

Ingegneria del Software I

Unified Modeling Language

Introduzione alla notazione

Introduzione al processo di sviluppo

1

Modeling

Se si vuole costruire una cuccia al proprio cane: un martello, dei chiodi, un metro, un po' di legno

Se si vuole costruire un grattacielo probabilmente lo si sta facendo con i soldi di qualcun altro (vorrà mettere bocca sulle dimensioni, le finestre, il numero di ascensori) e non si procederà all'impresa da soli (i ruoli coinvolti potrebbero essere centinaia, le persone fisiche migliaia).

E' NECESSARIO MODELLARE

2

Modelli ...

◆ Cos'è un modello?

- Un modello è una semplificazione della realtà
 - » Le quattro equazioni di Maxwell sintetizzano tutti i fenomeni elettromagnetici.

◆ Perché costruire modelli?

- I modelli rappresentano il linguaggio dei progettisti
 - » I modelli sono un veicolo di comunicazione
 - » I modelli descrivono in modo "visuale" il sistema da costruire
 - » I modelli specificano la struttura e il comportamento di un sistema, forniscono una guida nella costruzione del sistema e documentano le decisioni prese

3

... modelli ...

◆ Un solo modello non è sufficiente

- Ogni sistema non banale deve essere affrontato con un insieme di modelli quasi indipendenti
 - » Nel modello di un grattacielo, devo modellare gli ascensori, gli impianti di riscaldamento, l'impianto elettrico ...
- Un stesso modello può essere visto secondo diversi livelli di precisione.
 - » Modello di un impianto telefonico: connessioni tra centraline di piano ... diramazioni all'interno delle stanze ... doppino...
- I modelli sono uno strumento per gestire la complessità

4

Introduzione ad UML

Ingegneria del Software I

... modelli

Nello sviluppo dei sistemi software esistono due principali approcci alla definizione di un modello:

Algoritmico: i componenti del sistema sono le funzioni o le procedure, i progettisti si concentrano sulle problematiche legate alla decomposizione di algoritmi in sottoalgoritmi ...

Orientato agli Oggetti: i componenti del sistema sono gli oggetti che riflettono il dominio del problema, ognuno di essi ha un'identità, uno stato ed un comportamento.

Anche se UML è indipendente dalla metodologia di progetto si presta molto meglio ai modelli Orientati agli Oggetti.

5

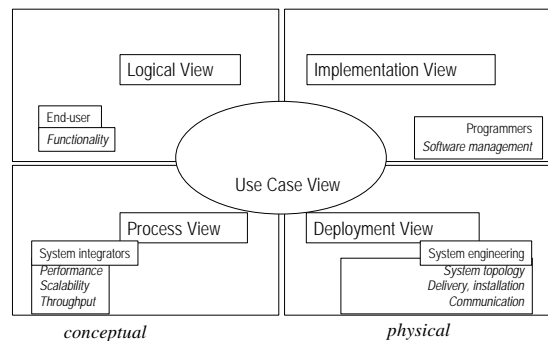
Architettura di un Sistema Software

È definita dall'insieme delle decisioni significative legate:

- all'organizzazione di un sistema software
- alla selezione degli elementi strutturali che compongono il sistema e delle relative interfacce,
- al comportamento specificato mediante collaborazioni tra gli elementi strutturali
- alla composizione di questi elementi strutturali e comportamentali in sottosistemi
- allo stile architeturale che guida l'organizzazione

6

Architettura software: 4 + 1 views



P. Kruchten, "The 4+1 View Model of Software Architecture".
IEEE Software 12 (6), pages 42-50, november 1995

7

Viste dell'Architettura del Sistema ...

Use Case View

E' relativa a come il sistema è visto dagli utenti, dagli analisti e dai testatori. Essa non specifica, realmente, il modo secondo cui è organizzato il sistema; piuttosto, evidenzia le forze che modellano l'architettura del sistema.

8

Ingegneria del Software I

... viste dell'Architettura del Sistema ...

Logical View

Comprende le classi, le interfacce e le collaborazioni relative alla realtà di interesse che si sta modellando. Essa è diretta espressione dei requisiti funzionali del sistema, ovvero dei servizi che il sistema offre

Process View

Le attività del sistema (sia quelle visibili direttamente all'utente che quelle di supporto) prevedono l'organizzazione di threads che potranno andare in concorrenza e/o richiedono attività di sincronizzazione.

9

... viste dell'Architettura del Sistema

Implementation View

Concerne le problematiche di configuration management relative ai file che ospitano i componenti il cui assemblaggio consente la creazione della release eseguibile.

Deployment View

Concerne gli elementi hardware che ospiteranno il sistema, in particolare, cura le problematiche relative alla distribuzione ed all'installazione.

10

Quante viste ?

- ◆ È necessario definire il modello più semplice per risolvere il problema
- ◆ Non tutti i sistemi richiedono tutte le viste
- ◆ Viste aggiuntive
 - Dati, sicurezza, ...

11

Che cosa è UML

UML (Unified Modeling Language) è un LINGUAGGIO, e non una semplice notazione per disegnare diagrammi, attraverso cui è possibile CATTURARE e TRASFERIRE la conoscenza relativa ad un soggetto (sistema software) ed esprimere tale conoscenza a scopi di comunicazione.

