjana, alta) i separat segons la categoria a la que pertanyi. El resultat de la suma dels punts funció de cada categoria serà multiplicat per el pes corresponent a la categoria.

[5] Per a més informació consultar la secció Referències.

Tipus de punt funció	Pes
Entrades	4
Sortides	5
Consultes	4
Fitxer lògics de dades	10
Interfícies a altres aplicacions	7

Taula 5 Classificació dels punts funció

El número resultant per a cada categoria serà sumat per a obtenir el *número de punts funció sense ajustar*, al qual se li aplicarà un factor de cost, anomenat factor de complexitat tècnica, per a obtenir el *número de punts funció ajustats*, que representarà una mesura de la seva mida funcional. El factor de complexitat tècnica té en compte una sèrie de factors tècnics i de qualitat derivats dels requeriments del programari a realitzar.

El mètode presenta algunes limitacions. Destaca el fet que està molt orientat als sistemes de gestió d'informació i no realitza bones estimacions per a programari amb un alt contingut de complexitat algorísmica i un número reduït d'operacions amb fitxers.

4.4 MKII-FPA

Va ser definit per Charles Symons el 1991, per a poder valorar més correctament la complexitat de les operacions interiors que realitzava el sistema. El mètode original dels punts funció es fixa molt més en el punt de vista de l'usuari i no té tant en compte els processos interns.

Symons va observar alguns problemes amb el mètode original, entre ells que la classificació que realitza el mètode dels punts funció en funció de la seva complexitat (simple, mitjana i alta) era massa simple i això provocava que mòduls amb una elevada complexitat no fossin correctament valorats.

El mètode, enlloc d'utilitzar les cinc classes de punts funció definides en el mètode clàssic, veu el sistema com una col·lecció de transaccions lògiques. Cada transacció està formada per un component d'entrada, un component de procés i un component de sortida. Una tipus de transacció lògica és definit com una única combinació de entrada/procés/sortida disparada per un event d'interès per l'usuari. Per exemple:

- Donar d'alta un client
- Modificar un registre
- Realitzar un llistat

El mètode també amplia la relació de factors de cost que té en compte. En general, el mètode ajusta més finament l'amidament de programari respecte al mètode original.

El mètode MKII-FPA, igual que el IFPUG-FPA, no realitza bones estimacions en determinats sistemes de programari, entre els que es troben aquells amb una elevada complexitat algorísmica o projectes basats en Internet.

Actualment està gestionat per la organització *United Kingdom Software Metrics Association* (UKSMA).

Capítol 5

Punts Característica

L'any 1986, l'empresa *Software Productivity Research*[6], va desenvolupar una tècnica per a poder aplicar els raonaments en que es basen els mètodes de punts funció, a aquells tipus de programari no suportat per el mètode original.

L'amidament de programari amb punts funció no presentava bons resultats quan es tractava d'avaluar programari amb una elevada complexitat algorísmica i un baix número d'entrades i sortides. Aquest era el cas, per exemple, de programari de sistemes, com sistemes operatius, sistemes de comunicacions (commutadors PBX), etc.

Quan la tècnica dels punts funció s'aplica a aquest tipus de sistemes el resultats que s'obtenen no es corresponen amb l'esforç real que representen aquesta mena de sistemes. L'esforç que prediu és inferior a l'esforç real. El motiu d'aquest desajustament és que el mètode original no té en compte la complexitat algorísmica de la funcionalitat a desenvolupar.

Des d'un punt de vista pràctic, només feia falta un mètode similar al de punts funció, però que, a més, fos sensible a les dificultats aportades afegides de complexitat algorísmica. Davant d'aquesta situació, varen desenvolupar un mètode que contemplava aquesta complexitat algorísmica i que varen anomenar *punts característica* (*Feature Points*).

En aquest sentit, la tècnica dels punts característica vindria a ser un superconjunt de la tècnica dels punts funció, ja que inclou els cinc paràmetres estàndard dels punts funció, però a més introdueixen un nou paràmetre, anomenat *algorismes*, que permet tenir en compte la complexitat algorísmica del sistema a desenvolupar. Aquest paràmetre per defecte té assignat un pes de 3.

[6] Per a més informació consultar el capítol Referències

A part, el mètode també difereix de l'original en el fet que redueix el pes del paràmetre *Fitxers de Dades Lògics* (de 10 a 7). Aquesta reducció en la ponderació del paràmetre reflecteix la pèrdua d'importància dels fitxers lògics en el programari de sistemes respecte al programari de gestió.

D'aquesta manera, per a aquelles aplicacions on el número d'algorismes i fitxers de dades lògiques són iguals, s'obté el mateix resultat utilitzant el mètode dels punts funció que amb el mètode dels punts característica. En canvi, quan es tracta d'aplicacions amb més algorismes que fitxers lògics (característica comuna en el programari de sistemes) el mètode dels punts característica retorna un valor total superior a l'obtingut amb els punts funció clàssics, més d'acord amb la realitat. Per contra, si el sistema a mesurar té més fitxers lògics que algorismes, el mètode dels punts característica retornarà un total inferior a l'obtingut amb el mètode dels punts funció.

Quan els punts funció i els punts característica són utilitzats per a mesurar sistemes d'informació típics, els valors obtinguts són pràcticament idèntics.

Per exemple, en avaluar un sistema d'informació de mida petita es va obtenir un resultat de 107 punts característica i de 107 punts funció. En canvi, en avaluar un sistema una mica més complex, com per exemple un commutador telefònic PBX, el total de punts funció obtingut va ser de 1.845, mentre que el total de punts característica va ser de 2.300 degut a l'elevada complexitat algorísmica del sistema.

Com que el mètode va donar bons resultats en el camps del programari de sistemes, també va ser provat experimentalment en projectes de diferent tipologia: programari de temps real, CAD, intel·ligència artificial i fins i tot en sistemes de gestió, obtenint bons resultats en tots aquests camps.

5.1 Definició d'algorisme

Com que els punts característica tenen en compte la complexitat algorísmica, és necessari definir el concepte d'algorisme. Un algorisme, segons la definició estàndard dels texts d'enginyeria del programari, és un conjunt de regles que han de ser correctament expressades i codificades per a solucionar un determinat problema computacional. Així, per exemple, serien considerats algorismes el càlcul d'una data juliana o la implementació del mètode del càlcul d'una nòmina.

La Taula 6 permet veure la semblança entre el mètode dels punts funció i els punts característica. Aquesta taula serà la utilitzada en l'aplicació del mètode i inclou el pes que té cada un dels sis paràmetres que té en compte.

Paràmetre	Ocurrències	Pes empíric	Total
Número d'algorismes	_____	x 3 =	_____
Número d'entrades	_____	x 4 =	_____
Número de sortides	_____	x 5 =	_____
Número de consultes	_____	x 4 =	_____
Número de Fitxers de dades	_____	x 7 =	_____
Número d'interfícies	_____	x 7 =	_____
Total sense ajustar			_____
Ajust per complexitat			_____
Total de punts característica ajustat			_____

Taula 6 Mètode Punts Característica

5.1.1 Regles per a avaluar algorismes

El rang de puntuació per a mesurar la complexitat d'un algorisme oscil·la entre 1 i 10. Un algorisme que utilitzi operacions aritmètiques bàsiques i algunes operacions condicionals tindrà una puntuació de 1. A un algorisme amb equacions complexes, operacions amb matrius, o processos lògics i matemàtics complicats se li podria associar un valor de 10.

El mètode aconsella puntuar per defecte amb 3 un algorisme normal amb ús normal de matemàtiques.

5.2 Escollir entre punts funció i punts característica

En aplicacions on el número d'algorismes és incert, o on la complexitat algorísmica no és significant, segurament els punts funció seran una bona elecció. Moltes aplicacions empresarials es troben en aquesta categoria, com per exemple aplicacions de gestió de vendes, aplicacions de gestió i bases de dades, i altres.

En canvi, en aplicacions on el número d'algorismes és comptable i on els algorismes tenen un pes important i són d'una certa complexitat, el mètode dels punts característica segurament serà una tècnica més encertada. En aquesta categoria hi trobem aplicacions científiques, d'enginyeria, aplicacions de control de sistemes com commutadors de sistemes de comunicació, sistemes CAD, sistemes operatius, sistemes de control de processos o sistemes en temps real.

