Capitolo 5

Completamento dell'analisi dei requisiti

Una modifica che eseguita in fase di analisi dei requisiti costa un dollaro, potrebbe costarne 1000 dopo il rilascio.
LVT

Introduzione

Il presente capitolo si pone un duplice obiettivo: presentare una tecnica dimostratasi particolarmente valida nell'arduo compito dell'analisi dei requisiti, e illustrare ulteriori manufatti (*artifact*) che permettono di completare il modello dell'analisi dei requisiti.

Per quanto concerne il primo punto, lungi dall'idea di voler affrontare per l'ennesima volta le problematiche connesse con con il processo di analisi dei requisiti, in questo paragrafo si focalizza l'attenzione sul problema della fatale incomunicabilità che a volte si instaura tra i clienti, competenti della propria area di business, e il team dei tecnici esperti nella costruzione di sistemi informatici. Il problema, in ultima analisi, si riconduce al trovare una piattaforma di dialogo comune e costruttiva, una tecnica che rappresenti un valido raccordo tra i due diversi linguaggi tecnici: quello dell'area business utilizzato dagli utenti e quello informatico parlato dagli esperti di sistemi software. Come si è avuto modo di constatare nei capitoli precedenti, il problema dei diversi linguaggi è così importante da influenzare anche le diverse versioni del modello dei casi d'uso caratterizzate dal transitare via via dal mondo business a quello tecnico informatico.

Un valido strumento di analisi è indubbiamente rappresentato dagli activity diagram; essi, oltre ai nomali vantaggi offerti da ogni formalismo grafico (aspetto accattivante, facilità di comprensione e memorizzazione, immediatezza, ecc.), ne offrono un altro tutto

particolare e insuperabile: sono molto simili agli antenati flow chart. Chi non ne ha mai realizzato o interpretato uno? Chi non è in grado di comprenderli? (Qui la costante vocina dell'autore si fa sentire…) Pertanto, i diagrammi di attività si delineano come strumento ideale di dialogo tra utenti e team tecnico, almeno nelle prime fasi del ciclo di vita del software.

Muovendosi da queste considerazioni, è possibile escogitare una tecnica molto efficace per catturare i requisiti utente, anche se un po' agli antipodi con quelle classiche. Generalmente si tenta di descrivere la proiezione statica del modello dei requisiti per mezzo dei casi d'uso, poi se ne dettaglia la descrizione dinamica e infine si uniforma il tutto. Nella tecnica presentata l'approccio è diametralmente opposto: prima si cerca di capire il comportamento dinamico della funzione oggetto di studio per mezzo degli activity diagram e poi si risale alla relativa proiezione statica. Naturalmente, una volta redatti gli activity diagram, i vari template e/o flussi degli eventi assumono un interesse molto relativo: mostrerebbero le stesse informazioni con un formalismo diverso. Ovviamente è consigliabile evitare manufatti ridondanti essenzialmente per questioni di manutenzione: ogni modifica va ripetuta per i vari manufatti.

Gran parte dei capitoli precedenti sono stati dedicati all'illustrazione della vista dei casi d'uso i quali, attraverso un formalismo amichevole e relativamente semplice, favoriscono i processi di analisi e descrizione delle funzionalità del sistema, conferendo particolare attenzione alla percezione che ne hanno gli attori. Il modello dei casi d'uso, pur essendo un artifact molto importante, da solo non è assolutamente in grado di fornire sufficienti informazioni per il pieno svolgimento delle restanti fasi del ciclo di vita del sistema. Inoltre, nello sviluppo di progetti Object Oriented guidati dal modello dei casi d'uso, esiste il rischio potenzialedi produrre modelli funzionali: tale rischio, oltretutto, è accresciuto da personale non espertissimo del formalismo ed era ben noto fin dall'inizio allo stesso Jacobson, tanto che egli stesso evidenziò (1991-92) la necessità di bilanciare il modello dei casi d'uso con altri manufatti, quali: il Domain Object Model, il documento di specifica delle interfacce, il documento di architettura (o meglio la versione disponibile nella fase di analisi dei requisiti) e così via. La completa realizzazione della vista dei requisiti del sistema, dunque, prevede la produzione di ulteriori manufatti complementari al modello dei casi d'uso.

> Il Domain Object Model, molto in breve è una versione del diagramma delle classi volto a fornire la rappresentazione Object Oriented del vocabolario vigente nello spazio del problema. In esso dovrebbero essere presenti unicamente entità appartenenti al mondo concettuale di business che il sistema in qualche modo dovrà automatizzare. A questo particolare modello è dedicato ampio spazio nel Capitolo 8.

Nei paragrafi successivi sono presentati i manufatti relativi alla specificazione delle interfacce, i test case, il documento dei requisiti non funzionali e le business rule.

Una tecnica di analisi dei requisiti basata sugli activity diagram

Presentazione

Prima di addentrarsi nell'illustrazione della tecnica, si ritiene opportuno presentare un primo basilare esempio. Si consideri di studiare un sottosistema di workflow atto a gestire situazioni di malfunzionamento (eccezioni) che potrebbero insorgere nei vari componenti di un qualsivoglia sistema. Strumenti di questo tipo sono di frequente applicazione in sistemi a elevata disponibilità e con pressanti esigenze di recupero di situazioni anomale. In altre parole in sistemi come quelli bancari. Apparati di questo tipo prevedono che, a seguito del verificarsi di una anomalia di un certo rilievo non risolvibile automaticamente venga generata opportuna comunicazione da inviare al workflow.

Compito di questo dispositivo è ricevere i dati relativi a tali anomalie, registrarle, assegnarvi la priorità e quindi convogliarle a un opportuno operatore umano in grado di risolvere l'anomalia. L'assegnazione della priorità tipicamente avviene in funzioni a parametri configurabili e la selezione dell'operatore a cui convogliare l'anomalia deve avvenire indirizzandosi a chi ha le competenze necessarie per risolvere quel particolare tipo di anomalia.

Il diagramma di fig. 5.1 è abbastanza semplice e non richiede particolari spiegazioni: è proprio questo il vantaggio di utilizzare i diagrammi di attività! Il sistema riceve i dati relativi ad un'anomalia, genera il relativo work item (in sostanza vengono estratti i dati significativi), li memorizza, effettua l'analisi di quelli ritenuti salienti: codice identificativo del sistema che ha generato l'eccezione, stadio in cui si è verificato l'errore, tipologia, ecc. Quindi verifica se sia possibile raggruppare il work item con altri pendenti originati dalla stessa anomalia. Nel caso in cui la verifica sia positiva, avviene il raggruppamento e quindi la priorità del gruppo di appartenenza viene estesa al nuovo work item, mentre in caso negativo (il raggruppamento non è possibile) è necessario assegnare la priorità in base sia ai dati salienti dell'anomalia, sia ad opportuni criteri presenti nel sistema; come si vedrà di seguito, le regole utilizzate per calcolare la priorità sono esempi di business rule.

Una volta superata anche questa fase, viene selezionato l'operatore appropriato, ancora una volta combinando i dati salienti dell'anomalia con quelli relativi agli operatori connessi al sistema — considerando ovviamente anche il carico di lavoro pendente su di essi — e quindi l'anomalia viene assegnata. Tipicamente, gran parte del lavoro viene risolto in fase di configurazione: tutte le tipologie di anomalie previste dal sistema vengono catalogate e raggruppate. Per ciascun gruppo identificato si predispone un'opportuna coda in cui inserire i dati delle varie anomalie. Le code sono quindi associate a ruoli da assegnare ai vari addetti. L'operatore riceve la segnalazione e, in base alle informazioni contenute, esegue determinate procedure atte a risolvere il problema. Risolta l'anomalia, notifica l'avvenuto assolvimento, eventualmente descrivendo le operazioni eseguite. Il sistema quindi riceve la segnalazione di avvenuta risoluzione del problema e, per anomalie non note, ne memorizza la soluzione.

Figura 5.1 — *Activity diagram di un gestore di anomalie.*

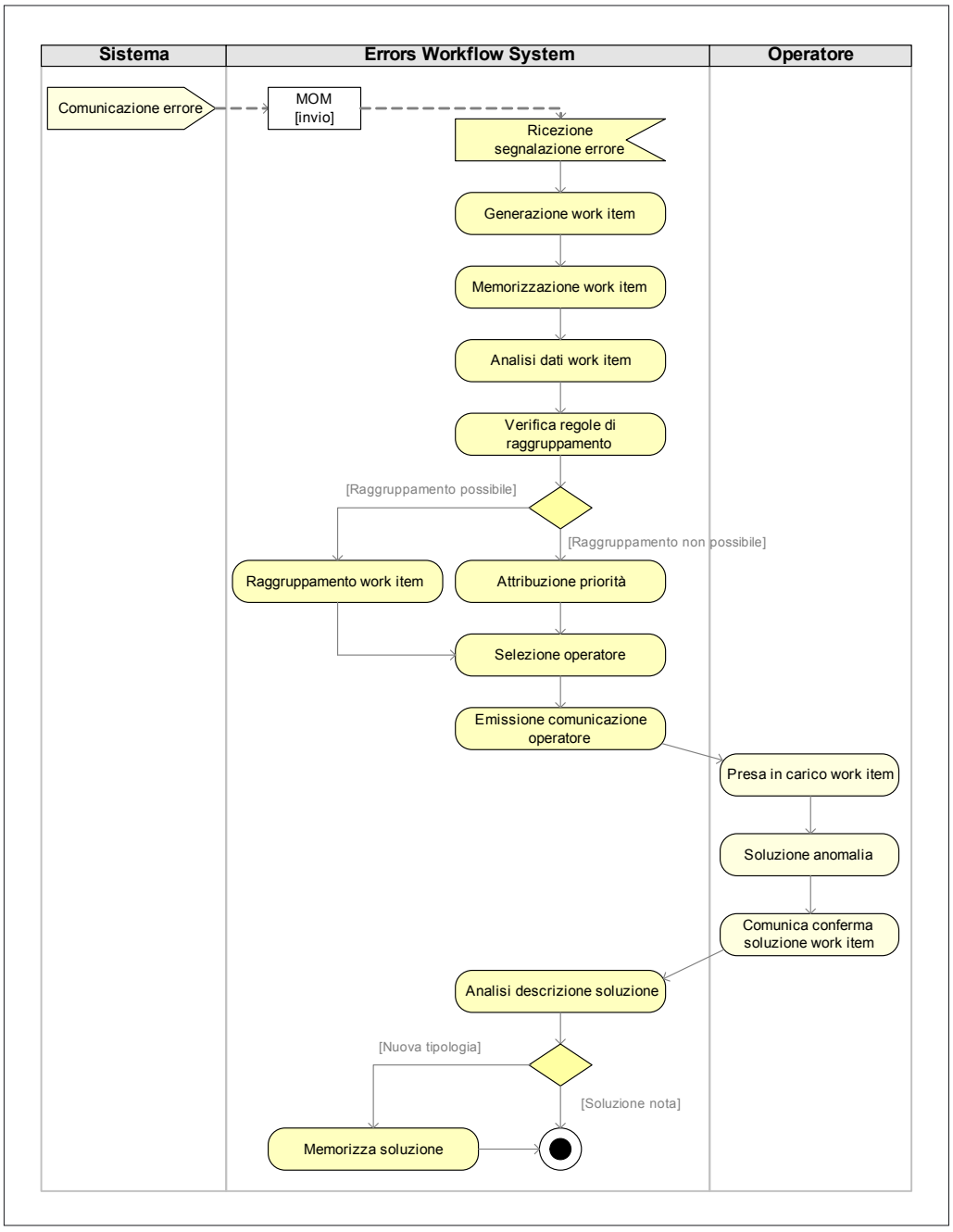
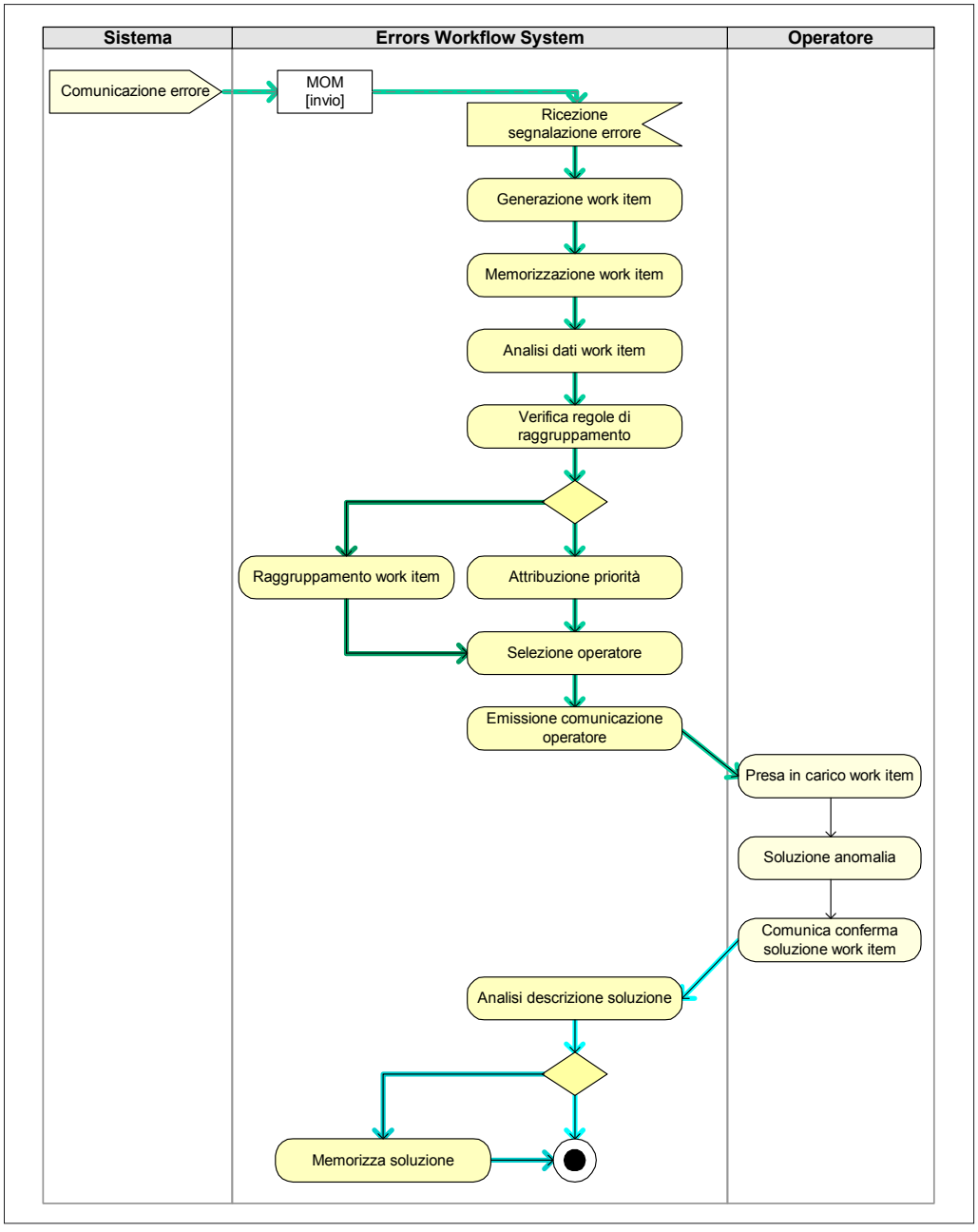


Figura 5.2 — *Activity diagram di un sistema di gestione delle anomalie con evidenziati i diversi flussi.*



Un passo propedeutico da utilizzarsi per "estrarre" i diagrammi dei casi d'uso dai relativi dall'activity diagram consiste nell'evidenziare i messaggi che il sistema scambia con i relativi attori. Generalmente, il percorso compreso tra due messaggi consecutivi rappresenta un'unica funzionalità da descriversi attraverso uno use case.

Per esempio la parte di activity diagram compresa tra la ricezione di una segnalazione di errore e la comunicazione del relativo work item all'`Operatore` rappresenta logicamente uno use case. Ora, verosimilmente, tale use case risulterebbe notevolmente complesso da rappresentare attraverso un solo caso d'uso, e quindi è opportuno ripartire alcune funzionalità in altri casi d'uso inclusi o estesi.

