

Modulo 1

Basi di UML

- 1.1** Perché si modella il software
- 1.2** Il processo di modellazione unificato
- 1.3** Che cos'è UML
- 1.4** Parti di UML
- 1.5** Strumenti di modellazione

Nessuna attività influenza tanto la qualità di un prodotto come l'architettura e la progettazione.

James O. Coplien, AT&T Bell Laboratories

Una delle domande che la maggior parte degli sviluppatori di fresca data si pongono è perché sia necessario modellare il software. Questa domanda viene posta di continuo, perché solo raramente si insegna agli sviluppatori a modellare prima che a codificare, in particolar modo se lo sviluppatore è un autodidatta. In effetti, la maggior parte degli sviluppatori ha imparato a programmare leggendo e cercando di imitare il codice altrui. Piano piano, in questo modo uno sviluppatore raggiunge una certa abilità, che diventa, per la maggior parte di loro, l'unica maniera concepibile per realizzare un programma. In questo libro, e in particolare in questo capitolo, si cercherà di evidenziare quanto la modellazione del software può migliorare il proprio modo di programmare.

1.1 Perché si modella il software

La ragione principale per cui si modella il software è cercare di raggiungere un elevato livello qualitativo del prodotto finito. Un prodotto ben studiato, dalla solida architettura alla fine paga: lo si è sentito molte volte, pur senza capire a fondo le motivazioni di questa affermazione. La qualità elevata non è qualcosa che capita: esistono diverse cose che devono accadere fra la fase di concepimento di un progetto e il prodotto finito, cose che rappresentano il risultato diretto della modellazione del software prima del suo sviluppo. L'elevata qualità discende direttamente dalla facilità di sviluppo, da tempi di sviluppo più corti, da documentazione migliore e da un numero di bug inferiore, dovuti a una fase di test migliorata.

È un dato di fatto che una buona struttura dura a lungo e una debole, per converso, finisce per fallire. Un prodotto che sia stato realizzato su basi solide, con metodi coerenti per il raggiungimento dei risultati, che sfrutta le procedure già realizzate e privo di bug è molto facile da modificare, in particolar modo per chi lo ha realizzato.

Si prendano, come esempio, i Lego. Sicuramente molti lettori ci hanno giocato almeno una volta nella vita.

Se ci si ricorda come si giocava, ci sono pochissime restrizioni su come assemblare i Lego: si possono mettere i mattoncini sulla testa dei personaggi, volendo, con la stessa facilità con cui si può mettere un tetto sopra quattro muri.

D'altro canto, le scatole di Lego sono generalmente realizzate per costruire una particolare struttura, sia essa una casa, una stazione spaziale, un ospedale o un castello. Chi ha provato a costruire una di queste strutture soltanto guardando la figura sulla scatola si può rendere conto di che cosa sia lo sviluppo software privo di modellazione: lo sviluppatore, semplicemente, si dice che è in grado di realizzare un determinato programma e parte con la programmazione. L'intenzione è buona, ma occorre un po' di tecnica.

Ora, si provi a ricordare il manuale di una ventina di pagine all'interno della scatola di Lego, che mostra esattamente, passo dopo passo, come costruire la struttura. Procedere in questo modo è molto più facile e probabilmente richiede molto meno tempo per ottenere il prodotto finale. E posso garantire personalmente che richiede molto meno tempo seguire le istruzioni per costruire la struttura, piuttosto che scrivere le istruzioni stesse. Garantisco altresì che ci vuole molto meno tempo a scrivere le istruzioni e quindi a realizzare la struttura, piuttosto che a procedere a tentoni cercando di arrivare al prodotto finito. È certo che la modellazione del software richiede più tempo dello sviluppo; ma è anche vero che il tempo di sviluppo può essere drasticamente ridotto mediante la modellazione e la documentazione del software.

Analisi, progetto e implementazione

In qualsiasi modo le si suddivida, ci sono tre fasi per creare un sistema di qualità: l'analisi, il progetto e l'implementazione. Mi piace pensare che un progetto di successo sia il risultato di molta analisi, un po' meno progetto e ancor meno implementazione, poiché se si fa il contrario ci si ritrova con un vero incubo fra le mani. Molta implementazione con poco progetto e ancor meno analisi solitamente porta a un prodotto pieno di bug e senza le funzionalità richieste dagli utenti. La Figura 1-1 illustra sei possibili scenari che si presentano quando si sviluppa un programma; il primo è l'ideale, mentre gli altri sono soltanto modi diversi per finir male.