Tot i que la tècnica dels punts característica encara és considerada una tècnica experimental, els resultats obtinguts en la seva aplicació, en comparació amb els punts funció, són interessants i prometedors. La Taula 7 presenta una equivalència entre punts funció i punts característica obtinguda empíricament en l'avaluació de diferents tipus d'aplicació. A la taula, es fa referència a aplicacions SIG (Sistemes d'Informació de Gestió, terme traduït de l'anglès MIS, *Management Information Systems*).

Tipus d'aplicació	Punts funció	Punts característica
Aplicacions SIG en batch	1	0,8
Aplicacions SIG interactives	1	1
Aplicacions de Bases de Dades	1	1
Commutadors de comunicacions	1	1,2
Control de processos	1	1,28
Aplicacions a temps real	1	1,35
Automatització de fàbriques	1	1,5

Taula 7 Ratis de conversió entre Punts funció i Punts característica

Alguns dels programes d'ajuda a l'estimació de costos i amidament que es poden trobar al mercat permeten treballar amb els dos mètodes i comparar-ne el resultat. És recomanable aplicar els dos mètodes en realitzar una estimació i escollir el que proporcioni un resultat més elevat.

Capítol 6

Recompte Indirecte (*Backfiring*)

Una de les dificultats d'abordar un pla d'estimació que exigeixi la mesura del programari en punts funció és l'elevat cost inicial que pot suposar el recompte d'un número suficient d'aplicacions que permeti l'estimació de la *productivitat* que s'utilitzarà per a realitzar les futures prediccions.

Una possibilitat per a reduir aquest cost és la utilització del mètode anomenat recompte indirecte (*Backfiring*). Aquest mètode fou desenvolupat per Caper Jones el 1995 [Jones 1995]. Consisteix en comptar el número d'instruccions lògiques d'aplicacions ja existents i utilitzar el resultat obtingut per a estimar el número de punts funció d'aquestes aplicacions aplicant uns factors de conversió. L'avantatge principal del mètode és la seva possible automatització, ja que existeixen eines en el mercat que permeten obtenir el número d'instruccions lògiques d'una aplicació o mòdul donat.

El fonament d'aquest mètode es basa en la correlació observada empíricament per Jones entre el número de punts funció d'un sistema i el número d'instruccions necessàries per a implementar la corresponent funcionalitat en un llenguatge concret.

L'empresa *Software Productivity Research*[7] presenta, a través de la seva pàgina web, una taula que conté les relacions d'equivalència obtingudes per Jones entre un elevat conjunt de llenguatges de programació i el número de línies necessari per a implementar un punt funció en cada llenguatge (veure Taula 8).

El recompte indirecte és un mètode completament empíric. Caper Jones assenyala que la quantitat i qualitat de les dades disponibles per a la elaboració de la taula de conversió no permet assegurar que el marge d'error sigui reduït. Per aquest motiu aconsella prendre els valors amb molta cautela.

[7] Per a més informació consultar el capítol Referències

Llenguatge	Instruccions per Punts Funció
Ensamblador Bàsic	320
Macroensamblador	213
C	128
COBOL 74	107
COBOL 85	91
C++	53
Java	53
NATURAL 2	46
Visual C++	34
Visual Basic 5	29
Powerbuilder	16
SQL	13

Taula 8 Recompte Indirecte. Relacions d'equivalencia

Per a utilitzar-lo, però, cal tenir en compte les condicions locals de la organització que pretén utilitzar el mètode, procurant, si és possible, validar aquests valors amb la seva manera de construir programari. Caper Jones destaca que no ha estat trobada cap relació consistent, discreta i definitiva entre el número d'instruccions lògiques de codi i els punts funció.

Malgrat aquests inconvenients, la senzillesa del mètode ha fet que tècniques clàssiques basades en línies de codi, com el COCOMO[8], en facin referència com a una possibilitat per a convertir amidaments realitzats en punts funció a línies de codi. D'aquesta manera, podran ser utilitzats amb el COCOMO i s'amplia el seu camp d'ús. Cal tenir present, però, que la taula del recompte indirecte va ser dissenyada inicialment per a convertir línies de codi a punts funció (i el mètode COCOMO pretén utilitzar-lo en sentit invers: convertir punts funció a línies de codi). Tot i això, és d'esperar que els resultats també es puguin considerar vàlids.

[8] Per a més informació sobre el mètode COCOMO consultar la secció 3.1.

Capítol 7

Punts Cas d'Ús

El terme *punts cas d'ús* prové del treball realitzat per Gustav Karner el 1993 mentre treballava per l'empresa *Objectory AB* (actualment *Rational Software*) sota la supervisió de Ivar Jacobson. El seu treball, anomenat "*Use Case Points - Resource Estimation for Objectory Projects*", és en realitat la seva tesi doctoral [Karner 1993]. El treball de Karner es basa sobre els conceptes dels punts funció originals d'Allen Albrecht.

7.1 Història dels casos d'ús

A finals dels anys 60, mentre treballava a Ericsson, Ivar Jacobson va idear el que més tard va rebre el nom de casos d'ús. En aquella època Ericsson va modelar el seu sistema com un conjunt de blocs interconnectats, que més tard varen esdevenir els subsistemes de l'UML. Els blocs s'establien a partir d'uns *casos de tràfic* prèviament especificats, més tard coneguts com a *casos d'ús*.

Jacobson va deixar Ericsson el 1987 i va fundar *Objectory AB* a Estocolm, on ell i els seus associats varen desenvolupar un procés de desenvolupament anomenat *Objectory*, una abreviació de "Object Factory".

El 1992, Jacobson ideà la metodologia de programari OOSE (*Object Oriented Software Engineering*), basada en els casos d'ús en la qual els casos d'ús participaven en totes les etapes de desenvolupament: anàlisi, disseny, validació i test.

El 1995, Jacobson i la seva empresa es van unir a *Rational Software Corporation*, empresa que en aquell moment estava unificant diverses metodologies amb l'objectiu de proposar-ne una com a estàndard internacional per al desenvolupament de programari orientat a objectes. El model OOSE es va unir al

treball de Grady Booch, i al mètode OMT (de Jim Rumbaugh i altres), per a acabar creant el UML (*Unified Modeling Language*), que finalment fou publicada com a estàndard per la OMG (*Object Management Group*).

7.2 Definició de cas d'ús

El terme *cas d'ús* implica la manera en que un usuari (actor) utilitza un sistema. És una col·lecció de possibles seqüències d'interacció entre el sistema a construir i els actors externs a aquest per a aconseguir un determinat objectiu. Els actors són persones o altres sistemes informàtics, i el sistema és una entitat que única que interactua amb els actors.

El propòsit d'un cas d'ús és trobar l'objectiu més immediat d'un actor, com per exemple consultar un registre. Per a aconseguir aquest objectiu s'han de realitzar una sèrie d'accions. Cada actor té un conjunt de responsabilitats, que s'enllacen amb responsabilitats d'altres actors. Els casos d'ús assenyalen les funcionalitats que el sistema ha de permetre a cada actor. I són aquestes funcionalitats i la seva complexitat el que cal mesurar per a aconseguir estimar la mida del sistema a desenvolupar i el cost que representarà per a una organització.

Els casos d'ús, per la seva naturalesa, s'utilitzen per a la elaboració dels requeriments d'un sistema. Un dels seus avantatges és la independència de la tecnologia que serà utilitzada per a la seva implementació, avantatge que comparteix amb el mètode dels punts funció.

És freqüent representar els casos d'ús gràficament. Un exemple el tenim a la Figura 7, extreta del material docent de l'assignatura Enginyeria de Programari III (Enginyeria Informàtica, UOC). En aquests tipus de representacions, les el·lipses representen els casos d'ús i les figures els actors del sistema.