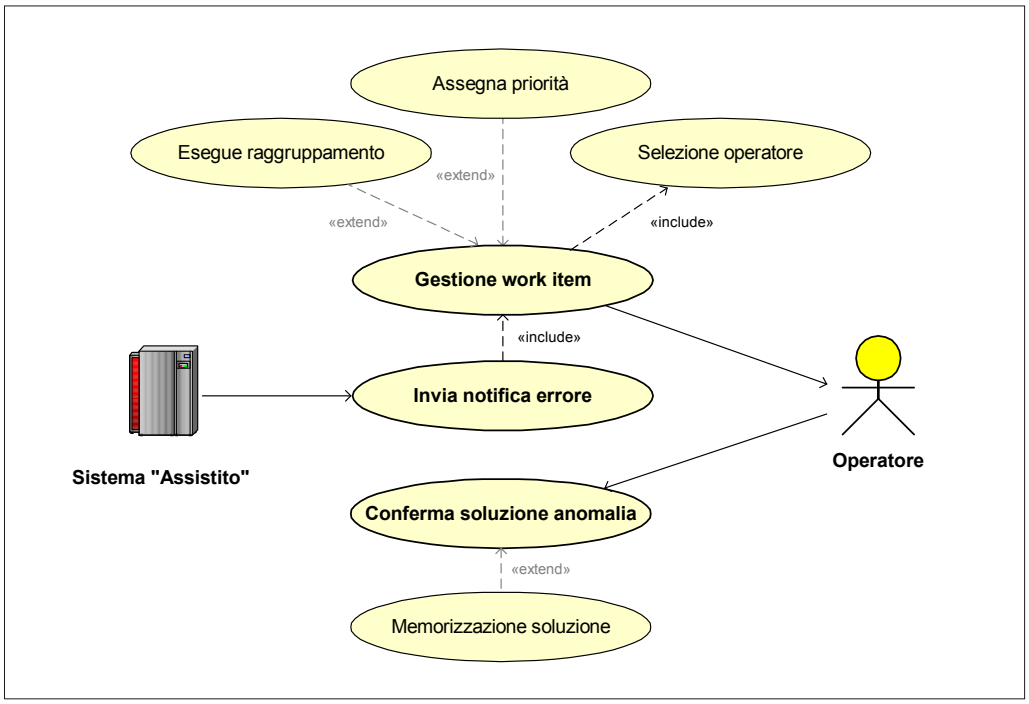
Analogamente la porzione di activity diagram compresa dalla ricezione del messaggio di conferma soluzione fino alla fine delle attività si presta a essere rappresentata per mezzo di un caso, eventualmente da espandere.

La parte di percorso relativa alla diramazione del flusso — quella generata dagli elementi condizionali che nella fig. 5.2 sono mostrati con una tonalità più scura rispetto a quella del flusso principale — rappresenta verosimilmente un'alternativa rispetto al flusso base. Tali flussi si prestano a essere descritti o attraverso specifici use case estendenti o, più semplicemente, attraverso dei flussi alternativi. La decisione dipende da vari fattori, come per esempio l'importanza che si desidera assegnare al flusso (use case distinti conferiscono maggiore enfasi), la corposità del comportamento da descrivere, gli obiettivi che permettono di raggiungere (se le post-condizioni sono diverse da quelle del flusso base allora molto probabilmente si tratta di uno use case separato) ecc.

Come si può riscontrare dall'analisi del diagramma dei casi d'uso mostrato in fig. 5.3, il relativo livello di dettaglio è decisamente inferiore a quello presente nell'activity diagram (ci sono meno informazioni). Molte attività ritenute di secondaria importanza (`Memorizza work item`, *Analisi dati work item*, ecc.) sono state inglobate in uno use case più generale (`Gestione work item`).

A questo punto può insorgere qualche problema. Gli utenti, nell'esaminare i diagrammi dei casi d'uso ottenuti da quelli delle attività realizzati con il loro contributo, tipicamente si attendono di vedere una corrispondenza biunivoca tra i casi d'uso e le attività presenti nei relativi diagrammi. Ciò chiaramente non sempre è consigliabile. Diversi tool commerciali permettono di tenere traccia automaticamente di tali corrispondenze. In

Figura 5.3 — *Use case del sistema di gestione anomalie.*



casi estremi è possibile realizzare apposite matrici di "corrispondenza", in cui si specifica l'allocazione delle attività ai casi d'uso. Il problema, ancora una volta, è che la gestione di tali tabelle è un'attività decisamente dispendiosa. Si consiglia pertanto di evitare per quanto possibile il ricorso a queste tabelle: richiedono molto tempo per essere realizzate e necessitano di essere riviste ogniqualvolta si ristrutturano i diagrammi dei casi d'uso a cui si riferiscono.

Un'altra constatazione è relativa alle attività inserite in diramazioni del flusso (costrutti `if then else`) presenti negli activity diagram (attività da svolgere solo al verificarsi di precise condizioni come per esempio `Assegna priorità` ed `Esegui raggruppamento`). Come specificato precedentemente, quelle ritenute importanti, e quindi da visualizzare anche nel diagramma dei caso d'uso (`Memorizzazione soluzione`), si prestano ad essere riprodotte per mezzo della relazione di estensione. In alcuni casi si può ricorrere anche a relazioni di generalizzazione. Ciò, quando possibile e badando bene al significato semantico, permette di rappresentare più chiaramente l'esecuzione alternativa di precise attività. Per esempio nel diagramma dei casi d'uso precedente si sarebbe potuto ricorrere a questo espediente per strutturare le attività di `Assegna priorità` ed `Esegui raggruppamento`. A essere onesti, in questo caso l'utilizzo sarebbe stato artificioso

Tabella 5.1 — *Matrice di mapping tra use case e activity.*

Activity \ Use Case	Invia notifica errore	Gestione work item	Esegue Raggruppamento	Assegna priorità	Selezione operatore	Conferma risoluzione anomalia	Memorizzazione soluzione
Comunicazione errore	×						
Ricezione segnalazione errore	×						
Generazione work item		×					
Memorizzazione work item		×					
Analisi dati work item		×					
Verifica regole di raggruppamento		×					
Raggruppamento work item			×				
Attribuzione priorità				×			
Selezione operatore					×		
Emissione comunicazione operatore		×					
Presa in carico work item	——	——	——	——	——	——	——
Soluzione anomalia	——	——	——	——	——	——	——
Conferma soluzione anomalia						×	
Memorizzazione soluzione							×

e quindi non di facile comprensione: il raggruppamento non può essere considerato una specializzazione dell'assegnazione della priorità sebbene, qualora eseguito, generi come effetto collaterale l'assegnazione automatica della priorità. L'iterazione con l'attore `Operatore` è stata suddivisa in due use case: uno inserito in quello denominato `Gestione work item` per l'invio della comunicazione all'attore e l'altra nel caso d'uso `Conferma risoluzione anomalia` al fine di ricevere il relativo feedback.

Si ricordi che nello UML è preferibile modellare le interazioni in modo asincrono. Ciò però non implica necessariamente che il sistema le realizzi secondo tale criterio: nel contesto dell'analisi dei requisiti è necessario catturare i requisiti utente e non le soluzioni.

Figura 5.4 — *Frammento semplificato di un activity diagram utilizzato per dettagliare la procedura di approvazione di primo livello delle richieste di variazione profilo utente.*

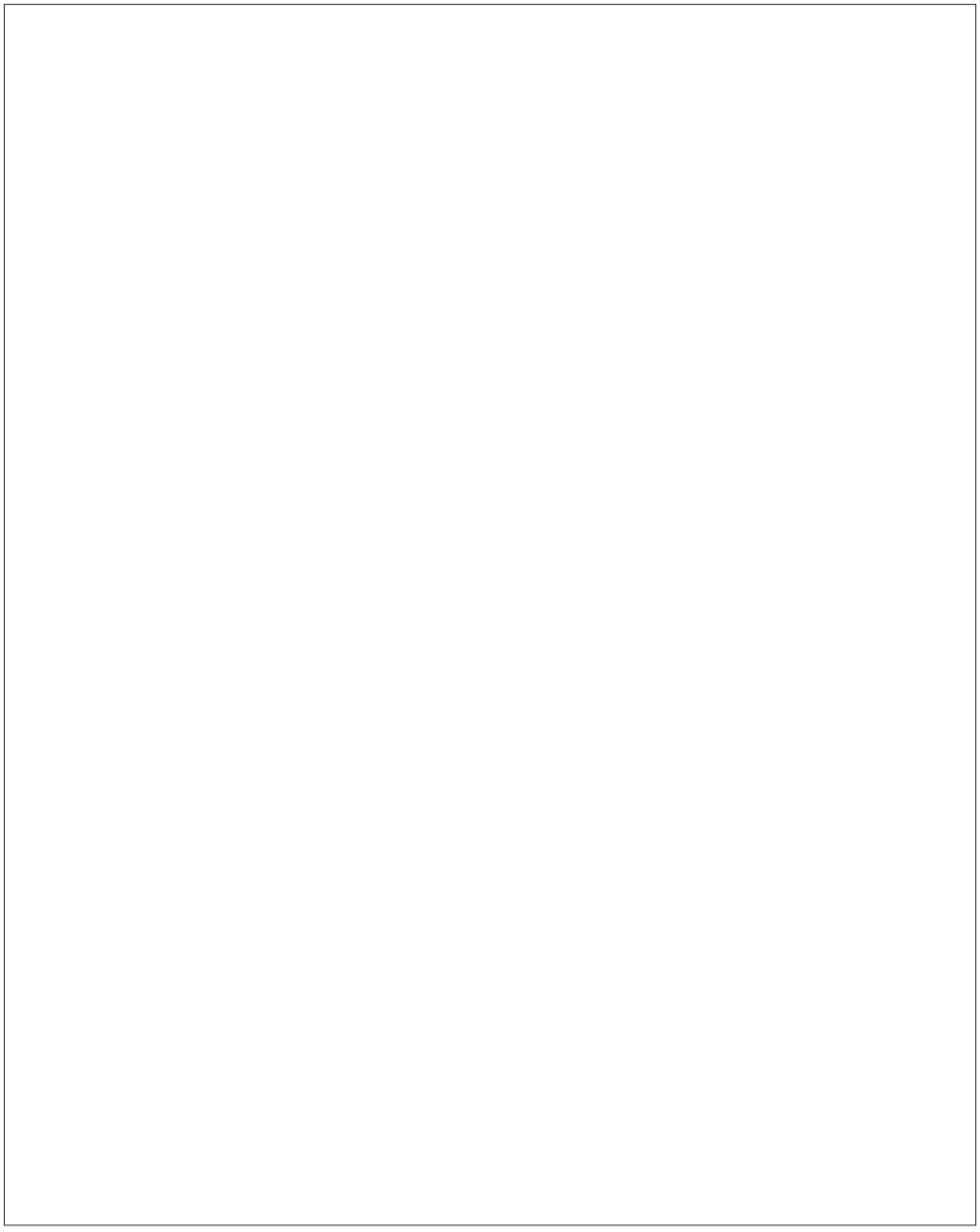
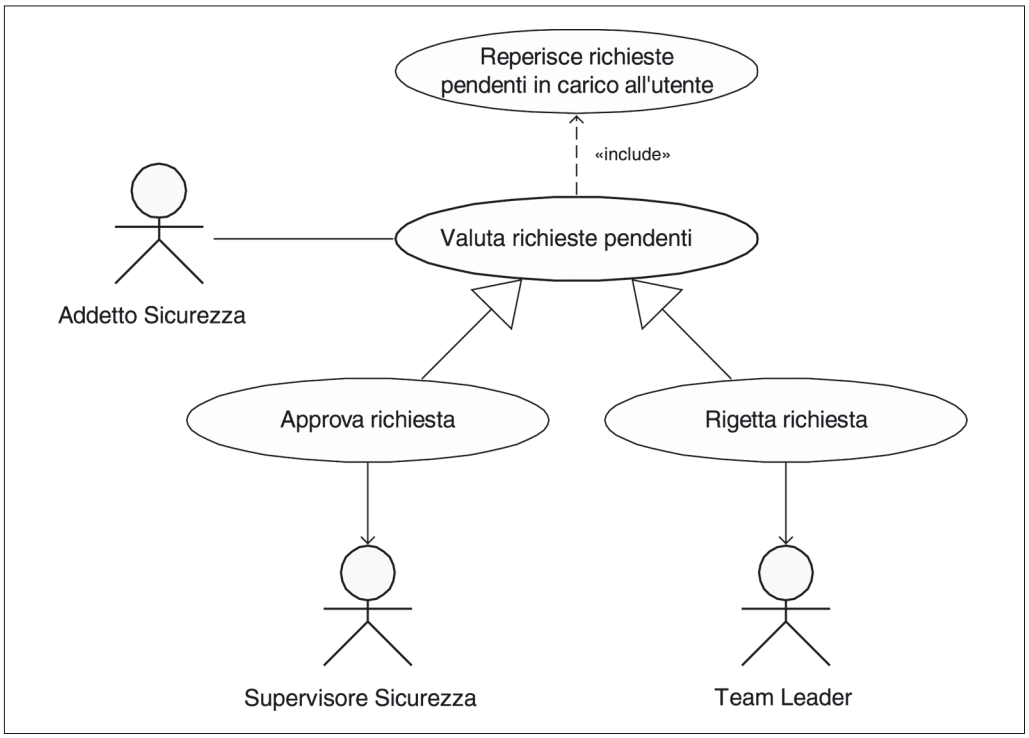


Figura 5.5 — *Frammento dello use case diagram relativo all'activity riportato nella fig. 5.4.*



Riflessione conclusiva di questo primo esempio: tipicamente è necessario riportare un ulteriore caso d'uso generale (`Gestione work item`) con funzioni sia di collante dei vari casi d'uso (una sorta di `main` che invoca sottoroutine), sia di raccoglitore delle funzionalità ritenute di eccessivo livello di dettaglio (il contenitore del flusso principale o *main scenario*).

Il secondo esempio presentato è stato introdotto al fine di illustrare l'utilizzo della relazione di generalizzazione. In particolare viene riportato un frammento del processo, vigente presso un'ipotetica organizzazione (per esempio una banca), atto ad assegnare ai propri dipendenti i profili (`profile`) per l'accesso al sistema. In questo contesto si suppone che, in ogni istante di tempo, ciascun utente disponga di un solo profilo operativo. Tale profilo stabilisce quali funzioni l'utente può eseguire e su quali dati. Per esempio, se un utente dispone di un profilo che abilita l'esecuzione del servizio di verifica dei conti correnti dei clienti di una determinata filiale, non è detto che lo stesso utente possa eseguire tale servizio per i conti correnti dei clienti di altre filiali. Il flusso ipotizzato è che il `Team Leader`, ogniqualvolta gli sia assegnato un nuovo dipendente, o quando questi cambi di

funzione/ruolo, compili un'apposita richiesta di assegnazione di un nuovo profilo per tale dipendente. Si ipotizza poi che tale richiesta debba passare attraverso due livelli di approvazione: una prima a carico di un addetto della sicurezza appartenente al dipartimento dell'utente e una seconda a carico del supervisore della sicurezza. La funzione analizzata di seguito è relativa alla prima approvazione.

Si consideri ora il diagramma delle attività riportato nella fig. 5.6. Si tratta della funzione denominata `Monitor of Off-Market Transactions`, ossia di un processo batch a carico del back office di un sistema finanziario atto a verificare che i tassi applicati ai `Trade` stipulati nel sistema di `Front Office` siano consistenti con quelli di mercato.