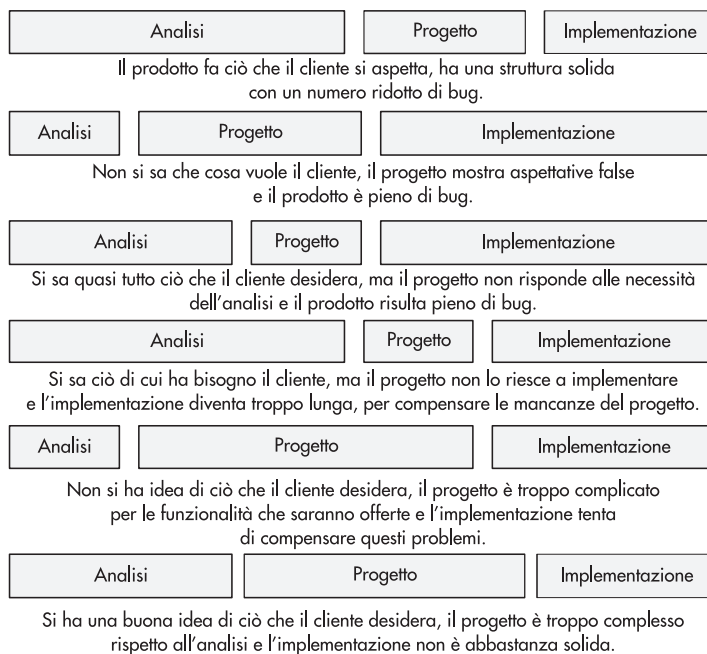


Figura 1-1 Diverse dosi di analisi, progetto e implementazione.

Durante la fase di analisi è necessario consultare un esperto del campo, ossia una persona che sia considerata un'autorità del sistema che si deve creare. Per esempio, se si deve realizzare un sistema per il check-in aeroportuale, chi sarà l'esperto? La compagnia aerea, l'impiegato al check-in o i passeggeri? Per rispondere alla domanda è necessario chiedersi chi dovrà utilizzare il sistema: sarà quindi, con ogni probabilità, l'impiegato al check-in.

Modelli del ciclo di vita del software

Ci sono tre modelli del ciclo di vita del software che hanno circolato per anni: a cascata, a spirale (o iterativo) e incrementale-iterativo. Il metodo a cascata, illustrato nella Figura 1-2, è molto semplice: si inizia con l'analisi, si progetta e quindi si sviluppa.

Questo metodo è probabilmente il più comune, poiché è il più prossimo al buon-senso: è molto più naturale degli altri due sistemi. Il metodo a spirale, o iterativo, è illustrato nella Figura 1-3, richiede di ripetere più volte il metodo a cascata fino a che non si giunge al prodotto finito.

Come si può vedere dalla Figura 1-3, questo processo inizia con l'analisi, continua con la progettazione e prosegue con l'implementazione, quindi si ripete partendo nuovamente dalla fase di analisi. Il metodo consente al team di sviluppo di completare man mano un progetto. Magari la prima fase prevede soltanto le funzionalità strettamente necessarie, la seconda ne aggiunge qualcuna che è semplicemente comodo avere e la terza ne implementa altre utilizzate di rado. D'altro canto, potrebbero essere necessari diversi passaggi fra le fasi di analisi, progetto e implementazione prima di avere un qualcosa che sia utile all'utente finale.

Il terzo modello del ciclo di vita di un software è quello iterativo-incrementale, illustrato nella Figura 1-4. Sostanzialmente, questo sistema suddivide un progetto in sottoprogetti e consente di applicare a ciascuno di essi il metodo a cascata. Invece di completare tutta la funzionalità di un'applicazione con ciascun passaggio, come accade per il metodo iterativo, il metodo iterativo-incrementale completa ciascun componente dell'applicazione all'interno di ciascuna fase.

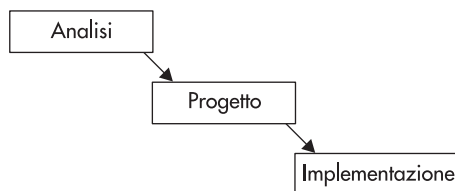


Figura 1-2 Il modello del ciclo di vita a cascata.