In particolare, esso consente di MODELLARE le caratteristiche del soggetto (sistema software) ed è il risultato dall'UNIFICAZIONE delle migliori (e più diffuse) metodologie e tecniche usate nell'industria per la realizzazione di sistemi informatici.

12

Introduzione ad UML

Ingegneria del Software I

Caratteristiche Principali ...

- Si tratta di un linguaggio, non di un metodo, quindi non indicherà cosa fare per sviluppare un sistema software.
- UML è un linguaggio per visualizzare, specificare, costruire, documentare gli artefatti di un sistema software
 - è indipendente da qualsiasi linguaggio di programmazione
 - non prescrive una sequenza di processo (non dice “prima bisogna fare questa attività, poi quest’altra, etc.”), pur coprendo l’intero processo di produzione, e quindi può essere utilizzato da persone e gruppi che seguono metodi diversi (è “indipendente dai metodi”).
 - è supportato da più strumenti

13

... caratteristiche principali ...

UML è uno standard aperto dell’OMG (Object Management Group)

- riunisce molte proposte esistenti
- è sponsorizzato dalle maggiori industrie produttrici di software
- definisce una notazione standard, basata su un metamodello integrato degli elementi che compongono un sistema software.
- è un linguaggio non proprietario (i suoi autori non hanno il copyright su UML)

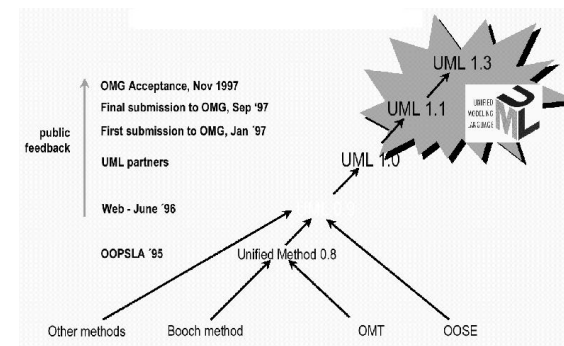
14

... caratteristiche principali

- UML riunisce aspetti dell’ingegneria del software, delle basi di dati e della progettazione di sistemi
- UML è utilizzabile in domini applicativi diversi e per progetti di diverse dimensioni
 - Enterprise MIS, Banking/ financial services, Telecommunications, Transportation, Defense/ aerospace, Retail, Medical electronics, Scientific
- UML è estendibile per modellare meglio le diverse realtà
- UML permette la specializzazione di concetti, notazione e vincoli per domini particolari
- UML consente l’approccio alla complessità architeturale dei problemi usando component technology, visual programming e patterns

15

Storia dell’UML



16

Introduzione ad UML

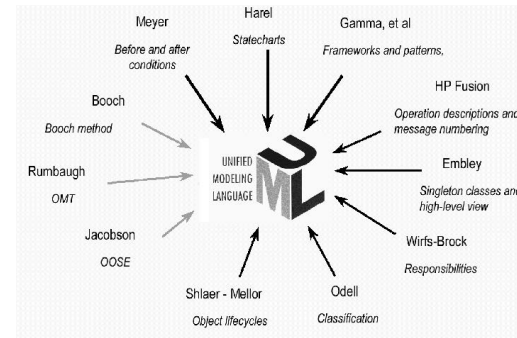
Ingegneria del Software I

Partner

- ◆ Aonix
- ◆ Colorado State University
- ◆ Computer Associates
- ◆ Concept Five
- ◆ Data Access
- ◆ EDS
- ◆ Enea Data
- ◆ Hewlett-Packard
- ◆ IBM
- ◆ I-Logix
- ◆ InLine Software
- ◆ Intellicorp
- ◆ Kabira Technologies
- ◆ Klasse Objecten
- ◆ Lockheed Martin
- ◆ Microsoft
- ◆ ObjecTime
- ◆ Oracle
- ◆ Ptech
- ◆ OAO Technology Solutions
- ◆ Rational Software
- ◆ Reich
- ◆ SAP
- ◆ Softeam
- ◆ Sterling Software
- ◆ Sun
- ◆ Taskon
- ◆ Telelogic
- ◆ Unisys
- ◆ ...

17

Principali contributi



18

Modello concettuale di UML

- ◆ UML è basato su un meta-modello integrato composto da numerosi elementi collegati tra loro secondo regole precise, che consentono di creare modelli particolari
 - *Building blocks*
 - » Elementi (costituiscono le principali astrazioni di un modello)
 - » Relazioni (consentono di mettere in relazione le astrazioni)
 - » Diagrammi (raggruppano collezioni di interesse di astrazioni)
 - *Regole* indicanti come comporre insieme i building blocks
 - *Meccanismi* che si applicano attraverso tutto lo UML
 - » Specifications (specifiche testuali che accompagnano quelle grafiche)
 - » Adornments (dettagli grafici o testuali aggiunti ai simboli base)
 - » Common divisions (per distinguere tra astrazioni e loro istanze)
 - » Extensibility mechanisms (Stereotypes, tagged values, constraints)

