2018
PO: cenni di ingegneria del software e di UML

Giorgio Bruno

Dip. Automatica e Informatica
Politecnico di Torino
email: giorgio.bruno@polito.it

Software Engineering

The systematic approach to the development, operation, maintenance and retirement of software (IEEE glossary of software terminology).

The application of scientific principles to:

- the orderly transformation of a problem into a working solution;
- the subsequent maintenance of that software through the end of its useful life (Alan M. Davis).

Software life cycle and software process

A software product has a life cycle that starts when the product is conceived and finishes when the product is no longer available for use. It is made up of a number of **phases**.

Each phase has a set of inputs, a set of outputs and is carried by means of a number of **activities**.

The activities and the order they are carried out form the **software process**.

The development of high quality software products requires the *software process* to be based on sound *software engineering principles* incorporating best *software engineering practices*.

Software life cycle

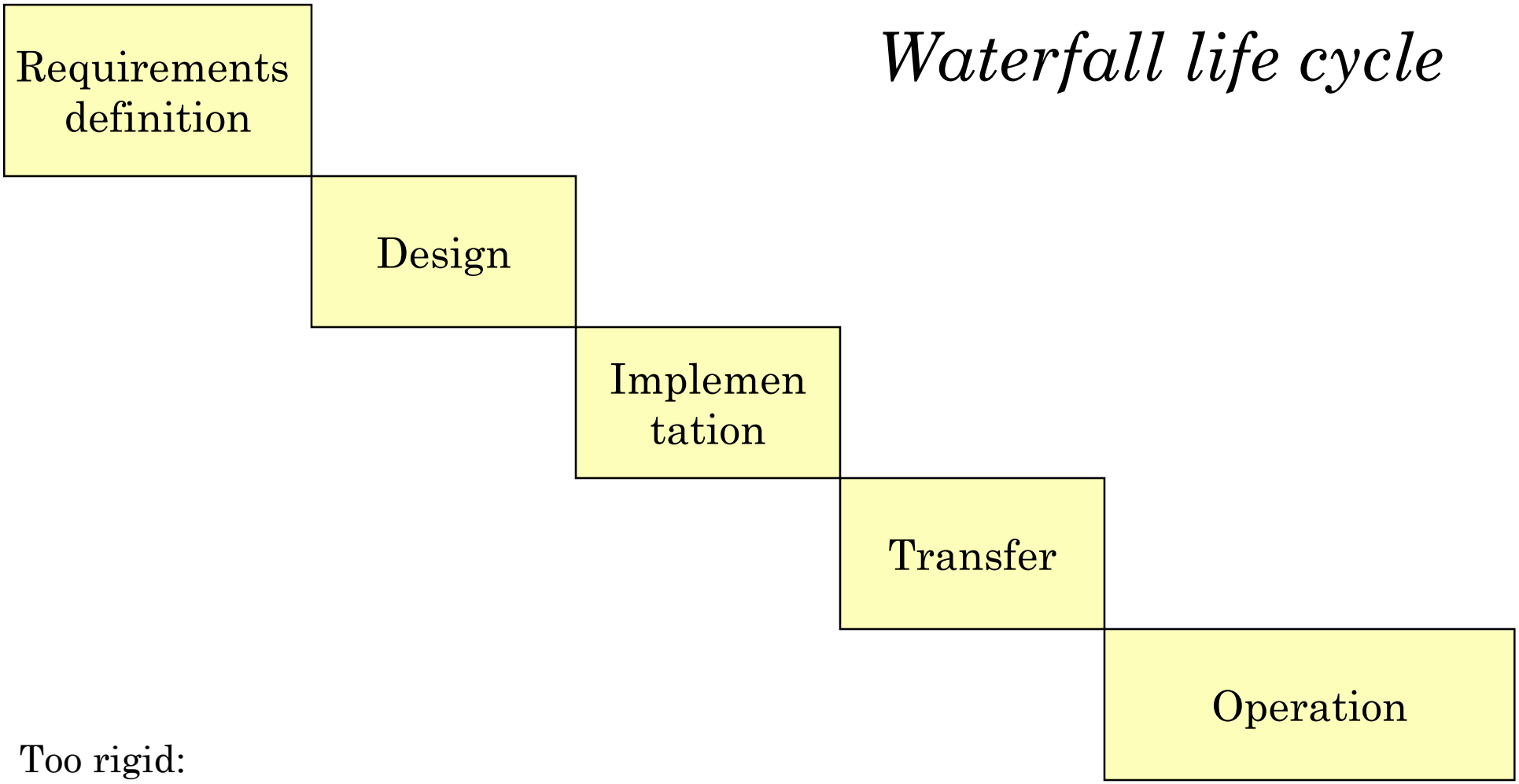
waterfall

- incremental
- evolutionary

unified

support activities

Waterfall life cycle



Too rigid:
software release at the very end (big bang release)
requirements frozen after the first phase

t

Phases

Requirements definition

overall description of the system in business terms, major functionalities and constraints

Architectural design

overall system architecture, interactions between components, linking requirements to components, make-or-buy decisions

Implementation

detailed design, development and testing

Transfer

acceptance testing and deployment (making the system ready for use)

Requirements definition

Software Requirement

Something that is wanted or needed (Webster's Dictionary)

Functional requirements

A functional requirement defines a capability or function that a system (or a subsystem) has to provide in order to **satisfy a customer need**.

It may concern a service, an activity, or a processing function that transforms input data into output data.

The focus is on what has to be done, not on how it is done.

The customer may be any actor involved in the process including the end user.

Functional requirements often include **structural** requirements.