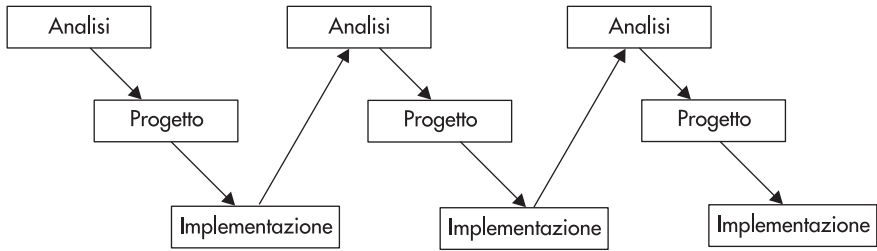


Figura 1-3 Il modello di ciclo di vita a spirale.

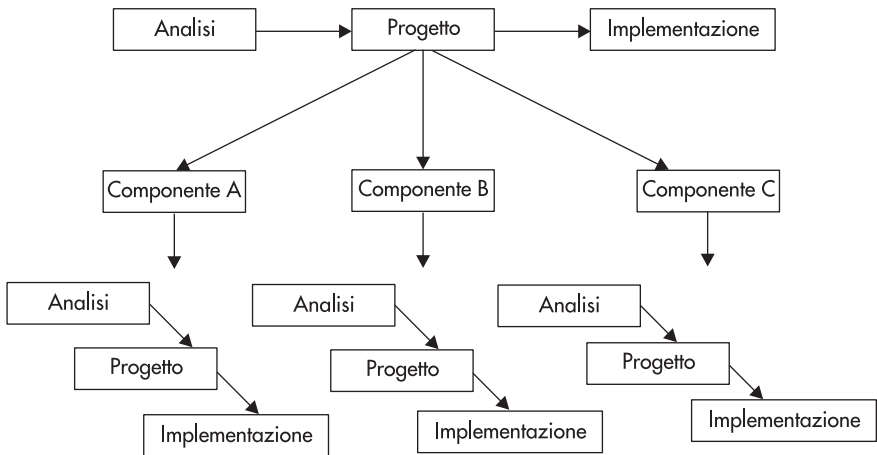


Figura 1-4 Il modello di ciclo di vita iterativo.

Con questo sistema, ogni componente completato non diventa necessariamente un prodotto utilizzabile da un cliente; ma esiste un punto in cui si possono unire vari componenti per creare un prodotto utilizzabile. Questo è il metodo da preferire, perché garantisce un'elevata percentuale di codice riutilizzabile. Se si separa la funzionalità del sistema in diversi componenti, come l'accesso ai dati, la logica e i controlli GUI, si sa esattamente dove si trova ciascuna caratteristica desiderata.



Esercitazione flash

1. Quali sono gli elementi che compongono i sistemi di qualità?
2. Chi dovrebbe essere l'esperto del campo per un chiosco ferroviario?
3. Descrivere il modello di sviluppo incrementale-iterativo.

1.2 Il processo di modellazione unificato

UML (Unified Modeling Language) è soltanto un linguaggio, non un modo di progettare un sistema: soltanto un modo per modellare un sistema. Per utilizzare UML è necessario utilizzare un metodo. Esistono diversi metodi di progettazione, ma il più comune, e probabilmente il primo che è stato utilizzato insieme con UML, è il metodo noto come Rational Unified Process (RUP, Processo Unificato Rational), chiamato anche semplicemente Unified Process (Processo Unificato).

È anche possibile imparare la notazione UML, ma questo non è del minimo aiuto per capire come applicare UML alla progettazione di un sistema. A causa di questo, si è pensato di utilizzare un'interpretazione del Processo Unificato per insegnare UML. Se si lavora presso un'azienda che già sta utilizzando UML, è possibile che l'azienda stessa implementi anche RUP, con i diagrammi inviati insieme con il prodotto Rational Rose da parte di Rational Software. Se invece si pensa di adottare UML nella propria azienda, è possibile che si vogliano sperimentare le proprie nuove competenze, insegnando ad altri in che modo UML può inserirsi all'interno delle attività quotidiane, invece di mostrare al proprio superiore come cambiare tutto il processo produttivo del software in modo da adattarlo alle esigenze del Processo Unificato.