19

Elementi

- ◆ *Elementi strutturali*, costituiscono la parte statica di un modello, rappresentano elementi sia concettuali che fisici:
 - use case, class, interface, collaboration, component, node
- ◆ *Elementi comportamentali*, costituiscono la parte dinamica del modello, rappresentano il comportamento del modello nel tempo e nello spazio:
 - interaction, state machine
- ◆ *Elementi di raggruppamento*, costituiscono la parte organizzazionale del modello, consentono la decomposizione del modello:
 - Package (subsystem)
- ◆ *Elementi annotazionali*, costituiscono la parte 'esplicativa' del modello, sono commenti per descrivere, evidenziare, rimarcare qualsiasi elemento del modello:
 - annotazioni

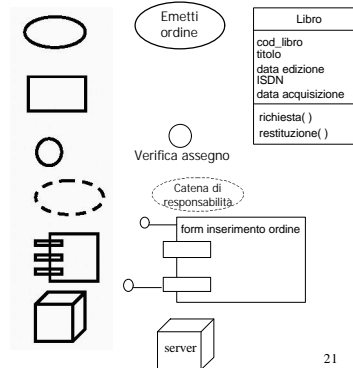
20

Introduzione ad UML

Ingegneria del Software I

Elementi strutturali

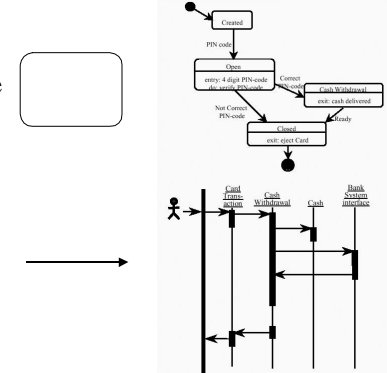
- ◆ Use case
- ◆ Class
- ◆ Interface
- ◆ Collaboration
- ◆ Component
- ◆ Node



21

Elementi comportamentali

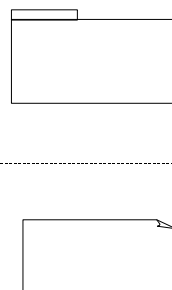
- ◆ State machine
- ◆ Interaction



22

Elementi di raggruppamento (package) e annotazioni

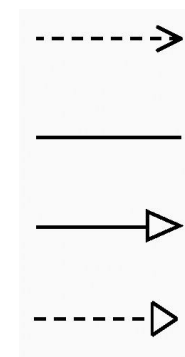
- ◆ Package
- ◆ Annotazioni



23

Relazioni

- ◆ Dependency
- ◆ Association
- ◆ Generalization
- ◆ Realization



24

Ingegneria del Software I

Diagrammi UML

- ◆ **Viste statiche**
 - Use Case Diagrams
 - Class Diagrams
 - Object Diagrams
 - Component Diagrams
 - Deployment Diagrams
- ◆ **Viste dinamiche**
 - Sequence Diagrams
 - Collaboration Diagrams
 - Statechart Diagrams
 - Activity Diagrams

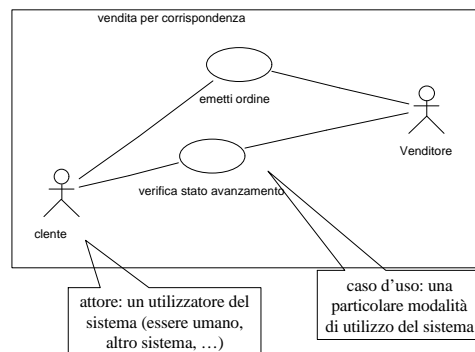
25

Use case diagram

- ◆ **Cattura le funzionalità del sistema dal punto di vista degli utenti**
 - Raccoglie un insieme di use case ed attori (un tipo speciale di classi) e le relazioni tra essi.
 - Rappresenta le modalità di utilizzo del sistema da parte di uno o più utilizzatori (attori).
 - Descrive l'interazione tra attori e sistema, non la "logica interna" della funzione.
 - Descrive l'organizzazione ed il modello del comportamento di un sistema.
- ◆ **Costruito nei primi passi dello sviluppo**
 - Ragionare sui casi d'uso aiuta ad individuare i requisiti funzionali.
- ◆ **Scopo**
 - Specificare il contesto di un sistema
 - Catturare i requisiti di un sistema
 - Validare l'architettura di un sistema
 - Guidare la realizzazione
 - Generare i casi di test
- ◆ **Sviluppato da analisti ed esperti di dominio**

26

Esempio di use case diagram



27

Class diagram

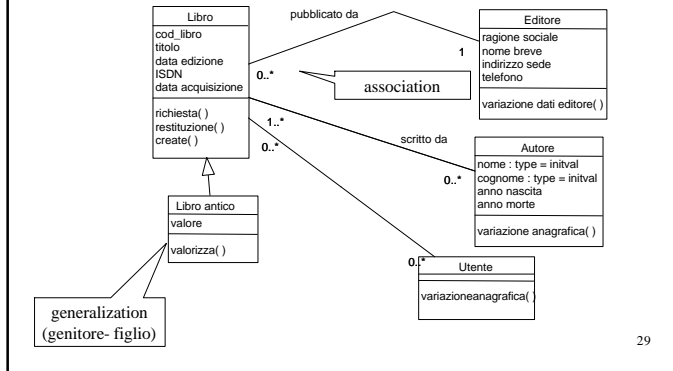
- ◆ **Cattura il vocabolario di un sistema**
 - Un class diagram rappresenta un insieme di classi, interface, collaborazioni e relazioni.
 - Specifica i vincoli e le relazioni che sussistono tra le classi.
 - Forniscono una rappresentazione statica di un sistema.
- ◆ **Costruito e raffinato per tutto lo sviluppo**
- ◆ **Scopo**
 - Nominare e modellare i concetti nel sistema
 - Specificare le collaborazioni
 - Specificare gli schemi logici di database
- ◆ **Sviluppato da analisti, progettisti e realizzatori**