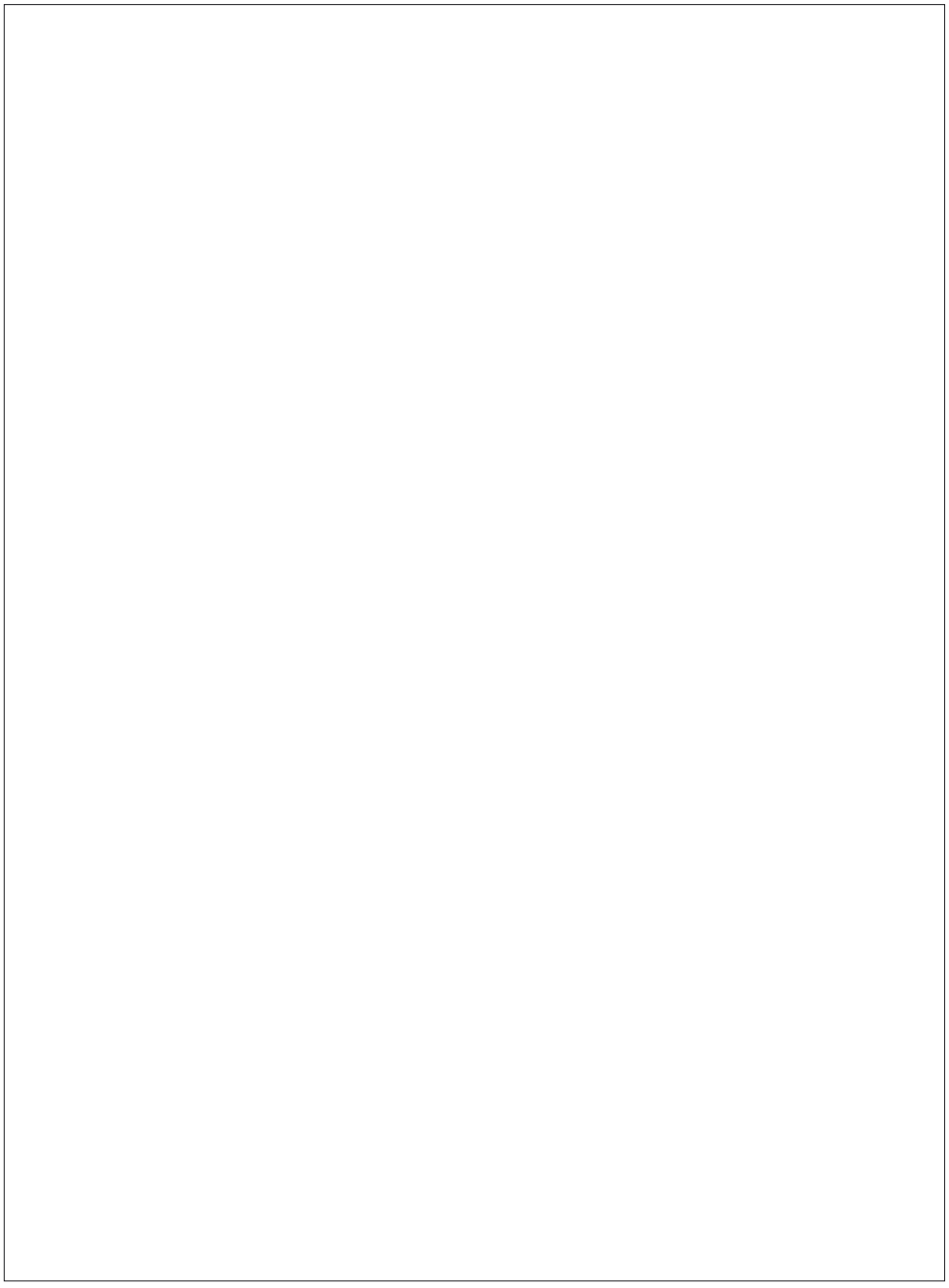
> Brevemente, con il termine *Trade* ci si riferisce a contratti di scambio di prodotti (in questo contesto finanziario e tipicamente in valuta), tra due parti: cliente e una organizzazione finanziaria.

In altre parole si vuole controllare che ai `Trade` siano applicati tassi compatibili con quelli di mercato. Per esempio, se un `Trade` è relativo a un prodotto `FX` (Foreign eXchange, scambio di valuta) tra `EUR` (Euro) e `GBP` (Sterlina del Regno Unito), e il relativo tasso di mercato nell'istante della stipula era uguale a `x`, si vuole accertare che quello realmente applicato sia all'interno di un opportuno intervallo (per esempio, ± 10%) del valore di mercato più le commissioni.

Il *Monitor of Off-Market Transactions* è uno dei controlli che avvengono presso le banche alla chiusura della attività di mercato. In ultima analisi si controlla l'operato dei vari `dealer` (personale addetto alla contrattazione e gestione dei `Trade` finanziari), con l'obiettivo di accertare che gli stessi non effettuino, per qualche misterioso motivo, condizioni di favore, magari ancora vantaggiose per l'organizzazione finanziaria ma comunque eccessivamente vantaggiose per il cliente (controparte). Tipicamente, i `Trade` risultati `Off-Market` vengono trasmessi al manager del `Front Office` ed al "controllore finanziario", i quali hanno la responsabilità di effettuarne la revisione

> Per questioni di completezza è necessario dire che esistono dei casi di eccezione — non riportati nella trattazione al fine di contenere la complessità e mantenere l'obiettivo sulla tecnica e non sul business bancario — solitamente denominati *Historical Rate Rollovers*. Brevemente, in alcuni casi può capitare che un cliente — probabilmente di particolare rilievo — stipuli un contratto con precise quotazioni di mercato in un istante determinato di tempo e, alla scadenza dello stesso, il cliente chieda di prolungarne il termine. Per esempio un cliente stipula un contratto per 1 000 000 di dollari da pagare in Euro. Alla scadenza, lo stesso richiede di prolungare il contratto per un altro mese. La procedura corretta sarebbe di chiudere il primo contratto e di effettuarne uno nuovo con i tassi di mercato aggiornati. Per specifici mercati e particolari clienti è invece permesso di stipulare un nuovo contratto con i tassi di quello precedente.

Figura 5.6 — *Diagramma delle attività del processo di controllo dei valori di tolleranza dei Trade.*



> Si tenga presente che l'obiettivo del presente paragrafo è ancora una volta illustrare la tecnica descritta in precedenza e non modellare una porzione del back office della banca, pertanto l'autore si scusa per eventuali approssimazioni.

Nel diagramma riportato in fig. 5.6 sono presenti imprecisioni ed errori tipici di un diagramma reale. Si è deciso di presentarli sia per avere ulteriori spunti di riflessione, sia per riportare situazioni più vicine alla realtà.

In primo luogo un errore tipico in cui è facile incorrere consiste nel tentare di modellare o comunque di descrivere le specifiche di sistemi al di fuori di quello oggetto di studio. In questo equivoco, a dire il vero, i clienti sono maestri: non appena cominciano a prendere dimestichezza con strumenti di analisi, quali essi siano (use case, activity diagram, ecc.), vengono affetti dalla sindrome denominata *specificomania*: vorrebbero scrivere le specifiche per ogni entità incontrata… Anche di sembianze umane!

Nel diagramma in questione il fantomatico servizio denominato `Servizio Telematico Dati Quotazioni Prodotti Finanziari` rappresenta evidentemente un sistema esterno a quello oggetto di studio: un attore a tutti gli effetti, il cui comportamento è stato però impropriamente modellato anche se solo parzialmente. Ciò non è un peccato mortale fintantoché sia ben chiaro che tali dettagli hanno unicamente valenza descrittiva e quindi non devono confluire negli use case diagram.

Un altro elemento di interesse potrebbe essere la presenza del sistema di gestione delle eccezioni descritto in precedenza. Ad esso vengono convogliate tutte le segnalazioni di situazioni anomale: problemi con la comunicazione con il sistema esterno, impossibilità di reperimento delle tabelle di tolleranza, ecc.

Tornando al processo Monitoring Off-Market Transactions, esso ha inizio alla chiusura della giornata finanziaria. La prima verifica da effettuarsi è relativa alla necessità di eseguire il processo di controllo del rispetto dei limiti di tolleranza. In caso di esito negativo si passa ad esaminare l'eventuale Trade successivo, altrimenti si prosegue verificando se nel sistema siano disponibili i dati relativi alle quotazioni di mercato inerenti il particolare Trade in un accettabile intervallo temporale del momento in cui è avvenuta la stipula del Trade stesso: si è interessati al valore di mercato del Trade all'atto della stipula.

Nel caso in cui i dati siano disponibili si procede con il processo, altrimenti si richiedono le quotazioni a un ipotetico `Servizio Telematico Dati Quotazioni Prodotti Finanziari`. Tale sistema è esterno a quello oggetto di studio (*outside scope*), sebbene riportarne la descrizione nel diagramma delle attività non sia peccato mortale.

Ricevuto il messaggio di ritorno, questo viene analizzato. Se la struttura e i contenuti attesi non sono conformi a quelli attesi viene generata una segnalazione di errore, inviata poi al `Sistema di Gestione delle Eccezioni`. Se invece la risposta ottenuta risulta conforme a quanto atteso, si salvano i dati contenuti e si prosegue per il normale flusso del processo.

Si tenga presente che il fatto che nelle specifiche utente il dominio del processo di verifica sia il singolo `Trade` non deve essere considerato un vincolo per il sistema finale. Gli

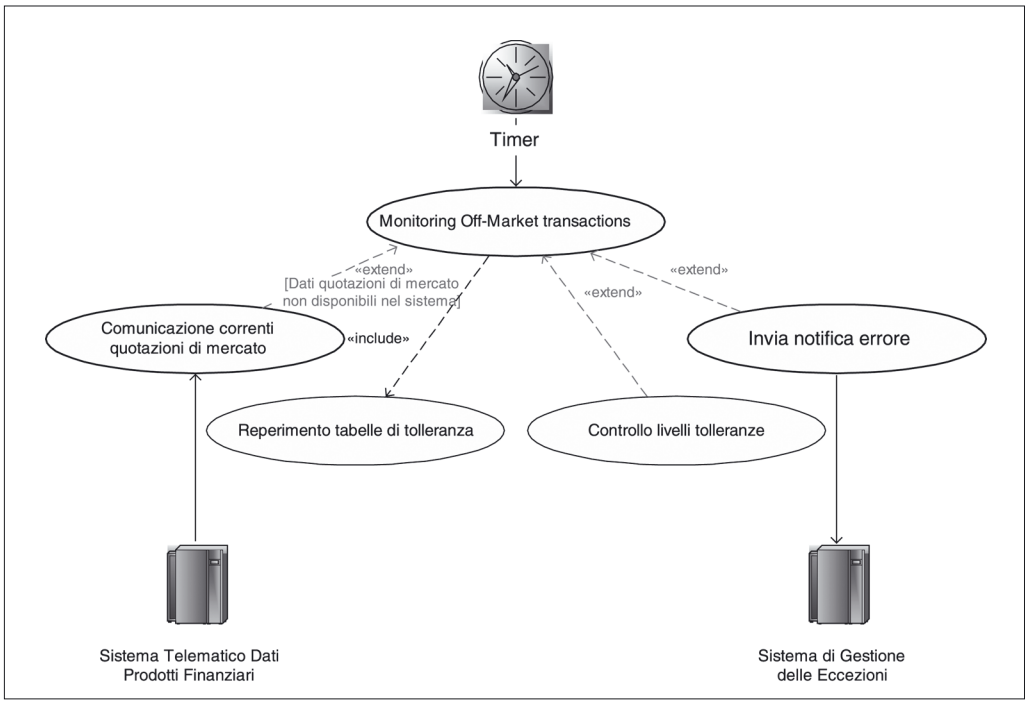
use case dei requisiti utente servono appunto per specificare cosa si attende l'utente e non come ottenerlo. Per esempio nulla vieta che il sistema finale presenti alcune ottimizzazioni, come per esempio accumulare le richieste di quotazioni in un apposito messaggio da inviare ad intervalli predefiniti invece di richiedere ogni singolo valore di quotazione appena necessario.

Ottenuti i dati si prosegue tentando di reperire le tabelle di tolleranza del particolare `Trade`; nel caso di impossibilità si procede tentando di reperire quelle standard. Fallendo anche questo secondo tentativo, si emette relativa segnalazione di errore ed il processo prosegue con l'esame del `Trade` successivo.

Ottenuti anche i dati relativi alle tabelle, si effettuano i vari calcoli, li si salva e quindi si effettua il tanto agognato controllo del rispetto dei vincoli di tolleranza. Se il `Trade` risulta al di fuori dei limiti previsti, come al solito si emette opportuna segnalazione al `Sistema di Gestione delle Eccezioni`, altrimenti il processo termina placidamente.

Il `Sistema di Gestione delle Eccezioni` dovrebbe riconoscere la tipologia del messaggio e quindi inserire appositi `WorkItem` nelle code relative al `Manager del Front Office` e al `controllore finanziario`, i quali sono formalmente chiamati a intrapren-

Figura 5.7 — *Use case diagram del processo di controllo dei livelli di tolleranza.*



dere opportune azioni (verifiche, sanzioni, ecc.) il cui esito deve essere memorizzato nel sistema. In fig. 5.7 viene mostrato il caso d'uso, che non dovrebbe presentare grosse sorprese.

Una perplessità che potrebbe sorgere analizzando i due diagrammi riguarda la ripetizione del ciclo di analisi per tutti i trade stipulati nell'arco del giorno. In particolare, mentre nel diagramma delle attività è ben evidente, nel rispettivo diagramma dei casi d'uso non lo è. Poco male, la ripetizione verrà definita nella descrizione del *main scenario* dello use case `Monitoring Off-Market Transactions`.

Vantaggi e svantaggi

I vantaggi del metodo dovrebbero essere ormai chiari:

1 semplifica il processo di instaurazione di una piattaforma comune di dialogo tra gli utenti e il team che dovrà sviluppare il sistema;

2. riduce il tempo necessario per addestrare il personale non tecnico all'utilizzo del formalismo dei diagrammi delle attività, e ne aumenta l'entusiasmo (in genere è molto più divertente giocare con dei diagrammi piuttosto che scrivere del testo);

3. la presenza di requisiti descritti tramite diagrammi li rende più chiari e più immediati e quindi riduce il rischio di incomprensioni;

4. agevola lo scambio delle informazioni e fornisce un'eccellente documentazione da utilizzare per addestrare il personale sia dell'area tecnica, sia di quella business;

5. i tool commerciali supportano bene questa strategia.

A fronte di questi vantaggi — a dire il vero se ne potrebbero riportare diversi altri — esiste però una serie di inconvenienti:

1. superata una certa complessità, i diagrammi delle attività tendono a divenire confusi e bisogna investire molto tempo nel tentativo di renderli lineari e comprensibili;

2. gli activity diagram, tipicamente, non riescono a sopperire completamente all'esigenza di redigere la descrizione dei casi d'uso. Ciò perché non è immediato inserire tutte le informazioni (trigger, pre- e postcondizioni, requisiti speciali, ...) ed è difficile specificare i dettagli di tutti i flussi nei diagrammi stessi;

3. disponendo sia della descrizione dei casi d'uso, sia dei diagrammi delle attività, diviene molto costoso mantenere sincronizzati i due modelli;

4. il processo di aggiornamento dei diagrammi di attività richiede molto più tempo rispetto all'aggiornamento del testo di un diagramma dei casi d'uso.

Dall'analisi dei vantaggi e degli svantaggi del metodo è possibile constatare che nonostante gli inevitabili inconvenienti, il metodo possiede delle peculiarità (maggiore coinvolgimento degli utenti, facilità di scambio delle informazioni, ...) che, specie in progetti non banali, sarebbe veramente un peccato non sfruttare. Il consiglio allora potrebbe essere quello di utilizzare comunque la tecnica dei diagrammi delle attività almeno fino al punto in cui i requisiti, e quindi i relativi modelli, raggiungono un'accettabile stato di stabilità. A questo punto si potrebbero placidamente trascurare questi diagrammi, concentrandosi unicamente sui diagrammi dei casi d'uso e sulla relativa descrizione.

Modello per le interfacce

Un aspetto molto importante della progettazione di un sistema è legato alla definizione del confine dello stesso. Un manufatto che ne favorisce la determinazione, e che agevola la definizione formale delle responsabilità che attori e sistema assumono gli uni nei confronti dell'altro, è il modello di specifica delle interfacce. Questo, una volta completato, dovrebbe rendere immediata la definizione delle classi "boundary" presenti nel modello di analisi (cui si accenna nel Capitolo 2 e che viene trattato nel Capitolo 8).

