Construcció dels sistemes d'informació

Marian Ortiz del Amo

PID_00187589

UOC
Universitat Oberta de Catalunya

Índex

Introducció

Hem treballat en la construcció de sistemes d'informació seguint l'explicació de com es construeix un pla de sistemes d'informació i com s'ha de gestionar un projecte TIC. Durant aquest mòdul aprofundirem en la idea de com es pot dur a terme l'anàlisi de processos i la construcció d'un sistema d'informació.

Tant si es produeix després d'una anàlisi de l'organització com durant la creació de l'estratègia corporativa, la reenginyeria de processos és el mecanisme més habitual per a obtenir la informació necessària per a operar un canvi en els processos i funcions de l'empresa. En aquest mòdul parlarem sobre els processos de reenginyeria i desenvoluparem de manera especial la primera fase, que consisteix en el **disseny de requisits de negoci**. La importància de les persones en la construcció d'un sistema és fonamental, molts teòrics obliden la forta influència de l'entorn en el disseny d'un sistema i això fa que les noves teories sobre el desenvolupament del programari cobrin cada vegada més importància.

La construcció de sistemes d'informació passa, en la majoria dels projectes, pel desenvolupament de programari com a element central. Per tant, explicarem en què consisteix aquest procés segons les metodologies clàssiques que encara estan en ús. A més, en aquest apartat reflectirem el debat que hi ha entre metodologies ortodoxes i heterodoxes, descrivint en què consisteixen les noves metodologies àgils i per què representen una revolució en el món del desenvolupament del programari.

S'ha separat l'explicació sobre presa de requisits de negoci de la de presa de requisits durant el procés de desenvolupament, perquè es produeixen en moments diferents i són dutes a terme per perfils diferents, i perquè volíem fer èmfasi especialment en l'aspecte estratègic dels primers enfront del caràcter més tècnic dels segons, encara que, com veurem, estan íntimament relacionats.

Objectius

En acabar la lectura del mòdul s'hauran assolit els objectius següents:

1. Saber en què consisteix la reenginyeria i la seva aplicació.

2. Fer una anàlisi de requisits de negoci de l'organització.

3. Adquirir la visió del procés de desenvolupament amb les fases tradicionals.

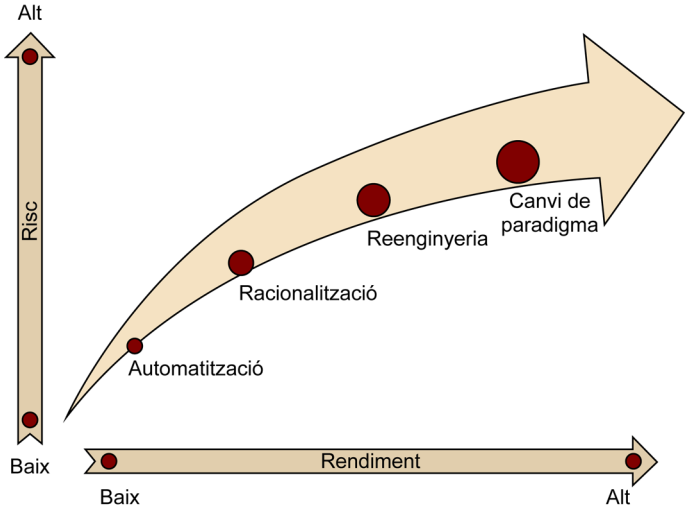
4. Conèixer els nous mètodes de desenvolupament àgils.

1. Reenginyeria de processos

1.1. Integració dels sistemes en l'organització

Seguint Laudon i Laudon (2006), les transformacions dels sistemes d'informació (SI) en l'organització passen per diferents fases que exemplifiquen en el gràfic següent:

Figura 9. Fases de canvi organitzacional.



Elaboració pròpia a partir de Kenneth C. Laudon; Jane P. Laudon (2006). *Sistemas de información gerencial* (8a. ed.). México DF: Prentice Hall.

1) Automatització

Es correspon amb la fase més primerenca de la digitalització, que podem situar al començament de la informàtica en les empreses. S'automatitzen algunes funcions bàsiques habitualment associades amb les nòmines o la comptabilitat. No s'utilitzen sistemes gaire sofisticats –per exemple, un full de càlcul– i, per descomptat, es donen solucions a problemes puntuals. En la literatura sempre trobem aquesta fase com una cosa ja superada, però la realitat és molt diferent. Les empreses petites i mitjanes, alguns departaments d'empreses grans, i fins i tot seccions de multinacionals poden estar en aquest estadi respecte a les funcions bàsiques. L'objectiu de l'automatització és estalviar costos directes sovint relacionats amb la manipulació de la informació; a més, com a objectiu secundari se sol aconseguir l'eliminació d'errors.

2) Racionalització

Es comença a tenir la preocupació de digitalitzar processos complets. Quan es comença a automatitzar s'observen punts febles en la cadena de funcions que formen un procés. Quan tenim digitalitzada la nòmina d'una empresa, obser-

vem que la informació sobre les variables mensuals (plusos, hores extres, viatges, absències, etc.) encara és un procés manual i és llavors quan es comença a gestar la racionalització del procés complet. I és que no té sentit automatitzar una funció si després hem de fer un treball manual considerable per generar les entrades de la funció que hem automatitzat perquè la resta del procés encara és manual. No passa gaire temps des de l'automatització fins a la racionalització i sol ser un procés obligatori. En empreses petites en què no es disposa d'un equip TI expert, habitualment es compren paquets (sistemes desenvolupats) bàsics la implantació dels quals, encara que no es faci de manera gaire professional, obliga a la racionalització.

3) Reenginyeria de processos

Requereix una visió de com s'organitza el procés complet. És més elaborada que la racionalització perquè implica un redisseny del procés. Els passos que es duen a terme en la reenginyeria són:

- Identificació dels processos i funcions (mapes de processos).
- Desenvolupament de la visió dels nous processos.
- Disseny dels nous processos.
- Test dels nous processos.
- Posada en marxa.

Descriurem amb més detall aquest concepte, que considerem un estadi òptim en l'aplicació dels sistemes d'informació a una empresa. Abans, però, expliquem la darrera fase del canvi organitzacional.

4) Canvi de paradigma

És un canvi radical de la visió del negoci, no s'ha de confondre amb grans implantacions de sistemes. Habitualment s'usa la reenginyeria perquè el més prudent és abordar parts específiques de l'empresa i portar-hi a terme un procés de reenginyeria. Un canvi de paradigma afectaria tota l'empresa i consistiria en una reformulació de processos, funcions, divisions, departaments, etc.

Mentre que la reenginyeria és el dia a dia en les empreses que tenen un grau tecnològic mitjà-alt, un canvi de paradigma és menys freqüent a causa dels costos i de l'envergadura. Amb tot, cal dir que el canvi de paradigma és molt utilitzat per les empreses que requereixen un canvi radical i que aposten per la innovació com a font d'avantatges competitius.

Tornem, doncs, a la **reenginyeria de processos**, que té aspectes en comú amb el canvi de paradigma perquè també s'aplica com a conseqüència d'una necessitat de canvi total. Una empresa amb un cicle (ja habitual) de millora contínua fa canvis i ajustos a mesura que sorgeixen els problemes o les noves ne-

cessitats, sovint provocats per canvis en el mercat, en la situació de l'empresa o per requisits dels clients. Ara bé, sovint cal replantejar-se tot el procés per moltíssims factors, entre els quals destaquem els següents:

- Processos antiquats que no reflecteixen totalment el flux de treball i que ocasionen problemes de rendiment (pèrdua de temps, incidències, etc.).