28

Introduzione ad UML

Ingegneria del Software I

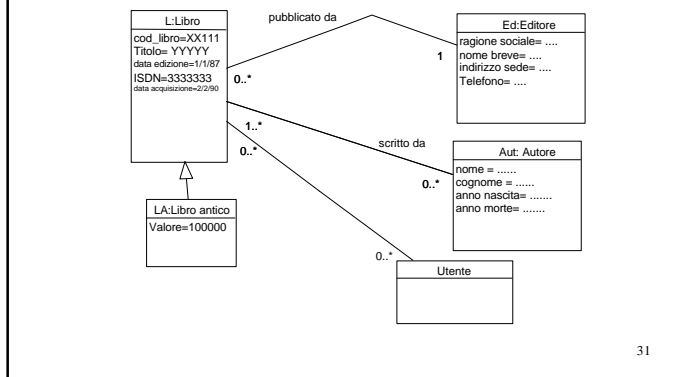
Esempio di class diagram



Object diagram

- ◆ Cattura le istanze e i legami
 - rappresenta un insieme di oggetti e le relazioni che intercorrono tra essi.
 - i suoi elementi sono una istanziazione di elementi specificati in qualche class diagram.
 - rappresentano la vista statica di un sistema (o vista statica di un processo) ma dalla prospettiva di un caso reale, o prototipale.
 - ◆ Costruito durante l'analisi e il progetto
 - ◆ Scopo
 - illustrare le strutture di dati / oggetti
 - specificare istantanee
 - ◆ Sviluppato da analisti, progettisti e realizzatori
- 30

Esempio di object diagram

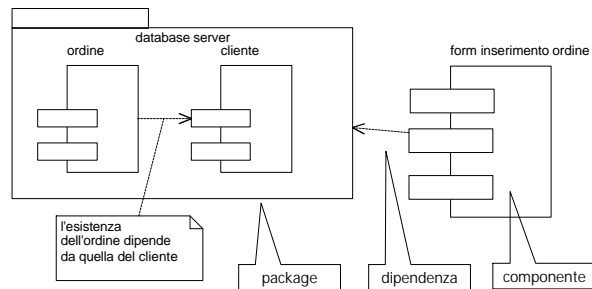


Component diagram

- ◆ Cattura la struttura fisica della realizzazione
 - evidenzia l'organizzazione e le dipendenze esistenti tra componenti, moduli software eseguibili, dotati di identità e con un'interfaccia ben specificata
 - Sono correlati ai class diagram, poiché un componenti, tipicamente, mappa una o più classi, interface o collaborazioni
 - ◆ Costruito come parte della specifica architetturale
 - ◆ Scopo
 - Organizzare il codice sorgente
 - Costruire una release eseguibile
 - Specificare un database fisico
 - ◆ Sviluppato da architetti e programmatori
- 32

Ingegneria del Software I

Esempio di component diagram



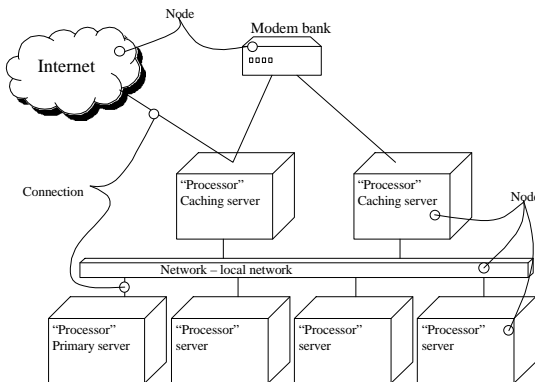
33

Deployment diagram

- ◆ Evidenzia la configurazione dei nodi elaborativi in ambiente di esecuzione (run-time), e dei componenti, processi ed oggetti ubicati in questi nodi
- ◆ Permette di rappresentare, a diversi livelli di dettaglio, l'architettura fisica del sistema, poiché cattura la topologia dell'hardware di un sistema
- ◆ Costruito come parte della specifica architetturale
- ◆ Scopo
 - Specificare la distribuzione dei componenti
 - Identificare i colli di bottiglia delle prestazioni
- ◆ Sviluppato da architetti, sistemisti di rete e di sistema operativo