Il Processo Unificato non è fisso e inalterabile; in effetti, è necessario fare in modo che si adatti alle proprie necessità e alle proprie competenze. È facile trovare scorcioie rispetto a molte delle specifiche, ma, in alcuni casi, è anche possibile scoprire che si ha bisogno di qualcosa di più di quanto RUP possa offrire. Proprio in questi casi entra in gioco l'evoluzione, che consente di modificare il proprio processo produttivo finché non si adatta perfettamente alla propria situazione.

-
1. Facilità di sviluppo, cicli di sviluppo più corti, migliore documentazione utente e meno bug dovuti al miglior testing.
 2. I passeggeri dei treni sarebbero un'ottima fonte di informazioni per sapere che cosa desiderano vedere, ma la ferrovia può sapere che cosa è disponibile da far vedere ai passeggeri.
 3. Durante la fase iniziale di analisi, progetto e implementazione, il sistema viene diviso in componenti, ciascuno dei quali ha le proprie fasi di analisi, progetto e implementazione.

SUGGERIMENTO



Esistono altri tre processi per la progettazione di sistemi software: il processo software a oggetti (OOSP, Object-Oriented Software Process (OOSP)), il processo OPEN (www.open.org.au) e il processo ICONIX Unified Object Modeling (www.iconixsw.com/Spec_Sheets/UnifiedOM.html).

Il Processo Unificato è basato sull'organizzazione di Ivar Jacobson Objectory, che si è fusa con Rational. Il processo, chiamato Objectory Process, è stato migliorato con l'apporto del processo di progettazione software proprio di Rational ed è stato pubblicato come Unified Process nel dicembre del 1998.

Un processo di sviluppo software è costituito da un insieme di fasi da seguire per portare un prodotto o un sistema dall'ideazione all'implementazione effettiva. Nel Processo Unificato le fasi sono quattro:

- **inizio:** identificare il sistema che si desidera sviluppare, compresi contenuti e casi d'uso;
- **elaborazione:** eseguire una progettazione dettagliata e identificare le basi del sistema;
- **costruzione:** scrivere il software;
- **transizione:** fornire il sistema agli utenti.

Fase di inizio

La fase di inizio del Processo Unificato è quella in cui si esegue l'analisi iniziale: si discute con l'esperto del campo su come dovrà essere il sistema. In questa fase è necessario identificare i requisiti per poi modellarli nei diagrammi dei casi d'uso. Il Capitolo 2 tratta specificamente di questa fase del Processo Unificato, mentre il Capitolo 10 aiuta a modellare il flusso esecutivo di sistemi complessi.

Fase di elaborazione

La fase di elaborazione del Processo Unificato è quella in cui avviene la progettazione del sistema. A partire dai casi d'uso, il team di progettazione si mette al lavoro per ottenere un'immagine unifica di come dovrebbe essere realizzato il sistema.

Il team di progettazione continua a suddividere iterativamente il sistema in sottosistemi, ciascuno dei quali può essere modellato separatamente. Non è un compito facile, motivo per cui una grande parte di questo libro è dedicata proprio alla progettazione di un sistema. Durante la fase di elaborazione i casi d'uso scoperti nella fase di inizio vengono elaborati e trasformati in un progetto globale del sistema, comprendente i sottosistemi e gli oggetti funzionali a esso correlati. Da questo punto in avanti si pro-

cede iterativamente con la trasformazione di questo modello in un progetto software, utilizzando diagrammi più dettagliati che, alla fine, serviranno a modellare le classi e i relativi membri.

Questo libro è pesantemente orientato alla fase di elaborazione del Processo Unificato, sfruttando la progettazione e l'analisi a oggetti nonché il linguaggio UML per preparare il team alla realizzazione di un prodotto basato su un progetto ben costruito.

Fase di costruzione

La fase di costruzione del Processo Unificato rappresenta l'effettiva realizzazione del prodotto a partire dal progetto del sistema. Nel Processo Unificato, la parte riguardante lo sviluppo è un processo incrementale-iterativo. Il codice viene sviluppato in porzioni, ciascuna delle quali, oltre a essere facilmente gestibile, viene portata avanti con dei piccoli cicli simili all'intero Processo Unificato. La fase di costruzione richiede solitamente modifiche al progetto e nuove fasi di analisi; è necessario essere sufficientemente flessibili per continuare, ma occorre anche attenersi, in generale, al risultato delle fasi precedenti, in modo che il progetto finale non si discosti troppo da quello originale. Spesso ci si troverà a tornare indietro a fasi precedenti, in particolare a quella dell'elaborazione, per progettare nuovi componenti a cui non si era pensato prima. Questo può accadere in particolare adottando il modello di ciclo di vita del software di tipo iterativo-incrementale, di cui si è parlato in precedenza.