- Canvi de visió sobre un procés concret (per exemple, el procés de nòmina no es farà més en paper, ara s'enviaran les nòmines en format electrònic a cada empleat o hi ha un canvi en la relació amb els nostres proveïdors, clients, etc.).

- Canvis organitzatius (fusions entre departaments, àrees o seccions, creació d'un nou departament, eliminació d'un departament, etc.).

La tendència natural de molts enginyers de processos, experts IT (Information Technologies) o qualsevol persona implicada en processos de reenginyeria és intentar solucionar aquests problemes amb tecnologia. Des de l'experiència d'anys en implantació de sistemes d'informació, només cal dir el següent:

> Si tenim un procés mal definit i busquem un sistema amb la confiança que, en adaptar-nos-hi, el procés canviarà, incorrem en un error greu de disseny.

Sovint, molts directors d'àrees busquen en els sistemes una manera ràpida de solucionar els problemes en els seus àmbits d'actuació. Passa el mateix amb les persones; sovint, especialment en mitjanes i petites empreses, pensem els processos tenint en compte les persones i no és exactament el mateix que pensar en rols. Si parlem que Pere Martínez serà el responsable de la definició del producte en el procés *X*, pensem en Pere Martínez amb unes capacitats i coneixements determinats, i podem incórrer en l'error d'afegir a les seves funcions les que Pere fa de manera natural per les seves capacitats encara que no tinguin res a veure amb el rol que exerceix. Això passa més sovint del que pensem i s'evita fàcilment dissenyant els processos aïlladament de les persones que els duran a terme. En el cas anterior, definirem les funcions que un director de producte (*product manager*) hauria de tenir, independentment del que faci un director de producte en la situació actual i independentment del que facin les persones en l'actualitat. Després la realitat ens farà ajustar i assignar persones als rols definits i entraran en joc altres funcions de la gestió de projectes o de la reenginyeria que s'encarregaran d'adaptar els nous processos a l'organització.

Els processos s'han de definir gairebé amb un punt de vista científic, desproveït de relació amb l'organització. Els implicats en el procés solen tenir avantatges i desavantatges com a motors de canvi, ja que tenen tota l'experiència per a saber en què fallen els processos actuals i com es poden millorar, però

tenen massa prejudicis i barreres psicològiques per a visualitzar canvis totals del propi treball. En el redisseny, és important ser amb les persones directament implicades per a obtenir informació, però també cal que les persones que originin el canvi estiguin fora d'aquests processos.

1.2. Requisits d'organització

Hi ha un *gap* tant en la pràctica de l'aplicació dels sistemes d'informació als processos corporatius com en la literatura respecte al que són els requisits d'organització. Som en la fase en què s'ha decidit informatitzar un o més processos, s'ha dut a terme la gestió de projectes de l'organització i es produeixen diverses reunions per a definir amb més detall en què consistiran els nous sistemes.

La figura fonamental en la gestió de requisits de l'organització serà el que anomenem ***analista de requisits***, d'ara endavant AR (*requirements analyst*) i que ha de reunir capacitats diferents de les merament tecnològiques. Enumerem les més importants:

Taula 3.1. Perfil d'un analista de requisits.

Habilitats	Coneixements
Creativitat. Ha de trobar sempre les millors solucions. La innovació serà important no solament en el disseny de programari, sinó també en la cerca de solucions tècniques alternatives. El disseny de solucions se sol plantejar com a resposta a un problema on els camins alternatius solen oferir solucions més òptimes i més econòmiques.	**De negoci**. Ha de conèixer l'empresa, el mercat i la situació socioeconòmica del projecte. És una part important del cos de coneixements que un RA hauria de tenir, però es poden trobar processos de redisseny, per exemple, que es duen a terme sense tenir en el compte la situació anterior, la història i les característiques de l'empresa.
Empatia. Ha d'entendre perfectament les necessitats dels seus interlocutors de negoci. S'ha d'entendre el client final, el seu negoci, els seus objectius, la seva història, etc. Tradicionalment, la informàtica ha estat allunyada del negoci recolzant-se de vegades en excés en el desconeixement tècnic de gestors, enginyers, responsables d'organització, etc. S'hauria d'integrar totalment en les empreses, ja que la distància tècnica es comença a estrènyer cada vegada més a causa que la informàtica s'ha estès amb rapidesa a totes les esferes de la vida i el coneixement tècnic s'ha universalitzat.	**Tècnics.** Ha de conèixer quines possibilitats té, quin equip de desenvolupament durà a terme el projecte, les limitacions tecnològiques, les restriccions de tot tipus que el puguin afectar, etc.
Assertivitat. Són útils en moltes professions, però en sistemes el client final sol presentar un grau de desconeixement més gran que en la prestació d'altres serveis. L'RA ha de ser capaç de liderar i dirigir per transformar requisits en funcionalitat sense que sigui un cost per al sistema mateix. Tot això implica basar-se en els requisits del client, però tenint la decisió última en temes purament tècnics (arquitectura de programari, de sistemes, etc.).	**De procés**. L'AR ha de conèixer en profunditat el procés de creació del programari.
Bona disposició al canvi. L'AR produeix continus canvis en el treball de les persones en l'exercici de la seva professió i per això ha de ser especialment sensible davant els efectes d'aquests canvis. A més, la falta d'experiència dels seus clients o la falta de visibilitat que es té sobre projectes de sistemes faran que aquests mateixos clients canviïn el projecte en innombrables ocasions, característica que s'ha d'integrar de manera positiva, més com a implicació per part d'aquests en el projecte que com a indecisió o falta de criteri.	

Font: elaboració pròpia.

Habilitats	Coneixements
Objectivitat. De vegades els enginyers de requisits posen barreres personals davant determinades solucions, potser per desconeixement, perquè ofereixen dificultats "extremes" o per prejudicis. Cal un exercici d'objectivitat per a donar la millor solució independentment de les dificultats per a dur-la a terme. Un exemple típic és considerar programari de sistemes o programari de desenvolupament que l'AR coneix com a úniques eines i menysprear un altre programari que pot ser més adequat per al projecte, encara que implicaria dificultats perquè requereix que l'AR actualitzi els seus coneixements.	

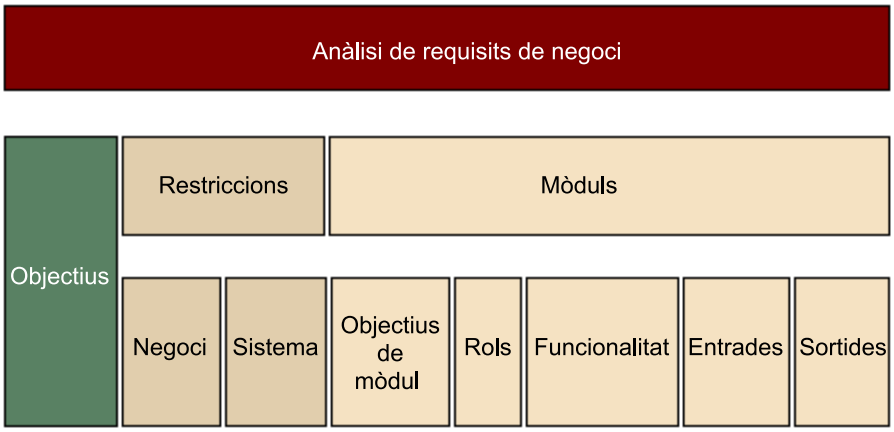
Font: elaboració pròpia.