34

Esempio di deployment diagram



35

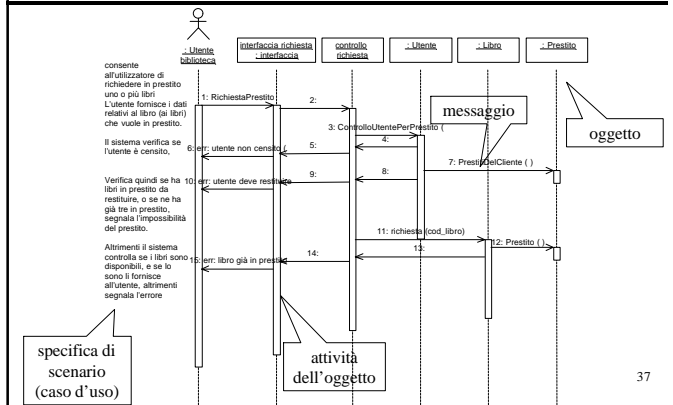
Sequence diagram

- ◆ Cattura la dinamica del comportamento (time-oriented)
 - ◆ Descrive dinamicamente le caratteristiche di un sistema software, evidenziando le interazioni dovute allo scambio di messaggi ed il loro ordinamento temporale
- ◆ Scopo
 - Modellare il flusso di controllo
 - Illustrare scenari tipici

36

Ingegneria del Software I

Esempio di sequence diagram

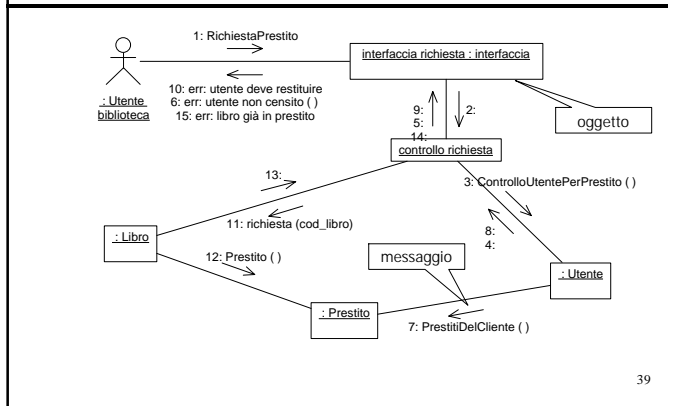


Collaboration diagram

- ◆ Cattura la dinamica del comportamento (message-oriented)
 - ◆ Anch'esso descrive dinamicamente le caratteristiche di un sistema software, evidenziando le interazioni dovute allo scambio di messaggi ed il loro ordinamento temporale
 - ◆ Diagrammi di sequenza e diagrammi di collaborazione esprimono informazioni simili, ma le evidenziano in modo diverso. Tra tali diagrammi è possibile istituire un isomorfismo, cioè è possibile trasformare uno nell'altro.
- ◆ Scopo
 - Modellare il flusso di controllo
 - Illustrare il coordinamento tra oggetti

38

Esempio di collaboration diagram



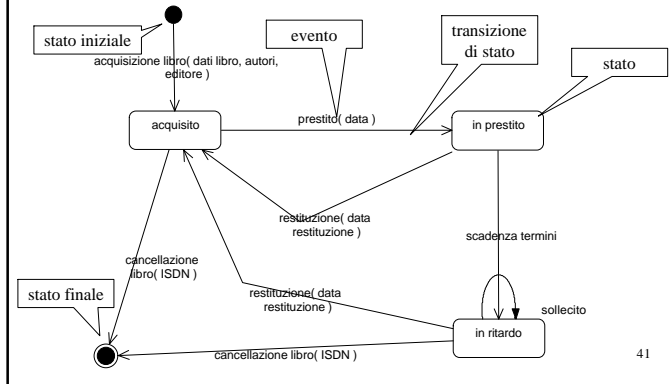
Statechart diagram

- ◆ Cattura la dinamica del comportamento (event-oriented)
 - ◆ Descrive il comportamento di un sistema in termini di stati, transizioni, eventi ed attività, quindi ne evidenziano la dinamica
- ◆ Scopo
 - Modellare il ciclo di vita di un oggetto
 - Modellare oggetti reattivi (interfacce utente, dispositivi, ecc.)