Constraints (or **non functional** requirements)

related to performance, compliance to standards, conventions and regulations, ...

Quality issues

Requirements should be unambiguous, complete, consistent and verifiable.

Unambiguous requirement: only one interpretation.

Consistent requirement: no conflicts with the other requirements.

Verifiable requirement: its fulfillment can be verified (with tests or other means) in a cost-effective way.

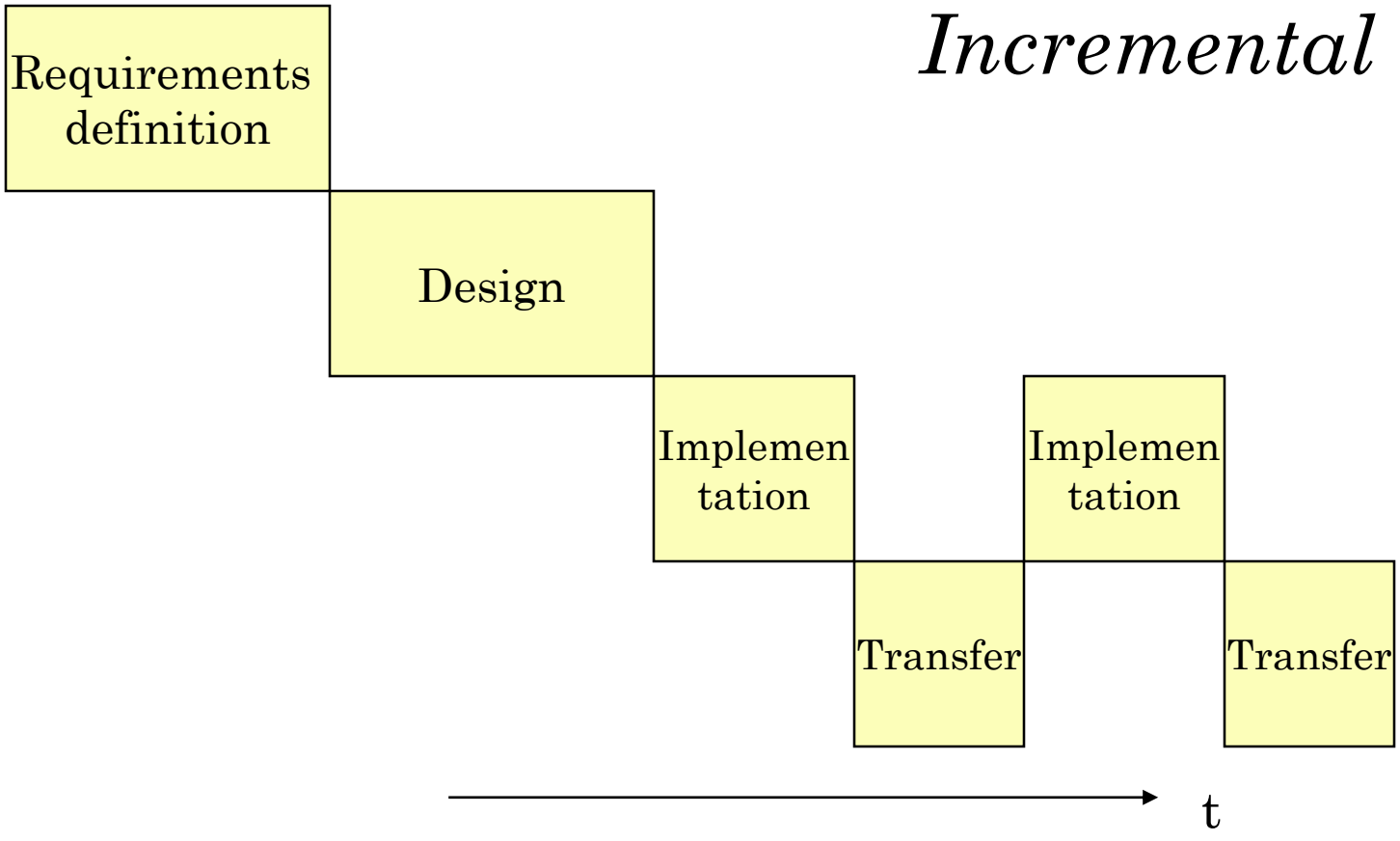
Complete requirements: they handle all the relevant issues.

Traceability of requirements

forward: from a requirement to the subsystem(s) fulfilling it.

backward: from subsystems to requirements.