El perfil és gairebé impossible de trobar a causa de la segmentació tradicional entre tècnics i persones de negoci, però aquest perfil reflecteix la necessitat d'establir un vincle entre tots dos mons. Les habilitats són difícils d'adquirir encara que es poden treballar. Respecte als coneixements, ha estat una falta tradicional dels enginyers allunyar-se del negoci, l'obsessió per viure només en el món de la tecnologia sense fusionar-ne els coneixements amb els dels clients, que en el món corporatiu són els directors i personal clau d'àrees dins de l'empresa.

La millor eina per a l'anàlisi de requisits de negoci són les entrevistes, però no és l'única, ja que no en tenim prou amb els coneixements del client (a partir d'ara parlarem sempre de client tenint en compte que client és qualsevol persona que dirigeix un departament, àrea o secció o un usuari clau que es converteix en el sol·licitant d'un sistema) per a establir quins seran els requisits d'un sistema.

Els nostres lliurables en la fase d'anàlisi de requisits de negoci contindrà els elements que es mostren en el diagrama següent.

Figura 10. Diagrama d'elements que formen els requisits de negoci.



Font: elaboració pròpia.

La gran dificultat és que aquesta informació d'estructura tan clara l'hem d'obtenir de persones que potser tenen una experiència en sistemes d'informació bastant reduïda i amb un discurs més centrat en les pròpies necessitats que en les del sistema. Hi ha moltíssimes excepcions, tantes com per-

sones. Podem tenir personal expert que ens sàpiga transmetre exactament les necessitats en un format "comprensible" perquè es puguin convertir en requisits de negoci. L'ofici de l'AR sempre consistirà a "separar el gra de la palla", a aprendre a distingir, dins d'un discurs generalment caòtic, els elements que encaixen amb el diagrama de dades que requereixen els requisits de negoci. A continuació descrivim en què consisteixen aquests requisits.

1) Objectius principals

És el més important dels requisits de negoci, responen a la pregunta "per a què vull un sistema d'informació?". Sovint la resposta és la presa de decisions o el coneixement d'un aspecte del negoci, que es transforma en un sistema d'informes. Tenir clar el comportament d'aquests grans *outputs* (resultats) ens ajudarà a tenir una visió global sobre el sistema. És important sempre tenir en compte que la persona que transmet les necessitats moltes vegades anteposa les pròpies a les del negoci. L'AR ha de saber separar els objectius principals d'altres, sense treure'n importància perquè seran més rellevants en àmbits petits (seccions, persones). Ara bé, l'interès de l'organització ha de prevaler perquè un sistema ha de beneficiar major nombre possible de persones.

Un risc habitual és perdre de vista els objectius principals, especialment quan el projecte és molt gran o complex. Els mateixos líders del projecte d'implantació canviaran els objectius sense gairebé adonar-se'n per diversos motius. Tenir els objectius principals originals a mà és una bona taula de salvació i un mecanisme bastant aclaridor quan es perd el significat d'un projecte. Quan aquests objectius es mantenen, només amb petites variacions, es demostra que la salut del projecte està intacta.

2) Restriccions i/o macrorequisits

Les restriccions generals són molt importants, i també és molt important qüestionar-les perquè de vegades es fonamenten en principis falsos. Una restricció pot ser el temps d'implantació ("s'ha de tenir llest un sistema d'informació en un any"), es pot referir a l'equip de desenvolupament ("l'equip de desenvolupament serà intern per a minimitzar costos") o, fins i tot, es podria referir a tecnologies. Recentment, hem tingut l'experiència de participar en un gran projecte d'una multinacional el requisit de la qual és que "qualsevol sistema d'informació ha d'estar dins de l'ERP", en aquest cas SAP. L'exemple anterior és un requisit que implica aprofundir en les possibilitats d'aquest ERP per a poder implementar nous mòduls, la investigació de l'ERP per a comprovar si s'hi adapta o no, etc. També es pot intentar influir per a canviar aquest tipus de restriccions que probablement tenen l'origen en decisions exemptes de coneixements tecnològics. Un macrorequisit és un requisit general de tot el sistema com, per exemple, "el sistema serà multiplataforma, multillenguatge i *multisite*" en el cas de treballar per a una multinacional amb diferents filials a tot el món. També hi ha múltiples requisits de tipus legal o de qualitat, per exemple: "el sistema ha de seguir la Llei de protecció de dades espanyola" o

"el sistema ha de seguir la política de seguretat de l'empresa". Els requisits més comuns són de negoci (normativa de la mateixa empresa) o de sistema (el sistema ha de tenir unes característiques determinades com en l'exemple del sistema *multisite* que hem comentat).

3) Rols

La informació sobre rols és vital per a poder conèixer tots els angles o punts de vista sobre un procés i per a saber-ne el grau de complexitat. La diversitat de rols té l'origen en el tipus d'organització (cultura d'empresa), és a dir, si cal identificar clarament la responsabilitat sobre cada acció, i també és important la grandària de l'organització. És clar que en organitzacions petites les persones tenen molts rols i aquests es redueixen o es fusionen.

4) Mòduls

Abans d'arribar a cap conclusió sobre el sistema que s'analitza, hem d'optar per l'estratègia de separar-lo en problemes diferents, no afrontar-lo des del principi com un tot, sinó dividir-lo per grups de funcionalitat, per rols, per funcionalitat associada a diferents seccions, etc. Hi ha moltes maneres de segmentar o modularitzar, però totes estaran clares quan sapiguem definir amb claredat els objectius generals. En projectes molt petits podem ser unimodulars. És sempre una recomanació iterar per mòduls, però per descomptat no necessàriament ha de passar així.

Vegeu també

Pel que fa a la recomanació d'iterar, vegeu l'apartat "Models heterodoxos del desenvolupament. Models àgils".

En els mòduls és on trobarem el detall dels requisits de negoci:

- ***Outputs* (resultats)**. Hem de definir els resultats que els diferents actors volen aconseguir. Per a dur a terme aquesta aproximació, cal identificar els rols que intervindran en cada mòdul, la feina que s'ha fet durant la definició de rols en paral·lel amb la identificació de mòduls. En la implantació d'un CRM, un mòdul pot ser la generació de mestres d'informació (per exemple, alta de clients i proveïdors). Haurem de saber amb exactitud quina anàlisi es farà d'aquesta informació. Si al final necessitem tenir un informe que analitzi la situació dels nostres clients respecte a les seves línies de crèdit (morositat), haurem d'incloure indicadors per saber-ne la situació i s'haurà de tenir en compte des del moment de la presa de requisits. Els resultats en el projecte o en el mòdul són el més crític, perquè són la brúixola del projecte, ens marquen exactament on volem anar.

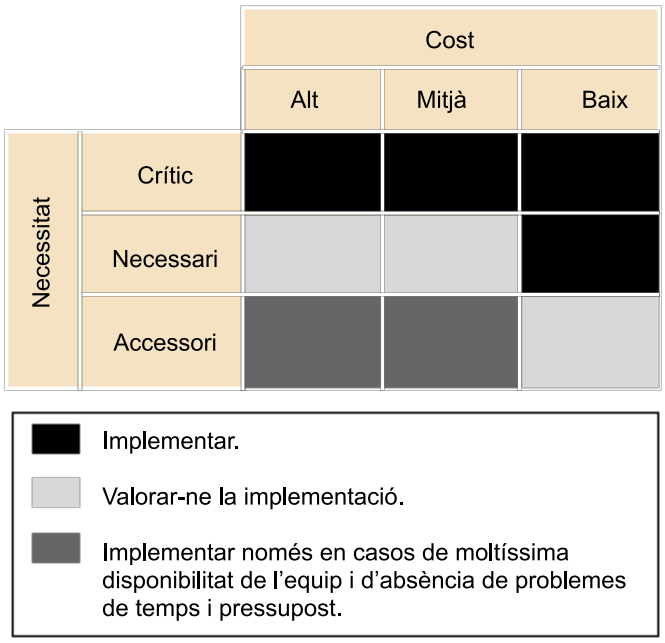
- **Funcions**. Per a cada mòdul hem de definir quina funcionalitat es necessita. La funcionalitat sempre s'expressa usant expressions com "volem imprimir informes des de cada lloc del sistema" o "necessitem generar notificacions de com reaccionen determinats esdeveniments del sistema". Se solen repetir perquè les accions usuari-sistema són limitades i perquè el cli-

ent no sol demanar les accions que no impliquen interacció amb l'usuari, sinó que això és un requisit tècnic, que pertany a l'etapa de disseny.

