Introducció a les metodologies àgils

Altres maneres d'analitzar i desenvolupar

Jorge Fernández González

PID_00184450

Universitat Oberta de Catalunya

Índex

Introducció

Algú de vosaltres us preguntareu què significa això d'"àgil" i si això comporta de manera intrínseca una mica com "fer malament les coses" o "deixar-ho a mitges". És cert que durant l'assignatura gairebé tot el que hem intentat ensenyar és precisament a ser metòdics, rigorosos i estrictes científicament parlant, encara que no és menys cert que us hem intentat inculcar també una mica d'esperit crític. Doncs bé, és precisament a aquest esperit crític al qual hem d'apel·lar en aquest mòdul per intentar comprendre que ser àgils no significa renunciar a formalismes ni deixar de ser estrictes i rigorosos.

Molts professionals que ens dediquem als sistemes d'informació hem dit en alguna ocasió una cosa semblant a: "això és en teoria i queda molt bonic però a la pràctica no pots perdre tant de temps fent l'anàlisi perquè quan acabes el sistema ja ha canviat i el client s'ha avorrit". Jo mateix sóc conscient que és fàcil caure en aquesta dinàmica quan la pressió ens cau al damunt i els terminis de lliurament s'apropen.

L'alta competitivitat actual fa que els sistemes d'informació s'hagin de desenvolupar ràpidament per a adaptar-se a l'organització. Les presses fan que el primer que es rebutgi en aquesta carrera "boja" cap a un desenvolupament ràpid sigui una anàlisi exhaustiva i que se substitueixi per una anàlisi superficial o simplement que s'elimini. Aquest és, sens dubte, el **gran error**: volem tenir un sistema desenvolupat ràpidament però amb el que realment ens trobem a les mans és amb un sistema ple d'errors, immanejable i que no es pot mantenir.

És difícil canviar les regles del mercat mundial, així que el que s'ha pensat és adaptar les metodologies d'especificació i desenvolupament a aquest entorn canviant i ple de pressions, en el qual obtenir un resultat ràpid, una cosa que es pugui veure, mostrar i sobretot utilitzar ha esdevingut crucial per a l'èxit de les organitzacions. La metodologia necessàriament ha de ser àgil, ha de tenir un cicle curt de desenvolupament i ha d'incrementar les funcionalitats en cada una de les seves iteracions preservant les existents, ajudant el negoci en lloc de donar-li l'esquena.

És per a aquest entorn per al qual han nascut les metodologies àgils.

Les metodologies àgils no són la gran solució a tots els problemes del desenvolupament d'aplicacions, ni tan sols es poden aplicar en tots els casos, però sí que ens aporten un altre punt de vista de com es poden arribar a fer les coses, de manera més ràpida, més adaptable i sense haver de perdre el rigor de les metodologies clàssiques.

L'objectiu d'aquest mòdul no és cap altre que mostrar que no sempre hi ha un camí únic per a fer bé les coses, no tot és blanc o negre, sinó que hi ha una gran gamma de grisos. Aquests tons grisos també els tenim en les metodologies d'especificació i desenvolupament.

En aquesta "Introducció a les metodologies àgils" pretenem mostrar l'actual ventall existent d'aquest tipus de metodologies, per passar després a centrar-nos en una que actualment té un especial èxit entre els professionals del sector: l'Extreme Programming, més coneguda per *metodologia XP*.

A mesura que anem veient aquesta metodologia, anirem insistint en els avantatges i en els inconvenients i conclourem amb un exemple d'aplicació d'XP en un entorn basat en Java. L'elecció de Java no és arbitrària, sinó que respon al fet que la idea d'agilitat en el desenvolupament d'aplicacions està molt lligada a la de programari lliure, sens dubte a causa que els nuclis impulsors d'ambdues idees són molt propers.

Per acabar, mostrarem com la idea d'àgil es va introduint amb força en diferents àmbits, com és el cas del disseny de bases de dades i objectes, en la documentació, etc.

Objectius

L'objectiu general d'aquest mòdul és que adquiriu una primera visió de les metodologies àgils perquè així tingueu la possibilitat de reconèixer quan un projecte és propici per a usar-les. L'objectiu general es descompon en els objectius parcials següents, que l'estudiant ha d'aconseguir:

1. Adquirir els conceptes del "Manifest àgil".

2. Conèixer les principals metodologies àgils actuals.

3. Desenvolupar un esperit crític que li permeti aplicar la metodologia més adequada a cada projecte en concret.

4. Conèixer amb detall la metodologia àgil que actualment té més èxit: Extreme Programming.

5. Entreveure cap on pot anar la metodologia de l'enginyeria del programari en els pròxims anys.

1. La necessitat de ser àgils

àgil

1) *adj*. Que es mou amb facilitat i rapidesa, que té els moviments ràpids i expedits. *Una dona àgil*. *Un cavall àgil*. *Cames, peus àgils*. *Els dits àgils d'un pianista*. *Àgil com una daina*.

2) FIG. Una veu àgil. Un esperit àgil.

Diccionari de l'Institut d'Estudis Catalans

"Que es mou amb facilitat i rapidesa, que té els moviments ràpids i expedits". *A priori* no sembla un adjectiu gaire adequat per a una metodologia. Per a buscar l'origen d'aquest adjectiu, ens hem de remuntar al subjecte inicial que el posseïa, que no és cap altre que l'organització o empresa. És aquesta la que ha de ser àgil, la que ha de moure tots els seus "membres" amb facilitat i desimboltura, en una orquestració que li permeti sobreviure en el món altament competitiu en el qual ens trobem. Per a això necessita un "cervell", un sistema d'informació que ha de créixer segons la demanda de necessitats que se li exigeixin, i que ha de créixer ràpidament i coherentment, de manera que l'organització es pugui adaptar ràpidament als canvis produïts al seu entorn.

La competència cada dia és més ferotge i els ritmes de canvis són més ràpids. Les organitzacions hauran de respondre immediatament a aquests canvis si volen sobreviure, hauran d'evolucionar en un termini curt de temps. Els sistemes d'informació (SI) no es poden quedar enrere, les organitzacions en depenen per funcionar i no han de ser un llast, sinó un avantatge competitiu. El paper dels SI i la seva velocitat d'adaptació als canvis marcarà la diferència entre una empresa pròspera i una en declivi. Els SI hauran d'ajudar que aquesta evolució contínua no només sigui una realitat, sinó que ho sigui amb el mínim cost de recursos possible. Han de permetre que les organitzacions evolucionin de manera eficaç, eficient i, és clar, **àgil**.

Així doncs, de manera transitiva, si l'organització ha de ser àgil i el SI que la controla ha de ser àgil, com a mínim la metodologia per a desenvolupar parts del SI també hauria de ser àgil.

Però quan realment va néixer l'adjectiu *àgil* aplicat al desenvolupament de programari va ser el febrer de 2001, a l'Snowbird Ski Resort de Utah (EUA). Es van reunir per descansar, esquiar i, ja que hi eren, conversar, un grup de disset experts de la indústria del programari per, precisament, veure què es podia fer per a adequar les metodologies a les necessitats de la indústria. Hi van assistir representants de diferents metodologies com Extreme Programming,

SCRUM, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development que ja oferien una alternativa als desenvolupaments tradicionals caracteritzats per la rigidesa i la documentació exhaustiva en cada una de les etapes i que s'havien mostrat inadequats per als projectes de petita envergadura (i de vegades fins i tot per als mitjans).

Després d'aquesta reunió es van crear dues coses que han donat molt a parlar en els últims anys, la primera de les quals és el "Manifest àgil per al desenvolupament de programari", que recull la filosofia de les metodologies àgils.

Webs complementàries

Podeu veure el contingut del "Manifest àgil per al desenvolupament de programari" i The Agile Alliance en les webs següents: http://www.agilemanifesto.org/ (http://www.agilealliance.org/)

La segona és una organització sense ànim de lucre, The Agile Alliance, dedicada a promoure els conceptes relacionats amb el desenvolupament àgil de programari i ajudar les organitzacions i empreses a adoptar l'agilitat.

1.1. El "Manifest àgil"

En el "Manifest àgil" es defineixen els **quatre valors** pels quals s'haurien de guiar les metodologies àgils.

> "Busquem millors maneres de desenvolupar programari i ajudar d'altres a desenvolupar-lo. En aquest treball valorem:
>
> - L'individu i les seves interaccions més que el procés i les eines.
> - El desenvolupament de programari que funciona més que l'obtenció d'una bona documentació.
> - La col·laboració amb el client més que la negociació d'un contracte.
> - El fet de respondre als canvis més que seguir una planificació.
> - D'aquesta manera, mentre major valor tinguem en la part dreta més apreciarem els de la part esquerra."
>
> "Manifest àgil"
>
> Firmat per Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland i Dave Thomas.

Vegem què significa cada un d'aquests valors:

1) L'individu i les seves interaccions més que el procés i les eines

Sens dubte, l'eina fonamental de l'enginyeria del programari i del desenvolupament d'aplicacions és el cervell humà. Unes jornades maratonianes de catorze hores de treball van en detriment de la qualitat del producte acabat. Però una persona sola no realitza un projecte, necessita un entorn en el qual desenvolupar el seu treball i un equip amb el qual col·laborar. Aquestes interaccions s'han de cuidar també. Un factor clau per a l'èxit és construir un bon equip, que els seus membres tinguin una bona comunicació, i que a més sàpiguen com construir el seu propi entorn de desenvolupament. Moltes vegades es co-

met l'error de construir primer l'entorn i esperar que l'equip s'hi adapti. Això ens resta eficiència, és millor que l'equip el configuri a partir de les seves necessitats i les seves característiques personals.

A més, les interaccions que faci l'equip amb l'usuari final haurien de ser igual de fluides i aquest usuari hauria de ser un membre més de l'equip, amb un objectiu comú, que és aconseguir que el projecte funcioni i li sigui útil.

Si tot això funciona bé, és obvi que l'elecció de les eines i el procés mateix de desenvolupament passen a estar en un pla totalment secundari en l'equació de l'èxit del projecte.

2) El desenvolupament de programari que funciona més que l'obtenció d'una bona documentació

Un dels objectius d'una bona documentació és poder anar-la a consultar quan calgui modificar alguna cosa del sistema. Sens dubte, és una arma de doble tall: una bona documentació actualitzada en cada modificació i ben mantinguda al dia et permet saber l'estat de l'aplicació i com realitzar les modificacions pertinents, però són pocs els que amb les pressions externes de temps i diners acaben actualitzant la documentació. Generalment quan s'ha d'arreglar un programa perquè alguna cosa falla, et concentres que funcioni, ja que és molt possible que tinguis usuaris aturats (fins i tot enfadats) i que l'empresa o l'organització perdi diners; en aquests casos, ni t'atures a mirar detingudament la documentació ni, quan acabes d'arreglar el programa, et poses a actualitzar-la.

En la filosofia àgil el primordial és evitar aquests errors, centrar el nostre temps a assegurar que el programari funciona i que s'ha testat exhaustivament, i intentar reduir la creació de documents "inútils", d'aquests que acaben no mantenint-se ni tan sols consultant-se. La regla no escrita és **no produir documents superflus**, i només produir aquells que siguin necessaris de immediatament per a prendre una decisió important durant el procés de desenvolupament. Aquests documents "han d'anar al gra", ser curts i centrar-se en la idea fonamental, oblidant-nos dels circumloquis que no aporten res a la comprensió del problema.

3) La col·laboració amb el client més que la negociació d'un contracte

La consultoria informàtica dels últims anys s'ha convertit en una lluita a mort entre el proveïdor del servei i el client que el contracta. Per una banda, el client intenta que es facin el major nombre de funcionalitats amb els mateixos diners i, per una altra, el consultor intenta que per aquests diners solament es realitzin les funcionalitats contractades inicialment. En les reunions de seguiment dels projectes és fàcil sentir frases del tipus "aquesta modificació no entra en el contracte", resposta generalment per la tan típica "doncs jo ja no tinc més

pressupost i així no podem treballar". Al final aquest tipus de projectes fa que el consultor informàtic sigui un híbrid entre analista i advocat, que desenvolupa habilitats legals per salvaguardar-se en cas de conflicte jurídic.

Tanmateix, perquè un projecte tingui èxit és fonamental la complicitat i el contacte continu entre el client i l'equip de desenvolupament. El client ha de ser i sentir-se part de l'equip. D'aquesta manera tots dos entendran les dificultats de l'altre i treballaran conjuntament per solucionar-lo.