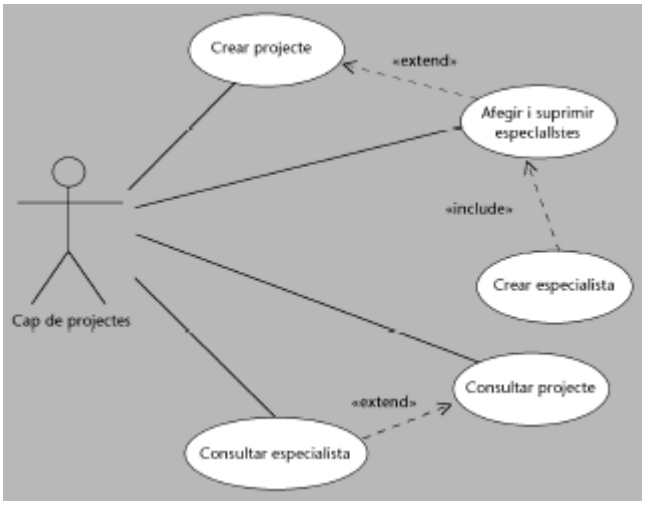

Figura 7 Exemple de diagrama de casos d'ús

7.3 Funcionament del mètode

A grans trets, el funcionament del mètode consisteix en el següent:

1) Comptar determinats aspectes dels requeriments (del model de casos d'ús) per a aconseguir un valor anomenat *unadjusted use case point*.
2) Avaluar les complexitats tècniques del projecte i l'entorn de la organització per a tenir-ho en compte en l'estimació.
3) Multiplicar el valor aconseguit en el pas 1 (*unadjusted point count*) per el factor obtingut a partir del pas 2 per a aconseguir el valor anomenat *adjusted point count*.
4) El número de punts cas d'ús ajustats serà traduït a una mesura del nivell d'esforç requerit (*LOE, Level of effort*) per a desenvolupar-lo, que és mesurat en unitats de hores-persona (o mesos-persona).

A grans trets, aquest és el procediment general. Com es pot veure, el raonament que segueix aquest mètode és molt similar al dels mètodes de punts funció.

Ara veurem el mètode en més detall.

7.3.1 Avaluació del model de casos d'ús

El treball que requerirà l'anàlisi i implementació d'una determinat programari està relacionat amb la complexitat del seu diagrama de casos d'ús. Per a avaluar la complexitat d'un model de casos d'ús es tenen en compte els actors que hi participen i els casos d'ús que hi intervenen. L'objectiu d'avaluar aquests dos

aspectes d'un model de casos d'ús és obtenir un valor que representi la complexitat i mida del model. Aquest valor és anomenat *punts cas d'ús sense ajustar* (*unadjusted use case points*, UUCP). Aquest valor és anomenat *sense ajustar* perquè no té en compte l'entorn en que es desenvoluparà el projecte, l'entorn de la organització. En posteriors fases del mètode, es mesura aquest entorn per a poder ajustar el número de punts cas d'ús.

Avaluació dels actors

El primer que cal determinar és el número d'actors que participen en el sistema. Els actors són elements externs del sistema que interactuen amb aquest. Exemples d'actors són: usuaris finals, altres programes, magatzems de dades, etc. Els actors es classifiquen segons la seva complexitat en simples, mitjans i complexes.

- Un actor qualificat com a simple seria un sistema o programa que interactua d'alguna manera amb l'aplicació a través d'una API (*Application Programming Interface*) dissenyada per a aquests efectes.

- Un actor és considerat mitjà en el cas que es tracti d'un usuari que interactua amb el nostra sistema a partir d'un entorn de línia de comandes o bé quan es tracta d'un sistema extern que interactua amb el nostre sistema utilitzant algun protocol (com HTTP, TCP/IP, etc.), sense utilitzar cap API. Aquests són considerats actors de dificultat mitjana per la complexitat d'avaluació dels resultats de les proves de test. Els resultats de les proves de test necessitaran ser verificats manualment a través de sentencies SQL, o verificant informació sobre els temps d'execució dels protocols de comunicació utilitzats, per exemple.

- Un actor serà qualificat com a complex quan representi una persona interactuant amb el sistema a través d'una interfície gràfica.

Un cop identificats els diferents actors del sistema cal ponderar-los segons la categoria a la qual pertanyen per a reflectir la seva complexitat. Aquesta ponderació es realitzarà aplicant un factor de pes a cada categoria d'actor. Aquests factors són els anomenats *factors de ponderació* (*Weighting factors*). En aquest sentit, a un actor simple li correspondrà un factor 1, un actor mitjà tindrà un factor 2 i els actors considerats complexos tindran un factor 3.

Tipus d'actor	*Descripció*	*Factor*
Simple	Interfície de sistema	1
Mitjà	Interfície interactiva o via un protocol	2
Complexe	Interfície Gràfica	3

Taula 9 Factors de ponderació per als actors

Amb la suma de tots els actors del sistema multiplicats per el seu factor corresponent s'obté un valor anomenat *pes dels actors sense ajustar* (*Unadjusted Actor Weights, UAW*). El UAW s'obté comptant el número d'actors de cada categoria, multiplicant-los per el seu factor de ponderació corresponent, i finalment, sumant el resultat d'aquest productes. A la Taula 10 es pot observar un exemple d'obtenció del pes dels actors sense ajustar (UAW) d'un model de casos d'us inventat, format per 2 actors simples, 2 actors mitjans i 4 actors complexes.

Tipus d'actor	*Descripció*	*Factor*	*Número d'actors*	*Resultat*
Simple	Interfície de sistema	1	2	2
Mitjà	Interfície interactiva o via un protocol	2	2	4
Complexe	Interfície Gràfica	3	4	12
Pes dels actors sense ajustar				**18**

Taula 10 Exemple d'obtenció del UAW

Avaluació dels casos d'ús

El següent pas consisteix en determinar una mesura del número de casos d'ús del sistema. En aquest cas, el valor obtingut és anomenat *pes dels casos d'ús sense ajustar* (*Unadjusted Use Case Weight, UUCW*). Com amb els actors, un cas d'ús té un pes associat segons la seva complexitat. Es distingeixen tres categories de complexitat per els casos d'ús: simple, mitjana i complexa.

La complexitat d'un cas d'ús és mesurada en funció del número de transaccions d'aquest.

Una transacció es defineix com un conjunt d'activitats que es realitzen de manera atòmica. És a dir, de manera que o bé es realitzen totes o no se'n realitza cap.

Un cas d'ús és considerat simple quan conté d'una a tres transaccions. Quan s'hi identifiquin de quatre a set transaccions serà considerat mitjà. I serà considerat complex quan tingui més de 8 transaccions. La Taula 11 presenta el factor de ponderació que s'aplicarà a cada cas d'ús segons la seva complexitat.

Categoria	Descripció	Factor
Simple	1 - 3 transaccions	5
Mitjà	4 - 7 transaccions	10
Complexe	més de 7 transaccions	15

Taula 11 Factors de ponderació per als casos d'ús

Per a obtenir el pes dels casos d'ús sense ajustar (UUCW) es comptaran el número de casos d'ús corresponents a cada categoria, es multiplicaran per el factor de ponderació corresponent i es sumarà el resultat d'aquests productes. La Taula 12 presenta un exemple de l'obtenció del pes dels casos d'ús sense ajustar per a un model de casos d'ús format per 24 casos d'ús de diferents categories.

Categoria	Descripció	Factor	Núm. Casos	Resultat
Simple	1 - 3 transaccions	5	8	40
Mitjà	4 - 7 transaccions	10	12	120
Complexe	més de 7 transaccions	15	4	60
Pes dels casos d'ús sense ajustar				**220**

Taula 12 Exemple d'obtenció del UUCW

Un cop obtinguts els valors corresponents al pes dels actors sense ajustar (UAW) i al pes dels casos d'ús sense ajustar (UUCW), la seva suma permet obtenir el valor anomenat *punts cas d'ús sense ajustar (Unadjusted Use Case Points, UUCP)*. L'equació per a obtenir aquest valor és la següent:

$$\boxed{UUCP = UAW + UUCW}$$

Equació 6 Punts cas d'ús sense ajustar

Continuant amb l'exemple proposat, s'obtindria el següent resultat:

$$UUCP = 18 + 220 = 238$$

7.3.2 Avaluació de l'entorn

Com el seu nom indica, el valor obtingut amb l'Equació 6 és un valor *sense ajustar*. Ajustar-lo significa tenir en compte aspectes concrets de l'organització on es realitzarà el desenvolupament i aspectes tècnics sobre el sistema a desenvolupar. En aquest sentit, els diferents aspectes a avaluar es classifiquen en dos tipus de factors: factors de complexitat tècnica i factors de l'entorn de desenvolupament.