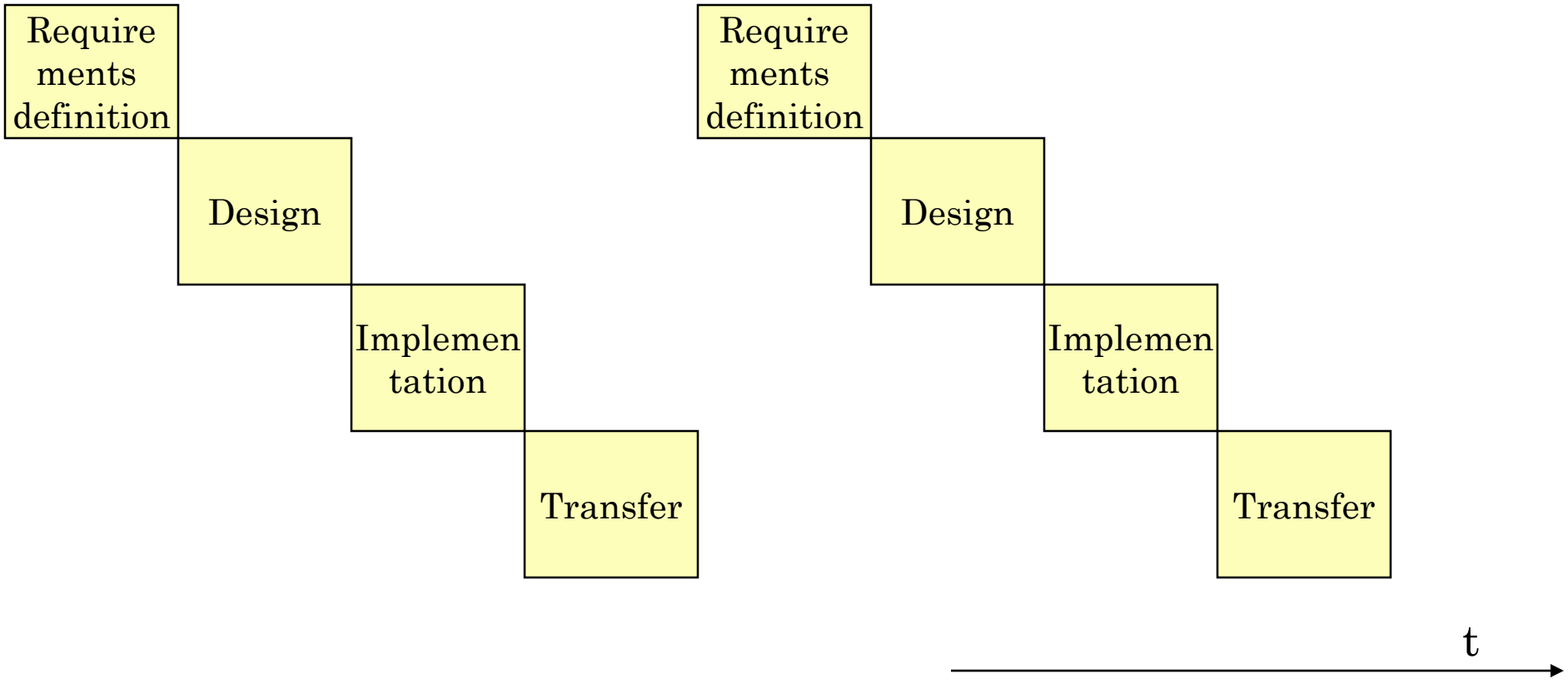
Incremental life cycle



Implementation and release in pieces rather than in one pass.

Incremental release demands a careful selection of the order in which the parts are released and the overall project cost may increase.

Evolutionary life cycle



All the phases can be repeated. This is useful when the initial requirements are poorly understood or unstable. It can give users an early operational capability. Each version of the system is a kind of prototype.

Prototyping

It is the technique of constructing a partial implementation of a system so that customers, users or developers *can learn* more about a problem or a solution to a problem.

Two approaches: throwaway and evolutionary.

A *throwaway prototype* should be **quick and dirty**. It can be used to validate a user interface, to check whether a particular architecture has the potential to meet the requirements, or to validate a particular algorithm.

On the contrary, since the *evolutionary prototype* evolves into the final product, it must exhibit all the quality attributes of the final product. It will not be particularly rapid.

Unified Process

It combines the incremental and evolutionary approaches, and is based on 4 phases. It uses UML diagrams.

A well known refinement is RUP (Rational Unified Process).

Inception: business case (business modeling), key requirements and constraints, risk assessment, preliminary project schedule and cost estimate.

Elaboration: detailed requirements analysis (resulting in use cases and class diagrams), design of the architecture and validation through prototypes, development plan.

Construction: implementation supported by various detailed UML diagrams.

Transition: deployment.

A type of activity does not coincide with a specific phase, although it may be the most important one in that phase.