4) El fet de respondre als canvis més que seguir una planificació

Una organització canvia constantment, s'adapta a les necessitats del mercat i reorganitza els seus fluxos de treball per ser més eficient. És difícil, doncs, que en el desenvolupament d'un projecte, aquest no pateixi cap canvi, ja que és segur que les necessitats d'informació de l'empresa hauran canviat. I no solament això, sinó que les possibilitats econòmiques de l'empresa poden influir sobre el nostre projecte, sia engrandint-lo o bé sia reduint-lo. Són molts els factors que alteraran la nostra planificació inicial del projecte. Si no l'adaptem a aquests canvis, correm el risc que, quan acabem, la nostra aplicació no serveixi per a res i el client s'hagi gastat els diners en va. L'habilitat de respondre als canvis de requisits, de tecnologia, pressupostaris o d'estratègia marca sens dubte el camí de l'èxit del projecte.

Com a conseqüència d'aquests **quatre valors**, el "Manifest àgil" també enuncia els **dotze principis** que caracteritzen un procés àgil diferenciant-lo d'un altre de tradicional en els quals aquest enfocament no s'havia aplicat prou; sempre s'havia deixat implícit però sense insistir-hi.

1) La prioritat és satisfer el client mitjançant lliuraments de programari tan aviat com sigui possible i continus que li aportin un valor.

2) Donar la benvinguda als canvis, fins i tot al final del desenvolupament, ja que els canvis donaran un avantatge competitiu al nostre client.

3) Fer lliuraments freqüents de programari que funcioni, des d'un parell de setmanes a un parell de mesos, amb el menor interval de temps possible entre lliuraments.

4) Les persones del negoci i els desenvolupadors han de treballar junts cada dia al llarg de tot el projecte.

5) Construir el projecte entorn d'individus motivats. Donar-los l'entorn i el suport que necessiten i confiar-hi.

6) El diàleg cara a cara és el mètode més eficient i efectiu per a comunicar informació dins d'un equip de desenvolupament.

7) El programari que funciona és la principal mesura del progrés.

8) Els processos àgils promouen un desenvolupament sostingut. Els promotors, usuaris i desenvolupadors han de poder mantenir un ritme de treball constant de manera indefinida.

9) L'atenció contínua a la qualitat tècnica i al bon disseny milloren l'agilitat.

10) La simplicitat és essencial. S'ha de saber maximitzar el treball que no s'ha de realitzar.

11) Les millors arquitectures, requisits i dissenys sorgeixen dels equips que s'han organitzat ells mateixos.

12) En intervals regulars, l'equip ha de reflexionar respecte a com arribar a ser més efectiu, i ajustar el seu comportament per aconseguir-ho.

Arribats a aquest punt, podem intuir que les maneres de fer les coses en l'enginyeria del programari van canviant, adaptant-se més a les persones i a les organitzacions en les quals han de funcionar les aplicacions.

Som a l'avantsala d'una revolució? Possiblement.

1.2. Metodologies àgils més importants

Són moltes les metodologies que tenen el qualificatiu d'àgils, algunes de les quals exploren diferents principis per aconseguir l'objectiu de satisfer plenament les necessitats del sistema d'informació que s'intenta implementar. Vegem-ne algunes de les més importants.

1.2.1. SCRUM

SCRUM és una metodologia que **neix aliena al desenvolupament del programari**, de fet els seus principis fonamentals van ser desenvolupats en processos de reenginyeria per Goldratt, Takeuchi i Nonaka en la dècada dels vuitanta.

Principi de requisits indefinits de Humphrey

Per a una nova aplicació, els requeriments no seran totalment coneguts fins que l'usuari no l'hagi usat.

Podríem dir que SCRUM es basa en cert **"caos controlat"**, però estableix certs mecanismes per controlar aquesta indeterminació, manipular l'impredictible i controlar la flexibilitat.

Lema de Wegner

És impossible definir completament un sistema iteratiu.

S'estableixen tres fases:

1) Pre-joc
2) Joc
3) Post-joc

En el **pre-joc** es defineixen i/o revisen les funcionalitats que ha de tenir el sistema, en el **joc** es distribueixen les tasques per a cada membre de l'equip, es treballa de valent i s'intenta aconseguir l'objectiu. Tots els membres de l'equip han de participar en una reunió diària que en cap cas no haurà de superar els trenta minuts. En la fase de **post-joc** s'avalua el lliurament de funcionalitats, es veuen les tasques pendents, s'avalua el progrés del projecte i es redefineix el seu temps de lliurament si fos necessari.

El principi d'incertesa de Ziv

En l'enginyeria del programari la incertesa és inherent i inevitable en el procés de desenvolupament d'aplicacions.

1.2.2. Dynamic Systems Development Method

Neix el 1994 amb l'objectiu de crear una eina RAD (Rapid Applications Development) unificada, mitjançant la definició d'un *framework* de desenvolupament (sense propietari ni ànim de lucre) per als processos de producció de programari.

DSDM

Es va originar a Anglaterra i s'ha anat estenent per Europa (no tant pels EUA). Existeix una organització que s'encarrega del seu manteniment i desenvolupament anomenada DSDM Consortium.

DSDM s'ha desenvolupat tenint com a idees fonamentals:

- Res no es construeix a la perfecció a la primera.
- La vella **regla del 80-20** és certa (el 80% de les funcionalitats del projecte es realitzen amb el 20% del temps, i el 20% restant, els detalls, consumeixen el 80% de la resta de temps).
- És improbable que algú conegui tots els requisits del sistema des del primer dia.

DSMD proposa cinc fases, de les quals només les tres últimes són iteratives malgrat que hi ha realimentació en totes les fases.

a) Estudi de la viabilitat. El primer que s'avalua és si DSDM es pot o no aplicar al projecte.

b) Estudi del negoci. S'estudia el negoci, les seves característiques i la tecnologia.

c) Modelatge funcional. En cada iteració es planegen els processos funcionals del negoci sobre el prototip.

d) Disseny i construcció. Aquí és on es construeix la major part del sistema. El prototip es torna apte perquè els usuaris el puguin utilitzar.

e) Implementació. Passem d'un prototip a un sistema de producció. S'entrena els usuaris perquè l'usin.

La idea dominant a DSDM és contrària a la intuïció inicial. En aquesta metodologia temps i recursos es mantenen com a constants i s'ajusta la funcionalitat d'acord amb això. És a dir, la idea no és "quant em costarà desenvolupar aquest sistema?", sinó que més aviat és "amb aquest temps i aquests recursos, quantes de les funcionalitats del sistema puc fer?".

1.2.3. Crystal Methodologies

No es tracta d'una única metodologia, sinó d'un conjunt de metodologies centrades en les persones que han de desenvolupar el programari, **l'equip és la base** d'aquestes metodologies creades per Alistair Cockburn.

Web complementària

http://alistair.cockburn.us/crystal/crystal.html

Desenvolupar aplicacions ha de ser com un joc en el qual tots cooperen, aporten la seva part d'invenció i es comuniquen. Per què oblidem aquesta part?

Crystal estableix una sèrie de **polítiques de treball en equip (Methods)** orientades a fomentar la millora d'aquestes habilitats. Depenent de la mida de l'equip, s'estableix una metodologia o una altra que es designa per un color: Crystal Clear (per a 3-8 persones), Crystal Yellow (per a 10-20 persones), Crystal Orange (per a 25-50), etc.

Exemple

No és el mateix cuinar per a quatre persones que per a vint i no és el mateix planificar un cap de setmana per a dues persones que per a quaranta, llavors per què utilitzem la mateixa metodologia per a un grup de tres desenvolupadors que per a un grup de quinze?

1.2.4. Feature-Driven Development

Impulsat per Jeff de Luca i Meter Coad, Feature-Driven Development (FDD) es basa en un cicle molt curt d'iteració, mai superior a dues setmanes, i en el qual l'anàlisi i els desenvolupaments estan orientats a complir una llista de característiques (*features*) que ha de tenir el programari que s'ha de desenvolupar.

Una característica:

- Ha de ser molt simple, i poc costosa de desenvolupar, entre un i deu dies.
- Ha d'aportar valor al client i ser rellevant per al seu negoci.
- S'ha de poder expressar en termes d'<acció>, <resultat> i <objecte>.

Exemples de característiques

- Calcular el balanç de situació.
- Concedir un crèdit a un client d'un banc.
- Comprovar la validesa del PIN d'una targeta.

La metodologia segueix cinc fases iteratives: Desenvolupament/Modificació d'un Model Global; Creació/Modificació de la llista de característiques; Planificació; Disseny de la característica, i Implementació de la característica.

1.2.5. Adaptive Software Development

Aquesta metodologia parteix de la idea que les necessitats del client són sempre canviants durant el desenvolupament del projecte (i després de lliurar-lo). El seu impulsor és Jim Highsmith, la novetat d'aquesta metodologia és que en realitat **no és una metodologia de desenvolupament de programari**, sinó un mètode (com un "cavall de Troia") per mitjà del qual inculcar una cultura adaptativa a l'empresa, ja que la seva velocitat d'adaptació als canvis marcarà la diferència entre una empresa pròspera i una en declivi.

Exemple

La incertesa i el canvi continu són l'estat natural dels sistemes d'informació, però sembla que moltes organitzacions encara no en són conscients. La idea de "finalitzar" un projecte no té sentit perquè s'ha de continuar adaptant.

Els objectius d'aquesta metodologia són quatre:

1) Conscienciar l'organització que ha d'esperar canvi i incertesa i no ordre i estabilitat.

2) Desenvolupar processos iteratius de gestió del canvi.

3) Facilitar la col·laboració i la interacció de les persones des del punt de vista interpersonal, cultural i estructural.

4) Marcar una estratègia de desenvolupament ràpid d'aplicacions però amb rigor i disciplina.

1.3. Quan puc aplicar una metodologia àgil?

Ja hem vist una sèrie de metodologies àgils, hem deixat de banda moltes altres (Lean Development, Pragmatic Programming, etc.), però encara no hem vist quan hem d'utilitzar una metodologia àgil.

Com? No és sempre? Doncs no. Com gairebé tot en aquesta vida, depèn. Depèn de cada projecte en concret, cada un necessita una metodologia adequada que li garanteixi l'èxit. Necessita que s'adeqüi no solament a les funcionalitats que ha de desenvolupar, sinó a més a l'equip de desenvolupament, als recursos disponibles, al termini de lliurament, a l'entorn sociocultural, etc.

Un bon professional no s'hauria de tancar en una sola metodologia de desenvolupament, sinó que hauria d'analitzar el projecte en concret i veure quina de totes les existents seria la més adequada al seu projecte. I si cap no el satisfà plenament, hauria de ser capaç d'adaptar-les. Les metodologies hi són per a ajudar-nos, però nosaltres hem de decidir quan i com és millor aplicar-les.

Tot i així, hi ha algunes regles bàsiques, de "sentit comú", per a discernir si hem d'aplicar una metodologia tradicional o una metodologia àgil. Però recordeu que són simplement una guia, no un manament.

Primer has de respondre les set preguntes següents:

1) Com és de gran el vostre projecte? Si és molt gran i/o involucra moltes persones en l'equip, el millor és que s'utilitzin metodologies més estrictes i que es basin en la planificació i control del projecte. Si es tracta d'un projecte petit i/o amb un equip reduït de desenvolupament, de tres a vuit persones, tens una oportunitat d'experimentar amb els mètodes àgils.

2) Els vostres requisits són dinàmics? Si us trobeu davant d'un àmbit d'actuació en el qual els fluxos de negoci canvien constantment i has d'adaptar-los com podria ser una gestió de força de vendes, llavors la metodologia àgil t'ajudarà a adaptar-t'hi. Si esteu en un àmbit en el qual el dinamisme és escàs –per exemple, en la creació d'un sistema comptable–, potser hauríeu d'utilitzar una metodologia orientada al pla que tinguin en compte els canvis previsibles i que us asseguri minimitzar el *reworking*.

Reworking

Són aquelles modificacions que es realitzen en l'aplicació a causa dels errors comesos durant la codificació. No es tracta d'ampliacions sinó de tornar a fer el treball perquè no compleix la qualitat necessària.

3) Què passa si el vostre sistema falla? Si us trobeu en un sistema crític en què qualsevol error representa pèrdues humanes o grans quantitats de diners, si us plau, no utilitzeu les metodologies àgils. Si dissenyeu un sistema per controlar un bisturí làser no es poden fer gaires proves, el millor és que funcioni a la primera i, en aquest sentit, les metodologies clàssiques garanteixen la qualitat.

4) El vostre client té temps per a dedicar-lo al projecte? Si no aconseguim que el nostre client s'involucri en la creació d'un sistema que en el fons és per a ell, ens serà impossible aplicar una metodologia àgil. En aquest cas el millor és utilitzar una metodologia en cascada amb una gran anàlisi inicial.

5) Quants júniors teniu en el vostre equip de desenvolupament? Les metodologies àgils requereixen d'una gran maduresa, experiència i una dosi de talent. Han de ser equips amb gent sèniors o semisèniors. Si el vostre equip el constitueixen principalment júniors, el millor és que no ho intenteu amb les metodologies àgils.