Avaluar aquests factors permet *ajustar* els punts cas d'ús per a un determinat projecte i organització. Aquest valor, que rep el nom de *punts cas d'ús ajustats* (*Adjusted use case points, AUCP*), s'obté de la següent equació:

$$\boxed{AUCP = UUCP * TCF * EF}$$

Equació 7 Punts cas d'ús ajustats

En aquesta equació, UUCP és el valor de punts cas d'ús sense ajustar obtingut a l'apartat anterior. I els paràmetres TCF i EF es corresponen a els factors que mesuren la complexitat tecnològica (*Technical Complexity Factor, TCF*) i l'entorn (*Environmental Factor, EF*) respectivament. A continuació veurem com s'obtenen aquests dos valors.

Aquest pas acostuma a ser laboriós, motiu pel qual algunes organitzacions opten per començar utilitzant uns valors nominals per a cada factor (una mesura estàndard) enlloc de mesurar-lo, i anar-los revisant a mesura que volen ajustar el mètode a les característiques de la seva organització.

Factors de complexitat tècnica

El factors de complexitat tècnica agrupen 13 aspectes diferents que fan referència a la complexitat tècnica del sistema a desenvolupar. Cadascun dels 13 factors té un pes associat que denota la importància (el pes) d'aquest en relació al total de factors. Tot i que el mètode proporciona una relació de pesos per defecte, cada organització pot alterar-los per a ajustar el mètode al seu model de funcionament. La Taula 13 presenta els diferents factors i el pes associat per defecte a cada factor.

Número de factor	Descripció	Pes
T1	Sistema Distribuit	2
T2	Objectius sobre temps de resposta o rendiment	1
T3	Eficiència de l'usuari final	1
T4	Complexitat de procés intern	1
T5	Reusabilitat del codi	1
T6	Facilitat d'instal·lació	0,5
T7	Facilitat d'ús	0,5
T8	Portabilitat	2
T9	Facilitat de manteniment (Mantenibilitat)	1
T10	Concurrència	1
T11	Criteris especials de seguretat	1
T12	Proporcionar accés directe a tercers	1
T13	Entrenament especial als usuaris	1

Taula 13 Factors de complexitat tècnica

Mesurar el conjunt de factors de complexitat tècnica consisteix en avaluar cada un d'aquests factors i puntuar-los en una escala del 0 al 5.

Un cop associat un valor a cada un dels factors, aquest valor es multiplica per el pes corresponent al factor. La suma de tots aquests productes (valor x pes) permet obtenir el total dels factors de complexitat tècnica. La Taula 14 mostra un exemple d'avaluació dels factors de complexitat tècnica per a un projecte determinat. Com es pot veure, la puntuació associada a cada factor va en relació a les necessitats del projecte, descrites en la columna de comentaris. La columna anomenada *Factor calculat* és el producte entre el pes de cada factor i el valor associat a cada factor per el projecte concret.

Factor	Descripció	Pes	Valor assignat	Factor calculat	Comentari
T1	Sistema Distribuit	2	0	0	És un sistema central
T2	Objectius sobre temps de resposta o rendiment	1	3	3	Velocitat limitada per la de l'usuari
T3	Eficiència de l'usuari final	1	5	5	Ha de ser eficient
T4	Complexitat de procés intern	1	1	1	No té càlculs complexes
T5	Reusabilitat del codi	1	0	0	No
T6	Facilitat d'instal·lació	0,5	5	2,5	Ha de ser molt fàcil d'instal·lar
T7	Facilitat d'ús	0,5	5	2,5	Ha de ser molt amigable
T8	Portabilitat	2	0	0	No
T9	Facilitat de manteniment (Mantenibilitat)	1	4	4	Baix cost de manteniment
T10	Concurrència	1	0	0	No
T11	Criteris especials de seguretat	1	3	3	Nivell de seguretat normal
T12	Proporcionar accés directe a tercers	1	5	5	Accés directe a usuaris WEB
T13	Entrenament especial als usuaris	1	1	1	Pocs usuaris. Sistema amigable.
Total de factors tècnics				**27**	

Taula 14 Exemple d'avaluació de factors de complexitat tècnica

L'objectiu de l'avaluació dels factors de complexitat tècnica és obtenir un valor que rep el nom de *Factor de Complexitat Tècnica* (*Technical Complexity Factor, TCF*), el qual és utilitzat en l'equació d'obtenció dels punts de cas d'ús ajustats (Equació 7). El factor de complexitat tècnica es calcula en funció del total de factors tècnics obtingut (veure Taula 14), i la seva equació és la següent:

$$\boxed{TCF = 0.6 + (0.01 * < Total\ \ factors\ complexitat\ tècnica >)}$$

Equació 8 Obtenció del factor de complexitat tècnica TCF

Seguint l'exemple utilitzat fins ara, aquest factor prendria el següent valor:

$$TCF = 0.6 + (0.01 * 27) = 0.6 + 0.27 = 0.87$$

Factors d'entorn

El factors d'entorn contemplen 8 aspectes diferents que fan referència a l'organització que realitzarà el desenvolupament del sistema. Cadascun d'aquests 8 factors té un pes associat que denota la importància d'aquest en relació al total de factors. Igual que amb els factors de complexitat tècnica, el mètode proporciona una relació de pesos per defecte, però cada organització pot alterar-los si ho creu convenient per a ajustar més el mètode al seu model de funcionament. La Taula 15 presenta els diferents factors i el pes per defecte associat a cada factor.

Número de factor	*Descripció*	*Pes*
E1	Familiaritat amb el model de projecte utilitzat	1,5
E2	Experiència amb l'aplicació	0,5
E3	Experiència en Orientació a Objectes	1
E4	Capacitació del cap de projecte	0,5
E5	Motivació	1
E6	Estabilitat dels requeriments	2
E7	Equip a temps parcial	-1
E8	Llenguatge de programació difícil	-1

Taula 15 Factors d'Entorn

Mesurar el conjunt de factors d'entorn consisteix en avaluar cada un d'aquests factors i puntuar-los en una escala del 0 al 5 segons la seva importància dins la organització.

La Taula 16 mostra un exemple d'avaluació dels factors de complexitat tècnica per a una organització determinada. Com es pot veure, la puntuació associada a cada factor va en relació a les característiques de l'organització, detallades en la columna de comentaris. La columna anomenada *Factor calculat* és el producte entre el pes de cada factor i el valor associat a cada factor. La suma de tots aquests productes (valor x pes) permet obtenir el total dels factors d'entorn.

L'objectiu de l'avaluació dels factors d'entorn és obtenir un valor que rep el nom de *Factor d'Entorn* (*Environment Factor, EF*), el qual és utilitzat en l'equació d'obtenció dels punts de cas d'ús ajustats (Equació 7). El factor d'entorn es calcula en funció del total de factors d'entorn (veure Taula 16), i la seva equació és la següent:

$$EF = 1.4 + (- 0.03 * < Total\ Factors\ Entorn >)$$

Equació 9 Obtenció del factor d'entorn EF

Seguint l'exemple utilitzat fins ara, aquest factor prendria el següent valor:

EF = 1.4 + (-0.03 * 12) = 1.04 - 0.285 = 0.755

Factor	*Descripció*	*Pes*	*Valor assignat*	*Factor calculat*	*Comentari*
E1	Familiaritat amb el model de projecte utilitzat	1,5	1	1,5	La major part de l'equip no està familiaritzada
E2	Experiència amb l'aplicació	0,5	4	2	La major part de l'equip i ha treballat durant anys
E3	Experiència en Orientació a Objectes	1	1	1	La majoria són programadors de COBOL
E4	Capacitació del cap de projecte	0,5	5	2,5	És expert i altament qualitficat
E5	Motivació	1	5	5	Molt alta
E6	Estabilitat dels requeriments	2	2	4	S'esperen alguns canvis
E7	Equip a temps parcial	-1	3	-3	Malauradament part de l'equip és a temps parcial
E8	Llenguatge de programació difícil	-1	1	-1	Visual Basic
Total Factors d'Entorn				**12**	

Taula 16 Exemple d'avaluació dels factors d'entorn

7.3.3 Obtenció dels punts cas d'ús ajustats

Un cop determinats els valors per als factors d'entorn i de complexitat tècnica, s'aplicarà l'equació que permet obtenir els punts cas d'ús ajustats (Equació 7). Aquesta equació té com a paràmetres els dos factors acabats de determinar i el número de punts cas d'ús sense ajustar (UUCP) obtingut a la primera etapa del mètode. L'equació, que tornem a reproduir aquí, té el següent aspecte:

AUCP = UUCP * TCF * EF

Continuant amb l'exemple seguit fins ara, l'aplicació d'aquesta equació presenta el següent resultat:

$$AUCP = 238 * 0.87 * 0.755 = 156.3303$$

Aquest valor obtingut és abstracció de la mida del producte a desenvolupar.