Fase di transizione

La fase di transizione va un po' al di là dello scopo di questo libro e tratta principalmente dell'installazione del prodotto presso i clienti, magari su un sito beta o anche presso il cliente vero e proprio.

Se si è già sviluppato software in precedenza, si sa già che questa non è la fine del ciclo di vita di un progetto software, tutt'altro: è normale che il completamento di un prodotto software, sia esso coronato da successo oppure no, venga seguito da una fase di manutenzione e aggiornamento. Alla fine di tutto, resta la fase del tramonto, durante la quale il prodotto sparisce piano piano dall'esistenza.



Esercitazione flash

1. Che cosa viene svolto durante la fase di inizio del Processo Unificato?
 2. Che cosa viene svolto durante la fase di elaborazione del Processo Unificato?
 3. Che cosa viene svolto durante la fase di costruzione del Processo Unificato?
-

1. L'analisi.
2. La progettazione.
3. L'implementazione.

1.3 Che cos'è UML

Il linguaggio UML è un linguaggio semantico che può essere applicato a qualsiasi processo di sviluppo software. UML è stato sviluppato da Grady Booch, Jim Rumbaugh e Ivar Jacobson, di Rational Software. La notazione di UML combina le migliori idee di questi tre progettisti, risultando in uno standard che gode di un supporto internazionale. UML utilizza diversi tipi di diagrammi, che possono essere utilizzati per modellare sistemi software a oggetti.

La storia

I primi due membri del team UML, Grady Booch e Jim Rumbaugh, iniziarono sviluppando la nuova notazione presso Rational, nel 1994. Avevano i propri metodi di progettazione, il metodo Booch e il metodo OMT (Object Modeling Technique, tecnica di modellazione di oggetti). Nel 1995, il dott. Ivar Jacobson si unì al team, portando il proprio metodo noto come Object-Oriented Software Engineering (OOSE). Jacobson viene comunemente considerato il padre dei casi d'uso. Dei tre metodi, il metodo OMT di Jim Rumbaugh è quello che ha dato il maggior contributo allo sviluppo di UML.

I tre iniziarono un cammino che doveva portare alla creazione di un linguaggio di modellazione unificato, perché tutti e tre i loro metodi si stavano pian piano riunendo in un'unica sintassi quasi da soli: a questo punto, i tre decisero di unire le proprie forze. In questo modo riuscirono a standardizzare la modellazione degli oggetti.

La prima versione di UML fu UML 0.8. Dopo il primo rilascio delle specifiche UML, i tre autori richiesero un feedback per il proprio lavoro. Nel 1995, fu raggiunto un accordo con il gruppo di lavoro Object Management Group (OMG) per rendere UML uno standard. Dal 1996, diverse aziende esterne furono coinvolte nella decisione di che cosa sarebbe dovuto diventare UML; il lavoro comune risultò nello standard UML 1.0. Nel 1997 altre aziende si unirono al gruppo di lavoro e contribuirono a creare lo standard UML 1.1. Questo libro utilizza la versione 1.4, approvata nel 2001. Attualmente, una nuova versione 2.0 sta per essere approvata; il libro dà per scontato che la versione corrente delle RFP (Request For Proposal, richieste di proposta) 2.0 verranno a far parte della specifica ufficiale 2.0.

Di seguito è riportato un elenco delle aziende e delle singole persone che hanno avuto un'influenza sostanziale sul progetto della notazione UML:

- Colorado State University: Robert France;
- Computer Associates: John Clark;
- Concept 5 Technologies: Ed Seidewitz;
- Data Access Corporation: Tom Digre;
- Enea Data: Karin Palmkvist;