6) Quina és la cultura empresarial de l'empresa en la qual es desenvoluparà el projecte? Les metodologies àgils requereixen cert ambient "informal", que fomenti la comunicació d'igual a igual. Si l'organització en la qual es vol desenvolupar el projecte té un alt grau de cerimònia i jerarquies estrictes, no hauríeu d'utilitzar les metodologies àgils.

7) Us ve de gust? Fa falta saber si realment es tenen ganes de provar una metodologia àgil. Aprendre coses noves representa gairebé sempre un nou sacrifici.

Si el vostre equip és petit i està format majoritàriament per gent amb talent i experiència, si el client final hi està involucrat i no imposa barreres de comunicació, si els requisits són altament canviants, si no és un projecte crític i no és massa gran, i us ve de gust provar una metodologia àgil, no en dubteu, és el moment d'experimentar aquesta manera de dissenyar i crear aplicacions.

1.4. Han mort les metodologies tradicionals? RUP is RIP?

Sens dubte, la resposta és no; segueixen molt vives i molt necessàries per a grans projectes amb grans equips de desenvolupament o per a entorns crítics com hem vist abans, però, tanmateix, ara tenen una sèrie de competidors en l'àmbit dels projectes més manejables i que requereixen una adaptació ràpida i resultats freqüents.

De les metodologies clàssiques, la més famosa és RUP (Rational Unified Process). Alguns podrien asseverar que s'oposa radicalment al que hem vist fins ara, però la realitat és que, com ja hem dit, cada projecte té la seva metodologia pròpia i en molts casos múltiples metodologies poden coexistir en el mateix projecte. Rational ha treballat estretament amb DSDM Consortium i han creat documents conjunts que demostren la compatibilitat del model **DSDM amb RUP**.

Hi ha múltiples estudis de com s'ha d'utilitzar UML **dins de la filosofia d'eXtreme Programming**; malgrat que XP desemfatitza l'ús de diagrames innecessaris, no fa falta dissenyar cada classe sinó les més representatives i/o complexes. L'**ús de patrons** no està prohibit en les metodologies àgils, però potser sí que està menys estès, ja que es prioritza un lliurament ràpid del producte i l'anàlisi de patrons pot alentir el cicle. Però sens dubte, els patrons més repetitius i estesos s'han d'utilitzar. Podem disposar d'una implementació dels patrons com a part de les eines amb què construirem el nostre sistema, malgrat que no ens posem a analitzar els seus patrons específics.

Com podeu veure, cada projecte, i fins i tot cada part d'un projecte, ha de tenir la seva pròpia metodologia independentment que sigui àgil, o que sigui orientada al pla; l'important és que ens serveixi i que sapiguem aprofitar les diferents qualitats que ens ofereixen.

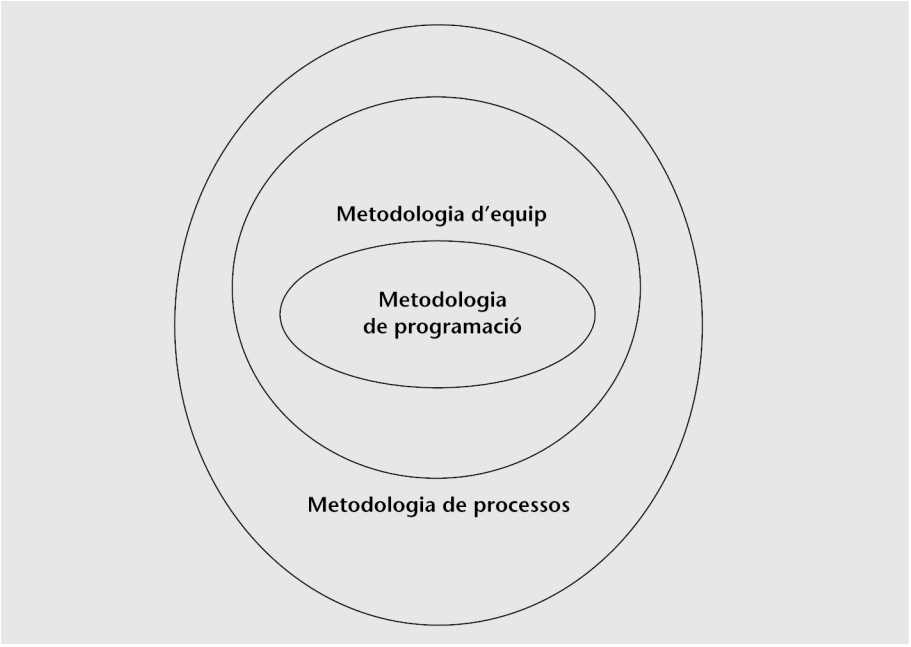
2. EXtreme Programming (XP)

La programació extrema (d'ara endavant XP) probablement marcarà un abans i un després en l'enginyeria del programari. En aquest segon punt del mòdul, intentarem mostrar tots els seus avantatges i desavantatges, i també el seu ambiciós punt de partida: el programari com a solució àgil i no com a projecte arquitectònic.

Creada per Kent Beck, Ward Cunninghamn i Ron Jeffries al final dels noranta, la programació extrema ha deixat de ser una simple idea per a un únic projecte a inundar totes les "factories de programari". Alguns la defineixen com un moviment "social" dels analistes del programari cap als homes i dones de negocis del que hauria de ser el desenvolupament de solucions en contraposició amb els legalismes dels contractes de desenvolupament.

Vivim una utopia? Potser, però actualment aquesta metodologia passa de boca en boca com si es tractés d'"una cosa secreta i pecaminosa", i molts caps de projectes busquen l'oportunitat per convèncer els seus directius i clients de posar-la en pràctica amb algun projecte pilot. Els resultats dictaran si XP passa a impregnar el desenvolupament del programari de les pròximes dècades o si simplement ens quedarem amb la idea d'aquella versió utòpica de desenvolupament de programari.

Per a assolir aquest objectiu de **programari com a solució àgil**, la metodologia XP s'estructura en tres capes que agrupen les dotze pràctiques bàsiques d'XP:



1) Metodologia de programació: disseny senzill, test, refactorització i codificació amb estàndards.

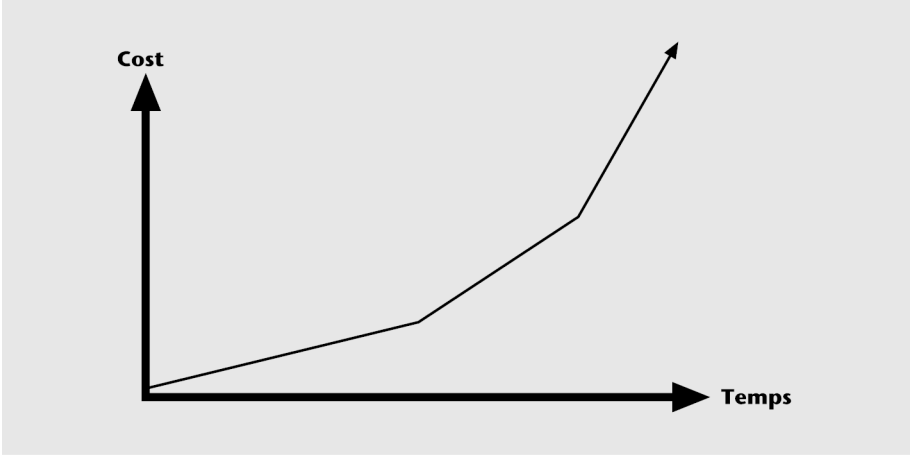
2) Metodologia d'equip: propietat col·lectiva del codi, programació en parelles, integració contínua, quaranta hores setmanals i metàfora del negoci.

3) Metodologia de processos: client *in situ*, lliuraments freqüents i planificació del joc.

Introduir el vessant de la relacions socials dins d'una metodologia és el que fa d'XP una cosa més que una guia de bones maneres. **Converteix la programació en una cosa molt més "humanitzada"**, en una cosa que permet a les persones relacionar-se i comunicar-se per trobar solucions, sense jerarquies ni enfrontaments. Els analistes i programadors treballen en equip amb el client final, tots estan compromesos amb el mateix objectiu, que l'aplicació resolgui o mitigui els problemes que té el client. El vessant social és fonamental en altres àrees del coneixement: per exemple, en les relacions de l'equip mèdic amb el pacient, o en les de bufet d'advocats amb un client, o en les de professorat i alumnes, cada un té la seva funció però l'objectiu és comú.

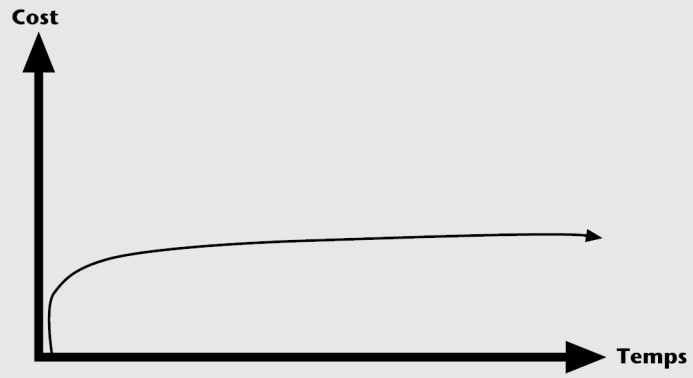
XP també humanitza els desenvolupadors. Un entorn agradable per al treball, que faciliti la comunicació i els descansos adequats, forma part d'aquesta metodologia.

Però on XP centra la major innovació és a desmuntar la idea preconcebuda del cost del canvi de les metodologies en cascada, és a dir, el que costa canviar alguna funcionalitat de la nostra aplicació a mesura que hi anem avançant. La idea generalitzada és que qualsevol modificació al final del projecte és exponencialment més costosa que al principi; si alguna cosa no estava especificada inicialment costa molt introduir-la al final del projecte.



El que XP propugna és que aquesta corba ha perdut vigència i que amb una combinació de bones pràctiques de programació i tecnologia és possible aconseguir que la corba no creixi sempre de manera exponencial.

XP pretén aconseguir una corba de cost del canvi amb creixement lleu, que en un principi és més costosa, però que al llarg del projecte **permet prendre decisions de desenvolupament al més tard possible** sense que aquesta nova decisió de canvi impliqui un alt cost en el projecte.



Només en veure aquesta gràfica és quan podem entendre el significat d'un dels lemes més repetits en la metodologia XP "Benvinguts els canvis".

XP no s'oblida de la rendibilitat necessària dels projectes, imprescindible en una economia tan competitiva com l'occidental. Per això proposa una equació d'equilibri entre el cost, el temps de desenvolupament, la qualitat del programari i l'abast de funcionalitats d'aquest.

2.1. Les variables XP: cost, temps, qualitat i abast

El punt de partida de la metodologia XP són les variables que utilitza per a cada projecte: **cost** (la inversió econòmica i en recursos), **temps** (el temps emprat, que determina la data de lliurament final), **qualitat** (del codi i de l'aplicació desenvolupada) i **abast** (conjunt de funcionalitats).

D'aquestes quatre variables només tres podran ser fixades pel client i/o pel cap de projectes; la quarta és responsabilitat de l'equip de desenvolupament i s'establirà en funció de les altres tres.

Què significa això? Doncs que s'eliminen les imposicions impossibles, s'eliminen frases del tipus:

"El client vol aquests requisits satisfets i implementats en dos mesos; l'equip és aquest i no hi n'haurà més, i el programari ha de superar tots els tests de qualitat, ja que és per a una farmacèutica."

En aquest cas el client ha establert dues variables, la de temps i els requisits, i el cap de projecte ha fixat l'equip i la qualitat. En què pot derivar això? Doncs simplement que segurament la qualitat no serà l'adequada o que no es compliran tots els requisits o que l'equip de desenvolupament haurà de fer jornades de catorze hores diàries.

L'equip de desenvolupament ha de tenir veu i vot i la possibilitat de fixar almenys una de les quatre variables. D'aquesta manera podrem establir un projecte viable.

Òbviament, amb XP no s'estableix el valor de totes les variables a la primera de canvi, és un procés gradual en el qual cada un dels responsables (client, cap de projecte i equip de desenvolupament) negocia el valor de les variables que tenen assignades fins a aconseguir una equació equilibrada i que satisfaci tothom.

Advertència: les interrelacions entre aquestes quatre variables no són sempre tan senzilles com semblen.

Exemple

Augmentar els recursos d'equips de maquinari més eficients potser inicialment disminuirà el temps de desenvolupament, però seguir augmentant les seves capacitats potser ja no aportarà cap benefici; o duplicar les persones de l'equip de desenvolupament no necessàriament ha de disminuir per dos el temps d'implementació; és més, en alguns casos fins i tot pot agreujar els problemes de coordinació i comunicació, incrementant-lo en lloc de reduir-lo.