7.3.4 Determinar l'esforç

Finalment, un cop obtingut el valor ajustat de punts cas d'ús del sistema a desenvolupar, només resta determinar l'esforç que aquest representarà. L'esforç es mesura en unitats de hores-persona. S'obté multiplicant el valor ajustat de punts cas d'ús (AUCP) per un *factor de conversió*. Aquest factor denota l'esforç necessari en hores-persona per a desenvolupar un punt cas d'ús ajustat (AUCP) segons una determinada combinació d'entorn i complexitat tecnològica. La següent és l'equació d'esforç del mètode dels punts cas d'ús:

$$\boxed{Esforç = AUCP * factor_de_conversio}$$

Equació 10 Equació d'esforç

Originalment, Karner va suggerir que una unitat de AUCP requeria sobre unes 20 hores-persona per a ser desenvolupada. Alguns autors indiquen que els seus resultats empírics demostren que aquest factor pot oscil·lar entre 15 i 30 hores-persona per a unitat de punt cas d'ús ajustat, depenent del rendiment de l'equip de treball i la dificultat tecnològica que requereix la implementació. Aquests aspectes, però, ja han estat avaluats en considerar els *factors d'entorn*, per aquest motiu alguns autors suggereixen que és possible basar-se en els factors d'entorn per a obtenir un valor més realista i ajustat del factor de conversió. I proposen un mètode.

Aquest mètode consisteix en agafar els factors d'entorn de E1 a E6 que estiguin per sota de 3 i comptar-los, i a continuació sumar-los al número de factors entre E7 i E8 que estiguin per sobre de 3.

- Si el resultat de l'operació és 2 o menys, es recomana utilitzar 20 hores-persona com a factor de conversió
- Si el total és 3 o 4, es recomana utilitzar 28 hores-persona
- I si el número obtingut és superior a 5 es recomana reconsiderar el projecte fins que s'obtingui un dels resultats anteriors, ja que en aquesta situació es considera que el risc que comporta el projecte és massa alt. Una altra opció, si no es vol reconsiderar el projecte, és utilitzar un factor de conversió de 36 hores-persona per punt cas d'ús.

És interessant comprovar que el factor proposat per Karner per a estimar l'esforç (20 hores-persona) és consistent amb els valors empírics obtinguts per el mètode RUP (*Rational Unified Process*), que estimen un esforç d'entre 150 i 350 hores per cada cas d'ús. Es pot comprovar en els següents dos exemples:

- Cas d'ús senzill: El cas d'ús més senzill representarà un usuari interactuant amb el sistema a través d'un sistema de comandes en línia. En aquest cas, segons les taules del mètode, el pes de l'actor serà de 2, i el pes del cas d'ús serà de 5 (el més petit). Això dona un valor de 7 punts cas d'ús sense ajustar (UUCP). Si es pren el factor de conversió proposat per Karner, s'obté un total de 7 * 20 = 140 hores-persona, molt similar a l'esforç proposat pel RUP.

- Cas d'ús complex: Un cas d'ús complex tindrà un pes de 15, i el pes pel que fa als actors serà de 3. Això dóna un total de 18 punts cas d'ús sense ajustar (UUCP). Si es pren el factor de conversió proposat per Karner, en aquesta ocasió s'obté un total de 18 * 20 = 360 hores-persona, que també s'acosta molt a les 350 hores estimades pel mètode RUP.

Seguint amb l'exemple utilitzat fins ara, si s'aplica el valor indicat per Karner en l'equació d'esforç (Equació 10) s'obté el següent resultat:

Esforç: 156.3303 * 20 = 3126.6 hores-persona

Cada organització pot mirar de determinar el seu factor de conversió a mesura que vagi assolint coneixement en l'aplicació del mètode. De totes maneres, cal tenir en compte que una variació de només 0.5 punts pot ocasionar grans diferències entre les estimacions realitzades.

7.4 Avantatges

Un dels avantatges més importants del mètode és que permet la seva aplicació en les primeres etapes del cicle de vida del programari, aconseguint en aquest punt una mesura aproximada del cost que pot representar cada funcionalitat. A més, la seva aplicació és relativament senzilla. Alguns autors fins i tot indiquen que un dels seus avantatges és que és possible anar realitzant les estimacions mentalment a mesura que es construeixen o s'estudien els requeriments, de manera que és possible començar a obtenir una idea de les persones que requerirà l'equip de

desenvolupament o del temps que caldrà invertir en el desenvolupament de l'aplicació.

Pel que fa a la qualitat de les estimacions obtingudes, aquestes seran tant acurades (o inacurades) com les obtingudes per qualsevol altre mètode aplicat en la mateixa etapa del cicle de vida del programari.

Una bona manera d'utilitzar el mètode és realitzar una primera estimació, i després treballar amb l'equip de desenvolupament per veure el rendiment que és capaç d'assolir i recalibrar les estimacions realitzades al principi per anar-les adaptant a la realitat del projecte.

També cal tenir en compte que les estimacions obtingudes són a molt alt nivell, i que serà important que a mesura que vagi avançant el projecte i es disposi de més informació detallada sobre les seves funcionalitats, es reconsiderin els casos d'ús i s'actualitzin les estimacions a aquesta nova situació.

Capítol 8

Model de Rayleigh

Durant la dècada dels 50, Peter V. Norden, del *IBM Development Laboratory*, va trobar una relació entre l'assignació de recursos humans a un projecte de recerca i desenvolupament i el temps de desenvolupament del treball a realitzar. Aquesta relació podia ser modelada a través d'una corba, que va ser anomenada corba de Rayleigh en honor a Lord Rayleigh, físic i matemàtic anglès del s. XIX.

La idea darrera de la corba de Rayleigh és que l'assignació de recursos per a un projecte de desenvolupament creix a l'inici d'aquest, arriba al seu punt àlgid, cau, i després s'allarga amb una cua durant un llarg període de temps. Un exemple d'aquesta corba el tenim a la Figura 8.

Aquesta corba mostra l'esforç destinat a un projecte des del disseny fins al lliurament del producte un cop acabat. L'eix horitzontal representa el temps. L'eix vertical representa l'esforç mesurat en mesos-persona. L'àrea per sota de la corba equival a l'esforç total que representa el projecte.

Aquest esforç es pot traduir al cost econòmic total del projecte multiplicant-lo per el cost mitjà de una unitat de mesos-persona, és a dir, el cost d'una persona treballant en el projecte durant un mes.

Larry Putnam, als anys 70, va aplicar aquest concepte al desenvolupament de programari. Posteriorment l'ha anat desenvolupant amb Ware Myers [Putnam 1992][Putnam 1996][Putnam 1997].