- Hewlett-Packard Company: Martin Griss;
- IBM Corporation: Steve Brodsky, Steve Cook;
- I-Logix: Eran Gery, David Harel;
- ICON Computing: Desmond D'Souza;
- IntelliCorp and James Martin & Co.: James Odell;
- Kabira Technologies: Conrad Bock;
- Klasse Objecten: Jos Warmer;
- MCI Systemhouse: Joaquin Miller;
- OAO Technology Solutions: Ed Seidewitz;
- ObjecTime Limited: John Hogg, Bran Selic;
- Oracle Corporation: Guus Ramackers;
- PLATINUM Technology Inc.: Dilhar DeSilva;
- Rational Software: Grady Booch, Ed Eykholt, Ivar Jacobson, Gunnar Övergaard, Jim Rumbaugh;
- SAP: Oliver Wiegert;
- SOFTEAM: Philippe Desfray;
- Sterling Software: John Cheesman, Keith Short;
- Sun Microsystems: Peter Walker;
- Telelogic: Cris Kobryn, Morgan Björkander;
- Taskon: Trygve Reenskaug;
- Unisys Corporation: Sridhar Iyengar, GK Khalsa, Don Baisley.

OMG

Object Management Group (www.omg.org) è un gruppo di più di ottocento produttori, sviluppatori e utenti software dedicato alla promozione della tecnologia a oggetti utilizzata nello sviluppo di sistemi di elaborazione distribuiti. OMG è un'associazione non-profit che fornisce una struttura comune per la realizzazione di applicazioni a oggetti.

Domande all'esperto

? Dove è possibile trovare ulteriori informazioni sulla modellazione del software?

Esistono moltissimi siti sul Web che riportano informazioni sulla modellazione del software. Alcuni di essi sono riportati di seguito:

- **www.uml.org**: Object Management Group, i supervisor di UML;
- **www.cetus-links.org**: 18.309 link su oggetti e componenti;
- **www.rational.com/uml**: la pagina su UML di Rational Software;
- **www.sdmagazine.com/uml**: il centro risorse su UML della rivista Software Development Magazine;
- **www.well.com/user/ritchie/oo.html**: la pagina Object-Oriented di Ricardo Davis;
- **www.uml-zone.com**: la UML Zone di DevX.



Esercitazione flash

1. Chi furono le tre persone a contribuire maggiormente alla notazione UML?
2. Quale organizzazione sovrintende alla specifica UML?
3. Qual è lo scopo di questa organizzazione?

1.4 Parti di UML

UML può essere suddiviso in due grandi parti: i diagrammi strutturali e i diagrammi comportamentali.

Diagrammi strutturali

UML ha due tipi di diagrammi considerati strutturali: i diagrammi di classe e i diagrammi di implementazione. All'interno di queste due categorie, si possono trovare quattro tipi specifici di diagrammi.

1. Booch, Rumbaugh e Jacobson.
2. Object Management Group (OMG).
3. Standardizzare la tecnologia a oggetti per lo sviluppo di sistemi distribuiti.

Diagrammi di classe e diagrammi di oggetto

Un diagramma di classe viene utilizzato per rappresentare le parti sottostanti (le classi) di un sistema, le relazioni fra di esse e la loro appartenenza ai vari sottosistemi. I diagrammi di classe comprendono attributi e operazioni, insieme con molti tipi di ruoli e associazioni.

Un diagramma di oggetto è molto simile a un diagramma di classe, ma, invece di avere a che fare con le classi, mostra gli oggetti che sono istanze di una classe. Questi diagrammi vengono normalmente utilizzati quando si tratta di mostrare un esempio di progetto; in altre parole, gli oggetti hanno a che fare con cose singole uniche, mentre le classi sono più generiche.

Classi e oggetti, che sono alla base di qualsiasi sistema a oggetti, forniscono la gestione della propria funzionalità e dei propri dati. Le classi vengono utilizzate non soltanto per modellare il campo di azione durante l'inizio della fase di elaborazione (come si vedrà nel Capitolo 6), ma anche per realizzare porzioni complesse del modello di progettazione del software durante le iterazioni seguenti della medesima fase di elaborazione (come si vedrà nel Capitolo 8).

UML viene utilizzato per progettare sistemi mediante i concetti della programmazione a oggetti. Il Capitolo 3 tratta delle basi di un sistema a oggetti, fra le quali c'è il concetto di classe con l'incapsulamento, l'astrazione, l'ereditarietà e il polimorfismo. Si apprenderà a modellare i diagrammi di classe con un elevato livello di dettaglio, applicando i concetti della programmazione a oggetti come l'ereditarietà multipla e gerarchica attraverso la generalizzazione.