També hi ha tasques que ja tenen totes les seves variables fixades per endavant per la seva pròpia naturalesa i sobre les quals no podem fer res per modificar l'equació. En aquest sentit, hi ha una frase molt coneguda que reflecteix aquesta situació: "nou dones no poden tenir un fill en un mes". Per molt que s'intenti, hi ha tasques que necessiten el seu temps i els seus recursos per a fer-les.

Un altre exemple paradoxal és el fet que en molts casos augmentar la qualitat acaba derivant en una disminució del temps d'implementació, sobretot en aquells projectes en els quals es preveu una gran quantitat de canvis durant el desenvolupament. No sempre baixar la qualitat del programari ens estalviarà costos.

Com podeu veure, les relacions entre aquestes quatre variables no són sempre evidents, però sí que hi ha una pregunta que ens farem sempre amb la metodologia XP: "**quina variable és la que hem de sacrificar?**".

Doncs depèn, però **la majoria de les vegades és l'abast**. La raó és simple: no sempre tenim tot el temps del món, el cost és una cosa que generalment es tendeix a minimitzar, i sense qualitat tot el procés d'XP deixa de tenir significat, no podríem mantenir les aplicacions si són un caos. Això ens deixa només una variable com la candidata més adequada a sacrificar.

Distingir entre els requisits més importants i els que aporten menys valor al negoci i donar prioritat als primers ens ajudarà a aconseguir que el projecte tingui a cada moment tanta funcionalitat com sigui possible. D'aquesta manera, quan ens apropem al termini de lliurament ens assegurarem que el principal està implementat i que només quedaran els detalls dels quals podem prescindir en cas de necessitat. L'objectiu és sempre maximitzar el valor de negoci.

2.2. Els quatre valors: comunicació, simplicitat, realimentació i coratge

Els creadors d'aquesta metodologia van voler mesurar la seva utilitat per mitjà de quatre valors, que representen aquells aspectes el compliment dels quals ens garantirà l'èxit en el projecte: comunicació, simplicitat, realimentació i coratge. Vegem què significa cada un d'ells:

a) Comunicació. Ha de ser fluïda entre tots els participants en el projecte; a més l'entorn ha d'afavorir la comunicació espontània, ubicant tots els membres en un mateix lloc. La comunicació directa ens dóna molt més valor que l'escrita, ja que podem observar els gestos del client, o l'expressió de cansament del nostre company.

b) Simplicitat. Com més senzilla sigui la solució, més fàcilment la podrem adaptar als canvis. Les complexitats augmenten el cost del canvi i disminueixen la qualitat del programari. En XP ens oblidarem de frases com "farem un sistema genèric que...", o "això ho poso per si de cas algun dia ho necessitem".

> Només s'utilitza el que en aquell moment ens doni valor, i ho farem de la manera més senzilla possible.

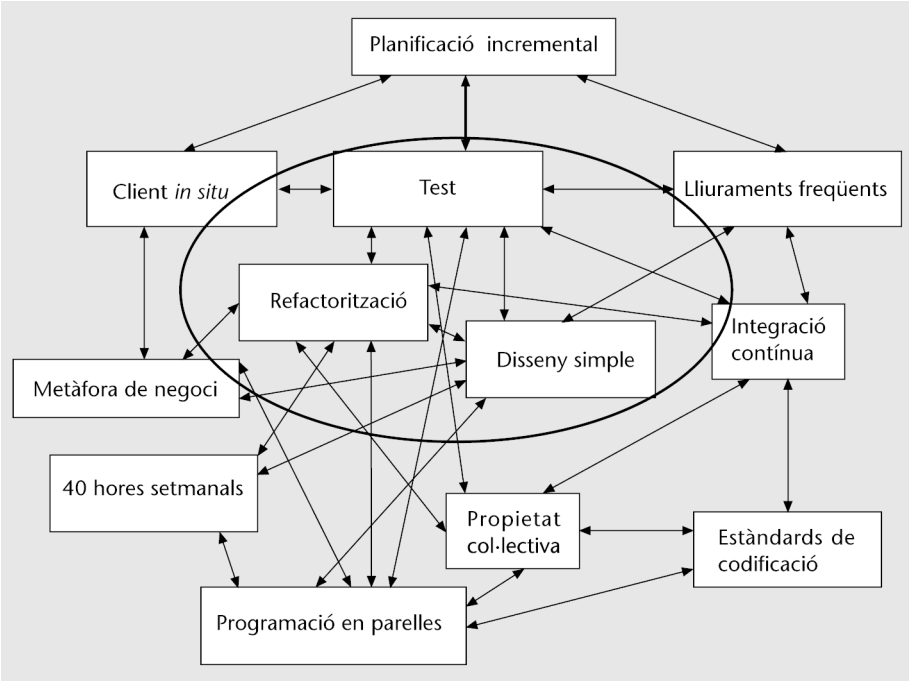
Algú de vosaltres pot pensar que això va en contra de tota la filosofia de disseny i utilització de patrons. Res més allunyat de la realitat. En un projecte XP, l'ús de patrons ens ajudarà a reduir el temps d'implantació, però el que no farem és dedicar temps a la implementació de patrons que no utilitzarem en aquest projecte; solament farem els que siguin necessaris per a aquest projecte, no utilitzarem temps del projecte per a beneficiar-ne un altre de futur que potser no arribarà mai. D'altra banda, res no ens impedeix desenvolupar un projecte que únicament es dediqui a desenvolupar patrons que més tard s'utilitzin en un projecte XP.

c) Realimentació. L'usuari ha d'utilitzar des del primer lliurament el programari desenvolupat, i ens ha de donar les seves impressions i les seves necessitats no satisfetes, de manera que aquestes històries tornin a formar part dels requisits del sistema.

d) Coratge. Coratge per a vèncer la frase més típica dels desenvolupadors: "si funciona no el toquis". Amb XP hem de tocar contínuament coses que ja funcionen, per millorar-les. Hem de canviar aquesta frase per la de "si funciona, el pots millorar". I això, us ho assegurem, requereix molt valor i coratge.

2.3. Les dotze pràctiques bàsiques d'XP

Kent Beck, Ward Cunninghamn i Ron Jeffries tenien molt clar les pràctiques que els havien donat millors resultats en els seus projectes, així que les van intentar aplicar totes juntes filant encara més prim. Aquest va ser l'embrió de la metodologia XP. Van crear les dotze pràctiques que es reforcen les unes amb les altres per obtenir els millors resultats. Les relacions que s'estableixen les unes amb les altres, les podem veure en el gràfic següent, que hem adaptat del que va proposar Kent Beck.



Al centre hem situat les pràctiques que més resultats ens poden donar en adaptar-les; no en són d'altres que el disseny simple, el test i la refactorització. Fins i tot si no volem prendre la totalitat de les pràctiques d'XP adoptant aquestes tres a la nostra metodologia habitual, podem tenir una millora substancial en els resultats obtinguts.

2.3.1. Disseny simple

"Ha de ser simple per a ser cert. Si no és simple, probablement no podrem desxifrar-ho."

Albert Einstein

No hi ha res tan encertat com aquesta frase per a il·lustrar la idea de disseny simple d'XP. Si el nostre disseny és simple, quan algú més el vegi l'entendrà, però si és complex probablement no el podrà desxifrar.

El principi és "utilitzar el disseny més senzill que aconsegueixi que tot funcioni", d'aquesta manera facilitarem el manteniment i minimitzarem els riscos de modificacions que es facin sense "entendre" el codi.

XP defineix un "disseny tan simple com sigui possible" com aquell que:

- No té codi redundant, ni duplicat.
- Supera tots els tests de funcionalitat, integritat i acceptació.
- No utilitza sintaxis complexes, és a dir, que queda clara la intenció dels programadors en cada línia de codi.
- Conté el menor nombre possible de classes i mètodes.

2.3.2. Refactorització

Així com en el Gènesi ens expliquen que al principi tot era caos i després tot va ser ordre, quan programem generalment passa el contrari. Inicialment tot són línies de codi ben ordenades i comentades, però a mesura que anem introduint canvis l'ordre es va perdent fins que allò deriva en una sèrie de línies de codi caòtiques, plenes de codi comentat de les diferents proves i amb comentaris obsolets que no es corresponen amb la realitat de la funcionalitat.

L'excessiu cost de les modificacions de les metodologies tradicionals es deu en gran manera a aquest **deteriorament progressiu del codi**, després de l'acumulació de modificacions.

Per a mantenir la corba del cost de canvi tan plana com sigui possible, en la metodologia XP s'aplica la **refactorització**, que no és altra cosa que modificar el codi per a deixar-lo en bon estat, tornant a escriure les parts que siguin necessàries, però sempre des d'un punt de vista global de la funcionalitat independentment del canvi que fem.

El codi final ha de conservar la claredat i senzillesa de l'original.

Els comentaris són molt importants per a mantenir aquesta claredat i senzillesa.

2.3.3. Test

És el pilar fonamental de les pràctiques d'aquesta metodologia –sense els tests, XP seria un fracàs total–, és el punt d'ancoratge que dóna la base metodològica la flexibilitat d'XP.

Si una funcionalitat no s'ha testat, només funciona aparentment. Els tests han de ser aplicats després de cada canvi, i **han de ser automatitzats**. Si no ho fem, podem incórrer en errors humans a l'hora de testar-los i això pot resultar fatal.

L'objectiu dels tests no és detectar errors, sinó evitar-los; no es tracta de corregir errors, sinó de prevenir-los.

Per a això, els tests sempre s'escriuen abans que el codi que s'ha de testar, mai després. D'aquesta manera estem obligats a **pensar per avançat quins** són els problemes que poden trobar quan usin el nostre codi, fa posar-nos en el comportament de l'usuari final, i aquest pensar per avançat evita molts problemes, ja que els preveiem en lloc de deixar que apareguin i després respondre-hi sobre la marxa.

Quan escrivim el codi ja sabem a què s'ha d'enfrontar, i d'aquesta manera els errors es minimitzen. L'objectiu del nostre programari ja no és complir unes funcionalitats, sinó passar uns tests.

És per això que l'usuari final ha d'ajudar el programador a desenvolupar els tests de manera conjunta, ja que en aquests tests estarà implícita la funcionalitat que volem que tingui. Un altre factor clau és que ha de ser el mateix programador qui desenvolupi els tests, perquè si no és així perdem l'avantatge de minimització d'errors.

Els tests han de ser de tres tipus: tests d'acceptació, tests unitaris i tests d'integritat; i sempre han d'estar automatitzats.

- **Test d'acceptació**. És creat conjuntament amb el client final i ha de reflectir les necessitats funcionals del primer.

- **Test unitari**. És creat pel programador per a veure que tots els mètodes de la classe funcionen correctament.

- **Test d'integritat**. És creat per l'equip de desenvolupament per a provar que tot el conjunt funciona correctament amb la nova modificació.

L'ús dels tests ens facilita la refactorització, cosa que ens permet comprovar de manera senzilla que els canvis originats per la refactorització no han canviat el comportament del codi.

2.3.4. Estàndards de codificació

L'equip de desenvolupament ha de tenir unes normes de codificació comuna, unes nomenclatures pròpies que tots els membres de l'equip puguin entendre. Per exemple, si el programa s'ha d'escriure a Java el millor és que tots utilitzin les "Code Conventions for the Java Programming Language" de Sun o una adaptació pròpia d'aquestes normes.

Web complementària

http://java.sun.com/docs/codeconv/index.html

El fet d'utilitzar una nomenclatura comuna permet que qualsevol persona de l'equip entengui amb més facilitat el codi desenvolupat per un altre membre; d'aquesta manera, facilitem les modificacions i la refactorització.

Per exemple, si decidim que a l'hora de anomenar les nostres variables d'una funció seguirem la norma següent:

1) La primera lletra ha de ser una *"v"* si és variable local o una *"p"* si ens ve per paràmetre d'entrada o una *"o"* si és paràmetre d'entrada i sortida.

2) La segona lletra ha d'indicar el tipus de dades: *"n"* si és numèrica, *"s"* si és una cadena de caràcters, *"d"* si és una data i *"b"* si és booleana.

3) La resta del nom ha de ser descriptiu del seu contingut lògic, separant significats amb el *"_"*.

Si jo ara en el codi veig la sentència següent:

```
If (vbPeriode_no_calculable) { odFinal_Periode = pdInici_Periode;}
```

tinc molta més informació que si hagués vist la següent:

```
If (pnc) { fp = ip;}
```

El funcionament és el mateix i ambdues sentències estan ben codificades, però la primera ens dóna el valor afegit de la comprensió de la codificació estàndard.