El Model de Rayleigh permet obtenir una perspectiva global que ajudi a realitzar una planificació realista del projecte. La corba comença amb l'etapa de Disseny. En aquest punt només són necessàries unes quantes persones. A mesura que els dissenyadors van dividint el disseny en parts més petites, són necessàries més persones per a continuar refinant cada una d'aquestes parts i creant més divisions. La codificació comença a l'esquerra de la part més alta de la corba. Poc abans de la

primera línia de punts comencen les proves de test, que finalitzen a l'alçada de la primera línia vertical de punts (IOC). A partir d'aquest moment es reduirà el nombre de persones implicades a mesura que es vagin solucionant els problemes localitzats a la fase de test. En assolir la segona línia vertical de punts, el producte ha assolit un nivell acceptable d'errors i és lliurat al client. A partir d'aquí tenen lloc les tasques de suport del producte i manteniment (22% restant).

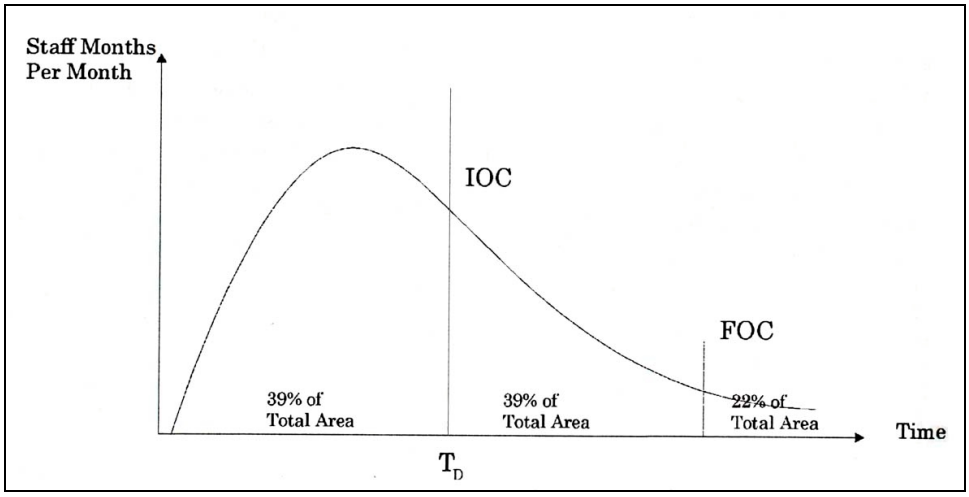

Figura 8 Corba de Rayleigh

8.1 Conseqüències de la corba de Rayleigh

L'aplicació del concepte al desenvolupament de programari va resultar en l'obtenció d'una *equació del programari*. Aquesta permet observar que la relació entre la quantitat de treball a realitzar (la mida del programari), l'esforç que representa (en mesos-persona) i el temps que caldrà dedicar-hi (en anys) ve moderat per un quart element anomenat *productivitat del procés*. Aquest és una mesura de la productivitat de tota la organització, un nivell que indica la professionalitat del procés amb la que la organització va desenvolupar el projecte.

La següent equació presenta el paràmetre de productivitat de l'organització que desenvolupa el projecte:

$$Productivitat = \frac{MidaProducte}{(Esforç / B)^{1/3} * TempsDesenvolupament^{4/3}}$$

Equació 11 Equació del programari

En aquesta equació tenim que:

- El paràmetre *MidaProducte* representa el número de Línies de Codi[9] del projecte lliurat (cal tenir en compte que aquests paràmetres fan referència a informació de projectes anteriors).

- El paràmetre *Esforç* fa referència al número de anys-persona que va requerir el projecte.

- El paràmetre *B* és un *factor especial de dificultat*. Aquest varia entre 0.16 (per productes entre 5.000 i 15.000 línies de codi) i 0.39 (per productes amb més de 75.000 línies de codi) [Putnam 1992] [Putnam 1997]. Aquesta constant serveix per a compensar la diferència de mida entre projectes. Projectes més grans involucren més persones, requereixen més comunicació i hi ha més possibilitat de malinterpretacions, que poden comportar una taxa d'errors més elevada. I aquesta relació no és lineal pel que fa a la mida del programari, d'aquí la necessitat de contemplar-ho amb aquest factor.

- El paràmetre *TempsDesenvolupament* s'expressa en anys.

Convencionalment, la productivitat s'expressa en funció del treball a realitzar i l'esforç necessitat per a realitzar-lo. És a dir:

$$Productivitat = \frac{QuantitatDeTreball}{Esforç}$$

Equació 12 Productivitat convencional

Aquesta relació és lineal (es pot representa amb una equació lineal). És important notar que l'equació de productivitat del model de Rayleigh (Equació 11) no és lineal. Per una banda, en l'equació del model de Rayleigh l'esforç perd considerablement la seva importància, ja que està elevat a 1/3, el que equival a l'arrel cúbica d'aquest. Per altra banda, s'hi afegeix el temps de desenvolupament, el qual és elevat a 1.33, que significa un pes bastant important en l'equació. D'aquesta

[9] Aquest mètode utilitza la mesura Línies de Codi. Hi ha mètodes que permeten passar Punts Funció a Línies de Codi. Veure l'apartat dedicat a la tècnica anomenada recompte indirecte (*Backfiring*).

equació se'n pot deduir que "augmentar els recursos humans en un projecte de programari que portar retard fa augmentar el seu retard" (Fred Brooks, 1975).

Cal tenir en compte que aquesta equació està mesurant la productivitat de tota una organització i no només la d'un sol individu. Mesurar la productivitat d'un sol individu només serveix per a conèixer la productivitat d'aquest. La suma de la productivitat individual de *n* individus no té res a veure amb la tindran treballant tots junts!

8.2 Utilitat del mètode

L'equació del programari (Equació 11) permet obtenir el nivell de productivitat d'una organització. A partir del nivell de productivitat d'una organització és possible predir molta informació de propers desenvolupaments. Per a obtenir el nivell de productivitat caldrà avaluar l'historial de la organització. Caldrà analitzar projectes anteriors: obtenir la mida del programari desenvolupat, calcular el factor especial de dificultat *B*, el temps que es va trigar en desenvolupar-lo i els recursos (en mesos-persona) que s'hi varen dedicar. Aquesta informació aplicada a l'equació de programari ens permetrà obtenir el nivell de productivitat de la organització. Si es vol millorar l'avaluació es pot considerar l'estudi de més d'un projecte dels realitzats per la mateixa organització.

A partir del nivell de productivitat d'una organització serà possible estimar informació sobre nous projectes a realitzar per l'organització. Per a un projecte estimat en una determinada mida, serà possible predir el temps de desenvolupament d'un projecte d'una determinada mida, i els recursos que caldrà dedicar-hi. Aquests recursos permetran obtenir el cost total del desenvolupament del producte pel que fa a recursos humans.

Les equacions següents ens permetran obtenir aquesta informació:

$$TDmin = 0.68 * (MidaProducte / Productivitat)^{0.43} \quad en\ anys$$

$$TDmin = 8.14 * (MidaProducte / Productivitat)^{0.43} \quad en\ mesos$$

Equació 13 Temps de Desenvolupament mínim

$$Esforç\ en\ anys\ -\ persona = 15 * B * TDmin^3 \quad amb\ T\ en\ anys$$

$$Esforç\ en\ anys\ -\ persona = 180 * B * TDmin^3 \quad amb\ T\ en\ anys$$

Equació 14 Equacions d'Esforç

$$Cost = Mesos\text{-}persona * Cost_Mes_Persona$$

Equació 15 Equació del cost

Les primeres dues equacions ens permeten obtenir una previsió del temps mínim de desenvolupament (en anys o en mesos, segons preferim) que requerirà el nostre nou projecte (del que hem estimat prèviament la mida).

Les equacions d'esforç (Equació 14) permeten estimar l'esforç requerit a partir del temps mínim de desenvolupament (expressat en anys). *B* és el factor especial de dificultat explicat per a l'Equació 11.

Finalment, l'Equació 15 permet obtenir una estimació del cost econòmic en recursos humans del nou projecte. Aquesta equació requereix conèixer el cost mitjà d'un mes-persona, és a dir, el sou mitjà que cobra una persona dedicada al projecte durant un mes.

Capítol 9

Una tècnica senzilla

En aquesta secció es presenta una tècnica extremadament senzilla que pot ser utilitzada quan no es disposa d'informació sobre el rendiment obtingut en projectes anteriors, quan no es disposa de nivells de productivitat anteriors.