Diagrammi di componente e diagrammi di deployment

Un diagramma di deployment modella il punto in cui i componenti andranno a finire quando saranno installati sui sistemi informatici e come questi sistemi interagiscono gli uni con gli altri. Un diagramma di componente viene utilizzato per illustrare come i componenti di un sistema interagiscono fra di loro, mostrando le dipendenze fra file sorgente e classi nonché a quali componenti appartengono. Nella specifica UML 2.0 proposta, i diagrammi di componente diventano una funzionalità di base, il che significa che diventano più importanti nella modellazione di quanto non siano considerati a tutt'oggi.

Diagrammi comportamentali

I diagrammi comportamentali vengono utilizzati per mostrare come il processo scorre fra componenti, classi, utenti e sistema. Ci sono cinque diagrammi comportamentali in UML.

Diagrammi di caso d'uso

I diagrammi di caso d'uso contengono casi d'uso e attori, illustrando la relazione esistente fra i due insiemi. I diagrammi di caso d'uso sono il punto d'inizio della fase di analisi durante la progettazione di un sistema. Inventati da Ivar Jacobson, i casi d'uso

sono il fondamento di un diagramma di caso d'uso; vengono uniti mediante associazioni e collegati agli attori in modo da visualizzare la struttura globale e la disponibilità di un sistema per il personale non tecnico, come gli utenti finali. I casi d'uso possono essere utilizzati per realizzare il diagramma del flusso di eventi principale di un sistema, così come va avanti quando non si presentano errori. Possono anche essere utilizzati per realizzare un diagramma dei flussi alternativi (correlati direttamente con le situazioni che richiedono la gestione di errori). Nel Capitolo 2 verranno esposti alcuni esempi che serviranno a interpretare e a creare diagrammi di caso d'uso basandosi sulle richieste degli utenti.

Diagramma di attività

Il diagramma di attività viene utilizzato per analizzare il comportamento all'interno di casi d'uso complessi e per visualizzare le interazioni fra di essi. I diagrammi di attività sono molto simili ai diagrammi di stato, poiché, in effetti, rappresentano un flusso di dati; a differenza di quelli, però, i diagrammi di attività vengono utilizzati per modellare i flussi di lavoro durante la progettazione dei casi d'uso. I diagrammi di attività vengono normalmente utilizzati per rappresentare le attività più complesse, aiutando a identificare i casi d'uso o l'interazione fra di essi e al loro interno. Poiché i diagrammi di attività vengono utilizzati presto durante il processo di modellazione del software, verranno trattati già nel Capitolo 4. I diagrammi di stato, invece, vengono utilizzati per mostrare i cambiamenti di stato del sistema e il raggiungimento di determinate posizioni (o stati).

Diagrammi di sequenza

I diagrammi di sequenza vengono utilizzati per mostrare l'interazione fra attori e oggetti e altri oggetti. Da un attore vengono inviati messaggi a un oggetto, da un oggetto vengono inviati messaggi a un altro oggetto e da un oggetto vengono inviati messaggi a un attore, in modo da rendere chiaro il flusso di controllo attraverso un sistema. I diagrammi di sequenza vengono utilizzati per chiarificare i casi d'uso, documentando come un caso d'uso viene risolto con l'attuale progetto del sistema. I diagrammi di sequenza modellano l'interazione fra istanze di alto livello delle classi, osservando nel dettaglio dove si trova il controllo del processo in ciascuna fase della comunicazione. I diagrammi di sequenza possono essere utilizzati per mostrare, all'interno di una determinata interazione, ogni possibile percorso o un percorso singolo. Dei diagrammi di sequenza di parla nel Capitolo 5, poiché vengono utilizzati per modellare il dominio, e nel modulo 9, perché vengono utilizzati per progettare il modello.

Diagrammi di collaborazione

I diagrammi di collaborazione vengono utilizzati per portare avanti i diagrammi di classe. Essi rappresentano l'interazione e la relazione fra oggetti creati nei precedenti passaggi del processo di modellazione del dominio. Questi diagrammi possono essere utilizzati anche per modellare i messaggi fra oggetti diversi. La navigazione delle as-

sociazioni avanzate fra questi oggetti viene spiegata mediante un diagramma di collaborazione; i diagrammi di collaborazione sono illustrati nel Capitolo 7.