2.3.5. Propietat col·lectiva del codi

Per a poder aplicar la refactorització i per a assegurar-nos que el disseny és simple i que es codifiquen segons els estàndards, hem d'eliminar una altra de les idees que estan molt arrelades en el món del desenvolupament d'aplicacions; la "propietat individual" del codi.

Frases com "que ho modifiqui qui ho va fer que segur que ho entén millor" o "qui ha tocat la meva funció?" deixen de tenir sentit en XP, ja que el disseny simple ens garanteix que serà fàcil d'entendre. La refactorització permet que qualsevol membre de l'equip refaci el codi per tornar-li la senzillesa en cas que s'hagi complicat, els tests automatitzats ens garanteixen que no hem modificat el comportament esperat del codi amb la nostra modificació, i la codificació amb estàndards ens dóna aquest grau de comprensió addicional.

En XP el codi és propietat de tot l'equip i qualsevol membre té el dret i l'obligació de modificar-lo, per a fer-lo més eficient o comprensible, sense que ningú s'hagi de sentir ofès.

2.3.6. Programació per parelles

No aniríem més ràpids amb dues persones programant en lloc d'una programant i una altra mirant? La veritat és que inicialment pot xocar, la programació sempre s'ha vist com una cosa solitària, i tenir dues persones davant d'un sol teclat i d'un sol monitor sorprèn en un principi, però té molts avantatges.

Penseu en com funciona la ment humana: ens és molt complicat pensar abstractament i després passar a pensar concretament, i si ho fem contínuament acabem desconcentrats i cometem errors. És per això que, quan programem, primer pensem l'estratègia de codificació que seguirem i després ens posem a codificar cada una de les parts, sense pensar novament en l'estratègia fins que no acabem alguna de les parts o de vegades la totalitat del codi, i llavors veiem els errors o les coses que ens hem deixat en l'estratègia de codificació original.

Pensar en conjunt i detalladament alhora ens és impossible amb un sol cervell. Doncs posem-ne dos.

En la programació en parelles un dels membres ha de pensar des del punt de vista tàctic i l'altre des del punt de vista estratègic, de manera que aquests dos processos sempre estiguin actius i així es redueixin els errors i es millora la qualitat del programa. Òbviament aquests dos rols s'han d'intercanviar cada poc temps entre els membres de la parella per a incloure totes les possibilitats tàctiques i estratègiques.

El nivell dels membres de la parella ha de ser equivalent, no serveix que un en sàpiga molt i l'altre no en tingui ni idea, han d'estar equilibrats i òbviament portar-se bé perquè tingui èxit.

També la rotació ha de ser molt important: cada membre de l'equip ha de ser capaç de treballar en qualsevol àrea de l'aplicació que es desenvolupi. D'aquesta manera no provocarem "colls d'ampolla" quan assignem les tasques.

El fet que assignem les tasques per parelles fa que el temps d'aprenentatge es redueixi gairebé a zero, simplement substituint un dels membres per un altre de nou. Així doncs, la rotació d'àrees i de parelles ens garanteix que podrem fer un repartiment més equitatiu del treball sense haver de dependre d'una sol persona per a un treball específic.

Un altre efecte que produeix la programació en parelles és el psicològic: disminueix la frustració de la programació en solitari, ja que tens algú que entén el problema just al costat, i amb el qual el pots compartir. A més, moltes vegades els problemes vénen per falta de concentració (es té la solució davant dels nassos i no es veu) i es perd molt de temps. La programació per parelles disminueix la necessitat de l'"Efecte tòtem".

"Efecte tòtem"

El programador fa diverses hores que intenta solucionar un problema i decideix cridar un company per veure si ell veu la solució. Comença a explicar-li el problema i automàticament troba la solució sense que el segon company hagi pronunciat cap paraula. Rep aquest nom perquè podríem substituir el segon company per un "tòtem" indi amb exactament els mateixos resultats.

2.3.7. Integració contínua

En XP no esperem que totes les parts estiguin desenvolupades per a integrar-les al sistema, sinó que a mesura que es van creant les primeres funcionalitats ja s'hi acoblen, de manera que el sistema es pot construir diverses vegades durant un mateix dia. Això es fa perquè les proves d'integració vagin detectant els errors des del primer moment i no al final de tot i així l'impacte és molt menor.

És responsabilitat de cada equip publicar tan aviat com sigui possible cada funcionalitat o cada modificació. La idea és que tots els membres de l'equip treballin amb l'última versió del codi.

2.3.8. Quaranta hores setmanals

No es pot treballar durant catorze hores seguides i fer-ho amb qualitat. Les setmanes de setanta hores de treball són contraproduents. Els equips d'XP estan dissenyats per a guanyar, no per a morir en l'intent. Al final de la setmana s'ha d'arribar cansat però satisfet, mai exhaust ni desmotivat. Treballar hores

extres mina la moral i l'esperit de l'equip. Si durant dues setmanes cal fer hores extres, llavors és que el projecte va malament i s'ha de replantejar alguna de les quatre variables.

2.3.9. Metàfora del negoci

Perquè dues o més persones es puguin comunicar de manera eficient, han de tenir el mateix vocabulari i compartir el mateix significat. El model de negoci que entén l'usuari final segurament no es correspondrà amb el que creu entendre el programador. És per això que en els equips d'XP s'ha de crear **una metàfora amb la qual l'usuari final es trobi còmode** i que serveixi a l'equip de desenvolupament a l'hora de modelitzar les classes i els mètodes del sistema.

La metàfora és una història comuna compartida per l'usuari i l'equip de desenvolupament que descriu com s'han de comportar les diferents parts del sistema que es vol implementar.

2.3.10. Client *in situ*

En més d'una ocasió, quan programem o analitzem el sistemes, ens sorgeix un dubte i pensem que quan vegem a l'usuari final l'hi preguntarem. Possiblement haurem de continuar treballant sense resoldre aquest dubte i si la nostra suposició ha estat errònia, molt del treball realitzat es pot perdre.

XP necessita que el client final formi part de l'equip de desenvolupament i estigui ubicat físicament al mateix lloc perquè així s'agiliti el temps de resposta i es puguin validar totes les funcionalitats tan aviat com sigui possible.

En XP el client sempre ha d'estar disponible per a la resta de l'equip, formant part d'ell i fomentant la comunicació cara a cara, que és la més eficient de les comunicacions possibles.

2.3.11. Lliuraments freqüents

S'han de desenvolupar tan aviat com sigui possible versions petites del sistema, que encara que no tinguin tota la funcionalitat, ens donin una idea de com ha de ser el lliurament final i ens serveixin perquè l'usuari final es vagi familiaritzant amb l'entorn i perquè l'equip de desenvolupament pugui executar les proves d'integritat.

Les noves versions han de ser tan petites com sigui possibles, però han d'aportar **un nou valor de negoci per al client**.

Donar valor per al negoci és el que ens aconseguirà que la visió final del client sigui la millor possible.

2.3.12. Planificació incremental

La planificació mai no serà perfecta, sinó que variarà en funció de com variïn les necessitats del negoci, i en cada cicle de replanificació es tornaran a establir les quatre variables de la metodologia XP.

Assumir una planificació estàtica no es correspon amb l'agilitat que li volem donar, ja que les necessitats del negoci poden canviar dràsticament mentre desenvolupem l'aplicació. En XP la planificació es va revisant contínuament, de manera incremental, prioritzant aquelles necessitats de negoci que ens aportin més valor.

La planificació es planteja com un diàleg permanent entre els responsables de la perspectiva empresarial i de la perspectiva tècnica del projecte.

2.4. El cicle de vida de la metodologia XP

Les dotze pràctiques havien donat separadament molt bons resultats a Kent Beck i companyia. Unificar-les en el cicle de vida d'una metodologia és el que va donar origen a XP.

La millor forma de comunicació és la presencial. Quan tenim una cosa important a dir ens agrada "dir-la a la cara", perquè no hi hagi equívocs. L'expressió, l'entonació i les mirades són molt importants.

El cicle de vida d'XP s'organitza com si fos una conversa client-desenvolupador.

1) Tinc una necessitat.

2) Puc ajudar-te? Què necessites?

3) Això és el que necessito.

4) Crec que ho entenc.
Si construeixo això et serveix?

5) Doncs sí. Podries fer-ho?

6) Ara mateix em poso a treballar.

7) Has acabat?

8) Hi estic posat.

9) Necessitaria també això.

10) És crucial? Trigaria molta estona.

11) Llavors no ho facis.

12) Ja tinc això. Et serveix?

13) No del tot.

14) I aquesta nova versió?

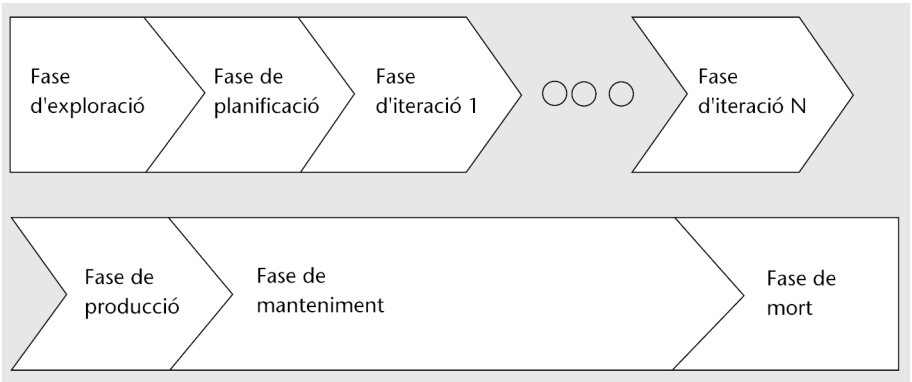
15) Sí, és just el que necessito. Gràcies.

16) De res. Si hi tens problemes avisa'm.

Aquest seria el desenvolupament ideal d'un projecte XP. Per a apropar-nos-hi, establim un cicle de vida dividit en sis fases.

Cicle de vida d'un projecte XP

1) Fase d'exploració
2) Fase de planificació
3) Fase d'iteracions
4) Fase de producció
5) Fase de manteniment
6) Fase de mort del projecte



2.4.1. La fase d'exploració

La fase d'exploració és la primera fase del cicle de vida de la metodologia XP, en la qual es desenvolupen tres processos:

1) Les històries d'usuari
2) L'*spike* arquitectònic
3) La metàfora del negoci

Tot comença amb les històries d'usuari. En aquesta fase els usuaris plantegen a grans trets les funcionalitats que volen obtenir de l'aplicació. Les històries d'usuari tenen el mateix propòsit que els casos d'ús, excepte en un punt crucial: les escriuen els usuaris i no l'analista. Han de ser descripcions curtes i escrites en el llenguatge de l'usuari, sense terminologia tècnica. Aquestes històries són les que guiaran la creació dels tests d'acceptació que han de garantir que aquestes històries s'han comprès i s'han implementat correctament.

No hem de confondre les històries d'usuari amb l'anàlisi de requisits. La principal diferència està en la profunditat d'anàlisi: amb els requisits volem arribar a l'últim detall per "no agafar-nos els dits davant del client", però en XP el client forma part de l'equip i li podem preguntar més coses durant la implementació, per la qual cosa el nivell de detall en les històries d'usuari ha de ser el mínim imprescindible perquè ens fem una idea general de la funcionalitat.

Les **històries d'usuari** han de ser:

- Escrites pel client final; en el seu llenguatge i sense tecnicismes.
- Descripcions curtes de la usabilitat i funcionalitat que s'espera del sistema.

Paral·lelament i conjuntament, es comença amb l'*spike* arquitectònic, en el qual l'equip de desenvolupament es comença a familiaritzar amb la metodologia, les eines, el llenguatge i les codificacions que s'usaran en el projecte.

En l'***spike* arquitectònic** l'equip de desenvolupament:

- Prova la tecnologia.
- Es familiaritza amb la metodologia.
- Es familiaritza amb les possibilitats de l'arquitectura.
- Realitza un prototip que demostri que l'arquitectura és vàlida per al projecte.

Finalitzades les històries d'usuari i l'spike arquitectònic, es passa a desenvolupar conjuntament la metàfora del negoci.

La metàfora del negoci:

- És una història comuna compartida per l'usuari i l'equip de desenvolupament.
- Ha de servir perquè l'usuari se senti de gust referint-se al sistema en els termes de la metàfora.
- Ha de servir als desenvolupadors per a implementar les classes i objectes del sistema.

Vegeu també

Com ja vam veure en l'apartat "Metàfora del negoci" d'aquest mòdul, aquesta metàfora és una de les dotze pràctiques bàsiques d'XP.

2.4.2. La fase de planificació

El resultat ha de ser una planificació (recordem que sempre flexible) del projecte.