Aquesta tècnica es basa en obtenir una estimació de qualsevol tipus de quantitat (mida, temps, dificultat, etc.) a partir de tres previsions: una optimista, una pessimista i una tercera corresponent a la més probable [Putnam 1992]. Aquestes previsions les anomena: previsió baixa, previsió més probable i previsió alta.

El mètode utilitza raonaments estadístics per a obtenir una previsió aproximada que garanteixi cert èxit en l'encert.

Les equacions d'aquest mètode són les següents:

$$Estimació=\frac{Baixa+4*MesProbable+Alta}{6}$$

$$EstimacióTotal=\sum Estimacions$$

Equació 16 Equacions d'estimació

Aquesta primera equació proporciona una estimació obtinguda a partir de les tres previsions realitzades.

$$DesviacióEstàndar=\frac{Alta-Baixa}{6}$$

$$DesviacióEstàndardTotal=\sqrt{\sum DesviacióEstàndard}$$

Equació 17 Equacions de desviació estàndard

La desviació estàndard de l'estimació ve donat de la previsió alta i baixa.

$$\%Error=\frac{DesviacióEstàndard}{Estimació}$$

$$\%ErrorTotal=\frac{DesviacióEstàndardTotal}{EstimacióTotal}$$

Equació 18 Equacions de %Error

El percentatge d'error indica la confiança en l'estimació realitzada a partir de l'estimació i la desviació estàndard que han donat les dues primeres equacions.

Aquesta darrera equació és una de les més important perquè permet determinar la validesa de les estimacions realitzades (optimista, pessimista i més probable). Si el percentatge d'error obtingut és superior a 20, cal considerar que l'estimació no és suficientment vàlida i que cal treballar una mica més en model per a poder realitzar unes estimacions més ajustades.

Una diferència gran entre la previsió optimista i la previsió pessimista significa que hi ha massa incògnites sobre quina pot ser l'evolució real del projecte. En aquesta situació, cal reduir el risc dividint les tasques en tasques més petites i manejables i realitzant noves previsions sobre aquestes parts. Amb aquest refinament s'aconseguiran previsions més realistes que permetran reduir la desviació i el percentatge d'error.

Caldrà repetir aquest procés fins que l'error obtingut sigui proper al 10 %.

Per a obtenir una estimació de un projecte sencer a partir de les estimacions de les seves parts utilitzarem les equacions anomenades *totals*.

Per exemple, suposem que es vol estimar la durada d'un projecte que creiem que pot durar 6 mesos. Creiem que si tot va molt bé en pot durar 5 i que si va molt

malament podria arribar a durar fins a 9 mesos. En aquesta situació, tenim que l'aplicació de la primera equació (Equació 16) donaria la següent estimació

$$Estimació = \frac{5 + 4 * 6 + 9}{6} = \frac{38}{6} = 6.33$$

La desviació estàndard per a aquestes previsions, segons l'Equació 17, és

$$DesviacióEstàndar = \frac{9 - 5}{6} = 0.66$$

I aplicant l'Equació 18 obtindríem el percentatge d'error:

$$\%Error = \frac{0.66}{6.33} = 10\%$$

És a dir, les previsions realitzades inicialment ens donen només un 10% d'error. Per tant, si ens sembla bé aquest percentatge, no caldrà que estudiem més el projecte. Si per contra, ens sembla un percentatge massa elevat d'error, estudiarem més a forns les funcionalitats del projecte per a veure si és possible ajustar més les tres prediccions inicials.

Aquest mètode és senzill, ràpid i efectiu quan no es disposa de dades sobre productivitats passades. Cal tenir en compte, però, que no és recomanable utilitzar-lo si es disposen d'aquestes dades històriques, ja que l'estimació no podrà ser mai tant precisa com ho seria l'aplicació d'un mètode més detallat.

Capítol 10

Avaluació d'estimacions

El procés d'obtenció d'estimacions conclou amb la seva avaluació. En aquest capítol es tracta l'avaluació des de tres perspectives: l'avaluació de les estimacions obtingudes, l'avaluació de la organització que les realitza i l'avaluació del mètode utilitzat per a obtenir-les.

10.1 Avaluació del resultat d'una estimació

És possible obtenir estimacions a partir de diversos mètodes. Alguns d'aquests són més senzills i altres més refinats. Cada situació i experiència determinarà quin és el mètode (o conjunt de mètodes) més apropiat per a realitzar estimacions sobre un projecte.

Park [Park 1996] va publicar una llista d'aspectes a tenir en compte a la hora d'avaluar unes estimacions. La llista està formada per un conjunt de preguntes que tot cap de projecte hauria de poder respondre afirmativament abans d'acceptar-les com a vàlides. La llista és la següent:

Taula 17 Guia per a l'avaluació d'estimacions

1. Els objectius de les estimacions són clars i correctes?
2. La tasca ha estat mesurada adequadament?
3. Les estimacions de costos i calendari són consistents amb els rendiments demostrats en altres projectes?
4. Han estat identificats i explicats els factors que afecten l'estimació?
5. S'han seguit els passos necessaris per a assegurar la integritat

del procés d'estimació?
6. Existeixen evidencies històriques de que és possible confiar en les estimacions realitzades per a la organització?
7. Ha canviat la situació des de la realització de les estimacions?

10.2 Avaluació d'organitzacions

El mateix Park va publicar més tard dues llistes més on enumerava els aspectes a tenir en compte en qualsevol organització a la hora de realitzar estimacions [Park 1996-b].

La primera de les dues llistes indica els requeriments necessaris en tota organització per a poder obtenir estimacions que mereixin la nostra confiança.

Taula 18 Sis requeriments per a obtenir estimacions de confiança

1. Disposar d'una base de dades històrica de la organització.
2. Procediments estructurats per a estimar la mida del producte i la seva reusabilitat.
3. Mecanismes per a extrapolar informació sobre compliments demostrats en anteriors projectes.
4. Generació de traces (*audit trails*) per a l'enregistrament de tots els resultats obtinguts en el desenvolupament per a poder-los estudiar i comparar amb les estimacions realitzades.
5. Integritat de l'organització a la hora de treballar amb uns costos i un calendari fixats.
6. Mecanismes de recollida de dades i retorn d'informació (*feedback*) que permetin enregistrar i interpretar correctament dades sobre tot el treball realitzat.

La segona llista presenta un conjunt d'indicadors que cal considerar per a avaluar la capacitat de cada organització per a obtenir estimacions correctes i fiables.

Taula 19 Set indicadors sobre la capacitat d'estimació d'una organització

1. La direcció de la organització coneix la seva responsabilitat per a mantenir i desenvolupar la seva capacitat d'estimació.
2. Les tasques d'estimació estan recolzades per un pressupost adequat.
3. El personal que realitza les estimacions ha estat format adequadament i disposa de les eines necessàries.
4. El personal destinat a tasques d'estimació té experiència i està capacitat.
5. Existeixen plans de carrera i de reconeixement de manera que les persones qualificades desitgen treballar en les tasques d'estimació.
6. Les persones dedicades a tasques d'estimació treballen juntament amb els equips de millora de processos per a quantificar i fer un seguiment dels processos de millora de programari.
7. La capacitat d'estimació de l'organització és quantificada, seguida i avaluada.

10.3 Avaluació de mètodes d'estimació

Per a ajudar a determinar quin mètode és millor per a un determinat projecte o organització pot ser útil disposar d'unes guies sobre les que mesurar-los. A continuació es detalla una taula de criteris per a avaluar la utilitat de models d'estimació de costos.

- **Definició**: El model defineix clarament què està estimant i què no?
- **Fidelitat**: Les estimacions obtingudes són properes als resultats actuals en projectes similars?
- **Objectivitat**: El model evita col·locar la major part del pes de les estimacions en factors subjectius?
- **Constructivitat**: L'estimador entén el millor el desenvolupament a realitzar després d'utilitzar el model? Pot explicar els resultats obtinguts?
- **Detall**: Permet assolir diferents nivells de detall en l'avaluació dels requeriments?
- **Estabilitat**: Els petits canvis realitzats en els requeriments realitzen petits canvis en els resultats?
- **Abast**: El model cobreix les necessitats del desenvolupament a realitzar?
- **Facilitat d'ús**: Les opcions del model són senzilles d'entendre i

d'especificar?