40

Ingegneria del Software I

Esempio di statechart diagram

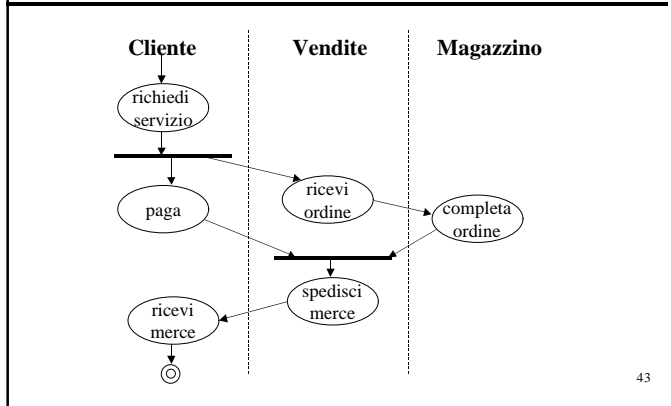


Activity diagram

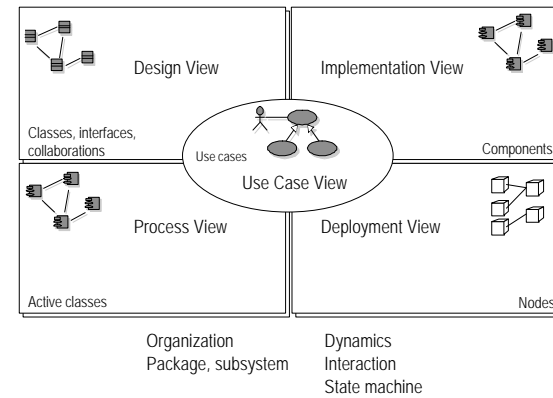
- ◆ Cattura la dinamica del comportamento (activity-oriented)
 - Mostrano il flusso da un'attività ad un'altra in un sistema.
 - Rappresentano sistemi di workflow, oppure la logica interna di un processo (di qualunque livello, dai business process ai processi di dettaglio).
 - Permettono di rappresentare processi concorrenti e la loro sincronizzazione.
- ◆ Scopo
 - Modellare business workflows
 - Modellare operazioni

42

Esempio di activity diagram



Architettura Software e UML



Introduzione ad UML

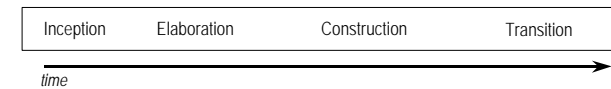
Ingegneria del Software I

UML e processo di sviluppo software

- ◆ UML è process independent
- ◆ Proposto uno Unified Process suddiviso in fasi e caratterizzato da tre aspetti essenziali:
 - Use case driven
 - » gli use case sono gli artefatti primari per stabilire il desiderato comportamento del sistema, per verificarne e validarne l'architettura, per il testing e per la comunicazione tra le persone coinvolte
 - Architecture-centric
 - » l'architettura del sistema è usata quale artefatto primario per concettualizzare, costruire, gestire ed evolvere il sistema in sviluppo
 - Iterative and incremental
 - » Continua integrazione del sistema e rilasci successivi

45

Unified Process: Fasi



- Inception* Definizione dello scopo del progetto, sviluppo dei business case
- Elaboration* Pianificazione dello sviluppo, specificazione e definizione dei requisiti e dell'architettura
- Construction* Implementazione del prodotto, test
- Transition* Beta test, rilascio del prodotto agli utilizzatori

46

Le fasi - Inception

Non esiste alcun documento di progetto del sistema, alcuna linea di codice, non si sono definiti gli oggetti che modelleranno il dominio di interesse. In questa fase si prova a rispondere a tre domande:

- 1) cosa dovrà fare il sistema per i suoi principali utenti ?
- 2) quale potrebbe essere un'architettura plausibile ?
- 3) qual'è il piano di sviluppo del sistema, qual'è il suo costo?

Infine, si individuano le funzionalità critiche del sistema oltre ai rischi critici (entrambi vanno prioritizzati).

47

Le fasi – Elaboration

Si sviluppano dettagliati use case in diretta risposta ai requisiti del sistema. Si avvia lo sviluppo dell'architettura del sistema, essa va rappresentata secondo le 5 viste di cui prima.

Si comincia dagli use case relativi alle funzionalità critiche. Al termine della prima iterazione della fase di Elaboration si produce la BASELINE.

L'esperienza maturata e le informazioni generate durante lo sviluppo della BASELINE vengono utilizzate dal project manager per preparare i piani di sviluppi per le iterazioni e le fasi successive.

Comunque, al termine di questa fase qualcuno dovrà provare a rispondere alla domanda:

“gli use case, l'architettura ed i piani di sviluppo sono sufficientemente solidi e stabili da consentirci di affrontare i rischi?”

48

Introduzione ad UML

Ingegneria del Software I

Le fasi - Construction

In questa fase si avvia la costruzione vera e propria del sistema. Si procede all'evoluzione dell'architettura BASELINE e mano a mano che l'architettura si stabilizza aumenta l'attività di codifica. Le release dovranno essere sempre meno prototipali.

Al termine di questa fase il sistema aderisce a tutti gli use case per esso definiti e si è pronti per consegnarlo agli utenti (beta). Ovviamente non è bug free.

49

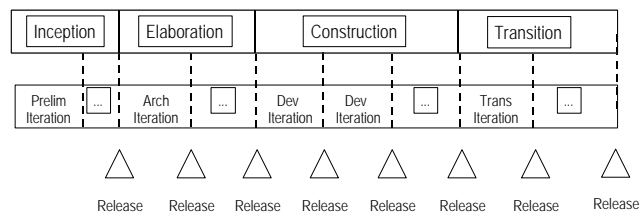
Le fasi - Transition

Questa fase prevede attività di beta-testing. Il sistema è rilasciato in uso ad utenti esperti che ne riportano difetti e malfunzionamenti

Ovviamente, il team di sviluppo deve porre rimedio a quanto sollevato dai beta-tester.