Unified Process

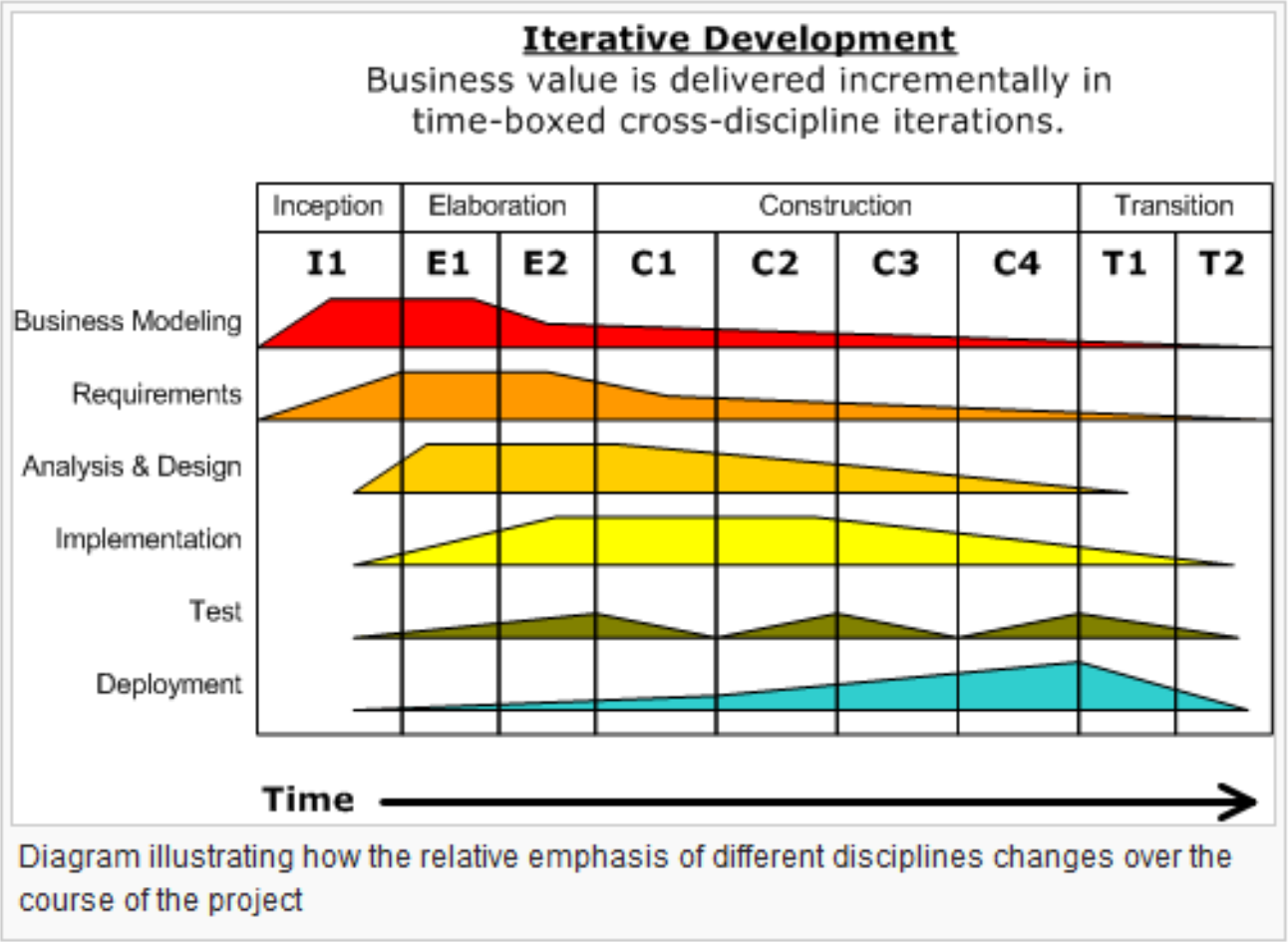


Diagram illustrating how the relative emphasis of different disciplines changes over the course of the project

from https://en.wikipedia.org/wiki/Unified_Process

Business modeling

Describes the context in which the intended software system will operate and the relationships with the other subsystems from a business point of view; it is a *bridge between business analysts and software analysts*.

Support activities

Project management: planning, staffing, monitoring, controlling and leading.

Configuration management: defining the elements of the system, controlling their release and change.

Verification (Are we building the product right?):

inspection (detecting defects through peer reviews), testing, static analysis.

Validation (Are we building the right product?): ensuring that the product meets the needs of the users.

Quality assurance: ensuring that standards and procedures are established and followed.

Nuovi approcci allo sviluppo del software

Agile development

Agile development

Manifesto for Agile Software Development
(http://www.agilemanifesto.org/)

Individuals and interactions over *processes and tools*
Working software over *comprehensive documentation*
Customer collaboration over *contract negotiation*
Responding to change over *following a plan*

Twelve principles

The Agile Manifesto is based on twelve principles:

1. *Customer satisfaction* by early and continuous delivery of valuable software
2. Welcome *changing requirements*, even in late development
3. Working software is *delivered frequently* (weeks rather than months)
4. Close, daily *cooperation between business people and developers*
5. Projects are built around *motivated individuals*, who should be trusted
6. *Face-to-face conversation* is the best form of communication (co-location)
7. *Working software* is the principal measure of progress
8. *Sustainable development*, able to maintain a constant pace
9. Continuous attention to *technical excellence and good design*
10. *Simplicity* - the art of maximizing the amount of work not done - is essential
11. Best architectures, requirements, and designs emerge from *self-organizing teams*
12. Regularly, *the team reflects* on how to become more effective, and adjusts accordingly

from https://en.wikipedia.org/wiki/Agile_software_development#cite_note-manifestoprinciples-15

Extreme programming (XP)

Sviluppo agile con pianificazione settimanale (planning game)

Requisiti basati su user stories

Pair programming

Test driven development

Continuous integration

Refactoring

Planning game

Release planning

- raccolta dei requisiti scritti come user stories con il coinvolgimento del committente;
- analisi benefici/tempi/costi;
- scelta dei requisiti da includere nella release.

Iteration planning

- definizione dei task;
- allocazione dei task agli sviluppatori;
- implementazione con test-driven development.

User story

Si scrive (su scheda) dal punto di vista dell'utente per esprimere un'esigenza.

Si usa un formato molto semplice per evitare oneri documentali e amministrativi.

Esempi relativi alla gestione di una biblioteca

Come *utente* della biblioteca vorrei poter segnalare i miei interessi per poter ricevere aggiornamenti sui nuovi libri disponibili.

Come *amministratore* vorrei poter avere informazioni di correlazione tra i libri presi in prestito in un dato periodo.

Come *utente* della biblioteca vorrei poter prenotare il prestito di un libro dato in prestito.

Pair programming

Si lavora in due sullo stesso programma; benefici:

coesione del gruppo (i partner cambiano) e on-the-job training per i nuovi assunti;

conoscenza distribuita;

review informale ma continua.

Test-driven development e continuous integration

Si scrivono prima i test del codice per chiarire i requisiti di ciò che si deve implementare.

I test servono per il regression testing dopo l'introduzione di nuove funzionalità.

Continuous integration significa integrare giornalmente il lavoro degli sviluppatori in una baseline condivisa (ref. configuration management).

L'integrazione comporta l'esecuzione dei test di integrazione (svolti anche da squadre esterne).

Attraverso small releases controllate si punta ad avere un sistema sempre coerente.