- ***Inputs***. Estretament relacionats amb els *outputs*, però el client rares vegades els esmenta. Depenent de la capacitat tècnica de la persona que ens transmeti els requisits de negoci, trobarem més o menys capacitat per a definir el que el sistema necessita per a ser complet. La informació que hem d'introduir en el sistema, la que el sistema futur espera, és de vital importància per a la seva construcció i cal afegir que els canvis en els *inputs* a la meitat de projecte, com també els de funcionalitat, poden posar en perill l'èxit del sistema. Per tant, en aquesta fase del projecte és important obtenir el màxim d'informació per a evitar, en la mesura que sigui possible, les sorpreses d'una falta de definició.

- **Prioritat**. Finalment, hem de ser conscients de la prioritat que té cada requisit o petició. No és fàcil aconseguir-ho perquè generalment l'AR no té la informació necessària per a distingir fàcilment si alguna cosa transmesa amb una importància crítica no ho és i viceversa. En aquest sentit, els requisits poden ser **crítics** (sense aquests, el sistema perd la raó de ser), **necessaris** (són importants per a reflectir el veritable flux de treball; se'n pot prescindir, però aquesta absència amenaça la qualitat del sistema) i **accessoris** (*nice to have*¸ és un aspecte que milloraria molt el sistema, però es considera prescindible). Ara bé, la necessitat o no d'un requisit no és l'única variable per a decidir si s'acceptarà o no; també hi intervé una altra variable que és el cost. La matriu següent ens dóna una idea del que convé acceptar o no unint totes dues variables.

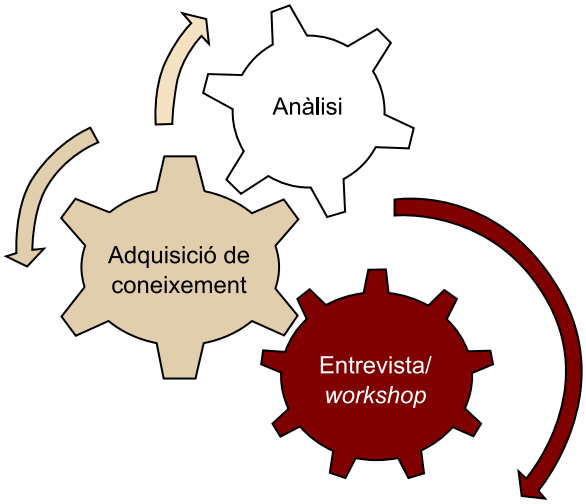
Taula 3.2. Taula de prioritització de requisits.



Font: elaboració pròpia.

Respecte a la manera en què es durà a terme la presa de requisits, és tan variada com les formes d'interacció, encara que el model que proposem és el d'**entrevistes iteratives**.

Figura 11. Mecanismes d'adquisició de requisits de negoci.



Font: elaboració pròpia.

En grans sistemes o projectes multisistema s'utilitza el ***workshop*** com a eina de contacte intensiu amb els diferents actors. Un *workshop* té com a avantatge que es duu a terme durant un temps més llarg i inclou gairebé tots els col·lectius implicats. Els desavantatges rauen en el fet que hem d'invertir molt més temps preparant-los (no sempre s'obté un retorn d'aquests temps de preparació) i que necessiten una planificació molt més formal que les entrevistes. L'elecció d'un mecanisme o un altre o la combinació de tots dos dependrà de la situació; per exemple, en projectes internacionals se sol utilitzar el *workshop* perquè així es viatja menys i s'estalvien costos innecessaris. Aquests tallers solen ser reunions intensives de cinc o sis hores al dia, mai més de quatre dies. La direcció o lideratge d'un *workshop* depèn de l'eficiència de l'AR responsable, però bona part de l'èxit es deu a la preparació prèvia dels continguts, a una agenda equilibrada i a la participació de tots els actors i experts en el procés o conjunt de processos que s'han de debatre.

En tots dos casos tindrà lloc el contacte amb les persones que tenen la informació (experts, client, sol·licitant, *key users*, etc.), amb una fase d'adquisició de coneixements i una altra d'anàlisi per a tornar insistir en el mateix de cicle d'entrevistes-investigació-anàlisi. D'aquesta manera, les entrevistes o tallers cada vegada parteixen d'un nivell de coneixements molt més alt, les preguntes són més especialitzades i les respostes molt més elaborades.

La fase de documentació o d'adquisició de coneixements és imprescindible en la presa de requisits. Si la nostra missió és treballar com a AR en un projecte que buscarà una solució de portal corporatiu per a una empresa d'assegurances, haurem de conèixer el negoci abans de la primera entrevista. Això s'aconseguirà amb documentació externa, Internet, altres empreses, saber fer (*know-how*) propi, etc. Quan parlem amb algú que no és expert en un

tema del qual nosaltres sí que som experts, la nostra tendència general és simplificar per a afavorir la comunicació. Això és el que passa en un *workshop* o entrevista amb experts en un tema si l'AR no ha adquirit prèviament un nivell alt de coneixements.

Finalment, l'anàlisi sobre la informació adquirida per mitjà dels integrants del projecte (entrevistes/*workshops*) o de manera externa (adquisició de coneixements) ens situarà en un nivell de coneixement del problema molt superior que ens permetrà aprofundir cada vegada més sobre temes relacionats directament o indirectament amb el problema que seran crucials per a analitzar els requisits. Durant les fases d'anàlisi no s'han de perdre de vista les tres perspectives temporals de l'obtenció de requisits:

- La **situació actual**. Aquesta perspectiva ens donarà informació sobre com funciona ara un procés.

- L'**experiència**, tant la tant de l'AR com la d'altres empreses que hagin passat pel mateix procés, ens donarà informació sobre què han fet altres en la nostra situació.

- La perspectiva de **futur**, que finalment haurem de contrastar amb les dues perspectives anteriors, parlarà sobre com volen les persones implicades que siguin els processos.

2. Models i etapes del desenvolupament tradicional

Un dels models de desenvolupament més seguits, més criticats i del qual s'han fet més versions és el model de desenvolupament en cascada de Benington (1956). Serà el model que explicarem com a punt de partida per a les variants que van sorgir després i que van refinar el model en cascada reinventant-lo, però que continuen en l'ortodòxia d'aquest mètode.

El model en cascada defineix unes fases seqüencials que avui dia continuen essent l'estàndard per excel·lència, encara que com veurem no està exempt de problemes. Les fases d'aquest model són les següents:

2.1. Requisits del sistema

S'assumeix que el sistema que s'està creant no estarà només en l'organització, sinó que conviurà amb altres sistemes. Els requisits del sistema estudiaran la integració amb altres sistemes i analitzaran quins requisits tindrà el sistema en el maquinari, les comunicacions, el programari, etc.