50

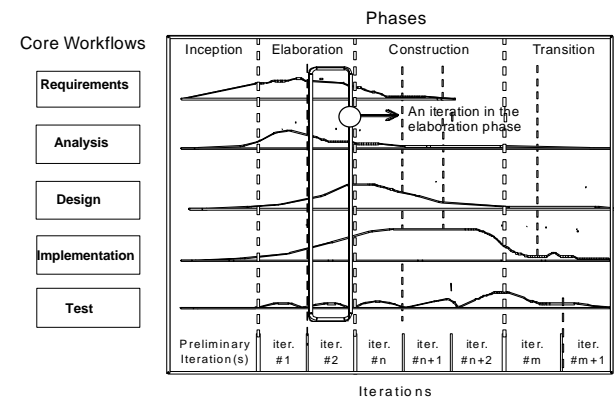
Fasi ed Iterazioni



Una iterazione è una sequenza di attività con predefinita pianificazione e predefiniti criteri di valutazione che porta ad un nuovo rilascio

51

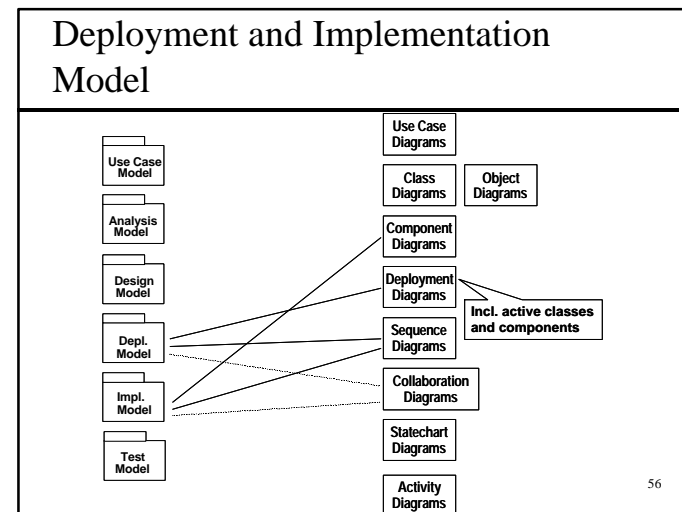
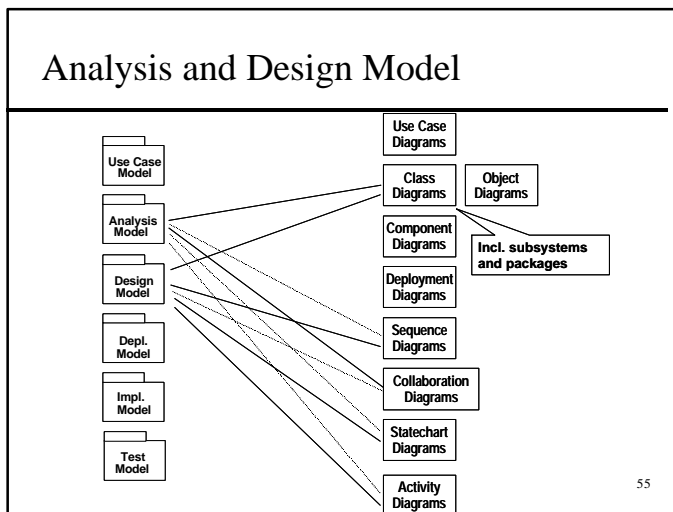
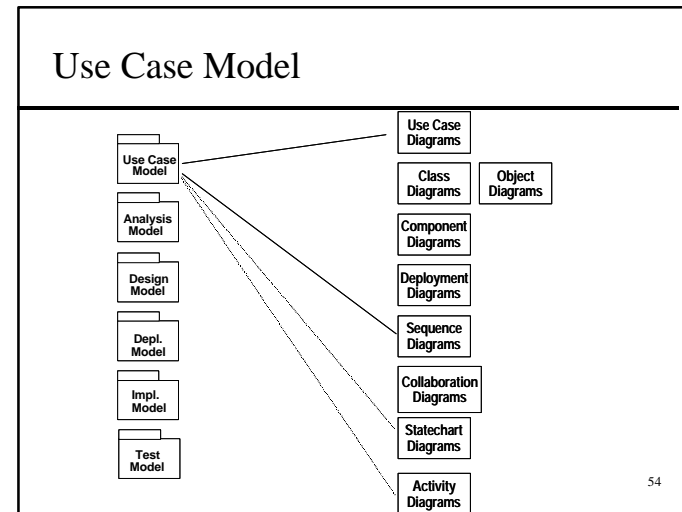
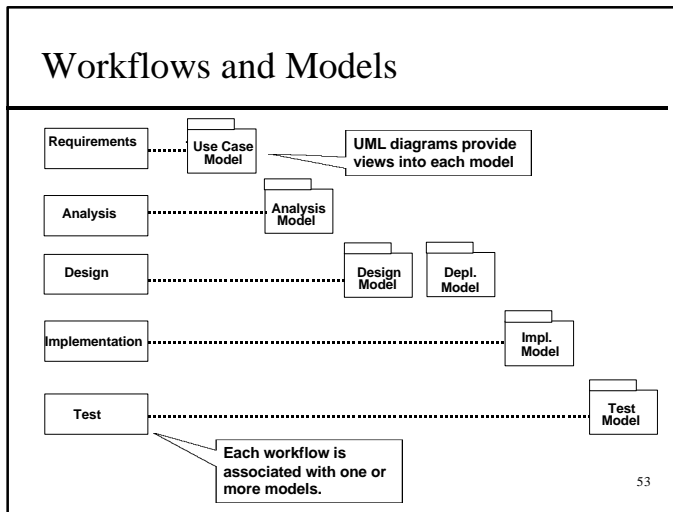
Unified Process