Refactoring

Il principio "design for change" non è ritenuto essenziale in quanto è difficile e rischioso cercare di anticipare i cambiamenti futuri.

Tuttavia, dopo una serie di modifiche un componente software può risultare *disordinato*, quindi occorre un lavoro di *risistemazione* per renderlo più leggibile e strutturato e agevolare estensioni future.

Software process

La qualità del prodotto dipende dalla qualità del processo di sviluppo.

Il software process comprende metodi, best practices, tecniche, attività e strumenti per lo sviluppo di un prodotto software in tutti i suoi componenti.

Col tempo il processo software di un'organizzazione si definisce sempre meglio e la sua implementazione si diffonde.

Il CMM (Software Engineering Institute, CMU) organizza lo sviluppo evolutivo in 5 step verso livelli di maturità crescenti.

Il CMMI fornisce una collezione di best practices.

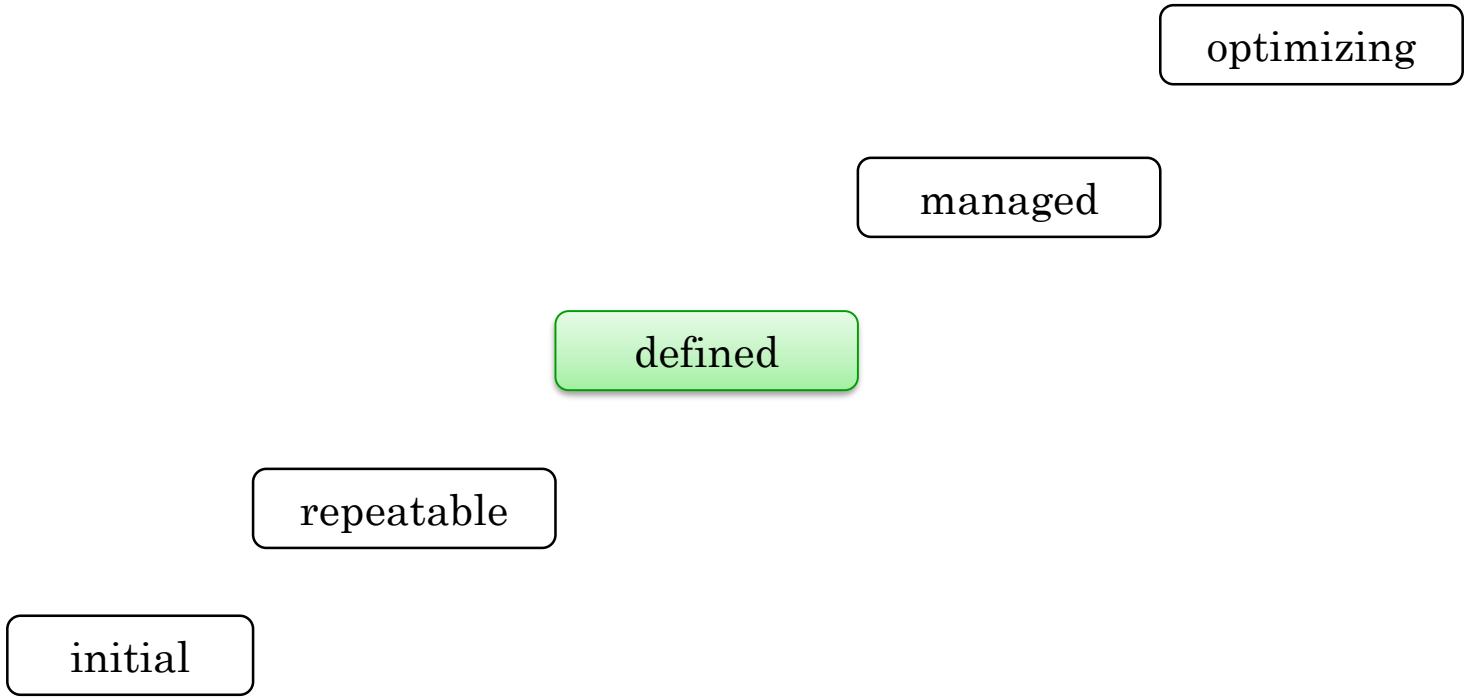
CMM (Capability Maturity Model)

Continuous process improvement is based on many small, evolutionary steps rather than revolutionary innovations.

The CMM provides a framework for organizing these evolutionary steps into *five maturity levels* that lay successive foundations for continuous process improvement.

These five maturity levels define an ordinal scale for measuring the maturity of an organization's software process and for evaluating its software process capability. The levels also help an organization prioritize its improvement efforts.

CMM



Levels

1) **Initial**. The software process is characterized as *ad hoc*, and occasionally even chaotic. Few processes are defined, and success depends on individual effort.

2) **Repeatable**. Basic project management processes are established to track cost, schedule, and functionality. The necessary *process discipline* is in place to repeat earlier successes on projects with similar applications.

3) **Defined**. The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's *standard software process* for developing and maintaining software.

4) **Managed**. Detailed measures of the software process and product quality are collected. Both the software process and products are *quantitatively* understood and controlled.

5) **Optimizing**. Continuous process improvement is enabled by quantitative *feedback* from the process and from piloting innovative ideas and technologies.

CMMI

www.cmmiinstitute.com

CMMI (Capability Maturity Model Integration) is a process improvement approach that provides organizations with the essential elements of effective processes, which will improve their performance. CMMI-based process improvement includes identifying your organization's process strengths and weaknesses and making process changes to turn weaknesses into strengths.

CMMI models are collections of best practices that help organizations to dramatically improve effectiveness, efficiency, and quality.