El procediment és el següent:

- El client lliura a l'equip de desenvolupament les **històries d'usuari** que ha confeccionat, però prioritzant-les de més a menys importància.
- L'equip de desenvolupament les estudia i **estima el cost** d'implementar-les:
  - Si l'equip de desenvolupament considera que la història d'usuari és massa complexa, llavors l'usuari final l'ha de descompondre en diverses històries independents més senzilles.
  - Si l'equip de desenvolupament no veu clar com implementar una part de la història d'usuari, pot realitzar un ***spike* tecnològic** per veure com es podria implantar i així poder avaluar el cost.
  - Una vegada tenim la llista d'històries prioritzades juntament amb el seu cost d'implementació, passem a convocar la **reunió del pla de lliuraments**.
  - El pla de lliuraments es compon d'una sèrie de plans d'iteració en el qual s'especifica quines funcionalitats s'implementaran en cada volta de la fase d'iteracions.
  - Participen d'aquesta reunió tant els usuaris com l'equip tècnic i cadascú ha d'aportar la seva visió del negoci de manera que s'obtinguin més ràpidament aquelles funcionalitats que donin el major benefici possible per al negoci.

– A cada iteració se li assigna un temps intentant que totes siguin més o menys idèntiques (encara que no és necessari).

– Es determina l'abast del projecte.

2.4.3. La fase d'iteracions

Com que hem dividit el projecte en iteracions, aquesta fase es repetirà tantes vegades com iteracions tinguem. Generalment cada iteració sol ser de dues a tres setmanes.

Es tracta el **pla d'iteració** de la manera següent:

- Es recullen les històries d'usuari assignades a aquesta iteració.

- Es detallen les **tasques que s'han de realitzar** per cada història d'usuari:
  – Les tasques han de ser d'un o tres dies de desenvolupament. Si són més grans, hauríem d'intentar dividir-les en diverses de més senzilles.

  – S'estima el cost de cada tasca. Si el total és superior al temps d'iteració, s'haurà de prescindir d'alguna història d'usuari que es passaria a la iteració següent. Si són moltes les històries d'usuari rebutjades, llavors cal tornar a estimar les quatre variables de la metodologia i tornar a planificar el projecte.

  – Si el temps total estimat de les tasques és inferior al temps d'iteració, es pot assumir una història d'usuari que correspongués a la iteració següent.

- Es prioritzen les tasques que més valor donaran al negoci, intentant que s'acabin històries d'usuari tan aviat com sigui possible.

- Es reparteixen les primeres tasques a l'equip de desenvolupament i la resta es deixa en una **cua de tasques sense assignar** d'on s'aniran agafant.

- Es convoquen **reunions de seguiment cada dia** per a veure si ens anem endarrerint en les estimacions o ens hi anem avançant i així poder rebutjar o incorporar-hi històries d'usuari.

El més important és que a cada moment de cada iteració realitzem la tasca que més valor possible dóna al negoci d'entre les que tenim pendents, de manera que, si hem de reduir l'abast del projecte, solament afecti les funcionalitats secundàries de la nostra aplicació.

2.4.4. La fase de producció

Arribem a aquesta fase en assolir la primera versió que l'usuari final decideixi que es pot posar en producció.

> Passarem l'aplicació a producció quan assoleixi les funcionalitats mínimes que aportin un valor real al negoci i una operativa arquitectònica estable.

És a dir, no esperem a tenir totes les funcionalitats implementades, sinó que quan tenim una cosa que els usuaris poden utilitzar i que ajuda el negoci, passem la primera versió a producció.

Paral·lelament se segueix amb les iteracions finals de projecte, però fixeu-vos que abans que finalitzi ja donem valor a l'organització, el ROI (retorn de la inversió) del projecte es comença a generar abans que aquest finalitzi la seva versió final.

En l'etapa de producció es realitzen també iteracions com en l'etapa anterior, però el ritme d'aquestes ja no és de dues a tres setmanes sinó mensuals.

Aquesta fase es manté fins que realitzem l'últim lliurament, amb el qual acabem l'àmbit de l'aplicació i passem a mantenir-lo.

Durant la fase de producció, el ritme de desenvolupament decau a causa que l'equip ha de resoldre les incidències dels usuaris. És per això que de vegades és necessari incorporar nou personal a l'equip.

2.4.5. La fase de manteniment

Una vegada l'abast del projecte s'ha aconseguit, i tenim totes les funcionalitats en producció, es revisen amb l'usuari aquelles noves històries d'usuari que s'han produït després de la posada en producció del projecte. Aquestes noves funcionalitats s'hi van incorporant segons el seu valor de negoci i el pressupost addicional del qual es disposi.

L'equip de desenvolupament es redueix a la mínima expressió, deixant alguns membres per al manteniment.

2.4.6. La fase de mort del projecte

Quan no queden més històries d'usuari per introduir al nostre sistema o quan es redueix progressivament el valor de les històries d'usuari que s'hi han implementat, el projecte entra en la fase de mort.

S'anirà desinvertint en el sistema fins a abandonar-lo totalment quan no aporti valor al negoci o quan les seves històries d'usuari hagin estat absorbides per un altre sistema d'informació.

2.5. Els diferents rols dins d'XP

Cada rol té unes funcions clares dins de la metodologia XP. Cada persona de l'equip pot executar un o diversos rols, o fins i tot canviar de rol durant les diferents fases del projecte.

Són moltes les extensions que s'han fet dels rols de la proposta original de Beck, però els rols que romanen sempre en qualsevol implementació de la metodologia XP són els següents:

Programador:

- Escriu les proves unitàries.
- Produeix el codi del programa.

Client:

- Escriu les històries d'usuari.
- Dissenya les proves d'acceptació.
- Prioritza les històries d'usuari.
- Aporta la dimensió de negoci a l'equip de desenvolupament.
- Representa el col·lectiu d'usuaris finals.
- Està sempre disponible per a consultes.

Encarregat de proves (*tester*):

- Ajuda el client a dissenyar proves d'acceptació.
- Executa les proves d'acceptació.
- Executa les proves d'integració.
- Difon els resultats entre l'equip de desenvolupament i el client.
- És el responsable de les eines automatitzadores de les proves.

Encarregat de seguiment (*tracker*):

- S'encarrega de realimentar tot el procés d'XP, mesurant les desviacions respecte a les estimacions i comunicant els resultats per a millorar les estimacions següents.

- Realitza el seguiment de cada iteració del procés d'XP tant en l'etapa d'iteracions com en la de producció.

- Reavalua la possibilitat d'incorporar-hi o eliminar històries d'usuari.

Entrenador (*coach*):

- S'encarrega del procés global.
- Garanteix que se segueix la filosofia d'XP.
- Coneix a fons la metodologia.
- Proveeix guies i ajuda els membres de l'equip a l'hora d'aplicar les pràctiques bàsiques d'XP.

Consultor:

- No forma part de l'equip.
- Té un coneixement específic d'una àrea en concret.
- Ajuda a resoldre un problema puntual, tant si es tracta de spike tecnològic com de valor de negoci.

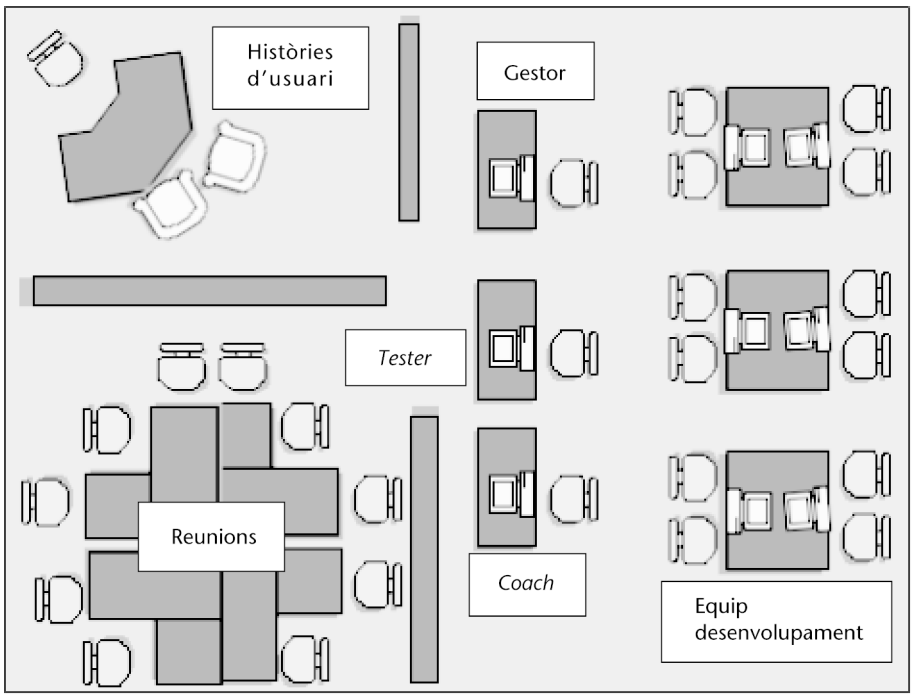
Gestor (*boss*):

- És el màxim responsable del projecte.
- Fa d'enllaç amb els clients.
- S'encarrega de coordinar i de garantir les condicions necessàries per al desenvolupament del treball.

2.6. L'entorn de treball

XP necessita, a més, un entorn que afavoreixi la comunicació. No serveix de res tenir el nostre equip dispers per diferents plantes i edificis, o que el client es trobi a cinc minuts pel passadís principal de l'oficina. Les barreres arquitectòniques i visuals s'han d'eliminar en la mesura que es pugui d'un projecte d'XP. La comunicació verbal és fonamental.

Una possible distribució d'un equip d'XP podria ser la següent:



A aquest esquema final seria convenient afegir-hi una sala de desconnexió en la qual els programadors fatigats poguessin fer un descans, prendre's un refresc, picar alguna cosa o simplement mirar per la finestra.

Un altre punt important és que el client ha de ser a prop i estar disponible, però ha de tenir un entorn que li permeti desenvolupar el seu treball habitual. Per aquesta raó en l'esquema es troba una mica apartat del gruix de l'equip.

2.7. Què en diuen els detractors d'XP

No tot són flors i violes per a la metodologia XP. També s'ha escrit molt en contra d'aquesta manera d'entendre la creació de programari. Vegem els arguments principals en contra:

a) XP sol funciona amb equips "brillants" de gent disciplinada i amb visió de negoci. Si tenim programadors "normals", en aplicar XP obtindrem l'efecte contrari, desenvolupament molt més lent i ple d'errors.

b) Les empreses no deixaran que un client s'integri al projecte de desenvolupament, i si aconseguim que ho consentin segurament serà aquell usuari final que tingui menys valor per al negoci i que no representi els problemes de l'empresa. Així les històries d'usuari i la seva assignació de prioritats no seran representatives i construirem la nostra aplicació sobre una fal·làcia.

c) Quan fas un projecte en una empresa el primer que et demanen és saber el temps i l'abast, abans d'acceptar-te un projecte. I amb XP totes dues són variables al llarg del projecte. Sense haver-les prefixat no es poden fer projectes per a terceres empreses i menys optar a un concurs públic.

d) XP es basa en les proves i els tests, però en un "entorn real" amb la pressió que solen patir els projectes de desenvolupament la primera cosa que es redueix a la mínima expressió són les proves. XP no és aplicable a un entorn d'una organització real.

e) La programació en parelles és una utopia. Al final deriva en la situació que una persona treballa i l'altra descansa. A més és difícil de justificar-la al client que paga el programari, ja que l'hora/persona mesurada és la que impera en el mercat. Com facturem si veu que de cada dues persones una només apunta coses a l'altra?

f) Els equips XP han de ser petits, no es podria realitzar amb equips de projectes grans.

g) Deixa de banda la documentació, una cosa fonamental per a controlar l'alta rotació en les empreses serveis de TI.

Com podem veure, no falten arguments, però saber si són sòlids o no depèn ja del vostre esperit crític. Però això sí, recordeu que, abans de criticar o lloar aquesta metodologia de projectes, potser seria convenient que comencéssim a aplicar-la en algun projecte poc important, per així poder-nos formar una opinió fonamentada en la nostra experiència.

2.8. Un exemple d'XP

Java és, sens dubte, el llenguatge de programació més accessible als estudiants i als programadors no professionals. A més, JEE és la plataforma de desenvolupament més estesa entre els moviments de les comunitats de programari lliure. Per aquesta raó, unida al fet que tots dos moviments estan molt vinculats entre si, hem cregut convenient realitzar un exemple del que seria un projecte XP en un entorn purament JEE.

Per a això tenim com a "voluntària" la ja famosa "botiga de mascotes" que, juntament amb l'"Hola, món", són els dos exemples més utilitzats en la història recent de la informàtica.