2.2. Anàlisi de requisits

Hem volgut parlar de presa de requisits de negoci a fi d'il·lustrar que aquesta fase d'anàlisi de requisits forma part del desenvolupament i que els requisits de què hem parlat en el punt anterior pertanyien al projecte. Podria passar que una vegada que tinguem descrits els requisits de negoci, no hi hagi la necessitat de dur a terme un desenvolupament. En aquest cas, les necessitats empresarials, en endavant BR (*business requirements*), no es transformaran en necessitats de programari, en endavant SR (*software requirements*). Per als projectes que deriven en un desenvolupament, sempre hi haurà una forta connexió entre els BR i els SR, però els hem volgut diferenciar perquè la manera en què es recullen els uns i els altres i, especialment, el lliurable que resulta de totes dues anàlisis és diferent. En aquesta fase dins el desenvolupament de programari, el lliurable se sol anomenar SRS o SRD (*software requirements specification* o *document*). El contingut d'un SRS pot ser molt variat, però proposem el model següent:

a) Introducció

- Concepte (descripció de l'objecte del document)
- Abast (del document)
- Equip (equip implicat en el desenvolupament)
- Glossari

b) Producte

- Descripció general
- Funcionalitat general
- Usuaris i rols
- Restriccions (tècniques)
- Dependències (possiblement d'altres sistemes)

c) Requisits

- Interfícies (d'usuari, de maquinari, comunicació)
- Requisits funcionals
- Requisits no funcionals (rendiment, seguretat, fiabilitat, disponibilitat, mantenibilitat i portabilitat)

d) Diccionari de dades

- Descripció de les dades utilitzades
- Model relacional proposat

e) Diagrama de classes

Com es pot veure, és una descripció molt tècnica del programari i molt detallada. En el subapartat "Requisits funcionals" es descriuran un a un tots els requisits. Una manera clàssica de representar els requisits són els casos d'ús. Un exemple de cas d'ús podria ser el següent.

Taula 3.3. Exemple de cas d'ús.

Codi	UC01-002
Nom	Validació en el sistema.
Rol	Tots els usuaris.
Descripció	Els usuaris necessitaran autenticar-se per a accedir al sistema. Aquest cas defineix com serà la validació.
Seqüència normal	1. L'usuari escriu un usuari. 2. L'usuari escriu la contrasenya. 3. El sistema valida l'existència de l'usuari. 4. El sistema comprova que l'usuari no està bloquejat (superació de 3 intents amb clau incorrecta). 5. El sistema valida la contrasenya respecte de l'usuari. 6. El sistema mostra la pantalla inicial.
Excepcions	3a. El camp usuari és buit (excepció ocorreguda en el pas 3). 3a1. El sistema alerta l'usuari de l'error. 3a2. El sistema torna al pas **1**. 3b. L'usuari no està donat d'alta en la BD. 3b1. El sistema alerta l'usuari de l'error. 3b2. El sistema torna al pas **1**. 5a. La contrasenya no és correcta. 5a1. El sistema alerta l'usuari de l'error. 5a2. El sistema torna al pas **1** (només estan permesos 3 intents).
Rendiment	La seqüència normal no pot excedir 0,02 segons.

Font: elaboració pròpia.

Freqüència	Es preveu que aquest UC es repetirà 1 vegada per execució.
Importància	Crític - important - *nice to have*
UC relacionats	UC01-001, UC04-001, UC04-002
Inputs	Usuari, contrasenya, *ok*
Outputs	Confirmació de validació o errors descrits en les excepcions.

Font: elaboració pròpia.

Amb aquestes instruccions completes, un desenvolupador pot implementar directament el codi. Els casos relacionats ens donaran una visió diagramàtica dels casos d'ús, ja que és important tenir visió relacional. En realitat, en el mercat hi ha sistemes que ens permeten controlar i organitzar més bé els casos d'ús d'un sistema, però molts equips de desenvolupament escriuen els seus casos en un processador de textos.

2.3. Disseny

El disseny fa referència a temes molt diferents, però tots formen part del disseny final del sistema. La documentació de disseny (SDS, *software design specification*) també pot variar d'un equip a un altre, però hauria de contenir una especificació de l'**arquitectura del programari**. El sistema estarà organitzat en capes, probablement seguint el model clàssic de tres capes (presentació, negoci, dades), però a més serà modular amb mòduls que agrupen una funcionalitat comuna. La descripció exhaustiva de com es comportaran aquestes capes, com s'agrupa la funcionalitat i per què, a més de com es relacionen uns mòduls amb altres i com es comuniquen les capes, constitueix la descripció de l'arquitectura del sistema. A més, el disseny podria incloure el **disseny d'interfície d'usuari**, consistent en un prototipatge complet de les pantalles que els usuaris finalment faran servir, i també el prototipatge de sistemes que consisteix en la descripció de com el nostre sistema resultant interaccionarà amb altres sistemes que ja hi ha. Això és l'**arquitectura del sistema**.

2.4. Codificació

Amb els documents de disseny i de requisits tècnics, l'equip de desenvolupadors ja pot **implementar** el que defineixen aquests documents. L'elecció de la plataforma de desenvolupament o d'altres dades tècniques com la base de dades, etc. marcarà la definició del mateix equip. Durant la codificació es produeixen els primers tests, guiats pel mateix equip de desenvolupament, però és important tenir en compte que els veritables tests els duran a terme per mitjà de *testers* en la fase següent del desenvolupament.

2.5. Prova

Aquesta fase marcarà la qualitat del programari, però l'experiència diu que la maduresa en les fases anteriors produeix un programari de més qualitat que exigeix menys atenció per part dels *testers*. Hi ha una varietat gairebé infinita de tests que podem dur a terme, però tot seguit destacarem els més importants.

Taula 3.4. Test en desenvolupament de programari.

Test d'integració	La fase de prototipatge de sistemes resultant del procés d'arquitectura de sistemes produirà un prototip d'interacció entre sistemes que es provarà durant la fase de disseny, però també en la fase de test executarem aquests tests d'integració per a comprovar que els sistemes interaccionen correctament. Per aquest motiu generarem els fitxers d'intercanvi amb les dades de test o executarem els mecanismes d'interacció (serveis web o *webservices*, etc.).
Test de rendiment	El creixement del maquinari ha fet que aquestes proves només es vegin com a necessàries en sistemes en què la disponibilitat és crítica o en què la rapidesa de processament està mil·limetrada (p. ex.: sistemes de producció). Són proves que, usant mecanismes definits expressament per al que es vulgui provar, asseguraran que la nostra aplicació tindrà un rendiment òptim una vegada que estigui en funcionament. Per exemple, la saturació de la base de dades per a assegurar l'escalabilitat d'una aplicació seria un mecanisme de test de rendiment.
Test d'interfície	Els tests d'interfície asseguren coses bàsiques en les interfícies d'usuari. Per exemple, que l'ortografia dels textos és correcta, que el focus en els desplaçaments es comporta de manera adequada, l'alineació dels controls en pantalla, que totes les icones i elements gràfics guarden coherència amb l'estil principal o que les ajudes contextuals estan presents i són correctes.
Test de funcionalitat	Prenent com a referència l'especificació de casos d'ús o requisits funcionals, els tests de funcionalitat s'usen per a assegurar que tot el que s'havia dissenyat s'ha implementat. En grans sistemes hi pot haver omissió de funcionalitat a causa de la gran quantitat de casos d'ús que s'han d'implementar. És una manera de donar coherència al sistema, d'assegurar que el que es va pensar és el que s'ha fet.
Test d'usabilitat	Proves de nou encuny que prenen cada vegada més importància en sistemes en què l'usuari té un protagonisme essencial. Els tests d'usabilitat es fan normalment en laboratoris d'usabilitat on s'observa el comportament de l'usuari durant l'experiència del programari. Hi ha mecanismes de disseny d'aquestes proves, com també de documentació i execució. El procés és més laboriós que en la resta de tests perquè intervenen les persones (usuaris), però garanteixen l'èxit de l'aplicació. A més dels resultats, el lliurable d'aquests tests serà un document de modificacions sobre les interfícies d'usuari com a resultat dels experiments amb els usuaris.