Spesso le entità esterne al sistema non sono esclusivamente utenti "in carne e ossa", bensì altri sistemi (legacy system, moduli agent, sensori, software forniti da terze parti, sistemi di fornitura di servizi *on demand*, ecc.). In questi casi è ancora più sentita la necessità di definire formalmente le interfacce tra il sistema e gli attori poiché, essendo dispositivi, potrebbero davvero essere meno accomodanti degli attori umani (almeno teoricamente!).

Analizzare tali interfacce è molto importante anche perché finiscono per influenzare direttamente l'elaborazione del modello dei casi d'uso: dovendo interagire con sistemi esterni di cui non si ha il controllo, è più facile che il sistema oggetto di studio si adatti alle interfacce predefinite piuttosto che il viceversa.

Considerando infine che il paradigma preferenziale dei sistemi di nuova generazione è il famoso component-based, secondo il quale il sistema è definito in termini di specifici componenti in grado di erogare servizi definiti nelle relative interfacce, si comprendono ancora più approfonditamente i vantaggi derivanti dall'iniziare a definire formalmente le interfacce del sistema già durante le prime fasi del processo di sviluppo del software. Sebbene le due tipologie di interfacce presentino un diverso livello di astrazione e obbiettivi sensibilmente diversi, esiste una chiara dipendenza delle interfacce dei componenti dalle rispettive del sistema: in ultima analisi i componenti realizzano i servizi richiesti al sistema.

Comunque, la definizione delle interfacce fornisce sia ulteriori requisiti da utilizzare per iniziare la modellazione del sistema in termini più architetturali, sia importanti informazioni per l'evoluzione del Domain Object Model: è lecito attendersi che i dati scambiati con gli attori siano opportuni sottoinsiemi di quelli presenti nel modello a oggetti. In caso contrario, probabilmente è opportuno accertarsi che nel Domain Object Model non siano presenti lacune.

La relazione tra le interfacce del sistema e il modello del dominio è così forte che spesso il processo di analisi delle interfacce viene eseguito dopo aver accertato la validità del modello a oggetti del dominio: d'altronde è lecito attendersi di comunicare informazioni a propria disposizione e di ricevere informazioni in grado di essere trattate.

Come ultimo punto si consideri il successo che stanno incontrando le infrastrutture di messaggistica (i famosi MOM*, Messaging Oriented Middleware*) nei sistemi di nuova generazione e il protocollo XML: è evidentissimo come molte delle interfacce — quelle relative allo scambio di messaggi tra dispositivi fisici — risultino prime versioni dei messaggi che viaggeranno nel sistema. Dunque iniziare a progettare anticipatamente questi messaggi agevola il processo di produzione del software.

Ricapitolando, è vantaggioso realizzare la definizione formale delle interfacce già durante la fase di analisi dei requisiti, perché:

- favorisce la definizione dei confini del sistema;
- offre ottimi spunti al modello dei casi d'uso;
- fornisce preziose informazioni o permette di bilanciare il modello a oggetti dominio;
- agevola l'individuazione delle interfacce dei sistemi basati sui componenti;
- favorisce la definizione dei messaggi circolanti nel sistema, qualora si ricorra ad un'infrastruttura di messaggistica.

La tab. 5.2 presenta un template rivelatosi utile per l'analisi iniziale delle interfacce sistema/attori. Qualora si desideri analizzare interazioni, specificatamente sistema/attori umani, probabilmente potrebbe risultare più opportuno, considerati i vari tool a disposizione, definire un prototipo di interfaccia utente con tutti i rischi del caso (cliente che confonde il prototipo con il sistema finale, perdita di generalità, ecc.) esaminati nei capitoli precedenti.

Nella sezione dedicata alle informazioni generali è necessario riportare il nome dell'interfaccia, il codice, la data e la versione. Per ciò che concerne il campo codice, tipicamente, si preferisce ricorrere a identificatori mnemonici.

Un possibile sistema di selezione del codice potrebbe consistere nell'utilizzare:

Tabella 5.2 — *Template utilizzato per l'analisi iniziale delle interfacce.*

Informazioni generali.			
NOME PROPOSTO:	CODICE PROPOSTO:	DATA:	VERSIONE:
BREVE DESCRIZIONE:			
Attori.			
NOME:	BREVE DESCRIZIONE:		FORNITORE/ FRUITORE
OBIETTIVI:			
Casi d'uso.			
NOME:	BREVE DESCRIZIONE:		FORNITORE/ FRUITORE
OBIETTIVI:			

Definizione inter faccia.					
GRUPPO	CLASSE NEL DOM (Se presente)	NOME ATTRIBUTO	TIPO/DIM	OBBL.	DESCRIZIONE

Occorrenze			
GRUPPO	NOME ATTRIBUTO	#	DESCRIZIONE

Osservazioni:

CODICE	AUTORE	DESCRIZIONE	STATO	DATA

- le prime tre lettere fisse al fine di indicare che si tratta della definizione di un'interfaccia (per esempio INT);

- un carattere atto ad individuare la fase del ciclo di vita di appartenenza del documento (ad esempio B = business, R = requirements, A =analysis, ecc.);
- tre lettere indicanti l'attore principale (per esempio AMM = amministratore);
- sei lettere indicante il contenuto (per esempio PRFUTN = per profili utente).

Nella sezione attori, come lecito attendersi, si dichiarano le entità esterne al sistema che utilizzano l'interfaccia, la relativa descrizione e i motivi che inducono l'utente ad utilizzare l'interfaccia stessa (obiettivi). In particolare è opportuno specificare se si tratta di un attore che fornisce i dati specificati nell'interfaccia o se li riceve dal sistema. L'intera sezione va ripetuta per tutti gli attori che interagiscono attraverso l'interfaccia stessa.

Per ciò che concerne la sezione dedicata ai casi d'uso, il discorso è del tutto analogo a quanto riportato poc'anzi per gli attori.

Nel caso generale, l'interfaccia può essere utilizzata da diversi attori e casi d'uso.

Nella definizione degli attributi che compongono l'interfaccia, è necessario tenere a mente che questi dovrebbero essere presenti nel Domain Object Model (modello a oggetti del dominio); in caso contrario è opportuno accertarsi che nel modello a oggetti non siano presenti lacune.

Nel caso in cui l'attributo sia contemplato nel modello del dominio, occorre precisare la relativa classe di appartenenza (seconda colonna del template), altrimenti è necessario specificarne il tipo e la dimensione.

Un campo che potrebbe destare qualche perplessità è quello relativo al gruppo. Si tratta di un artificio utilizzato per indicare la necessità di ripetere più volte un gruppo di attributi che non necessariamente coincidono con la definizione di una classe.

Nella definizione di un'interfaccia, tipicamente, alcune informazioni prevedono diverse occorrenze. Ora, se si tratta di singoli attributi, non sussiste alcun problema: è sufficiente riportare il nome dell'attributo nella sezione dedicata alle occorrenze e quindi specificare la frequenza. Nel caso di classi il discorso è pressoché simile. Nel caso in cui però si tratti di un insieme di attributi (magari sottoinsieme di una classe) che è utile ripetere, ecco che l'informazione "gruppo" risulta utile: si associano logicamente tra loro un insieme di attributi e quindi se ne specifica la frequenza nella sezione dedicata alle occorrenze. È chiaro che se si specifica un gruppo come elemento da ripetere, non ha senso specificare il nome di un attributo e viceversa.

Per chiarire quanto esposto nei periodi precedenti, si riporta un semplice esempio relativo a un'interfaccia atta a erogare dei dati rappresentanti il listino prezzi previsto da un determinato hotel organizzato in base alle stagioni. In particolare nel listino sono riportati i prezzi relativi a una sistemazione standard in stanza singola e doppia.

Dopo aver letto il presente paragrafo, i lettori più attenti potrebbero chiedersi: "perché non rappresentare direttamente tali interfacce attraverso lo UML e in particolare attraver-

Tabella 5.3

so il formalismo del diagramma delle classi?". Si tratta di un quesito legittimo. In effetti una rappresentazione grafica apporterebbe moltissimi vantaggi:

- quelli tipici dei formalismi grafici (immediatezza, maggiore chiarezza, migliore memorizzazione, ecc.);
- altri di integrazione con i tool commerciali di supporto (i vari diagrammi relativi alle interfacce potrebbero essere associati alle relazioni attori/use case e quindi essere visualizzati attraverso un doppio click eseguito sulla relazione stessa);
- agevolare il "tracciamento" del manufatto durante le varie fasi del processo, e così via..

Tutto quindi farebbe propendere per tale soluzione, e in effetti è quella più apprezzata dal personale tecnico. A fronte di questi vantaggi però esiste un unico grande svantaggio: gli utenti di solito si sentono più a loro agio con fogli elettronici, documenti e *template*, piuttosto che con i diagrammi UML. Poiché in questa fase gli utenti recitano ancora un ruolo fondamentale, la decisione sul formalismo da utilizzare deve essere assolutamente basata su ciò che riesce a stimolare e coinvolgere maggiormente l'utente.

I famosi test case

I test case, come lascia presagire il nome, sono artifact fondamentali per la fase di test, uno degli ultimi stadi del processo di sviluppo del software, o di ogni iterazione, qualora si utilizzi un processo iterativo ed incrementale. Ciononostante si è deciso di presentarli in questo capitolo poiché esiste un vincolo singolare tra i casi d'uso ed i casi di test: i primi descrivono le funzionalità che il sistema dovrà realizzare mentre i secondi servono ad accertarsi che il sistema fornisca realmente i servizi promessi.

Nel mondo dell'ideale si desidererebbe verificare ciascuna linea di codice di cui un sistema è costituito e ogni possibile percorso. Ciò indubbiamente favorirebbe l'individuazione di anomalie, però probabilmente un tale processo di verifica potrebbe risultare un "minimo" oneroso e non sempre possibile, sia perché tipicamente si utilizzano anche sottosistemi/librerie di cui non si hanno i codici sorgenti, sia perché i possibili percorsi tendono ad essere un'infinità, magari numerabile, ma comunque di infinità.

Tale approccio empirico, tra l'altro, potrebbe anche finire per non raggiungere l'obiettivo primario: verificare che il sistema realizzi le funzionalità di cui l'utente ha realmente bisogno. Per questo motivo è necessario realizzare un opportuno piano di test che tenga

conto di specifiche priorità basate principalmente sulle necessità degli utenti, al fine di conferire maggiore enfasi ai test relativi alle funzionalità eseguite più frequentemente.

In generale, i test andrebbero eseguiti in ciascuna fase del ciclo di sviluppo del software e non appena un artifact si renda disponibile o subisca degli aggiornamenti. L'obiettivo è neutralizzare eventuali errori o lacune prima che i relativi effetti possano essere amplificati dal passare del tempo o meglio dall'evoluzione viziata del processo di sviluppo del software. Risulta fondamentale poi che il personale che esegue le varie verifiche non sia lo stesso che ha prodotto il manufatto: ciò perché i modelli mentali utilizzati per la verifica sono gli stessi di quelli adottati per la produzione e quindi non si tende a effettuare test che violino la struttura del manufatto stesso.

I test di maggiore importanza, tuttavia, sono indubbiamente quelli eseguiti dopo ogni *build* (implementazione ottenuta come risultato di una nuova iterazione).

I manufatti di maggiore interesse nella fase di verifica sono indubbiamente i test case, i quali rivestono molta importanza non solo nei processi più accademici (come il RUP della Rational), ma anche in quelli più "leggeri" come XP (eXtreme Programming). In effetti, quest'ultimo prevede la definizione dei test case, magari direttamente attraverso il codice (test unit), come prerequisito irrinunciabile dell'attività di codifica (quanti programmatori XP ricordano questo "dettaglio"?).

Nei processi tradizionali, la prima versione dei test case è ottenuta direttamente dal modello dei casi d'uso. Questo fornisce un ottimo punto di partenza, ma da solo non è assolutamente sufficiente: in un sistema tante cose possono non funzionare correttamente — come enunciato chiaramente dalla "legge di Murphy": "se c'è la possibilità che qualcosa vada male, sicuramente andrà male — e non tutte si prestano a essere evidenziate per mezzo dei casi d'uso.

Certo è che il sistema deve essere verificato approfonditamente prima di essere consegnato. Nessuno — teoricamente — gradirebbe fornire un sistema non robusto, pronto ad andare in crash, o in uno stato indefinito, alla prima situazione anomala non considerata.

> Si provi a considerate cosa potrebbe accadere in settori diversi, magari nella fabbricazione di automobili… Si potrebbe candidamente confidare al cliente che la

> versione disponibile del sistema frenante dell'autovettura da lui acquistata non funziona correttamente… "ma poco male, perché si tratta di un'anomalia già riscontrata e di cui a breve verrà resa disponibile apposita patch!"…

In un sistema complesso molte sono le caratteristiche da verificare: requisiti funzionali, quelli non funzionali, interazione con l'utente, ecc.

Il tutto però deve avvenire in modo organico secondo un piano sistematico studiato a priori. In molte organizzazioni la fase di test è un'attività da svolgersi nell'eventuale lasso di tempo che intercorre tra l'ultima release del sistema — congenitamente rilasciata in ritardo — e la data di consegna del progetto. In tali ambienti l'algoritmo di test utilizzato è il famoso criterio di "ricerca a carponi": si provano a eseguire alcuni servizi con qualche dato di prova pensato al momento (tale esercizio in gergo è indicato con la frase "fare il giro del sistema").

Per alcuni manager l'attività di test è dovuta alla negligenza del team di sviluppo, dimenticando che è semplicemente impossibile verificare alcune parti prima dell'integrazione del sistema e che, comunque, come in ogni altra attività *errare humanum est*. Chiaramente in tutto c'è un fondo di verità, e indubbiamente profondere attenzione e impegno maggiori al proprio lavoro sicuramente riduce il numero di potenziali errori. Inoltre il paradigma Object Oriented, grazie alla naturale tendenza a produrre una miriade di piccoli oggetti cooperanti e quindi più facilmente verificabili, aiuta a realizzare sistemi di maggiore qualità…

Nonostante ciò vale la pena utilizzare un criterio sintetizzato da una celebre frase proferita dall'ex presidente statunitense Reagan nella cornice storica dei difficili rapporti con l'allora URSS: "Trust but verifiy" ("Fidati ma verifica").

> La corretta e completa definizione del piano di test esula dai fini del presente testo, ma si è ritenuto opportuno fornire alcune utili indicazioni. Su questo argomento esistono molti trattati, tra i quali molto importante è quello proposto dallo standard IEEE n1998d.

Nel contesto dei processi di sviluppo iterativi e incrementali, lo sviluppo del piano di test è più complesso giacché va eseguito ad ogni iterazione. In particolare, in ogni progettazione del piano di test bisogna svolgere attività aggiuntive come:

- esclusione di test case divenuti obsoleti con la nuova versione del sistema;
- realizzazione di nuovi test case;
- aggiornamento di specifici test case.

Tipicamente un piano ben congegnato è composto almeno dalle seguenti sezioni:

- **Definizione degli obiettivi**. Si tratta di una vera e propria introduzione nella quale si dichiara quale siano il sistema da verificare, gli obiettivi del piano, e così via.
- **Dettaglio dei test**. In questa sezione si stabilisce quali parti del sistema si intende verificare e quali no. Probabilmente, il modo migliore per rappresentare queste informazioni è ricorrere a una tabella in cui elencare le verifiche previste dal piano di test e quelle invece trascurate.
- **Allocazione delle risorse**. Questa è la sezione in cui si stabilisce a chi toccherà il gravoso compito della verifica del sistema, chi ne assumerà la responsabilità, ecc.
- **Elenco dei rischi**. Nella sezione dedicata ai rischi è necessario identificare i fattori che potrebbero annullare, o comunque ridimensionare, quanto sancito nel piano dei test.