52

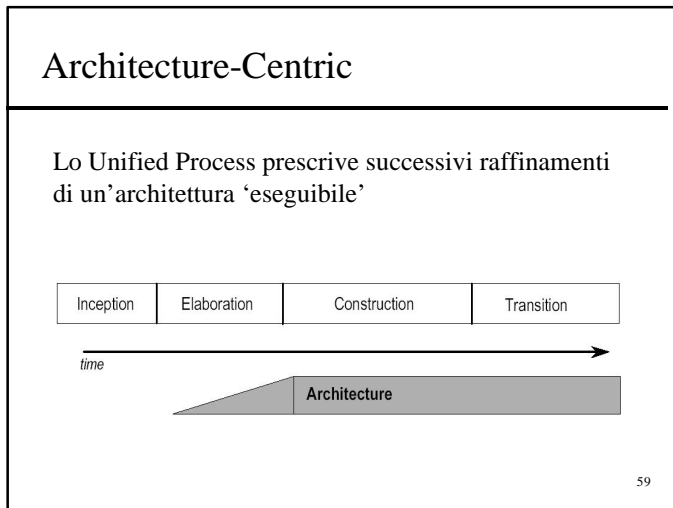
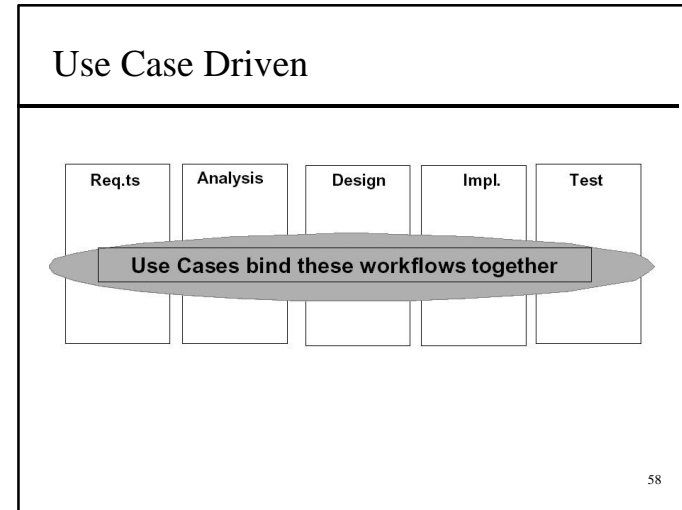
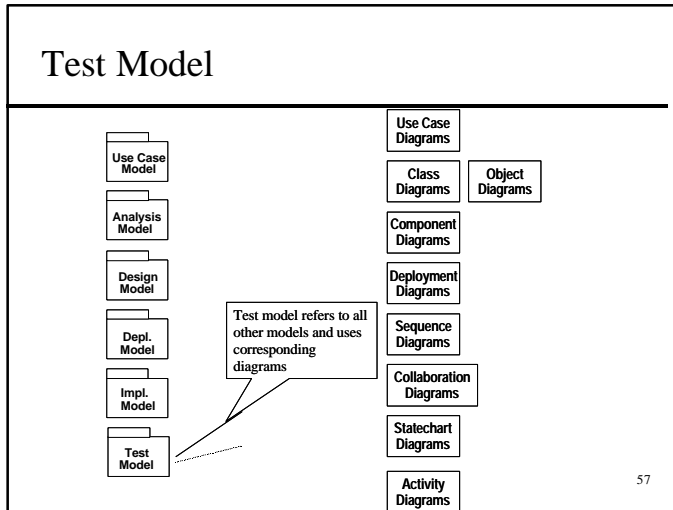
Introduzione ad UML

Ingegneria del Software I



Introduzione ad UML

Ingegneria del Software I



Risorse UML

- ◆ Grady Booch, James Rumbaugh, Ivar Jacobson, *The Unified Modeling Language User Guide*, Addison- Wesley, 1999.
- ◆ James Rumbaugh, Ivar Jacobson, Grady Booch, *The Unified Modeling Language Reference Manual*, Addison- Wesley, 1999.
- ◆ Ivar Jacobson, Grady Booch, James Rumbaugh, *The Unified Software Development Process*, Addison- Wesley, 1999.
- ◆ Terry Quatrani, *Visual Modeling with Rational Rose and UML*, Addison- Wesley, 2000
- ◆ Martin Fowler with Kendall Scott, *UML Distilled Second Edition: A brief guide to the standard object modeling language*, Addison- Wesley, 1999.
- ◆ Doug Rosenberg and Kendall Scott, *Use Case Driven Object Modeling with UML*, Addison- Wesley, 1999
- ◆ Philippe Kruchten, *The Rational Unified Process: An Introduction*, Addison- Wesley, 1999
- ◆ Rational UML Resource Center: <http://www.rational.com/uml/index.jsp>
- ◆ OMG Resource Page: <http://www.omg.org/uml/>
- ◆ Cetus links on UML: http://www.cetus-links.org/oo_uml.html

60

Introduzione ad UML