Font: elaboració pròpia.

El disseny de proves és un procés molt important que s'ha de definir durant les primeres fases del desenvolupament.

2.6. Manteniment

El manteniment d'un sistema ha d'estar definit en la gestió del mateix projecte, és a dir, molt abans d'arribar a aquesta fase. Aquesta fase fa referència a la definició, per part de l'equip de desenvolupament, de mecanismes de manteniment de l'aplicació. És important tenir en compte que el nombre d'hores dedicades al manteniment de l'aplicació superarà amb escreix les hores dedicades al seu desenvolupament. El manteniment serà molt més costós (afectarà el disseny) com més malament s'hagin definit les primeres fases de desenvolupament. La generació de versions sobre el mateix sistema pot resultar positiu quan el sistema ha estat dissenyat per a créixer sense alterar-ne la qualitat. Quan això no passa, tenim un sistema "apedaçat" que sempre serà de qualitat fràgil. Per aquesta raó són tan importants les fases de requisits i de disseny.

És clar que aquest sistema no està exempt de problemes. Les fases seqüencials tan ordenades exigeixen un model monolític en què sovint és difícil anar cap enrere (*roll back*) o corregir defectes en algunes de les fases. A més, exigeix una definició tècnica completa quan l'usuari (client o patrocinador) encara no ha vist o no sap en què es convertiran tants requisits en paper. Per tant, ofereix molta opacitat per al client que, per norma general, manca dels coneixements tècnics necessaris per a imaginar el sistema tal com serà només recorrent a la desmanegada documentació que es genera en aquest model en cascada.

Hi ha hagut moltíssimes variacions sobre aquest model i aquí hem volgut explicar els fonaments d'aquest model amb algunes de les característiques d'aquestes variacions (prototipatge) perquè entenem que resumeix força bé com són les fases del desenvolupament per al model clàssic, encara en vigor i massivament seguit per moltíssims equips de desenvolupament.

3. Models heterodoxos del desenvolupament. Models àgils

El model en cascada no sols explica tot el que ha passat en els últims cinquanta anys en desenvolupament del programari, sinó que també ens ajuda a entendre com funciona el desenvolupament perquè, per a ser equilibrats, el model va establir una manera de pensar que no és del tot absent en els mètodes més revolucionaris. Ens referim a la definició de les fases, conceptes que encara s'usen en els equips més heterodoxos. No obstant això, al principi dels noranta va passar el que Carlos Reynoso ha anomenat *la revolució dins del desenvolupament del programari*.

Lectura recomanada

Carlos Reynoso (2004). *Métodos heterodoxos en el desarrollo de software*. Buenos Aires: Universidad de Buenos Aires.

El model en cascada havia derivat en una mena de models diferents que bàsicament incloïen iteracions per a facilitar el *roll back*, prototipat per a apropar-se més a la visió del projecte i altres mecanismes que corregien els seus problemes. L'estàndard CMMI (*capability maturity model integration*), creat per al Ministeri de Defensa dels EUA pel SEI (Software Engineering Institute), el màxim exponent del model tradicional de desenvolupament, presenta una documentació excessiva, una relació amb el client rígida basada en un entorn contractual i, segons Reynoso, hereta tots els problemes del model anterior.

Mentre passava això, es produïa l'espontani moviment del desenvolupament col·laboratiu (*open source*) els fonaments del qual escapen de l'organitzat i estricte model tradicional; a més, moltes veus crítiques s'havien alçat contra els vells models amb ànim de construir models completament diferents. Més o menys sobre l'any 1990 comencen les crítiques al sistema tradicional amb diferents iniciatives de modelatge completament innovadores, i el 2001 disset persones s'uneixen en una reunió informal a les muntanyes de Utah, en la qual obtenen l'Agile Manifesto. La portada del manifest diu el següent:

Manifest pel desenvolupament àgil del programari

Estem descobrint maneres millors de desenvolupar programari tant per la nostra pròpia experiència com ajudant tercers. Gràcies a aquest treball hem après a valorar:

- **Individus i interaccions** sobre processos i eines
- **Programari que funciona** sobre documentació extensiva
- **Col·laboració amb el client** sobre negociació contractual
- **Resposta davant el canvi** sobre seguir un pla

Això és, encara que valorem els elements de la dreta, valorem més els de l'esquerra.

A hores d'ara s'han documentat molts fracassos dels models anteriors al manifest àgil, també podem disposar de documentació sobre grans èxits, però indubtablement s'ha demostrat que el sistema anterior comporta problemes seriosos. Hem parlat de la relació contractual i estricta amb el client que proposa sistemes com CMMI, però no cal oblidar que la resposta davant el canvi s'erigeix com el factor més important per a dissentir dels mètodes anteriors. En

la metodologia tradicional, la reacció al canvi implicava repetir moltes parts del procés, i els canvis representaven més un trauma per a desenvolupadors i analistes que un repte. Vegem quins són els dotze principis que se signen i s'acorden durant aquesta reunió:

1) La nostra prioritat principal és satisfer el client mitjançant el **lliurament primerenc i continu de programari amb valor**.

2) Acceptem que els requisits canviïn, fins i tot en etapes tardanes del desenvolupament. Els processos àgils aprofiten el canvi per a proporcionar avantatge competitiu al client.

3) Lliurem programari funcional sovint, entre dues setmanes i dos mesos, amb preferència en el període de temps més curt possible.

4) Els responsables de negoci i els desenvolupadors treballem junts de manera quotidiana durant tot el projecte.

5) Els projectes es desenvolupen entorn d'**individus motivats**. Cal donar-los l'entorn i el suport que necessiten, i confiar-los l'execució del treball.

6) El mètode més eficient i efectiu de comunicar informació a l'equip de desenvolupament i entre els seus membres **és la conversa cara acara**.

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

8) Els processos àgils promouen el **desenvolupament sostenible**. Els promotors, desenvolupadors i usuaris hem de ser capaços de mantenir un ritme constant de manera indefinida.

9) L'atenció contínua a l'**excel·lència tècnica i al bon disseny** millora l'agilitat.

10) La **simplicitat**, o l'art de maximitzar la quantitat de treball que no s'ha fet, és essencial.

11) Les millors arquitectures, requisits i dissenys emergeixen d'**equips autoorganitzats**.

12) A intervals regulars l'equip reflexiona sobre com pot ser més efectiu per, a continuació, ajustar i perfeccionar-ne el comportament en conseqüència.

El principi més innovador perquè resulta provocador respecte al model anterior és el 2, en què conclouen que no sols els canvis no resulten traumàtics sinó que són bons per al projecte. Hi ha moltes metodologies que subscriuen aquest manifest, encara que només aprofundirem en la més utilitzada. Entre

les metodologies que s'adhereixen al manifest hi ha Scrum, XP (*extreme programming*), DSDM (*dynamic system development method*), Crystal, ASD (*adaptative software development*), *pragmatic programming*, FDD (*feature driven development*). Hi ha més metodologies àgils que no estaven presents en la creació del manifest com són Evo, Agile Modeling, Lean Development o RUP. A continuació explicarem com funciona la metodologia àgil que té més adeptes: Scrum (dada de 2011 de l'enquesta feta per VersionOne).