CMMI® for Development, Version 1.3

CMMI-DEV, V1.3 2010

Process areas (22)

Each process area includes a set of goals and practices.

2

Configuration Management

Project Planning

Project Monitoring and Control

Requirements Management

Process and Product Quality Assurance

…

3

Organizational Process Definition

Risk Management

Technical Solution

Verification

Validation

…

Software quality

ISO/IEC 9126 sostituito da ISO/IEC 25010 (del 2011)

Propone un product quality model adatto a software e computer systems e basato su 8 caratteristiche suddivise in sotto-caratteristiche

1. functional suitability
2. reliability
3. performance efficiency
4. usability
5. security
6. compatibility
7. maintainability
8. portability

Propone inoltre un quality in use model relativo al giudizio da parte degli utenti.

ISO = International Organization for Standardization

IEC =International Electrotechnical Commission

Functional suitability

1. completeness
2. correctness
3. appropriateness

In che misura il prodotto fornisce le funzionalità richieste?

Tutte, in maniera corretta e appropriata?

Performance efficiency

1. time behavior
2. resource utilization
3. capacity

In che misura sono soddisfatti i requisiti relativi ai tempi di risposta, all'utilizzo delle risorse, ai limiti massimi stabiliti?

Compatibility

1. co-existence
2. interoperability

condivisione di risorse e scambio di informazioni tra prodotti o elementi

Security

1. confidentiality
2. integrity
3. non-repudiation
4. accountability
5. authenticity

i dati sono accessibili soltanto alle persone autorizzate

il sistema impedisce accessi e modifiche non autorizzati, tiene traccia delle azioni effettuate, associa le azioni ai soggetti che le eseguono, verifica l'autenticità dei soggetti che vogliono interagire con il sistema

Usability

1. appropriateness recognizability
2. learnability
3. operability
4. user error protection
5. user interface aesthetics
6. accessibility

1. facilità di riconoscere l'appropriatezza da una prima impressione o dalla documentazione
2. facilità di imparare ad usare il sistema

Reliability

1. maturity
2. availability
3. fault tolerance
4. recoverability

In che misura il prodotto è affidabile durante l'uso (1), è disponibile (2), funziona anche in presenza di difetti hardware e software (3), è in grado di riprendersi dopo un guasto (4).

Maintainability

1. modularity
2. reusability
3. analysability
4. modifiability
5. testability

impatto che la modifica di un componente ha sugli altri componenti,
riutilizzo di componenti in altri prodotti,
capacità di determinare l'impatto di una modifica,
capacità di subire modifiche senza peggiorare la qualità,
facilità di stabilire criteri di test e di verificarli con l'esecuzione dei test.

Portability

1. adaptability
2. installability
3. replaceability

facilità di adattamento ad ambienti diversi

Quality in use model

1. Effectiveness
2. Efficiency
3. Satisfaction
4. Freedom from risk
5. Context coverage

Punto di vista dell'utente sul sistema.

Dependability

Caratteristica composta da

1. availability
2. reliability
3. safety
4. security

safety: non ci sono rischi per l'utente

security: il sistema impedisce gli accessi non autorizzati

Misure

Si possono effettuare su caratteristiche esterne (misure black box) o interne (misure white box).

Esempi di misure esterne

Availability: % di tempo in cui il sistema funziona (può essere causa di penali se non mantenuta)

Reliability: n. di guasti nell'unità di tempo (il n. dovrebbe diminuire nel tempo)

Metriche interne

Danno indicazioni sulla complessità del software ma è difficile stabilire relazioni precise con la qualità del prodotto.

Esempi di metriche interne

N. di linee di codice di un programma (LOC)

Metriche di Halstead

n1 = numero di operatori (e keyword) distinti; N1 = n. totale

n2 = n. di operandi distinti; N2 = n. totale

n = n1 + n2; vocabolario del programma

N = N1 + N2; lunghezza del programma

Complessità ciclomatica (McCabe)

Metriche interne con un linguaggio object-oriented

N. medio (min, max) di linee per classe, di metodi per classe, di linee per metodo.

Fan-in/fan-out per metodo (n. metodi chiamanti/ n. metodi chiamati)

Profondità dell'albero di inheritance

N. di sottoclassi immediate per ciascuna classe

Dipendenze tra classi (es. tra C1 e C2): indica quanti metodi di C2 sono chiamati in C1.

Modeling

Conceptual models

Using models to study the properties of complex systems is common to all disciplines. Examples of models are scaled physical systems and mathematical representations.

In general, *a model is an abstract and rigorous representation* of a system: it enables the user to reason about the important properties of the system.

Unique to software development is the notion of *operational models*, i.e., models that can executed in a suitable support environment or that can automatically be turned into operational software.

model = code

model → code

model-driven development

UML

UML (Unified Modeling Language)

is made up of a number of modeling languages/notations; it is not a global methodology.

OMG standard: http://www.omg.org

Major kinds of diagrams

- Structural: class models, object models
- Behavioral: use cases, activity models, state models, sequence models

OCL (Object Constraint Language)

Class models

The system under consideration is made up of individuals called *objects*, and objects belong to *classes*. There are systematic links between objects and they are represented by links, called *relationships*, between their classes.

Classes may have properties called *attributes*.

A *class model* shows the classes, attributes and relationships.