- **Projecció**: El model no demana informació que no serà coneguda fins al final del desenvolupament?
- **Parsimònia**: El model evita la utilització de factors redundants o factors que pràcticament no contribueixen en els resultats?

Taula 20 Avaluació de mètodes d'estimació

Consideracions finals

Una idea interessant és que enlloc de destinar recursos per a realitzar una estimació abans de començar un projecte, potser seria millorar dedicar aquests esforços a canviar l'actitud de la organització pel que fa a estimacions prematures. Aquestes són, sense dubte, la clau de la gran quantitat de projectes amb problemes de pressupost i calendari.

La realització d'estimacions prematures és una pràctica molt arrelada en els projectes de programari i obliguen als caps de projecte a conduir els seus equips a complir tasques amb uns recursos poc realistes, la qual cosa significa aplicar directament una gran pressió sobre l'equip de desenvolupament, esgotar els recursos previstos abans de finalitzar i aconseguint satisfer només parcialment els requeriments inicials. Cal intentar realitzar estimacions al llarg de tota la vida del projecte, per a poder corregir el més aviat possibles les possibles desviacions.

Si al responsable del projecte se li exigeix una estimació concreta i tancada dels recursos, cal contemplar una reserva de recursos per a imprevistos o bé establir un mecanisme per a poder retallar funcionalitats del sistema fins que sigui viable ajustar-lo al pressupost.

Pel que fa a les possibles línies de treball, hi ha dos mètodes que pràcticament acaben de començar i podrien tenir un important desplegament: el mètode basat en casos d'ús i el mètode COSMIC-FFP. Tots dos semblen formar part de l'estat de l'art pel que a mètriques d'estimació de programari.

Cal tenir present que totes els organitzacions relacionades amb els diferents mètodes estudiats habiliten canals de comunicació per a que empreses i particulars puguin cedir les seves dades històriques per a poder ser analitzades i així contribuir a l'ajustament dels mètodes. Els pesos, les constants, i en definitiva, la seva evolució, depenen d'aquests programes d'avaluació de resultats.

Referències

- [Boehm 1981] B. Boehm, "*Software Engineering Economics*", Prentice Hall, 1981.
- [Boehm 1987] B. Boehm i W. Royce, "*TRW IOC Ada COCOMO: Definition and Refinements*", Software Engineering Institute, 1987.
- [Boehm 1988] B. Boehm, "*A Spiral Model for Software Development and Enhancement*", Computer, Maig 1988.
- [Boehm 1996] B. Boehm, "*Anchoring de Software Process*", IEEE Software, Juliol 1996.
- [Boehm 2000] B. Boehm, "*Software Cost Estimating with COCOMO II*", Prentice Hall, 2000.
- [Humphrey 1995] W. Humphrey, "*A Discipline for Software Engineering*", Addison-Wesley, Reading, Massachusetts, 1995.
- [Jones 95] Jones, Capers. "*Backfiring: Converting Lines of Code to Function Points*" IEEE Computer 28, November 1995.
- [Karner 1993] G. Karner, "*Use Case Points - Resource Estimation for Objectory Projects*", Objective Systems SF AB (Rational Software), 1993
- [Park 1996] R. Park, "*A Manager's Checklist for Validating Software Costs and Schedule Estimates*", American Programmer, June 1996.
- [Park 1996-b] R. Park, "*Assessing an Organization's Estimating Capabilities*", American Programmer, July 1996.
- [Putnam 1992] L. Putnam and W. Myers, "*Measures for Excellence: Reliable Software On Time, Within Budget*", Yourdon Press, Englewood Cliffs, N.J. 1992.
- [Putnam 1996] L. Putnam and W. Myers, "*Controlling Software Development, Executive Briefing*", IEEE CS Press, Los Alamitos, California. 1996.
- [Putnam 1997] L. Putnam and W. Myers, "*Industrial Strength Software: Effective Management Using Measurement*", IEEE CS Press, Los Alamitos, California. 1997.

Altres fonts d'informació

- D. Phillips, *The Software Project Manager's Handbook*, IEEE Computer Society Press, Los Alamitos, California, 1998

- W. Humphrey, *Introduction to the Personal Software Process (PSP^{SM})*, SEI Software Engineering Institute, Addison Wesley, 1997

- GlobalTester.com

  Plana web mantinguda per Jeff Nyman que ofereix molta documentació sobre control de qualitat i altres temes relacionats. És un excel·lent repositori d'informació. El seu contingut és obert i distribuït sota la llicència *GNU Free Documentation License*. L'adreça de la plana web és www.globaltester.com.

- SPR - *Software Productivity Research*

  Software Productivity Research és una empresa dedicada a la consultoria en enginyeria del software. Ha desenvolupat diversos productes destinats a facilitar l'aplicació de mètodes com els punts funció o punts característica. Fou fundada per Capers Jones l'any 1984. L'any 1986 va desenvolupar el mètode dels punts característica (*Feature Points*). A través de la seva pàgina web (www.spr.com) distribueix els seus productes i ofereix els seus serveis. Aquesta també conté informació sobre diferents mètodes d'estimacions, així com informació sobre la tècnica del recompte indirecte (*backfiring*) que fou desenvolupada per el fundador de l'empresa, Caper Jones.

- AEMES - Associación Española de Métricas del Software

  AEMES és una organització sense ànim de lucre constituïda per un grup d'empreses i institucions amb l'objectiu de promoure la utilització i millora de mètodes quantitatius de gestió del procés de desenvolupament de programari. Està ubicada a la Facultat d'Informàtica de la Universitat Politècnica de Madrid. Entre altres activitats, edita periòdicament un butlletí informatiu per als seus associats. Més informació a la seva web: http://aemes.fi.upm.es

- Longstreet Consulting Inc.

  Consultaria creada per David Longstreet. Conté informació sobre punts funció i temes relacionats amb mètriques de programari. www.SoftwareMetrics.com

- Softstar Systems

  Empresa fundada el 1985 dedicada al desenvolupament de programari d'ajuda a l'estimació de programari. Desenvolupen el programa anomenat COSTAR, basat en el model COCOMO. L'adreça de la seva pàgina és www.softstarsystems.com.

- IFPUG, *International Function Point Users Group*

  La IFPUG és una organització sense ànim de lucre dedicada a l'estudi i desenvolupament del mètode IFPUG-FPA, evolució directe del mètode original basat en punts funció desenvolupat per Allen Albrecht. Aquesta organització manté el manual anomenat *Function Point Counting Practices Manual* on es detalla el funcionament del mètode i que actualment és considerat un dels referents principals sobre la tècnica dels punts funció. També organitza conferències, seminaris i emet certificacions entre altres activitats. Per a més informació, l'adreça de la seva plana web és www.ifpug.org.

- USC, *University of Southern California*

  Aquesta universitat disposa d'un departament anomenat *Center for Software Engineering* (CSE), que entre altres temes, porta a terme els treballs relatius a la millora i desenvolupament del mètode COCOMO II. Aquest departament és dirigit per Barry Boehm, creador del mètode original. A través de la seva pàgina web es pot accedir a tota la documentació sobre el mètode (en totes les seves versions) i es pot participar en la recollida de estadístiques sobre l'aplicació del mètode.
  La pàgina web de la universitat és www.usc.edu.
  La pàgina web del CSE dedicada al mètode COCOMO II és http://sunset.usc.edu/research/COCOMOII/index.html.

- comp.software.measurement

  News dedicades a l'amidament de software i temes relacionats. Molt útil per a seguir les novetats en conferències, call-for-papers, etc. i per a resoldre qualsevol dubte.

- UQAM, *Université du Quebec à Montreal*

  El *Laboratoire de Recherche en Gestion des Logiciels*, dins de l'*École de technologie supérieure* d'aquesta universitat canadenca, desenvolupa i manté el mètode COSMIC-FFP. La pàgina web d'aquest departament és www.lrgl.uqam.ca.