3.1. Scrum

La paraula *scrum* defineix un comportament típic del rugbi en què l'equip avança passant-se la pilota de jugador a jugador. Segons aquest símil, aquesta metodologia és adaptativa, àgil i disposa de l'autoorganització com a element fonamental d'estructura de l'equip. El punt de partida era derrocar la predictibilitat com a pedra de toc de les antigues metodologies; quan un projecte comença, CMMI compta que les fases se succeiran de manera predictible i repetitiva. Les metodologies àgils parteixen de la base que la predictibilitat en desenvolupament del programari no existeix, per això un principi fonamental és l'acceptació del canvi com una cosa intrínseca a aquest tipus de projectes. Podem trobar la definició completa d'aquest nou mètode en l'article de Takeuchi i Nonaka.

Referència bibliogràfica

H. Takeuchi; I. Nonaka (1986). "The new new product development game". *Hardvard Business Review*.

Els principis o trets fonamentals de l'*scrum* són els següents:

- Els equips autodirigits, sense cap, són autoorganitzables. No tenen jerarquies, només tenen l'*scrum master*, que els ajuda a salvar impediments, però no té autoritat. Són "porcs" enfront de les "gallines" (observadors) amb relació a una rondalla que es pot resumir així: en els ous amb cansalada la gallina s'involucra i el porc es compromet.
- No s'afegeix cap tasca extra sobre una tasca triada.
- Hi ha trobades diàries amb les tres preguntes bàsiques: què heu fet des de l'última trobada?, quins obstacles hi ha per a assolir la meta?, què fareu per a la propera trobada?
- Iteracions de trenta dies (o més freqüents si és possible) que es diuen **esprint**, amb demostracions als participants del projecte.
- Al principi de cada iteració hi haurà un plantejament adaptatiu amb el client, és a dir, s'enfoca la iteració amb els canvis sorgits del treball anterior, no se segueix el pla inicial sense més, sinó que el projecte es va adaptant a les noves necessitats sorgides del client i del projecte en si mateix.

L'*scrum* defineix bàsicament cinc rols:

- ***Scrum master***. És una cosa semblant al director de projecte la funció del qual serà protegir l'equip de desenvolupament evitant obstacles que impedeixin assolir les metes en les diferents iteracions.

- ***Product owner***. És triat per l'*scrum master* i pel client, s'encarrega de la presa de decisions final sobre l'abast i el progrés. És el responsable oficial del projecte.

- ***Team***. L'equip de desenvolupament la característica principal del qual és l'autonomia i la capacitat autoorganitzativa. Es reorganitza a si mateix si ho necessita.

- **Client**. Que en les metodologies àgils adquireix una participació molt més directa en els projectes.

- ***Management***. Pren les decisions més importants, treballa al costat de l'*scrum master*.

Es recomana que cada equip *scrum* no estigui format per més de deu membres; per tant, si el nostre projecte supera aquesta xifra s'han de generar equips *scrum* per a aconseguir que l'equip assoleixi aquesta xifra. Més persones amenaçarien la capacitat autoorganitzativa.

El cicle de vida *scrum* el descrivim a continuació.

1) Planificació

Són les etapes prèvies al "joc", en què es descriu la visió, les expectatives sobre el producte final i en què s'assegurarà el finançament. La majoria de crítiques als sistemes àgils vénen per aquesta fase, el sistema *scrum* no disposa d'una predicció exacta de l'abast, ja que accepta el canvi com a part del joc; per tant, o assumim aquest joc i les desviacions que pugui comportar o haurem de girar cap a les maneres més tradicionals que emmalalteixen d'altres faltes. De la fase de planificació obtindrem un document important en *scrum*, el ***product backlog***, que defineix el que cal fer, la relació de requisits d'alt nivell (de negoci) amb la seva importància i criticitat dins el projecte. També es definiran l'arquitectura i el disseny, a més dels prototips. Dins de la planificació també hi haurà la definició d'iteracions. En aquesta subfase hi haurà més requisits perquè exigeix un aprofundiment dels requisits d'alt nivell detectats en el *product backlog*.

Un exemple del *product backlog* directament obtingut del text de Sutherland i Schwaber (2011) és el que exposem a continuació.

Referència bibliogràfica

J. Sutherland, K. Schwaber (2011). *The scrum papers: nut, bolts, and origins of an agile framework*. En línia a: http://jeffsutherland.com/ScrumPapers.pdf.

Taula 3.5. Product backlog.

							Iterations					
Item area	**Item**	**Details**	**Source**	**Priority**	**Estimate of value**	**Initial estimate of effort**	**1**	**2**	**3**	**4**	**5**	**6**
Customer requirements	Com a usuaris, volem posar un llibre en un carretó.	See wiki […]	Customer	1	6	9	x					
Customer requirements	Com a usuaris, volem eliminar un llibre del carretó.	See wiki [...]	Customer	2	6	9						
Systems	Millorar el rendiment del procés de transacció.		Team	5	7	15						
Systems	Actualitzar els servidors a Apache 2.2.		Team	6	8	15						
Customer requirements	Com a socis, volem crear una "llista de desitjos".	See wiki […]	Màrqueting	4	9	8		x				
Customer requirements	Com a socis, volem afegir o eliminar ítems de la llista de desitjos.	See wiki […]	Marketing	2	9	7						

Font: elaboració pròpia a partir de J. Sutherland; K. Schwaber (2011). *The scrum papers: nut, bolts, and origins of an agile framework*. En línia a: http://jeffsutherland.com/ScrumPapers.pdf.

La persona encarregada d'estimar la prioritat és l'*scrum master*. Convé que aquesta tasca la faci una sola persona, que serà qui mantindrà la visió de tot el projecte. Cal estimar el valor que el requisit té per al client i també assignar un valor que indicarà una estimació de l'esforç. Aquesta llista (incompleta en l'exemple) haurà de donar una idea clara sobre quins requisits s'assignaran a cada iteració o esprint.

2) Desenvolupament

Aquí és on es produeixen les iteracions, es farà un esprint per cada iteració i el resultat final serà un alliberament (*release*) intermedi que serà presentable a l'equip *scrum* complet. Els esprints tindran una descomposició de les tasques que s'han d'implementar en cada un, i per a cada esprint s'efectuaran les activitats de reunió de planificació de l'esprint (un per iteració), la definició d'acumulació de requisits en l'*sprint backlog* (abast de l'esprint) i l'estimació.

A continuació, un exemple de *sprint backlog*.

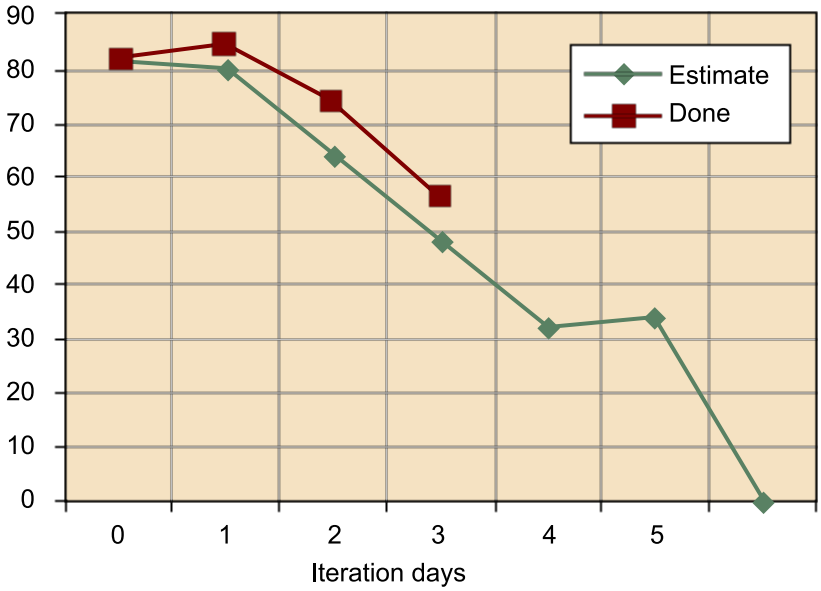
Taula 3.6. *Sprint backlog*.