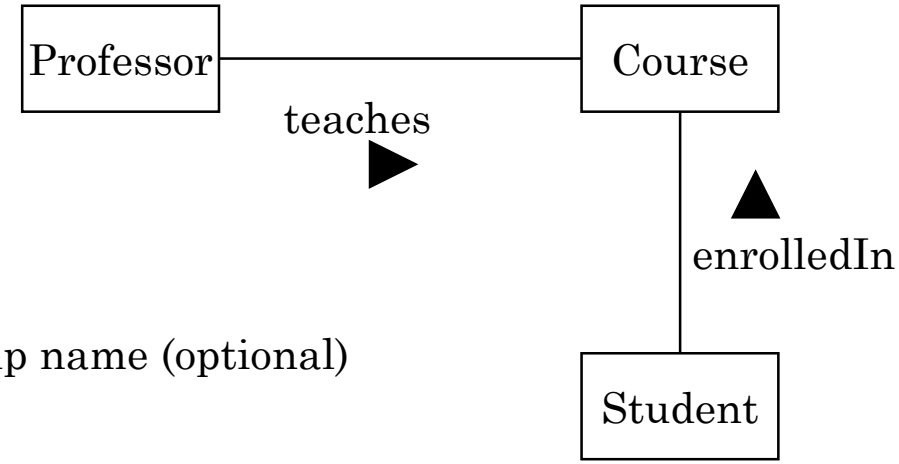
An *object model* is a particular realization of a class model: it shows a number of interrelated objects for illustrative purposes.

Several flavors:

- *Class model*: technical (implementation) perspective
- *Domain model*: high level (conceptual, business) perspective
- *Information model*: data base perspective

Example of domain model

In universities, there are professors and students; professors teach courses and students are enrolled in courses.



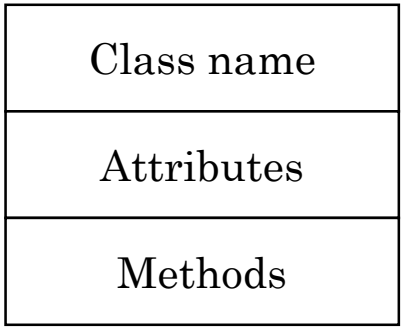
teaches: relationship name (optional)

There are links between professors and courses and between students and courses.

Two binary **associative relationships**; only *binary* relationships are considered.

Classi

Il simbolo di una classe può contenere attributi e metodi.

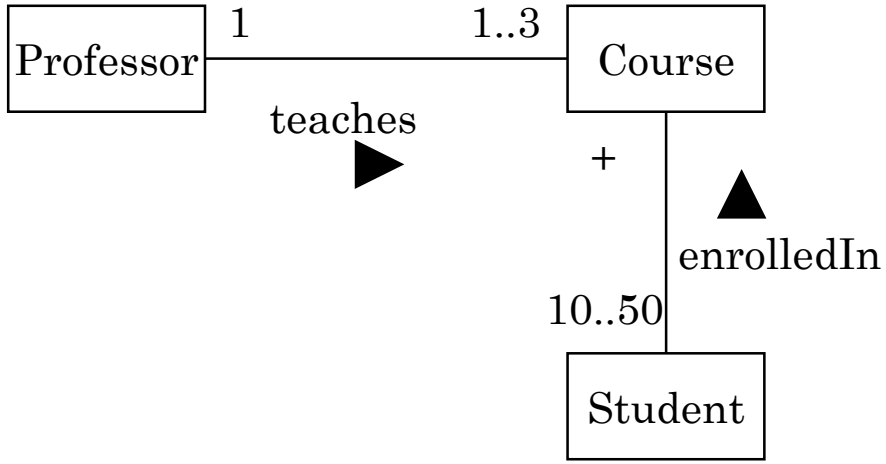


Vista completa

Constraints

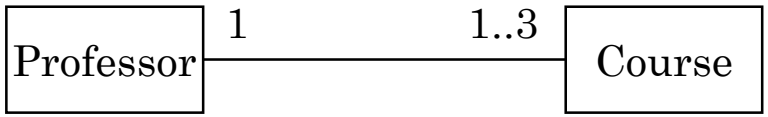
A course is taught by one professor. Professors teach 1..3 courses.

Students are enrolled in a number of courses. The minimum number of enrollees in a course is 10; the maximum is 50.



Multiplicity

How many associations a given object may be involved in?



Indicator	**Meaning**
0..1 or 0,1	Zero or one
*	Zero or more
"n"	One or more (not known a priori)
n	*n* (where *n* > 1)
0..n	Zero to *n* (where *n* > 1)
m..n	m to *n* (where *n* > m)

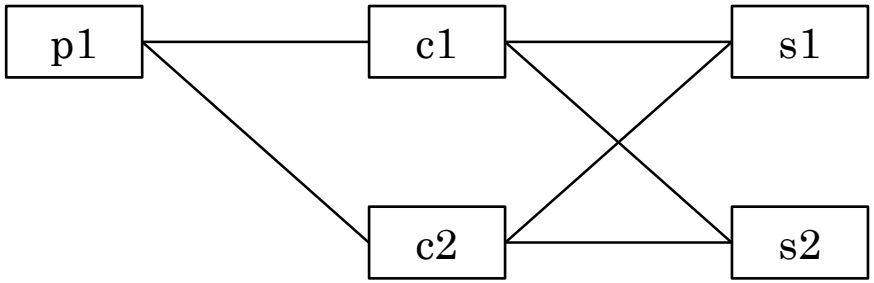
m, n denote integer values

"n" is the character n

La molteplicità definisce implicitamente regole di validazione.