Il "modello" di test non è composto esclusivamente dai test case, sebbene ne rappresenti l'artifact fondamentale, bensì anche dalle test procedure nonché da eventuali componenti di test. Le *test procedure* specificano come eseguire i test case, ossia specificano se eseguirne uno o una serie, o tutti, se eseguire completamente ciascuno di essi o solo particolari sezioni e così via.

I componenti di test servono per eseguire alcuni test in maniera automatica, per semplificare la realizzazione di altri: per esempio potrebbe rendersi necessaria la realizzazione di un componente atto a simulare un sistema remoto con il quale quello oggetto di studio deve interagire, la configurazione di un apposito sistema di simulazione dell'utente, *robot*, e così via.

Come visto in precedenza, si prestano a fornire il punto di partenza per la realizzazione del piano di test del sistema poiché i casi d'uso specificano i requisiti funzionali del sistema.

In particolare la descrizione del flusso di eventi di ciascun caso d'uso prevede la descrizione della successione di passi che globalmente realizzano il servizio (scenari principale e alternativi). Non solo; qualora l'esecuzione del flusso possa essere inquinata dal verificarsi di situazioni anomale, nel flusso stesso è necessario riportare la descrizione di tali eccezioni corredate dalla relativa gestione.

In sintesi la descrizione del flusso degli eventi dei casi d'uso permette di programmare sia i **positive** (verifiche positive), che i **negative test** (verifiche negative).

I primi servono ad accertare la **correttezza** del sistema: in condizioni ideali (dati di ingresso validi, ambiente completamente operativo, ecc.) il sistema eroga correttamente il

servizio esaminato e quindi produce i risultati attesti. I secondi in maniera diametralmente opposta ai primi, concorrono a verificare **robustezza** e/o *fault tolerance* del sistema, ossia si verifica la capacità del sistema di riconoscere e gestire correttamente situazioni anomale: dati di input non corretti, ambiente parzialmente non operativo (per esempio connessioni non funzionanti), e così via.

> Brevemente con il termine *correttezza* si fa riferimento alla capacità del sistema di fornire i servizi così come definiti nel documento delle specifiche. Sebbene si tratti di una caratteristica imprescindibile del sistema — non dovrebbe essere di particolare interesse disporre di un sistema, magari molto accattivante, efficiente, ecc., se poi semplicemente non fornisce i servizi richiesti — la pratica dimostra che non è così facile da conseguire. Come analizzato nei capitoli precedenti, già il prerequisito per ottenere la correttezza — corretta analisi e definizione dei requisiti utente — è un'attività molto insidiosa e per certi versi controversa.
>
> Con il termine di *robustezza* si fa riferimento alla capacità del sistema di riconoscere situazioni anomale e quindi di gestirle correttamente. Pertanto, dovrebbe essere chiaro che la robustezza è il logico completamento della correttezza. Anche questa proprietà importantissima del sistema nasconde diverse insidie, alcune condivise con la correttezza del sistema (difficoltà di fornire delle specifiche precise), altre relative all'eccessiva genericità del concetto di "situazione anomala" (mancato rispetto di business rule, malfunzionamento di dispositivi hardware e software, ecc.).

Nella tab. 5.4 viene riportato un modulo in grado di semplificare la descrizione dei casi di test. L'utilizzo del modulo di tab. 5.4 dovrebbe risultare piuttosto intuitivo. In ogni modo di seguito viene fornita qualche informazione supplementare.

La prima sezione è dedicata alla catalogazione/reperimento del modulo. In particolare vi trovano posto il codice, la descrizione del test case, la data e la versione.

Dall'analisi dei campi successivi è possibile evidenziare la diretta dipendenza tra i casi d'uso e quelli di test: una modifica ai primi, tipicamente, si ripercuote sui secondi. Il campo versione si rende necessario al fine di tener traccia delle eventuali variazioni del modello.

La sezione successiva del template è utilizzata al fine di evidenziare da quale caso d'uso dipende il test case descritto nel modulo.

In merito al processo di attribuzione delle priorità è necessario conferire particolare attenzione alle precedenze stabilite più o meno implicitamente dall'utente. Le priorità, in ultima analisi, permettono di ripartire più opportunamente l'intervallo di tempo a disposizione della fase di test, dedicando maggiore attenzione alle funzionalità con priorità più elevata. Queste vanno assegnate considerando sia la frequenza con la quale ciascuno use case viene eseguito dai relativi attori, sia alla criticità della funzione stessa (si consideri per esempio il caso d'uso relativo all'aggiornamento del conto corrente).

Tabella 5.4 — *Modulo per la semplificazione della descrizione dei casi di test*

Figura 5.8 — *Legame di dipendenza tra Use Case e Test Case.*

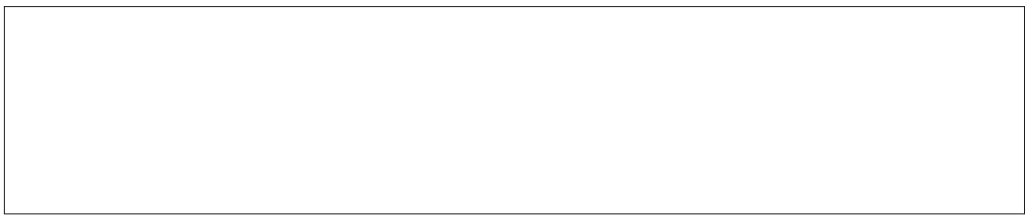


Figura 5.9 — *Schematizzazione di un flusso degli eventi di un generico caso d'uso.*

Alcuni tecnici particolarmente scaltri utilizzano una tecnica che prevede l'assegnamento di opportuni "pesi" alle valutazioni degli utenti… Chiaramente l'entità del peso è direttamente proporzionale all'importanza del particolare utente. Un conto è stare a sentire le lagnanze di un povero utente frustrato, un altro è prestare attenzione alle lamentele di un manager irritato.

Per ciò che concerne il campo denominato Set up, esso è utilizzato per riportare l'elenco delle operazioni propedeutiche all'esecuzione del test. Se, per esempio, si dovesse verificare il funzionamento di servizi relativi a un sistema di account bancario, alcune operazioni propedeutiche potrebbero consistere nell'inserire un nuovo account intestato al sig. Paolo Rossi, quindi effettuarvi un trasferimento di importo *x*, e così via. Lo svolgimento della procedura di Set up (magari eseguita attraverso un apposito componente di test) è volto a porre il sistema in uno stato iniziale predefinito (come per esempio la memorizzazione di prestabiliti dati di prova nella base dati) da utilizzarsi come base di partenza per eseguire i vari test.

Per ciò che concerne la sezione dedicata alla descrizione del test, e in particolare il campo scenario, è necessario aver presente come è di solito costituito un flusso degli eventi di un generico caso d'uso (come mostrato nel capitolo precedente).

Un generico flusso degli eventi prevede le tre tipologie (fig. 5.9):

- Principale: è l'unico obbligatorio e descrive la sequenza delle attività che permettono di fornire il servizio. In questa sequenza viene assunto che tutto "funzioni" bene e che quindi non intervenga alcuna eccezione;

- Alternativo: si tratta di una variante al flusso principale, non viziata da errori, che quindi è in grado di erogare del servizio, sebbene venga utilizzato un percorso alternativo;

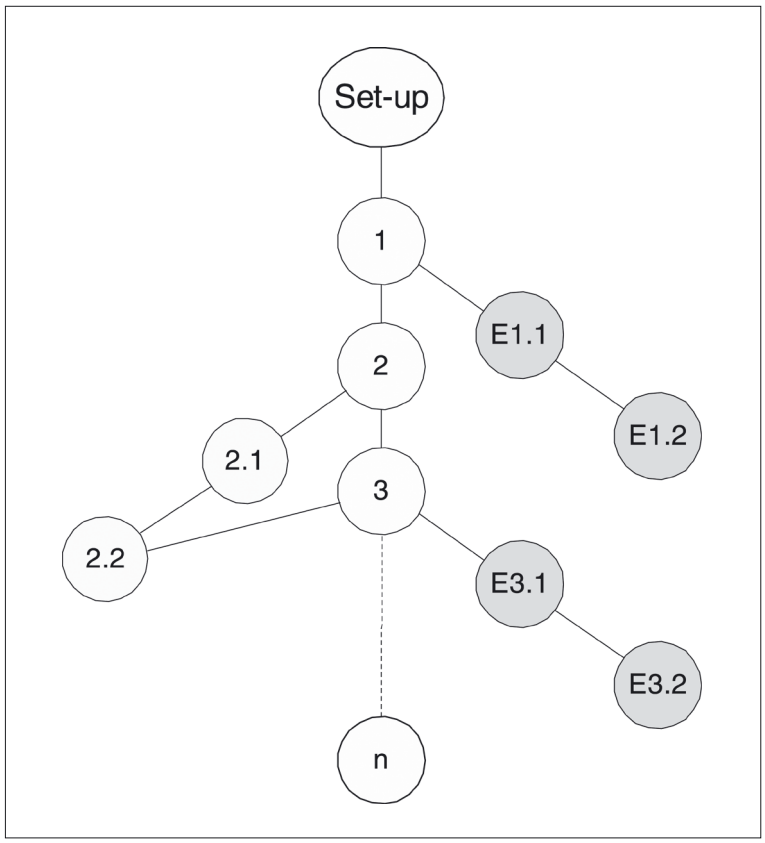
- Errore: evidenzia determinati errori che possono verificarsi durante l'esecuzione dei predetti flussi. L'insorgenza di un errore, evidentemente, inibisce l'erogazione del servizio. Ciascuno di essi viene corredato dall'insieme delle azioni che il sistema deve compiere per la relativa gestione.

A questo punto è possibile chiarire cosa si intenda con i campi scenario da inserire nel modulo dei test case. Essi rappresentano percorsi di verifica, ossia opportune sequenze di passi selezionati dal corrispondente caso d'uso.

Tabella 5.5 — *Frammento di caso d'uso relativo ai vari flussi (principale, alternativi e di errore).*

Flusso principale	
1.	Descrizione del primo passo
2.	Descrizione del secondo passo
…	…
N	Descrizione dell'ennesimo passo
1° Flusso alternativo	
2.1.	1° passo del 1° flusso alternativo relativo al 2° passo del flusso principale.
2.2	2° passo del 1° flusso alternativo relativo al 2° passo del flusso principale.
2° Flusso alternativo	
i.1	1° passo del 2° flusso alternativo relativo al i-esimo passo del flusso principale.
…	…
i.n	n-esimo passo del 2° flusso alternativo relativo al i-esimo passo del flusso principale.
1° Flusso di errore	
1.1	1° passo del 1° flusso di errore relativo al 1° passo del flusso principale.
…	…
1.n	n-esimo passo del 1° flusso di errore relativo al 1° passo del flusso principale.
2° Flusso di errore	
…	…

Figura 5.10 — *Esempio di albero dei percorsi da verificare che potrebbe scaturire dall'analisi di un caso d'uso. Come si può notare vanno verificati tutti i percorsi: il principale e ogni percorso alternativo e di errore.*



Gli *scenario* da verificare (fig. 5.10) su base obbligatoria sono quelli che permettono di erogare il servizio:

- flusso principale: passi 1, 2, 3, ..., n;
- primo flusso alternativo: 1 , 2, 2.1, 2.2, 3, ..., n e così via.

Poiché però si è interessati anche a verificare che il sistema sia in grado di riconoscere e gestire situazioni anomale (verifiche negative), altri scenari da verificare (fig. 5.10) sono dati dalle sequenze: 1, E1.1, E1.2 e 1, 2, 3, E3.1, E3.2, dove la lettera E maiuscola evidenzia appunto che si tratta di un passo appartenente ad un flusso di errore.

Chiaramente il numero di possibili scenario in un caso reale è decisamente molto elevato e, verosimilmente, è impossibile verificarli tutti. Pertanto risulta importante individuare gli *scenario* più frequenti, quelli più critici e così via.

Per ogni scenario, oltre a riportare una breve descrizione delle azioni intraprese (per esempio: selezionato il conto corrente del sig. Paolo Rossi, effettuato un pagamento per un importo *y*, ecc.), si vuole evidenziare l'esito del test. In caso di insuccesso (il sistema non si è comportato come previsto) è opportuno catalogare l'errore assegnandovi un identificatore univoco.

Ciò è utile sia per riportare una breve descrizione nell'apposita sezione, sia per l'utilizzo di software che permettono di memorizzare gli errori scaturiti e di assegnarli automaticamente al personale competente.

Le rimanenti sezioni del modulo risultano così intuitive da non richiedere ulteriori spiegazioni.

Sebbene molti tecnici siano portati a credere che il testing sia un'attività piuttosto semplice da compiersi su base randomica (il famoso criterio a carponi), magari da assegnarsi all'ultimo arrivato, in realtà le cose non sono esattamente così.

Un piano di test adeguato richiede una sofisticata comprensione del sistema oggetto di verifica. È necessario essere in grado di sviluppare una vista astratta di quelle che sono le dinamiche del flusso del controllo del sistema, comprendere lo spazio del problema, capire i requisiti funzionali e non, e così via. In sintesi, anche se non si eseguono attività di disegno, è necessario avere una comprensione del problema e del sistema paragonabile a quella di un architetto.

Per terminare questa breve sezione dedicata ai test case, si vuole fornire un suggerimento ai tecnici tester: munirsi di un semplice registratore vocale da utilizzare durante l'esecuzione dei vari test. Si tratta di uno strumento molto produttivo per tenere traccia della storia dei test eseguiti, ossia della successione di azioni che hanno generato l'errore. (Inserito record *X*, eseguita funzione *Y*, aggiornato dato *Z*, e così via). A chi non è mai capitato di analizzare un report prodotto dal team di test, magari molto dettagliato riportante anche la descrizione minuziosa dell'eccezione generata, con l'unico neo di non sapere come riprodurlo? Un errore, per essere studiato e quindi risolto, come minimo deve poter essere riprodotto, altrimenti la relativa analisi risulta decisamente molto complessa o addirittura impossibile: un errore riproducibile è un errore mezzo risolto!

Nella decisione del tempo e delle risorse umane da dedicare alle fasi di test, si tenga bene a mente che ciò che non si spende prima, si paga dopo… ma con gli interessi.

Requisiti non funzionali

Nel corso dell'illustrazione dei processi di sviluppo del software, nonché nei capitoli precedenti, si è menzionato frequentemente il manufatto relativo ai requisiti non funzionali. Si tratta di informazioni di importanza vitale, tipicamente analizzate e definite durante la fase di analisi dei requisiti, che però, ahimè, non sempre, e non tutte, si prestano a essere rappresentate per mezzo di un ben definito formalismo grafico. Pertanto, generalmente, la relativa descrizione è realizzata per mezzo del classico documento.

I requisiti non funzionali rientrano nel contesto più generale dell'analisi delle specifiche di un sistema software (SRS *Software Requirements Specification*, specifica dei requisiti roftware) e pertanto è possibile reperire molta documentazione al riguardo. Particolarmente degne di nota sono lo standard IEEE Std 830-1998 e la relativa rielaborazione (*Supplementary Specification*, specifiche supplementari) incorporata nel processo della Rational (RUP).

La rappresentazione dei requisiti del sistema, sia pure limitando l'attenzione a quelli non funzionali, è un'attività molto importante nel contesto del ciclo di vita del software. In ultima analisi si specificano le caratteristiche alle quali l'utente è interessato. Il relativo adempimento, quindi, concorre a determinare il successo o l'insuccesso dell'intero progetto.

Il documento dei requisiti non funzionali è di particolare valore nelle fasi di disegno, implementazione e test del sistema. In questo contesto non solo è importante realizzare le funzioni definite nel modello dei casi d'uso, ma bisogna rispettare anche vincoli legati a performance, sicurezza, affidabilità, ecc.

I requisiti individuati, generalmente, vengono suddivisi in tre categorie di importanza:

- **essenziali**, ossia requisiti che devono essere necessariamente soddisfatti per l'accettazione del sistema; eventuali insolvenze determinano il rifiuto del sistema da parte del cliente;

- **condizionali**, ovvero requisiti che se soddisfatti permettono di realizzare un sistema evoluto, ma che, se non presenti, non determinano l'insuccesso del progetto;

- **opzionali**, si tratta di requisiti che non forniscono un considerevole valore aggiuntivo e quindi potrebbe o meno valer la pena soddisfare.