Per desgràcia no ens podem estendre massa en la descripció detallada de la codificació, però sí que intentarem mostrar les decisions més importants durant el cicle de vida del projecte.

2.8.1. Fase d'exploració

El client es posa en contacte amb nosaltres i ens comunica que necessita crear una web de la seva botiga de mascotes, per a la qual cosa té un pressupost limitat. L'equip de desenvolupament el compondran una o dues persones, que realitzaran tots els rols dins de la metodologia XP.

El primer que farem és anomenar el projecte –Projecte Extrem Tot Animals– i esperarem que les sigles no siguin un indicatiu del resultat final.

El segon pas és sol·licitar i ajudar el client a escriure les **històries d'usuari**.

Història d'usuari 1: Presència a Internet
Vull que la gent que busqui la meva botiga de mascotes la pugui trobar buscant al Google, i així trobar la meva adreça, el meu correu electrònic, i els telèfons de contacte.

Història d'usuari 2: Catàleg de productes
A més, m'agradaria que els clients poguessin veure els meus productes i els preus a la web.

Història d'usuari 3: Comandes de productes
I comprar-los.

Història d'usuari 4: Carretó de la compra
No d'un en un, sinó diversos alhora.

Història d'usuari 5: Llistes de productes
I que el programa recordés els productes que més compra.

Història d'usuari 6: Productes vinculats
I que li suggerís nous productes per comprar, segons els que hagi comprat.

Història d'usuari 7: Ofertes i promocions
I poder destacar les ofertes de la setmana i descomptes en comprar diversos productes alhora.

Història d'usuari 8: Targetes client
I que acumulessin punts per comprar per Internet per a descomptes o regals.

Història d'usuari 9: Pagament per mòbil
I que poguessin pagar la comanda amb una trucada de mòbil en lloc de posar la targeta de crèdit.

Paral·lelament hem iniciat l'***spike* arquitectònic** basat en JEE.

Nota

Vegeu JEE, JavaBeans i els Enterprise JavaBeans a l'assignatura *Enginyeria del programari de components i sistemes distribuïts* del Grau d'Enginyeria en Informàtica.

Els dos membres de l'equip de desenvolupament es decanten per una arquitectura clàssica en tres capes per al projecte web.

L'únic dubte que tenien era si utilitzar només JavaBeans o Enterprise JavaBeans a l'hora de la persistència de les dades de les transaccions econòmiques. Per a això contacten amb diversos bancs que realitzen pagaments en línia i descobreixen que en la majoria dels casos l'única cosa que han de fer és redireccionar a les pàgines del banc quan es vagi a realitzar el pagament i una

vegada finalitzat, el banc redireccionarà el client a la pàgina de la botiga que els indiquin. Amb aquesta informació fan algunes proves i es decanten per JavaBeans, simplificant així l'arquitectura.

Els dos membres de l'equip de desenvolupament estan familiaritzats amb les eines OpenSource i després de fer diverses proves trien les següents:

Webs complementàries

Per a ampliar informació sobre les eines OpenSource podeu veure: http://ant.apache.org, http://www.junit.org, http://tomcat.apache.org/ i http://www.jboss.org/

- **Ant**. Eina OpenSource de codificació i compilació de programes Java.
- **Junit**. Eina OpenSource de creació i execució automatitzada de tests unitaris sobre classes Java i especialitzada en Extreme Programming.
- **Tomcat**. Com a servidor web OpenSource on residirà l'aplicació, permet l'ús de Java Server Pages. S'ha descartat JBoss ja que hem decidit que no necessitarem EJBs.

Totes les eines escollides són compatibles les unes amb les altres i estan plenament integrades.

L'últim pas és desenvolupar la **metàfora del negoci** perquè usuari i equip de desenvolupament se sentin còmodes i se sàpiga que parlen en els mateixos termes.

- *Domini*. Nom pel qual es coneixerà la nostra botiga a Internet, per exemple: "tot animals".
- *Botiga en línia*. Es tracta de la pàgina web que serà la nostra representació a Internet del negoci de la botiga de mascotes. És el punt d'entrada. En posar l'adreça http://www.totanimals.com al navegador apareixerà la nostra botiga.
- *Menú*. Són les diferents opcions que tindrem a l'hora de veure la informació de la botiga. Seria com les diferents portes d'accés a la botiga física.
- *Seccions*. La botiga en línia s'organitza en seccions, igual que la botiga física. A cada secció trobarem productes.
- *Barra de navegació*. Seria com caminar per la botiga física. A la botiga en línia posarem una zona que permetrà anar al producte següent, veure tots els productes de la secció, canviar de secció, etc.
- *Catàleg de productes*. El componen tots els productes que tindrem a la botiga en línia. Seria com el catàleg imprès, però amb la capacitat de poder canviar les pàgines fàcilment.

- *Productes*. Són els productes que vendrem a la botiga en línia. Igual com a les botigues físiques, un producte es pot exposar en una o més seccions.

- *Carretó de la compra*. Correspon a la zona de la botiga en línia on anirem dipositant els productes que volem comprar.

- *Procés de pagament*. El seu funcionament és molt semblant a la idea següent. Imagina que a la botiga real els clients no ens paguen a nosaltres a la caixa enregistradora, sinó que se'n van amb el carretó de la compra al banc, on paguen, i el banc després ens la paga a nosaltres. L'intercanvi de diners no el fem a la nostra botiga en línia sinó al banc.

- *Altres conceptes*. Que l'usuari i l'equip de desenvolupament creguin necessitar amb tantes analogies i/o metàfores com es cregui necessàries.

2.8.2. Fase de planificació

El client **prioritza** les històries d'usuari.

Història d'usuari 1: Presència a Internet
Història d'usuari 2: Catàleg de productes
Història d'usuari 6: Productes vinculats
Història d'usuari 3: Comandes de productes
Història d'usuari 7: Ofertes i promocions
Història d'usuari 4: Carretó de la compra
Història d'usuari 8: Targetes client
Història d'usuari 5: Llistes de productes
Història d'usuari 9: Pagament per mòbil

L'equip de desenvolupament **avalua el cost** d'implementació de cada una de les històries en jornades de vuit hores per parella de programadors.

Història d'usuari 1: Presència a Internet (deu jornades)
Història d'usuari 2: Catàleg de productes (vint jornades)
Història d'usuari 6: Productes vinculats (quatre jornades)
Història d'usuari 3: Comandes de productes (sis jornades)
Història d'usuari 7: Ofertes i promocions (deu jornades)
Història d'usuari 4: Carretó de la compra (deu jornades)
Història d'usuari 8: Targetes client (quatre jornades)
Història d'usuari 5: Llistes de productes (quatre jornades)
Història d'usuari 9: Pagament per mòbil (dues jornades)

Després de la reunió del **pla de lliuraments** s'estableix que la iteració serà bisetmanal (deu jornades) i que els membres de l'equip efectivament seran dos.

La planificació és la següent:

- Iteració 1
  Història d'usuari 1: Presència a Internet (deu jornades)
  Primer pas a producció

- Iteració 2
  Història d'usuari 2: Catàleg de productes amb les seccions més importants (deu jornades)

- Iteració 3
  Història d'usuari 2: Catàleg de productes amb les seccions restants (deu jornades)
  Segon pas a producció

- Iteració 4
  Història d'usuari 6: Productes vinculats (quatre jornades)
  Història d'usuari 3: Comandes de productes (sis jornades)

- Iteració 5
  Història d'usuari 7: Ofertes i promocions (deu jornades)
  Tercer pas a producció

- Iteració 6
  Història d'usuari 4: Carretó de la compra (deu jornades)
  Quart pas a producció

Queden fora de l'àmbit del projecte les històries d'usuari següents:

Història d'usuari 8: Targetes client (quatre jornades)
Història d'usuari 5: Llistes de productes (quatre jornades)
Història d'usuari 9: Pagament per mòbil (dues jornades)

És a dir, sis iteracions de dues setmanes cada una, amb quatre passos a producció. En aquesta planificació a partir de la segona setmana d'haver iniciat el projecte ja donem valor al negoci, en haver implementat la història d'usuari 1, que era la més important en la priorització del client.

2.8.3. Fase d'iteracions

En l'exemple actual, aquesta fase finalitzaria en la primera iteració, ja que en aquest moment passaríem a la fase de producció. El més important d'aquesta primera fase és crear la carcassa de l'aplicació final, sobre la qual ho anirem construint tot; és a dir, a més de la pàgina web que cobriria la història

d'usuari 1, hauríem de crear també les classes Java –Botiga, Secció, Catàleg, Producte, Carretó– i crear els seus primers tests unitaris i els seus primers tests d'integració.

Si ens centrem en la classe Producte, una possible primera implementació podria ser aquesta:

```
/* Classe producte */
/* Versió 1.0 */
package uoc.botiga;

public abstract class Producte {
   public void setPreu(double preu) {this.preu = preu; }
   public double getPreu(){ return preu }
   protected double preu;
}
```

Després d'acabar la implementació, l'altre membre de la parella podria dir que la funció setPreu no és del tot clara, que seria millor utilitzar la nomenclatura del projecte:

- La primera lletra ha de ser una *"v"* si és variable local o una *"p"* si ens ve per paràmetre d'entrada o una *"o"* si és paràmetre d'entrada i sortida.

- La segona lletra ha d'indicar el tipus de dades: *"n"* si és numèrica, *"s"* si és una cadena de caràcters, *"d"* si és una data i *"b"* si és booleana.

- La resta del nom ha de ser descriptiu amb el seu contingut lògic, separant significats amb el *"_."*

Així, la classe Producte quedaria de la manera següent:

```
/* Classe producte */
/* Versió 1.1 */
package uoc.botiga;

public abstract class Producte {
   public void setPreu(double pnPreu) {this.vnPreu = pnPreu; }
   public double getPreu(){ return vnPreu }
   protected double vnPreu;
}
```

Sens dubte això no es queda aquí, perquè recordem que un dels dos programadors ha de pensar des d'un punt de vista estratègic. Per això veu clarament que tots els objectes del projecte tindran tres propietats fixes –Identificador, Nom i Descripció–, així que incita a crear una classe amb aquestes tres propietats de les quals hereti la resta.

```
/* Classe producte */
/* Versió 1.2 */
package uoc.botiga;

public abstract class Producte extends ObjecteBasic{
   protected double vnPreu;
   public void setPreu(double pnPreu) {this.vnPreu = pnPreu; }
   public double getPreu(){ return vnPreu }
}
```

```
/* Classe Objecte Bàsic*/
/* Versió 1.0 */
package uoc.botiga;

public abstract class ObjecteBasic {
   protected int vnIdentificador;
   protected String vsNom;
   protected String vsDescripcio;

   /* Propietat Identificador */
   public void setIdentificador(int pnIdentificador)
       throws Exception { this.vnIdentificador = pnIdentificador; }
   public int getIdentificador(){ return vnIdentificador; }

   /* Propietat Nom*/
   public void setNom (String psNom){ this.vsNom = psNom; }
   public String getNom (){ return vsNom; }

   /* Propietat Descripcio*/
   public void setDescripcio(String psDescripcio)
       { this.vsDescripcio = psDes-cripcio; }
   public String getDescripcio (){ return vsDescripcio; }
}
```

No ens hem d'oblidar de fer els tests unitaris per a cada un dels objectes. Per a això utilitzarem les classes de JUnit, en especial la classe abstracta Assert, que ens permet una gran varietat de comprovacions, i la seva implementació en la classe TestCase. Una altra classe interessant és TestSuite, que ens permet automatitzar una gran quantitat de tests unitaris.

Web complementària

La jerarquia de la classe Assert la podem trobar a: http://junit.sourceforge.net/javadoc/org/junit/package-tree.html

Seguint amb l'exemple, la primera versió del test unitari de la classe Producte seria la següent:

```
package uoc.botiga.test;

import.uoc.botiga.Producte;
import junit.framework.TestCase;

public class TestUnitariProducteextends TestCase {
   public void testProducte( Producte pPTest) {
   // Creo una altra instància d'un producte
   final Producte producte1 = new Producte;
   // Comprovo la creació d'objectes per veure que s'han instanciat bé
      assertNotSame(producte1, pPTest);
      assertSame(producte1, producte1);
      assertSame(pPTest, pPTest);
// Assigno les propietats a producte 1 els mètodes set
   producte1.setPreu(pPTest.getPreu());
// correcte i concordant amb el mètode set
   assertEquals(pPTest.getPreu(), producte1.getPreu ());
// Finalment comprovo les propietats lògiques que ha de complir
   la classe Producte
// Propietat 1: Preu no ha de ser mai més petit que zero.
   assertFalse(pPTest.getPreu() <0);
// Propietat 2: Preu no ha de ser mai més gran que 999.999.
   assertTrue(pPTest.getPreu() <999999);
}
}
```

En acabar aquesta primera iteració hauríem de tenir la pàgina d'entrada inicial i una carcassa de tota l'aplicació, i també els tests unitaris i d'integració, i les automatitzacions d'aquests tests.