				Iteration days					
Product backlog item	**Sprint task**	**Volunteer**	**Initial estimate of effort**	**1**	**2**	**3**	**4**	**5**	**6**
Com a usuaris, volem posar un llibre en un carretó.	Modificar la base de dades		4	4					
	Crear una pàgina web (GUI)		8	4	4				
	Crear una pàgina web (lògica de negoci)		13		4	8	1		
	Escriure tests d'acceptació		13				7	6	
	Actualitzar l'ajuda per al comprador		3					2	1
Com a usuaris, volem eliminar un llibre del carretó.	Modificar la base de dades		4	4					
	Crear una pàgina web (GUI)		8	4	4				
	Crear una pàgina web (lògica de negoci)		13		4	8	1		
	Escriure tests d'acceptació		13				7	6	
	Actualitzar l'ajuda per al comprador		3					2	1

Font: elaboració pròpia a partir de J. Sutherland; K. Schwaber (2011). *The scrum papers: nut, bolts, and origins of an agile framework*. En línia a: http://jeffsutherland.com/ScrumPapers.pdf.

L'*scrum master* prendrà nota del progrés de la iteració dia a dia i generarà una vista especial del progrés quantitatiu que es diu *burndown chart*. A continuació, un exemple que mostra com en el dia 4 de la iteració hi ha una desviació entre el treball fet i el que s'esperava.

Figura 12. Burndown chart en scrum.



Font: elaboració pròpia a partir de J. Sutherland; K. Schwaber (2011). *The scrum papers: nut, bolts, and origins of an agile framework*. En línia a: http://jeffsutherland.com/ScrumPapers.pdf.

L'aspecte més innovador és la voluntarietat del treball: és important que cada integrant de l'*scrum team* triï què és el que vol fer. Aquest tret i la capacitat autoorganitzativa ens assenyalen que les metodologies àgils només funcionarien en equips de desenvolupament madurs.

Per a cada iteració es faran diverses reunions:

- ***Daily scrum***. Cada dia 15 minuts en què s'analitzen les tres preguntes bàsiques: què heu fet des de l'última trobada?, quins obstacles hi ha per a assolir la meta?, què fareu per a la propera trobada? Es fa dempeus per a assegurar-se que serà curta i hi intervé l'equip de desenvolupament al costat de l'*scrum master*, que haurà de prendre nota dels obstacles per a intentar resoldre'ls com a part de la seva missió en l'*scrum team*. En equips formats per diversos equips *scrum*, hi haurà una altra reunió diària d'integració en què es respondran les mateixes preguntes referides a l'equip (què ha fet el vostre equip des de l'última trobada?, etc.).

- ***Sprint planning meeting***. La reunió de planificació de l'esprint es farà una vegada al començament de cada esprint i s'hi definirà quines activitats es duran a terme en cada esprint, i es planificaran les activitats, estimació i dates.

- ***Sprint review***. Revisió del producte lliurable obtingut al final de l'esprint, la qual cosa es diu *alliberament intermedi* o *increment*, que és el producte que es pot ensenyar a la resta de l'equip (client, *management*, etc.). Com a nota peculiar està prohibit el PowerPoint, la demostració es basa al 100% en el producte acabat. Durant les demostracions *sprint review* sorgiran nous

problemes i canvis que s'han d'incorporar o gestionar amb vista a la iteració següent (missió de l'*scrum master* i decisió del *product owner*).

- ***Sprint retrospective***. L'equip *scrum* reflexiona sobre el que podria haver anat millor en l'esprint acabat. És possible la reflexió sobre si la metodologia és útil. S'ha d'animar l'equip a reflexionar en aquestes coses mentre treballa en la iteració perquè aquesta reunió sigui més resolutiva.

El mètode *scrum* s'ha pres en consideració com a representació de les metodologies àgils. Com s'ha comentat, amb aquesta metodologia tampoc no se superen tots els problemes que ens podem trobar en un projecte de creació d'un sistema d'informació. Aquest tipus de metodologia resol molts dels problemes assenyalats en les metodologies clàssiques, però n'incorporen d'altres que no existien. Els continus canvis incorporats satisfaran més el client que no té el procés clar, però n'incrementaran l'abast i generaran una desviació en hores que difícilment seria assumida per cap equip que treballés amb les antigues metodologies. L'*scrum* i la resta de metodologies àgils presenten un camí sobre el qual s'ha de caminar apartant els obstacles que vagin apareixent. És com qualsevol metodologia que hem d'adaptar per a cada cas concret. Les millors empreses de desenvolupament usen en l'actualitat l'*scrum* (Google, Microsoft, Siemens, etc.), fet que ens indica que les metodologies àgils no són únicament una moda passatgera.

Resum

Aquest mòdul ha explicat com es construeixen sistemes d'informació en l'empresa, centrant-se en el punt de partida que és la reenginyeria de processos. Al començament del mòdul hem vist que les empreses, segons el grau de digitalització, es poden trobar en diversos estadis: automatització, racionalització, reenginyeria i canvi de paradigma. Aquest últim és el grau més madur de digitalització o d'implantació de sistemes d'informació.

La reenginyeria és l'anàlisi de processos de l'organització amb una motivació clara de canvi i d'optimització. Per això podem dir que és el punt de partida d'un sistema d'informació, sempre deixant clar que podria ser que un procés de reenginyeria no tingués com a resultat un sistema d'informació, encara que avui dia és bastant complicat evitar aquesta relació. Després de l'anàlisi dels processos, en la reenginyeria es produeix l'anàlisi de requisits. En l'anàlisi de requisits hem vist tant els elements propis d'aquesta anàlisi (definició d'objectius, rols, funcions, modularitat, prioritat, *inputs* i *outputs*) com la manera de dur-la a terme utilitzant entrevistes iteratives.

Així mateix, s'ha aprofundit en el model de desenvolupament en cascada perquè aquest és el model de construcció de sistemes d'informació més extensament utilitzat i perquè es considera un model clàssic. Amb tot, la tendència de les empreses tecnològiques amb equips de desenvolupament nombrosos és adoptar nous models emergents de desenvolupament que s'anomenen *mètodes àgils*. El canvi de filosofia de l'un a l'altre és palès: el model tradicional és més estàtic, més rígid, més rigorós, encara que també hem vist que té més capacitat predictiva respecte a l'abast.

Finalment, s'ha explicat detalladament en què consisteix un dels mètodes àgils més estesos, l'*scrum*.

Activitats

1. Dissenyeu un procés de reenginyeria per a una empresa que conegueu.

2. Analitzeu les diferències que hi ha entre els nous models àgils de desenvolupament i el model tradicional en cascada.

3. Establiu els requisits tècnics per a un programari de gestió de contactes.

4. Feu una investigació sobre el model de desenvolupament alemany en *V* i analitzeu les diferències amb el model en cascada que s'ha presentat en aquest mòdul.