Nell'elaborazione del documento dei requisiti non funzionali bisogna porre particolare attenzione ad alcune proprietà molto importanti, come:

- **correttezza**: il documento viene considerato corretto se, e solo se, ciascun requisito menzionato rappresenta una reale caratteristica che il sistema dovrà soddisfare;

- **completezza**: il documento dei requisiti non funzionali è definito completo se sono incorporate tutte le caratteristiche di interesse (ciascun requisito significativo, vincoli di disegno, prestazioni, …). La definizione dello standard IEEE prevede la presenza di sezioni supplementari come per esempio quella dedicata alle interfacce esterne, che però, considerata l'importanza, probabilmente è più opportuno analizzare in un apposito documento;

- **precisione**: un documento dei requisiti è preciso (ossia non ambiguo) se, e solo se, ciascuna delle caratteristiche richieste si presta a essere interpretata in modo univoco, non solo da chi ha prodotto il documento, ma anche dai lettori. Condizione necessaria, ma non sufficiente, è che ciascun requisito disponga di un nome univoco. Utilizzando come strumento di specifica il linguaggio naturale (tipicamente l'inglese che quindi poi non per tutti è così naturale…), il quale per definizione non è preciso, la caratteristica di non ambiguità non è sempre facilmente ottenibile;

- **consistenza** nel senso di consistenza interna. In particolare il documento è internamente consistente se, e solo se, nessun requisito risulta in conflitto con un altro.

Come ogni documento di qualità che si rispetti, è opportuno includere la classica introduzione al fine di fornire un'appropriata overview del documento. Se ne dichiarano gli obiettivi, l'area di interesse (il sistema a cui fa riferimento il documento), il vocabolario con le definizioni, gli acronimi utilizzati, e così via.

Terminata la sezione "cerimoniale", comunque importante e d'obbligo, si inizia la descrizione delle caratteristiche non funzionali di interesse.

In generale la prima ad essere descritta è l'**usabilità** (*usability*) del sistema che prevede informazioni del tipo:

- **tempo necessario all'addestramento** del personale che dovrà utilizzare il sistema (utenti). Particolarmente importante è distinguere tra le varie tipologie di utente (amministratore, utente generico, ecc.), nonché definire i vari livelli di produttività (per esempio **marginale** quando l'utente è in grado di eseguire le attività basilari, oppure **operativa** quando l'utente è in grado di svolgere tutte le attività richieste durante una tipica giornata lavorativa, e così via);

- **tempo richiesto mediamente all'utente per lo svolgimento delle attività tipiche**: si individuano (dalle informazioni riportate nella descrizione dei casi d'uso) i servizi

richiesti più frequentemente e quindi, per ciascuno di essi, si determina la stima del tempo necessario per l'espletamento, considerando sia il tempo umano, sia quello richiesto dall'elaborazione;

- **requisiti base di utilizzabilità**: si descrivono gli ambienti conosciuti dall'utente che, quindi, concorrono a rendere il sistema maggiormente amichevole (*user-friendly*);

- **direttive per la definizione standard di interfacce utente** (Sun user interface guidelines, IBM's CUA standars, GUI standards per Microsoft Windows 95, ecc.).

Un'altra caratteristica molto importante è relativa all'**affidabilità** del sistema (*reliability*), la quale, tipicamente è definita attraverso le seguenti informazioni:

- **disponibilità**: in particolare è necessario specificare la percentuale di disponibilità del sistema (esempio 96%), oppure formule del tipo 24 × 7 (24 ore per 7 giorni alla settimana) e così via. Altrettanto importante è definire eventuali non disponibilità e stime del funzionamento degradato del sistema. Degradato in quanto parallelamente allo svolgimento dei task utente si eseguono operazioni di manutenzione che rendono momentaneamente non disponibili (in parte o todo) determinate componenti del sistema. Per esempio in un sito per il commercio elettronico durante l'aggiornamento del listino dei prodotti alcune tabelle/record non possono essere accedute.

- **tempo medio tra due errori consecutivi** (MTBF, *Mean Time Between Failure*). La definizione di questa stima è piuttosto singolare. Chiaramente tutti desidererebbero un valore tendente all'infinito + 1. Ciononostante, molte cose possono andare storte: errori software, guasti hardware, caduta delle comunicazioni, ecc. Chiaramente bisogna porre molta attenzione nella definizione di questa informazione (*verba volant scripta manent*, e se la fortuna è cieca la sfortuna ci vede benissimo…);

- **tempo massimo di non disponibilità** consentito al sistema per la riparazione di eventuali guasti (MTTR, *Mean Time To Repair*). Chiaramente in questo contesto si vorrebbero valori prossimi allo zero, sebbene valori tipici sono dell'ordine delle decine di minuti/ore.

- **accuratezza** o **risoluzione** che richiedono misure in termini di eventuali standard presenti presso l'organizzazione o l'area di business oggetto di studio. Lavorando in sistemi che trattano copiosamente dati numerici, potrebbe risultare importante definire l'accuratezza in termini di posizioni decimali valutate e arrotondamenti. Queste informazioni potrebbero assumere un valore importantissimo qualora i numeri

rappresentino cifre di denaro e gli arrotondamenti possano fare la differenza (ogni riferimento agli ambienti bancari è puramente voluto…);

- **bug** e/o **difetti**: sono stime definite in rapporto alle linee di codice (si consiglia di aggiungere molte linee di commento) e vengono effettuate per le diverse categorie di errore: minori, significanti, critici. Chiaramente è necessario definire formalmente cosa si intende, nel contesto del sistema oggetto di studio, con tali categorie di errore.

Un altro insieme di requisiti non funzionali di particolare interesse è quello relativo alle procedure di **backup**. In questa sezione è necessario specificare quale strategia di backup si intende adottare, quando fisicamente eseguire i vari backup, su quale tipo di dispositivo, ecc. Altrettanto importante è dichiarare le procedure da attuare qualora si renda necessario eseguire il recovery, e la porzione di informazioni che si rischia di perdere nell'intervallo che intercorre tra l'esecuzione dell'ultimo backup e il sopraggiungere di fallimenti catastrofici. Esistono sistemi per i quali la perdita di dati, anche se relativa a pochi di minuti, è così gravosa — si considerino per esempio i sistemi centrali delle compagnie aeree — che, al fine di correre il minor rischio possibile, si realizzano configurazioni hardware particolarmente ridondanti, come per esempio 12 unità di backup in linea, con tempo di disallineamento dell'ordine dei 10 minuti.

Altro gruppo di requisiti non funzionali decisamente importanti, specie per sistemi Internet o comunque esposti a connessioni remote, è quello della **sicurezza** (*security*). Si tratta di requisiti così importanti che, probabilmente, meriterebbero anch'essi di essere trattati in un apposito documento. In questa sezione è necessario definire le politiche di riconoscimento (autenticazione) degli attori (coppia login/password, firma digitale, riconoscimento della porta fisica richiedente la connessione, ecc.), le diverse tipologie di utente con la relativa matrice dei servizi fruibili, la modalità con cui proteggere i dati, sia quelli memorizzati permanentemente, sia quelli scambiati per mezzo di connessioni, algoritmi di crittazione, difese da parte di eventuali hacker, log con la storia delle azioni eseguite dagli utenti, ecc.

Il problema della sicurezza è diventato assolutamente impellente con l'avvento di Internet. Infatti, se da un lato il web fornisce alle imprese la possibilità di disporre di un network mondiale a costi contenutissimi, dall'altro si tratta di un rete "libera", virtualmente aperta a tutti. È necessario, quindi, fornire opportuni meccanismi atti a proteggere i dati circolanti nonché l'organizzazione stessa.

Una sezione assolutamente tipica per il documento dei requisiti non funzionali è quella relativa alle **performance del sistema**.

In questa sezione è necessario riportare informazioni del tipo:

- **tempo di risposta delle transazioni più significative**. È lecito attendersi che i servizi richiesti più frequentemente e quelli assoggettati a stringenti vincoli temporali, siano erogati il più rapidamente possibile: la definizione di tale valore richiede l'individuazione delle funzioni ai requisiti testé enunciati e quindi la relativa stima;

- **throughput** (numero di lavori eseguiti nell'unità di tempo, tipicamente transazioni per secondo). Chiaramente non ha senso stimare questo valore per tutte le transazioni, ma solamente per quelle ritenute più significative coerentemente a quanto definito nel punto precedente: spesso si preferisce fornire questi insiemi di valori sia in condizioni normali sia in condizioni di picco;

- **capacità**, definita in termini di numero di utenti simultanei che il sistema deve essere in grado di gestire in condizioni normali e in condizioni di funzionamento degradato da attività di manutenzione;

- **utilizzo delle risorse** ovvero percentuale di utilizzo delle principali risorse del sistema: connessioni, database, server, ecc.

Un'altra sezione degna di interesse è quella relativa alla **supportabilità** (*supportability*) del sistema. Con questo termine di fa riferimento a tutti quegli standard e a quelle direttive che semplificano la manutenibilità del sistema da costruire: standard di codifica, componenti atti al controllo del sistema, *naming convention* (standard per l'attribuzione dei nomi), *change cases*, accessi per la manutenzione, e così via.

Proseguendo nell'analisi dei requisiti non funzionali è necessario specificare eventuali **vincoli di disegno** esistenti. Esempi tipici sono: linguaggi di programmazione, paradigma utilizzato (Object Oriented, component-based, ecc.), architettura di riferimento, database management system, network, piattaforma software e hardware, legacy system presenti, ecc. Da notare che il dettaglio di queste informazioni dovrebbe essere presente nel SAD (*Software Architecture Document*, documento di architettura software).

Un elemento spesso dimenticato, che invece assume una certa importanza per l'utente, è relativo alla **documentazione utente**. Pertanto è necessario definire i requisiti relativi alla documentazione online, sistemi di aiuto, manuali utente, e così via.

Un altro gruppo di requisiti importante è relativo ai **componenti software/sistemi** della cui licenza si dispone che devono essere necessariamente utilizzati o che si intende utilizzare. Per esempio, in determinati progetti, potrebbe essere obbligatorio utilizzare uno specifico database management system perché magari già esiste un'applicazione che utilizza tale database con cui bisogna interagire, o perché si ha alle proprie dipendenze perso-

nale esperto o semplicemente perché l'azienda dispone delle necessarie licenze (probabilmente è più opportuno inserire anche questa tipologia di informazioni nel SAD).

Il template del documento relativo ai requisiti non funzionali (detto *Supplementary Specifications*, specifiche supplementari) provvisto dalla Rational, come accennato in precedenza, prevede la definizione delle interfacce fornite dal sistema, sia quelle utente, sia quelle dirette verso altri sistemi software ed hardware.

Come ripetuto più volte, si tratta di un manufatto di importanza così elevata che probabilmente è il caso di realizzare un apposito documento.

Le ultime sezioni del documento dei requisiti non funzionali sono dedicate a "requisiti" relativi a vincoli di utilizzo di software, eventuali restrizioni, copyright e standard da utilizzare definiti nell'ambiente business oggetto di studio. Quindi è necessario analizzare eventuali standard/vincoli legali, come per esempio avvisi di copyright da citare, eventuali notifiche, e così via.

Il documento viene concluso con l'obbligatoria sezione dedicata ai riferimenti.

Le famose regole del business

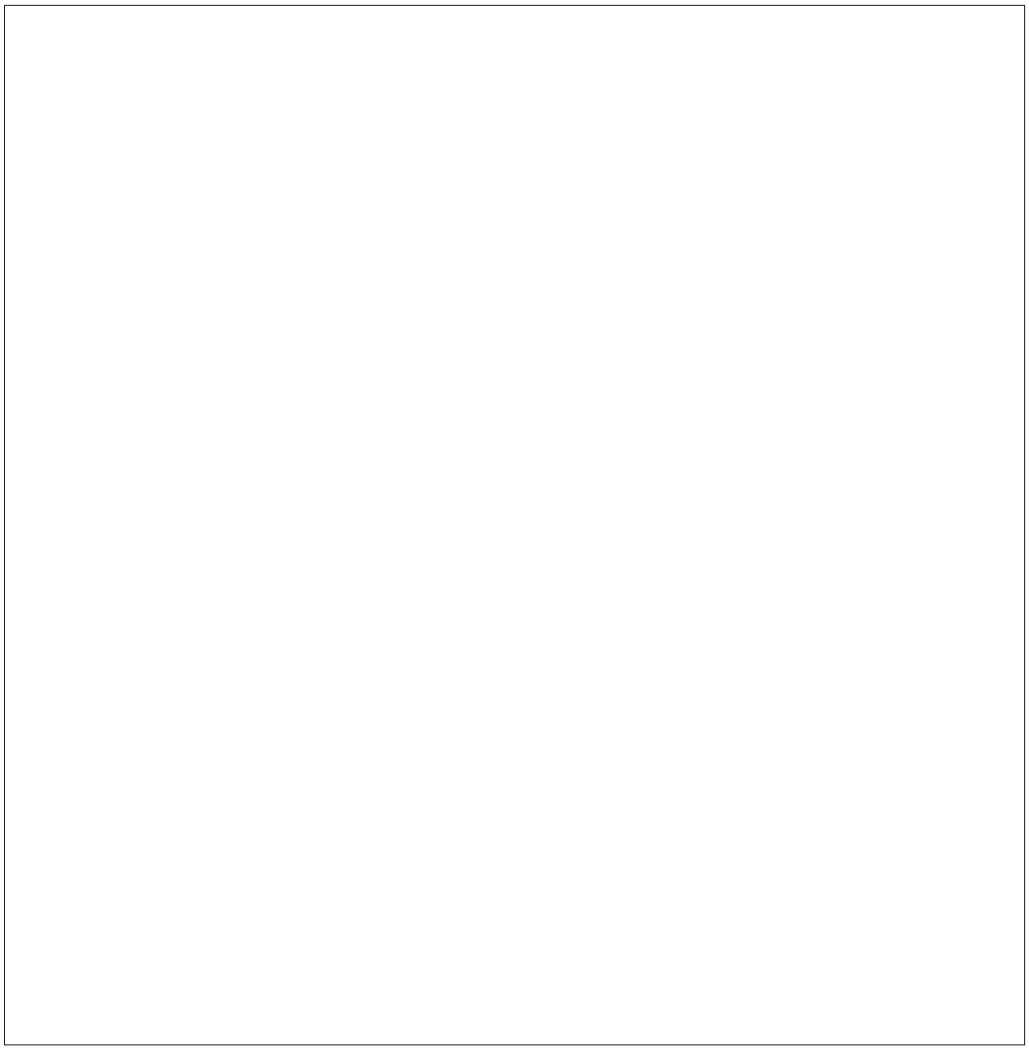
Si presenta ora un manufatto (*artifact*) di notevole importanza per la modellazione dei requisiti utente e, più in generale, per l'intero processo di sviluppo: le famose regole del business (*business rules*). Si tratta nuovamente di un concetto che, anche per via della difficoltà di fornirne una definizione precisa, si presta facilmente a essere equivocato. In questo paragrafo si tenterà di fornire informazioni ed esempi reali nella speranza di contribuire a neutralizzare la confusione che regna attorno al concetto. Al momento in cui viene scritto il libro, anche le regole del business sono fonte di dibattito, specie per ciò concerne le modalità di documentazione.

In generale con "regole del business" ci si riferisce a princìpi, policy, leggi, dettagliati algoritmi di calcolo, relazione tra oggetti, ecc. che influenzano direttamente il dominio del problema che il sistema, in qualche misura, dovrà automatizzare. Pertanto rappresentano informazioni che, in ultima analisi, dovranno confluire nell'implementazione del sistema. Non a caso nelle architetture moderne multi-tier è presente uno strato denominato Business Service.