L'última part d'aquesta primera fase són els tests d'acceptació que ha creat el client amb l'ajuda del membre de l'equip amb el rol d'encarregat de proves. Si es passen amb èxit, posarem aquesta iteració en producció. En el nostre cas els tests d'acceptació haurien de reflectir les necessitats de la Història d'usuari 1: Presència a Internet.

La creació de tests i els algorismes de testeig donarien per a tot un capítol d'aquest llibre, però per desgràcia està fora de l'abast d'aquesta introducció.

2.8.4. Fase de producció

Durant aquesta fase es realitzarien les iteracions de la 2 a la 6, segons el pla de lliuraments que havíem dissenyat. Cap a la meitat de la segona iteració ens adonem que avancem més del que ens esperàvem i que en lloc de trigar les deu jornades previstes, trigarem vuit jornades, de manera que ens plantegem assumir certes tasques de la iteració següent, per a la qual estimem ara que trigarem també vuit jornades en lloc de deu. La planificació queda així:

- Iteració 1
  Finalitzada

- Iteració 2
  Història d'usuari 2: Catàleg de productes amb les seccions més importants (vuit jornades)

Història d'usuari 2: Catàleg de productes amb les seccions restants. Part 1 (dues jornades)

- Iteració 3
  Història d'usuari 2: Catàleg de productes amb les seccions restants. Part 2 (sis jornades)
  Història d'usuari 6: Productes vinculats (quatre jornades)
  Segon pas a producció

- Iteració 4
  Història d'usuari 3: Comandes de productes (sis jornades)
  Història d'usuari 7: Ofertes i promocions. Part 1 (quatre jornades)

- Iteració 5
  Història d'usuari 7: Ofertes i promocions. Part 2 (sis jornades)
  Tercer pas a producció

- Iteració 6
  Història d'usuari 4: Carretó de la compra (deu jornades)
  Quart pas a producció
  Ens queda ara una Iteració 5 amb només sis jornades i amb un buit per a quatre jornades addicionals. Després de parlar amb el client decidim incloure en aquest buit una de les funcionalitats que havíem descartat, la Història d'usuari 8: Targetes client (quatre jornades). Així, la Iteració 5 queda de la manera següent:

- Iteració 5
  Història d'usuari 7: Ofertes i promocions. Part 2 (sis jornades)
  Història d'usuari 8: Targetes client (quatre jornades)
  Tercer pas a producció

Durant les iteracions següents no hi ha cap incidència i el projecte acaba tal com s'havia replantejat en la Iteració 2.

2.8.5. Fase de manteniment

Durant els dos primers mesos després de l'acabament de la fase de producció, es van anar solucionant algunes deficiències, sobretot de rendiment de l'aplicació, ja que en XP es fa prevaler que les coses funcionin primer i que després siguin eficients. Una vegada realitzades aquestes millores i a causa que el nombre de visites a la botiga real havia crescut de manera espectacular, es decideix invertir una mica més i implementar la Història d'usuari 9: Pagament per mòbil (dues jornades).

2.8.6. Fase de mort

Durant cinc anys la web va funcionar fins que l'empresa va ser absorbida per una multinacional de venda de mascotes.

3. Ser àgils no és només programar àgils

De què ens serviria tenir una gran elasticitat a les cames si no poguéssim doblegar els genolls? Exactament el mateix ens passa en el desenvolupament de sistemes d'informació. No ens serveix de res ser àgils en la creació de programari si no ho som en la resta de facetes organitzatives del nostre treball.

L'agilitat va arribant a totes les facetes de l'enginyeria del programari:

- Agile Modeling ens permet realitzar models que s'adapten a les metodologies aquí vistes. L'èxit és degut a la seva gran adaptabilitat, ja que pot ser usada tant en metodologies RUP, com en XP o qualsevol altra metodologia de desenvolupament de programari.

Web complementària

Agile Modeling pretén ser "una guia per a l'art de modelar", mai per a la ciència de modelar.

- Agile Data Method ens ajuda en la creació i manteniment de les bases de dades que donen suport als projectes àgils, en els quals el canvi és constant i de vegades traumàtic per als administradors d'aquelles.

Web complementària

http://www.agiledata.org/

- Agile Documentation ens ajuda a centrar-nos en la documentació fonamental dels projectes àgils que utilitzin Agile Modeling.

Web complementària

http://www.agilemodeling.com/essays/agileDocumentation.htm

- Agile Legacy Integration Modeling ens permet dissenyar de manera àgil models d'aplicacions que s'integrin amb sistemes heretats de tipus ordinador central o *host*.

- Agile Qualsevolcosa. Són moltes les iniciatives que es desenvolupen actualment entorn dels diferents aspectes que falten per cobrir, però sens dubte molts autors aprofiten la tirada de les metodologies àgils per a poder donar als seus projectes l'etiqueta "de moda". Només el temps dirà qui eren visionaris i qui eren simples oportunistes.

Resum

En aquest mòdul hem vist com cada projecte (o part d'aquest) es pot realitzar amb una metodologia diferent i que és responsabilitat nostra triar la que s'adapta més bé a cada situació en concret. Són moltes les metodologies àgils, però hem vist que totes tenen característiques i objectius similars: n'hi ha que posen més l'accent en la comunicació humana dels projectes, d'altres a donar valor al negoci tan aviat com sigui possible, d'altres a assumir que tot és canviant, però totes són útils en determinats projectes, cobreixen una necessitat o mancança que s'ha detectat en les metodologies de desenvolupament d'aplicacions.

D'entre aquestes metodologies, hem conegut la que ha cobrat més força: la metodologia Extreme Programming. Els quatre valors sobre els quals es fonamenta, les dotze pràctiques que proposa i les quatre variables dels projectes ens han fet constatar que es tracta d'una metodologia molt madura, gens menyspreable i perfectament aplicable a una gran varietat de projectes. També hem vist que XP és compatible amb altres metodologies tant si són àgils com DSDM, com clàssiques com RUP.

Ens hem introduït en els diferents àmbits en què l'agilitat irromp amb força, dels quals el més prometedor és Agile Modeling. Conèixer tots els àmbits i totes les propostes ens dóna un punt de referència per a la reflexió, ens permet saber on poden fallar les metodologies i així poder-les aplicar amb més bon criteri.

Amb aquest mòdul, hem entrevist tot un nou ventall de possibilitats a l'hora de dissenyar, implementar i gestionar projectes d'enginyeria del programari. De vosaltres depèn ara posar-les o no en pràctica.

Activitats

1. Descriviu tres projectes en els quals treballaríeu amb metodologies àgils. Trieu quina d'aquestes metodologies utilitzaríeu i comenteu en què fonamentaríeu aquesta decisió.

2. Dels tres projectes de l'activitat anterior, escolliu-ne un per a desenvolupar-lo mitjançant XP. Dissenyeu les històries d'usuari, la metàfora del negoci i planifiqueu les fases.

3. Creeu una metodologia pròpia que s'adapti al "Manifest àgil" i expliqueu com l'aplicaríeu en un altre dels projectes de la primera activitat.

Exercicis d'autoavaluació

1. Les metodologies àgils són sempre la millor opció per a la majoria de projectes?

2. Les metodologies clàssiques i les metodologies àgils són incompatibles?

3. Quan no hauríem d'aplicar una metodologia àgil?

4. De les dotze pràctiques d'XP, quines tenen relació amb una metodologia d'equip? I quines són els tres més importants?

5. En XP, en què consisteix l'*spike* arquitectònic de la fase d'exploració?

6. Quins són els quatre valors del "Manifest àgil"?

Solucionari

Exercicis d'autoavaluació

1. Les metodologies àgils no són la gran solució a tots els problemes del desenvolupament d'aplicacions, ni tan sols es poden aplicar a tots els projectes, però sí que ens aporten un altre punt de vista de com es poden arribar a fer les coses: de manera més ràpida, més adaptable i sense haver de perdre el rigor de les metodologies clàssiques.

2. Rotundament no. Es poden compaginar perfectament, fins i tot en un mateix projecte; de fet, hi ha múltiples estudis de com aplicar DSDM i XP amb RUP en un mateix projecte.

3. En els casos següents:

- Amb un equip molt gran i/o format majoritàriament per gent sense experiència.
- Si el client final no hi està involucrat i/o imposa barreres de comunicació.
- Si els requisits gairebé no són canviants.
- Si és un projecte crític.
- Si és un projecte massa gran.
- Si no us ve de gust provar una metodologia àgil.

4. Les pràctiques que tenen relació amb la part metodològica de treball en equip són: propietat col·lectiva del codi, programació en parelles, integració contínua, quaranta hores setmanals i metàfora del negoci.

Les tres pràctiques més importants de les dotze que hi ha són el disseny simple, el test i la refactorització. Fins i tot, si no volem adoptar la totalitat de les pràctiques d'XP, adoptant aquestes tres a la nostra metodologia habitual podem tenir una millora substancial en els resultats obtinguts.

5. En l'*spike* arquitectònic l'equip de desenvolupament:

- Prova la tecnologia.
- Es familiaritza amb la metodologia.
- Es familiaritza amb les possibilitats de l'arquitectura.
- Realitza un prototip que demostri que l'arquitectura és vàlida per al projecte.

6. Els quatre valors del "Manifest àgil" són els següents:

- Valorar més l'individu i les seves interaccions que no pas el procés i les eines.
- Valorar més desenvolupar programari que funciona que no pas obtenir una bona documentació.
- Valorar més la col·laboració amb el client que la negociació d'un contracte.
- Valorar més respondre als canvis que seguir una planificació.

Glossari

característica *f* Funcionalitat simple i poc costosa de desenvolupar que aporta valor al client del programari que s'ha de desenvolupar.
en **feature**

feature *f* Vegeu **característica**.

història d'usuari *f* Descripció curta de la usabilitat i funcionalitat que s'espera del sistema, escrita pel client final, en el seu llenguatge i sense tecnicismes.

Manifest àgil *m* Manifest que recull els valors que haurien de complir les metodologies de desenvolupament de programari. Va ser firmat per Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland i Dave Thomas.

metàfora del negoci *f* Història comuna compartida per l'usuari i l'equip de desenvolupament que descriu com s'han de comportar les diferents parts del sistema que es vol implementar.

metodologia àgil *f* Metodologia que compleix el manifest àgil.

refactorització *f* Modificació del codi per a deixar-lo en bon estat, tornant a escriure les parts que siguin necessàries però sempre des d'un punt de vista global de la funcionalitat independentment del canvi que fem.

***spike* arquitectònic** *m* Període durant el qual l'equip de desenvolupament prova la tecnologia i es familiaritza amb la metodologia que s'aplicarà durant el projecte.

test d'acceptació *m* Test creat conjuntament amb el client final que ha de reflectir les necessitats funcionals del primer.

test d'integritat *m* Test creat per l'equip de desenvolupament per a provar que tot el conjunt funciona correctament amb la nova modificació.

test unitari *m* Test creat pel programador per a veure que tots els mètodes de la classe funcionen correctament.

Bibliografia

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Boston: Addison-Wesley.

Beck, K.; Martin F. (2001). *Planning Extreme Programming*. Boston: Addison-Wesley.

Burke, E. M.; Coyner, B. M. (2003). *Java™ Extreme Programming Cookbook*. Sebastòpol: O'Reilly & Associates Inc.

Canós, J. M.; Letelier, P.; Penadés, M. C. (2003). *Metodologías ágiles en el desarrollo del software*. València: Universidad de Valencia.

Cockburn, A. (2001). *Agile Software Development*. Boston: Addison-Wesley.

Cronin, G. (2003). *eXtreme Solo: A Case Study in Single Developer eXtreme Programming*. Auckland: University of Auckland.

Highsmith, J. (2002). *Agile Software Development Ecosystems*. Boston: Addison-Wesley.

Hightower, R; Lesiecki, N. (2002). *Java Tools for Extreme Programming*. Nova York: Wiley & Sons Inc.

Reynoso, C. (2004). *Métodos heterodoxos en desarrollo del software*. Buenos Aires: Universidad de Buenos Aires.

Salo, O.; Abrahamson, P.; Ronkainen, J.; Warsta, J. (2002). *Agile Software Development Methods - Review And Analysis*. Universitat d'Oulu: VTT Publications.

Schwaber, K.; Beedle, M. (2002). *Agile Software Development with Scrum*. Nova Jersey: Prentice Hall.

Wake, W. C. (2000). *Extreme Programming Explored*. Boston: Addison-Wesley.

Wallace, D.; Raggett, I.; Aufgang, J. (2002). *Extreme Programming for Web Projects*. Boston: Addison Wesley.