Diagrammi di stato

I diagrammi di stato vengono utilizzati per modellare il comportamento dei sottosistemi, le interazioni con le classi e le interfacce del sistema, nonché per rendere chiari i casi d'uso. Molto simili a un modello di macchina a stati, i diagrammi di stato vengono utilizzati durante la fase di passaggio fra l'analisi e la progettazione e rappresentano un ottimo metodo per visualizzare il flusso di un'applicazione. Nel Capitolo 10 verrà spiegato come leggere un diagramma di stato e come crearne uno a partire da entità e casi d'uso complessi.



Esercitazione flash

1. Quale diagramma si usa per modellare le richieste degli utenti?
 2. Quali sono i due tipi di diagrammi di implementazione?
 3. Quale diagramma si usa per modellare casi d'uso complessi?
-

1.5 Strumenti di modellazione

Esistono diversi strumenti di modellazione, che vanno da quelli gratuiti a quelli che costano decine di migliaia di euro. Lo strumento scelto per la modellazione di un sistema con UML dipenderà dalle proprie esigenze, dalle proprie preferenze nonché, naturalmente, dal budget che si ha a disposizione. L'elenco degli strumenti di modellazione UML è molto lungo; a mio avviso, due dei migliori sono Rational Rose, di Rational Software, e Visio, di Microsoft, ma anche carta e penna non sono da scartare.

Rational Rose

Rational Software, azienda fortemente coinvolta nella standardizzazione di UML e che ha assunto tutti e tre i creatori originali di UML, è stata una delle prime a mettere insieme un package completo per UML. Anche se può apparire caro, Rational Rose è un pacchetto veramente completo che consente di effettuare un reverse engineering del proprio codice per creare dei modelli, di modificare questi modelli e quindi di aggiornare il codice in modo che rifletta le modifiche apportate. Questo modo di proce-

-
1. Il diagramma di caso d'uso.
 2. I diagrammi di componente e di deployment.
 3. Il diagramma di collaborazione.

dere viene chiamato *round-trip engineering* ed è uno dei punti di forza di Rational Rose.

Rational Rose ha anche una completa gestione degli oggetti, ottenuta mediante un repository di classi e diagrammi tale che una modifica in una classe appartenente a un diagramma si riflette immediatamente in tutti i diagrammi che contengono detta classe. Questa caratteristica facilita una progettazione rapida del sistema e riduce gli errori rispetto agli strumenti che non considerano globali queste informazioni, perché mantengono i diagrammi non correlati gli uni con gli altri.

Visio

Visio, ora un prodotto di Microsoft, è grandemente migliorato rispetto alle funzionalità UML offerte. È stato aggiunto un gestore di oggetti, che offre la gestione dei componenti come Rational Rose. Gli oggetti (classi, casi d'uso, attività, package e così via) creati in un diagramma possono essere migrati ad altri diagrammi con un semplice drag-and-drop.

Anche se Visio può, per certi versi, essere comparato a Rational Rose, è molto più economico e consente di creare diagrammi più o meno di qualsiasi genere, da layout di rete a piantine dell'ufficio, fino alle istruzioni stradali. Uno dei punti di forza di Visio è il fatto che si tratta di un componenti di Microsoft Office, sebbene non venga venduto solitamente insieme con Microsoft Office. Ciò significa che l'interfaccia, i controlli e la funzionalità sono quelli standard di prodotti come Word ed Excel. È quindi molto facile capire come funziona Visio.

Carta e penna

Ovviamente, ci sono anche i sistemi classici. Anche se è possibile spendere 10.000 euro per uno strumento di progettazione UML, carta e penna restano strumenti indispensabili. Fra l'altro, si tratta di strumenti molto facili da usare rispetto a uno strumento elettronico, soprattutto se si tratta di buttar giù un'idea.



Modulo 1 – Verifica

1. Perché si modella il software?
2. Quali sono i tre passaggi principali nella modellazione del software?
3. Quali sono i tre modelli principali del ciclo di vita di un software?
4. Quali sono le quattro fasi del Processo Unificato?
5. Chi sono le tre persone che maggiormente hanno contribuito alla creazione di UML?

6. Qual è stata la prima versione di UML?
7. Che cosa significa l'acronimo OMG e che cosa fa OMG?
8. Quali sono i diagrammi strutturali in UML?
9. Quali sono i diagrammi comportamentali in UML?
10. Quali strumenti si possono usare per modellare con UML?