Come si può notare la definizione è piuttosto ampia: si va da leggi sia scritte che non, a policy, a specifiche procedure di calcolo, ecc. Quello che è sicuro è che queste hanno un notevole impatto sul sistema da realizzare e che quindi è assolutamente necessario considerarle e documentarle appropriatamente. Il rischio che si corre nel non analizzarle propriamente è di realizzare un sistema, magari efficientissimo e ben congegnato, ma che semplicemente non risponde alle necessità dell'utente. Per esempio, viola le policy dell'azienda committente o non contempla regole che, seppur non formali, di fatto hanno un notevole impatto nel business, e così via.

Da quanto specificato fino a questo punto, molte regole del business non sono altro che particolari requisiti utente. Allora una domanda che potrebbe sorgere è "perché utilizzare un manufatto diverso da quello dei casi d'uso? E, in caso, quale criterio utilizzare per decidere che un gruppo di funzionalità/specifiche debbano essere descritte per mezzo di un apposito manufatto?".

Figura 5.11 — *Diagramma delle business rule nel contesto del processo di sviluppo del software.*



Le motivazioni alla base della realizzazione di un apposito manufatto sono diverse. Una prima giustificazione è relativa alla tipologia della regole di business. Qualora per esempio si debba descrivere dettagliatamente un algoritmo, evidentemente i casi d'uso non rappresenterebbero la modalità migliore. Verosimilmente in questi casi è opportuno ricorrere a formalismi più adeguati come quello dei diagrammi delle attività. Altri metodi che possono essere utilizzati sono relativi al livello di dettaglio e alla complessità delle norme da descrivere. Qualora si abbia la necessità di specificare dettagliatamente delle regole, e nei casi in cui queste risultino particolarmente complesse o articolate, potrebbe essere opportuno descriverle in un apposito manufatto utilizzando i formalismi ritenuti, caso per caso, più appropriati (pseudocodice, tabelle diagrammi di flusso, ecc.). È opportuno che i casi d'uso non scendano eccessivamente nei dettagli, sia per non dar luogo alle anomalie descritte nei precedenti capitoli, sia per non perdere di vista l'aspetto generale della funzionalità che si vuole illustrare. Ancora, per questioni legate alla minimizzazione del lavoro di revisione dovuto a cambiamenti delle specifiche utente, potrebbe essere intelligente isolare requisiti potenzialmente soggetti a ripensamenti.

Per esempio, una specifica dell'area della sicurezza che prevederebbe di sospendere l'account di un utente a seguito di 3 consecutivi fallimenti del log-in, rappresenterebbe una buona candidata a divenire business rule. Ciò permetterebbe di neutralizzare gli aggiornamenti da apportare qualora l'utente decidesse di aumentare a 4 o a 6 il numero di consecutivi fallimenti, in quanto le modifiche avverrebbero in un solo punto. Ancora, qualora delle regole siano presenti in diversi manufatti, potrebbe essere il caso di inserirne la descrizione in un apposito documento e quindi semplicemente referenziarle.

Da tener presente che, durante il processo di estrapolazione delle business rule, bisogna fare attenzione a non esagerare nella catalogazione, altrimenti si corre il rischio sia di utilizzare in modo non appropriato moltissimo tempo, sia di avere use case eccessivamente parametrici da risultare praticamente incomprensibili o svuotati di ogni contenuto.

Come si può notare le regole del business hanno un impatto non indifferente in molte fasi e numerosi modelli del processo di sviluppo di sistemi software. In alcuni casi giungono addirittura a guidare la stessa realizzazione degli artifact. Un esempio è dato dai test case. Uno dei loro compiti è infatti verificare che il sistema software inglobi correttamente le business rule. Per esempio, una delle funzionalità dei sistemi di back office delle banche è verificare che le tariffe concordate con i clienti con cui si sono stipulati dei contratti (trade) siano in un determinato intorno delle quotazioni di mercato. Questa funzionalità si

rende necessaria al fine di controllare l'operato dei broker, in altre parole per verificare che nessuno si comporti irregolarmente, magari accordandosi con qualche cliente per tariffe più vantaggiose in cambio di qualche ricompensa. La regola potrebbe prevedere che, se i limiti iniziali non sono rispettati ma comunque la variazione risulti ristretta ad un certo fattore (per esempio 10%), il sistema emetta un warning di cui il responsabile del team dei broker dovrà prendersi carico, mentre se risultasse superiore anche a quest'ultimo limite, allora lo stesso trade dovrebbe essere bloccato.

Chiaramente, uno dei compiti dei casi di test è verificare che questi controlli funzionino effettivamente come sancito da queste regole. Una verifica potrebbe richiedere la somministrazione al sistema diversi trade con parametri fuori mercato al fine di controllare che il sistema sia in grado di riconoscere le varie versioni di anomalie e quindi intraprendere i necessari provvedimenti.

Altri manufatti strettemente vincolati alle business rules, sono il modello dei casi d'uso e quello a oggetti del dominio (o la versione business). I primi contengono veri e propri riferimenti per i motivi visti in precedenza, mentre i secondi forniscono una rappresentazione Object Oriented delle regole. In sostanza leggendo il diagramma delle classi, e in particolare scorrendo le varie associazioni che legano le entità descritte, è possibile evidenziare tutta una serie di regole business, che probabilmente però non è il caso di trascrivere tutte in un apposito documento. Sarebbe anche un'attività molto utile, sia per verificare il modello, sia per renderlo fruibile a una platea non esperta dei diagrammi delle classi (come per esempio gli utenti che comunque dovrebbero validarlo). Però, in progetti di medio/grandi dimensioni, il tempo richiesto per la redazione del documento ne rende poco fattibile la produzione.

Questa categoria di regole è, tipicamente, etichettata come strutturali. Per esempio, nel contesto di un sistema di una compagnia di assicurazioni, un apposito diagramma delle classi, potrebbe illustrare le seguenti regole: una compagnia di assicurazione può stipulare diversi contratti; ciascuno di essi può essere riferito a una o più persone, o a una singola azienda, ecc. Da questo punto di vista, le regole del business possono essere considerate unità di conoscenza del business. Spesso i diagrammi delle classi non sono sufficienti ad esprimere correttamente le regole e quindi è necessario avvalersi di ulteriori formalismi, come il linguaggio OCL.

I diagrammi delle attività si prestano a descrivere regole del business denominate computazionali e comportamentali, mentre per ciò che concerne i diagrammi degli stati, esse rappresentano gli stimoli che determinano l'avvio del ciclo di vita di particolari oggetti, le transizione da uno stato a un altro, ecc.

Spesso l'acquisizione delle regole del business è un'attività tutt'altro che facile. Non è infrequente, infatti, il caso in cui molti utenti, anche esperti dell'area business, giunto il momento di definire formalmente determinate regole vigenti nel proprio dominio, comincino a vacillare. Un esempio? I sistemi di backoffice bancari. A questo punto ci si dovrebbe interrogare su quale sia la novità…

Tabella 5.6 — *Modello per le regole di business.*

BUSINESS RULE: *Codice*	*Nome della business rule.*		**Data:**
			Versione: 0.00.000
Descrizione:	*Descrizione generale della business rule.*		
Dettaglio:	*Dettaglio della business rule.*		
Esempio:	*Esempio chiarificatore.*		
Criticità:	*Impatto della business rule nel sistema..*		
Sorgente:	*Sorgente dalla quale è stata prelevata o è scaturita la regola business.*		
Business rule riferite:	*Codice*	*Nome della regola business oggetto di documentazione*	
CICLO DI VITA			
Azione	**Data**	**Descrizione**	**Autore**
Definizione			
Aggiornamento			

Si consideri la realizzazione del sottosistema di sicurezza del sistema informatico di un'azienda. Una policy vigente potrebbe prevedere che ogni utente disponga di un unico ruolo di sicurezza, ossia di un'assegnazione che abiliti l'esecuzione di specifici servizi e ne inibisca altri. Per esempio un ruolo di contabilità potrebbe abilitare i servizi relativi alla compilazione dei salari dei dipendenti, ed escludere quelli relativi alla pianificazione dei progetti aziendali. Pertanto, la realizzazione di un sistema che preveda diversi ruoli di sicurezza per uno stesso utente sarebbe chiaramente una violazione della policy. I sistemi ovviamente devono essere in grado di automatizzare sia le regole scritte sia quelle esistenti di fatto, altrimenti si realizzerebbe un sistema molto povero. In molte banche, per esempio, è prevista un'ulteriore regola che sancisce che modifiche ai dati appartenenti a determinate aree siano sottoposte a un doppio controllo. Per cui un utente inserisce determinati dati e un altro li controlla ed, eventualmente, li valida. Questi due utenti chiaramente si trovano a uno stesso livello gerarchico. Cosa avviene spesso nella realtà? Semplice, lo stesso utente effettua un doppio login nel sistema con due differenti utenze e quindi inserisce e convalida i dati!

Altri esempi di regole di business potrebbero essere: la procedura utilizzata per calcolare il codice fiscale, quella per verificarne l'esattezza (calcolo del codice di controllo, ultimo carattere del codice fiscale), per verificare l'esattezza della partita IVA, ecc.

A questo punto il primo problema che si pone è quale sia la forma migliore per descrivere le regole del business. La risposta più semplice e immediata potrebbe essere quella di ricorrere alla sempre valida forma testuale, magari resa più accattivante per mezzo di un apposito modello (tab. 5.6).

Le varie informazioni riportate sono piuttosto immediate da comprendere. L'unica cosa a cui è necessario porre particolare attenzione è il codice: è importante assicurarne l'univocità dal momento che viene riferito in altri manufatti (diagrammi dei casi d'uso, diagrammi delle classi, e così via).

Tabella 5.7 — *Esempio di regole business.*

BUSINESS RULE: BR-CSF-023	Spawned Deals	**Data:** 20 Ago 2001 **Versione:** 0.00.002
Descrizione:	Spawned Deals è la procedura che, tipicamente, si rende necessaria qualora un *FX Trade* (*Foreign eXchange*) sia relativo a valute desuete.	
Dettaglio:	La necessità di eseguire questa procedura si presenta quando per le valute scambiate nel *FX Trade* non esiste la corrispondente *dealing position* nel *book* di appartenenza del *trade* stesso. Tipicamente ciò avviene per scambi tra valute desuete. In queste condizioni, il *Trade* effettuato nel front office dà luogo ad altre due istanze nel back office, in quanto viene introdotta, come valuta di intermediazione, il dollaro americano (USD). Si dice quindi, che il *trade* è "spawned against USD".	
Esempio:	Si supponga di ricevere un FX Trade del tipo: 1223 MYR/SGD MYR=*x* e SGD = *y* Ossia uno scambio tra *x* Malaysian Ringgit ed *y* Singapore Dollar Poiché questo scambio è decisamente infrequente, non esiste un corrispondente dealing position e quindi è necessario dar luogo allo spawned contro dollari US 1223 MYR/SGD MYR=*x* e SGD = *y* 1224 MYR/USD MYR=*x* e USD =*z* 1225 SGD/USD SGD= *z* e USD= *y*	
Criticità:	Importante.	
Sorgente:	Analisi del sistema di legacy.	
Business rule riferite:		

CICLO DI VITA			
Azione	**Data**	**Descrizione**	**Autore**
Definizione	10-6-00	Definizione iniziale	LVT
Aggiornamento	20-6-00	Inserimento esempio	RV
Aggiornamento	20-8-00	Correzione esempio.	RV

Si consideri l'esempio di tab. 5.7. La regola business considerata fa riferimento a una procedura che si applica nel back office di sistemi bancari, in determinate condizioni, nella gestione del prodotto denominato FX (*Foreign eXchange*): scambio di valute estere. Il principio di base è molto semplice: si scambia una certa quantità di una valuta con un'altra quantità di un'altra valuta. Ebbene questa evoluzione del baratto è il prodotto che

Tabella 5.8 — *Esempio di regole business relative alle SSI.*

muove qualche decina di miliardi di dollari al giorno, prevalentemente tra New York, Londra e Tokyo per fini, ovviamente, speculativi.

Sempre nel sistema bancario, altre entità molto importanti sono le famose Istruzioni di pagamento. Ne esistono due versioni: quelle da applicare (SSI*, Settlement Standing Instruction*) e quelle applicate (*Settlement Instruction*). La differenza consiste nel fatto che le prime forniscono regole generali, nel senso che specificano per ogni trade effettuato, in base a diversi criteri (cliente, prodotti trattati, valute, ecc.), quali conti correnti utilizzare per addebitare e prelevare, mentre le seconde sono particolari istanze delle prime, che una volta applicate perdono di generalità e vengono associate al trade. Ciò permette di modificare le SSI, magari per esempio perché un cliente vuole utilizzare un nuovo conto corrente, evitando che le modifiche si riversino in regole già applicate. Chiaramente di queste regole esistono quelle associate ai clienti, e quelle della banca. In altre parole bisogna conoscere sia i conti correnti dei clienti sia quelli della banca cui addebitare e accreditare (tab. 5.8).

Molto spesso alcune regole del business sono rappresentate da algoritmi necessari per calcolare specifici valori (regole computazionali). Per esempio, nei sistemi bancari è necessario calcolare i fattori di rischio di determinati trade. In questi casi, non è infrequente che, per la descrizione della regola di business, alla soluzione testuale ne sia preferita una più intuitiva basata sui diagrammi delle attività. In questo contesto, tipicamente, essi sono utilizzati come particolari versioni dei famosi diagrammi flowchart. In ultima analisi, si tratta di scegliere una forma diversa per mostrare le stesse informazioni.

In generale, le regole del business si prestano a essere rappresentate per mezzo di diversi formalismi, come il diagramma delle classe e l'Object Constraint Language. Si consideri, per esempio, la regola secondo la quale ogni conto corrente può appartenere a diversi clienti ma, nel caso in cui si tratti di un'azienda, allora il cliente deve essere unico. La regola del business rappresenta un vincolo relativo a entità che effettivamente esistono nell'area business oggetto di studio e pertanto il formalismo dei diagrammi delle classi, accompagnato da un vincolo in OCL, rappresenta il formalismo più idoneo da utilizzarsi.

Affrontato il problema della forma, è necessario passare a quello inerente la sostanza.

Quali direttive seguire per assicurarsi che le regole siano ben analizzate e rappresentate? La risposta migliore è seguire l'usuale principio di agevolare la comunicazione: fare in modo che le regole del business siano il più possibile complete ed espresse nella forma più chiara possibile. Pertanto è necessario cercare di specificare regole coesive, focalizzando l'attenzione su una singola regola alla volta. In breve, è importante che ad ogni regola del business corrisponda un unico concetto ben definito. Nella pratica, capita di imbattersi in business rule che non rispettano questi princìpi. Generalmente regole sifatte tendono a generare confusione nel lettore, soprattutto qualora quest'ultimo non sia esperto dell'area business oggetto di analisi.

Figura 5.12 — *Algoritmo di calcolo della stringa di login. Da notare che il diagramma rappresenta un requisito e quindi non è stata introdotta alcuna ottimizzazione. Queste appartengono allo spazio delle soluzioni. Un esempio potrebbe essere quello di utilizzare una strategia più "furba" per la risoluzione dei conflitti. Per esempio, dopo un certo numero di tentativi falliti, si potrebbe risolvere il problema effettuando una ricerca nella tabella utente utilizzando come chiave i primi caratteri che generano il conflitto e quindi sfruttare il primo "buco" tra due codici successivi.*

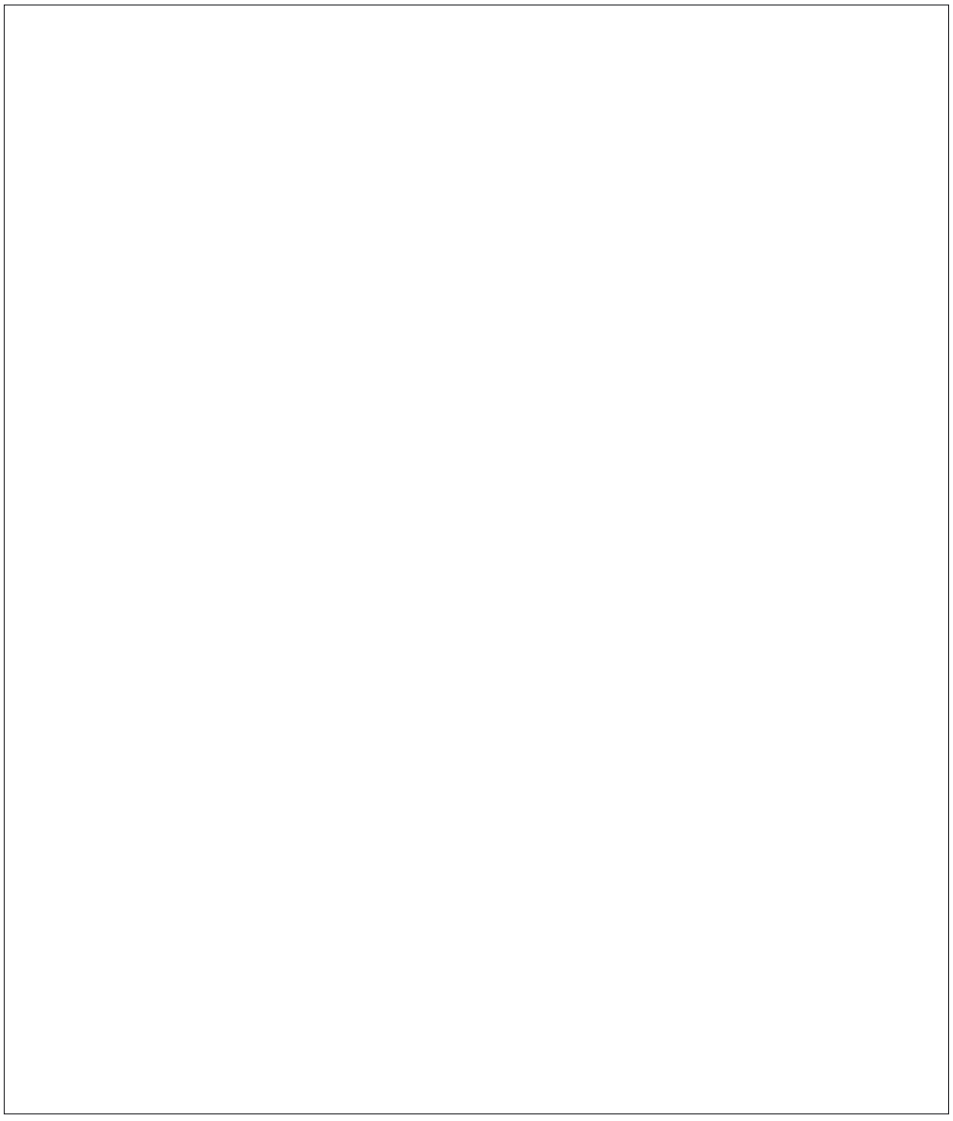


Tabella 5.9 — *Esempio di regole business.*

Anche se, teoricamente, le regole del business dovrebbero essere analizzate, descritte e catalogate durante l'analisi dei requisiti, nella realtà ciò non è sempre possibile. Tipicamente queste possono emergere in qualsiasi fase del processo di sviluppo: l'importante

però è che ciò non avvenga in fase di codifica. Per esempio, spesso può accadere che specifiche regole emergano durante il disegno del sistema, magari perché sia necessario realizzare uno strato di integrazione con uno specifico sistema di legacy, e quindi sia necessario rispettarne le regole interne. L'importante è, ovviamente, che tutte le regole siano confermate dai clienti e/o dagli esperti dell'area business e che siano catalogate in un unico repository.

L'attività che maggiormente favorisce l'estrapolazione delle business rule è sicuramente la realizzazione dei casi d'uso e del modello a oggetti del dominio. Una buona norma da seguire consiste nel dare luogo ad una prima classificazione già durante i colloqui iniziali con il cliente, nei quali si tende ad acquisire quanta più documentazione possibile relativa all'area business oggetto di studio.

Una volta catalogate è importante mantenere le regole del business in un apposito artifact a parte, per una serie di ovvi motivi. Il primo è che queste tendono a essere referenziate in più manufatti e quindi è opportuno centralizzarne la descrizione. Ciò, tra l'altro, permette di neutralizzare il propagarsi degli effetti delle modifiche. Nel caso contrario una volta modificata una regola del business bisognerebbe ritrovare tutti i vari punti in cui sarebbe stata ripetuta e quindi aggiornare di conseguenza.

Chiaramente l'importanza di classificare correttamente le regole del business è una necessità particolarmente sentita in sistemi di medie e grandi dimensioni, mentre negli altri spesso è possibile utilizzare un approccio a maggior grado di pragmatismo.

Ricapitolando…

Il presente capitolo è completamente incentrato sue due tematiche portanti: la presentazione di una tecnica dimostratasi particolarmente valida nell'arduo compito dell'analisi dei requisiti utente e l'illustrazione di una serie di manufatti (artifact) che permettono di completare il modello dell'analisi dei requisiti.

Il problema che tenta di risolvere la tecnica di analisi dei requisiti proposta è trovare una piattaforma di dialogo comune e costruttiva tra personale tecnico e clienti, in altre parole trovare un efficace raccordo tra i due diversi linguaggi tecnici: quello dell'area business e quello informatico. In questo scenario, gli activity diagram si sono dimostrati ottimi strumenti di analisi dei requisiti poiché, oltre ai normali vantaggi offerti da ogni formalismo grafico, ne offrono un altro tutto particolare e insuperabile: sono molto simili ai popolarissimi antenati flowchart. Questa discendenza favorisce la realizzazione di una particolare tecnica di analisi: si definisce dapprima il comportamento dinamico della funzione oggetto di studio per mezzo degli activity diagram e poi si ricostruisce la proiezione statica attraverso i diagrammi dei casi d'uso.

Una volta redatti gli activity diagram, la redazione dei flussi degli eventi, teoricamente, assume un interesse molto relativo: mostrerebbero le stesse informazioni con un formalismo diverso (linguaggio naturale). Pertanto si potrebbe far a meno di descrivere il comportamento dinamico attraverso i template mostrati nel capitolo precedente (o qualsiasi altro formato). Il problema sorge qualora i diagrammi realizzati siano o possano essere soggetti a variazioni di un certo rilievo. Non è infrequente il caso in cui il tempo necessario per modificare i vari diagrammi (aggiornamento del diagramma, conferimento di un aspetto

soddisfacente, ecc.) è molto superiore a quello occorrente a dar vita alle varie descrizioni e ad aggiornarle. Quindi, nella pratica, spesso si preferisce utilizzare i diagrammi delle attività per produrre una versione iniziale dei requisiti utenti e, una volta stabilizzati, si passa al formato testo. C'è anche da dire che realizzare uno stesso manufatto con diversi formalismi, se da un lato fa perdere molto tempo, dall'altro risulta un'ottima tecnica per analizzare quanto prodotto da una prospettiva diversa e quindi eseguirne una valida revisione (trattasi del famoso "contentino").

L'attività di estrapolazione dei diagrammi dei casi d'uso prevede un passo propedeutico che consiste nell'isolare i messaggi che il sistema scambia con i relativi attori: si tratta dello stimolo di avvio (*trigger*) che viene elaborato per produrre specifici effetti. Il percorso compreso tra due interazioni consecutive, tipicamente, rappresenta un'unica funzionalità da descriversi attraverso un apposito use case (eventualmente suddiviso in altri al fine di controllarne la complessità, evidenziarne parti riutilizzabili, ecc).

Il passaggio dai diagrammi delle attività alla relativa rappresentazione realizzata per mezzo dei casi d'uso, generalmente, non avviene attraverso una corrispondenza biunivoca tra le attività (intese come elementi) dei primi e i casi d'uso dei secondi. Ciò, talune volte, può generare l'esigenza di realizzare opportune matrici di corrispondenza, al fine di tracciare l'evoluzione dei requisiti e semplificare la lettura dei diagrammi da personale non esperto.

Il modello dei casi d'uso, pur essendo un artifact molto importante, da solo non è assolutamente in grado di fornire sufficienti informazioni per il pieno svolgimento delle restanti fasi del ciclo di vita del sistema. Pertanto, la completa realizzazione della vista dei requisiti del sistema prevede la produzione di ulteriori manufatti complementari al modello dei casi d'uso. In particolare, nel presente capitolo sono stati presentati manufatti di notevole importanza, come il dettaglio delle interfacce, i test case, il documento dei requisiti non funzionali e le famose business rule.

Un aspetto molto importante della progettazione di un sistema è legato alla definizione del confine dello stesso. Un manufatto che ne favorisce la determinazione, e che agevola la definizione formale delle responsabilità che attori e sistema assumono gli uni nei confronti dell'altro, è il modello di specifica delle interfacce. Questo, una volta completato, dovrebbe rendere immediata la definizione delle classi *boundary* presenti nel modello di analisi. I vantaggi derivanti dalla definizione di queste interfacce diventano più evidenti nei casi in cui il sistema interagisca con altri dispositivi. La definizione formale delle interfacce permette, inoltre, di verificare completezza e correttezza del modello a oggetti del dominio: nelle interfacce dovrebbero comparire esclusivamente attributi ed entità rappresentate nel modello a oggetti del dominio; qualora ciò non avvenga, verosimilmente, è opportuno revisionare i due artifact per verificare l'esistenza di eventuali inconsistenze.

Questo manufatto è in grado anche di fornire importanti direttive alle restanti fasi del ciclo di sviluppo del software; in particolare permette di agevolare il disegno delle interfacce, particolarmente utili in sistemi in cui vi è una netta separazione tra la definizione dei comportamenti e la relativa implementazione.

I test case, come lascia intuire il nome, sono artifact fondamentali per la fase di test, però derivano direttamente dai casi d'uso. Mentre questi ultimi descrivono le funzionalità che il sistema dovrà realizzare, i test case servono ad accertarsi che il sistema fornisca realmente i servizi promessi.

I processi moderni prevedono la produzione del sistema attraverso una serie ripetuta di iterazioni, una sequenza di miniprogetti, e quindi i test devono essere eseguiti ad ogni rilascio di nuove versioni. Chiaramente, come suggerisce il buon senso, occorre verificare quanto prodotto non appena disponibile, non solo per ciò che riguarda il sistema finale, ma per ogni manufatto.

I test case devono essere inseriti in un contesto ben pianificato, detto piano dei test. Tenendo presenti i processi di sviluppo iterativi e incrementali, tipiche attività da pianificare sono esclusione di test case diventati obsoleti, realizzazione di nuovi test case, aggiornamento di specifici test case. Il documento relativo al piano di test dovrebbe prevedere la definizione degli obiettivi, il dettaglio dei test, la allocazione delle risorse, l'elenco dei rischi, ecc.

Il "modello" di test, tipicamente, è composto dai test case, dalle test procedure e da eventuali componenti di test.

Nella programmazione dei test è molto importante verificare sia che il sistema eroghi correttamente il servizio esaminato e quindi produca i risultati attesti (*positive test*) in condizioni ideali (dati di ingresso validi, ambiente completamente operativo, ecc.), sia che in situazioni anomale (dati di input non corretti, ambiente parzialmente non operativo, ecc.) il sistema sia in grado di riconoscere tali situazioni anomale e quindi eseguire le relative procedure (*negative test*). Tali test possono essere pianificati esaminando il flusso degli eventi dei casi d'uso; in particolare il flusso principale e quelli alternativi permettono di programmare i positive test (verifiche positive), mentre i flussi di errore danno luogo i negative test (verifiche negative).

I requisiti non funzionali sono informazioni di importanza vitale, tipicamente analizzate e definite fin dalle prime fasi di analisi dei requisiti, che però non sempre si prestano a essere rappresentate per mezzo di un ben definito formalismo grafico. Pertanto, generalmente, la relativa descrizione è realizzata per mezzo del classico documento.

Essi sono di grande valore nella fasi di disegno, implementazione e test del sistema. In queste fasi non solo è importante realizzare le funzioni definite nel modello dei casi d'uso, ma bisogna rispettare vincoli legati a performance, sicurezza, affidabilità, ecc.

Le proprietà che devono possedere i requisiti non funzionali sono: correttezza, completezza, precisione e consistenza.

Tipicamente, il documento dei requisiti non funzionali prevede le seguenti sezioni:

- utilizzabilità (*usability*), nella quale si affrontano problematiche del tipo: tempo necessario all'addestramento del personale che dovrà utilizzare il sistema (utenti), tempo richiesto mediamente all'utente per lo svolgimento delle attività tipiche, requisiti base di utilizzabilità, direttive per la definizione standard di interfacce utente, ecc.;

- affidabilità del sistema (*reliability*), che prevede temi quali: disponibilità, tempo medio tra due errori consecutivi (MTBF, *Mean Time Between Failure*), tempo massimo di non disponibilità consentito al sistema per la riparazione di eventuali guasti (MTTR, *Mean Time To Repair*), accuratezza o risoluzione, bug e/o difetti, ecc.;

- procedure di *backup*, in cui è necessario specificare: strategia di backup stessa, procedure di recovery, ecc.;

- sicurezza (*security*): i requisiti appartenenti a quest'area sono così importanti che, probabilmente, meriterebbero anch'essi di essere trattati in un apposito documento. In questa sezione è necessario definire le politiche di riconoscimento dell'utente, i ruoli con la relativa matrice dei servizi fruibili, la modalità con cui proteggere i dati, sia quelli memorizzati permanentemente, sia quelli scambiati per mezzo di connessioni, gli algoritmi di crittazione, le difese verso eventuali hacker, i log con la storia delle azioni eseguite dagli utenti, ecc.;

- *performance* del sistema, descritte in termini di: tempo di risposta delle transazioni più significative, *throughput* (numero di lavori eseguiti nell'unità di tempo, tipicamente transazioni per secondo), capacità, utilizzo delle risorse, ecc.;

- *supportability* del sistema, ossia standard e direttive che semplificano la manutenibilità del sistema da costruire (per esempio standard di codifica), componenti atti al controllo del sistema, *naming convention* (standard per l'attribuzione dei nomi), *change cases*, accessi per la manutenzione, e così via;

- vincoli di disegno, come, per esempio: linguaggi di programmazione, paradigma utilizzato (Object Oriented, component-based, ecc.), architettura di riferimento, database management system, network, piattaforma software e hardware, legacy system presenti, ecc.;

- documentazione, componenti software/sistemi di cui si dispone la licenza che devono essere necessariamente utilizzati o che si intende utilizzare, ecc.

L'ultima parte del capitolo è dedicato alle famose *business rules*, per le quali non esiste una definizione rigorosa. Tipicamente, con questi termini, ci si riferisce a princìpi, policy, leggi, dettagliati algoritmi di calcolo, relazione tra oggetti, ecc. che influenzano direttamente il dominio del problema che il sistema, in qualche misura, dovrà automatizzare. Pertanto rappresentano informazioni che, in ultima analisi, dovranno essere implementate dal sistema. Si tratta di una definizione piuttosto ampia; è certo però che queste regole hanno un notevole impatto sul sistema da realizzare e quindi è assolutamente necessario considerarle e documentarle appropriatamente.

Teoricamente, le regole del business dovrebbero essere analizzate, descritte e catalogate durante l'analisi dei requisiti; nella realtà ciò non è sempre possibile. Tipicamente queste possono emergere in qualsiasi fase del processo di sviluppo: l'importante però è che ciò non avvenga in fase di codifica.

Le modalità con cui documentare le regole del business sono diverse e attualmente, nella comunità informatica, non esiste una visione univoca. Il modo più semplice e generale possibile è ricorrere al classico documento, eventualmente reso più accattivante da appositi moduli. Alcune business rule poi, sono algoritmi necessari per calcolare informazioni specifiche. In questi casi non è infrequente che, per la rela-

tiva descrizione, alla soluzione testuale ne sia preferita una più intuitiva basata sui diagrammi delle attività. Altre volte le regole del business si prestano ad essere rappresentate per mezzo di formalismi, come per esempio il diagramma delle classi e l'Object Constraint Language.

Chiaramente l'importanza di classificare correttamente le regole del business è una necessità particolarmente sentita in sistemi di medio e grandi dimensioni, mentre negli altri, spesso, è possibile utilizzare un approccio più pragmatico.