Object model

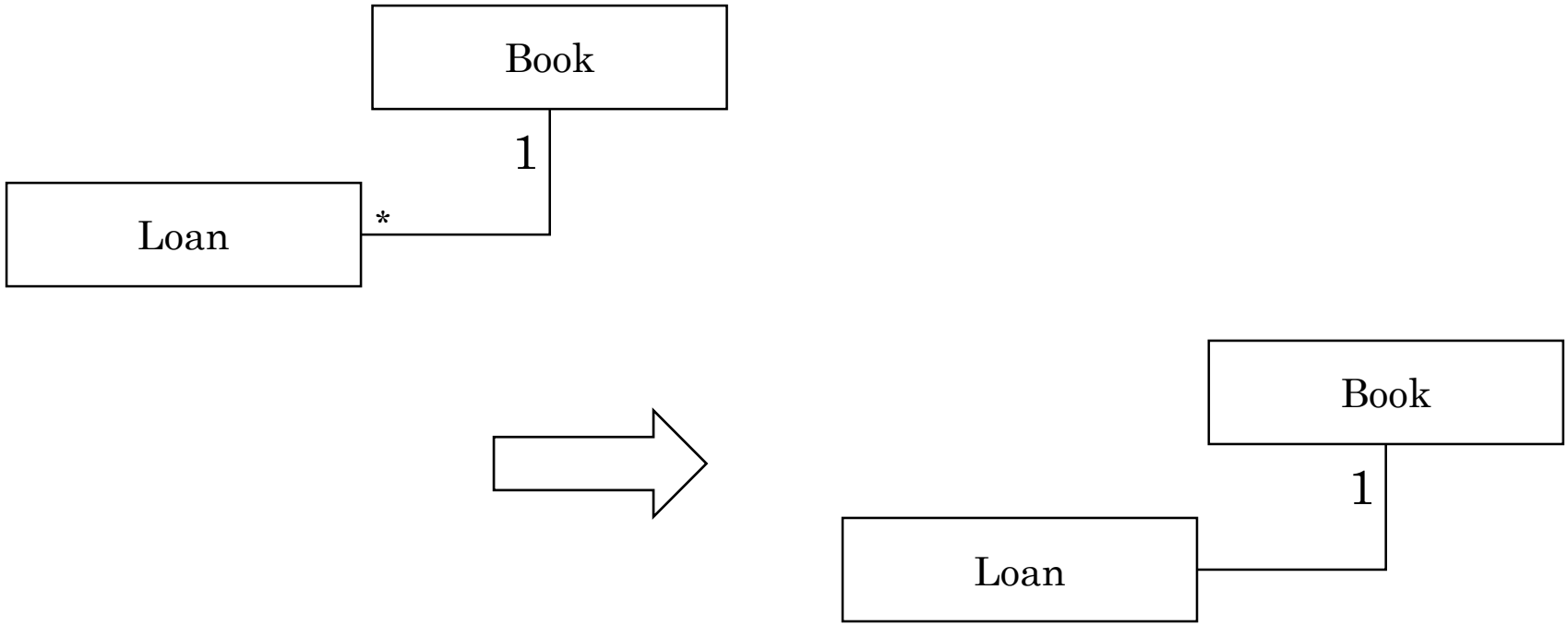
Professor Course Student



Definite che cos'è un object model.

Il modello è valido?

Default multiplicity (*)

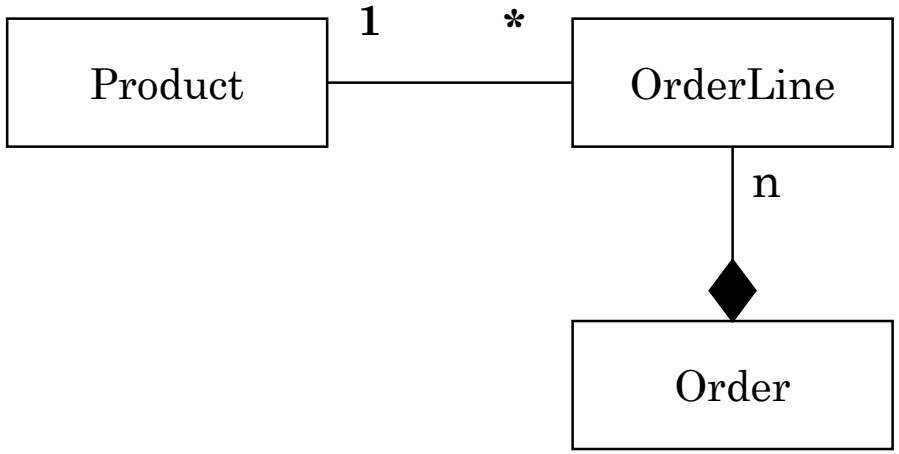


Composition

Objects can be made up of other objects.

When a compound object is deleted, all its components are automatically deleted.

A component exists only in connection with a container.



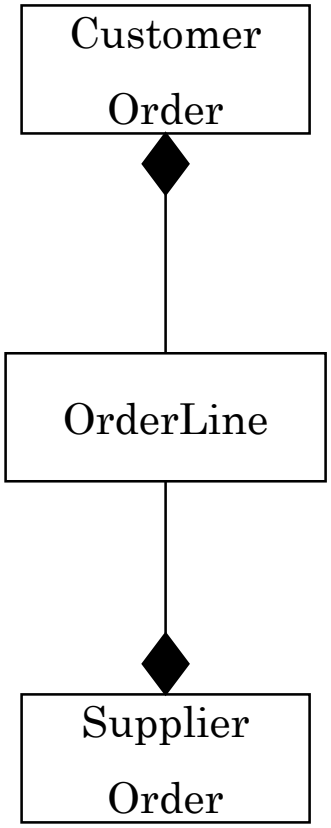
An order is made up of order lines; each order line refers to a product and includes the number of units required.

Attributes:

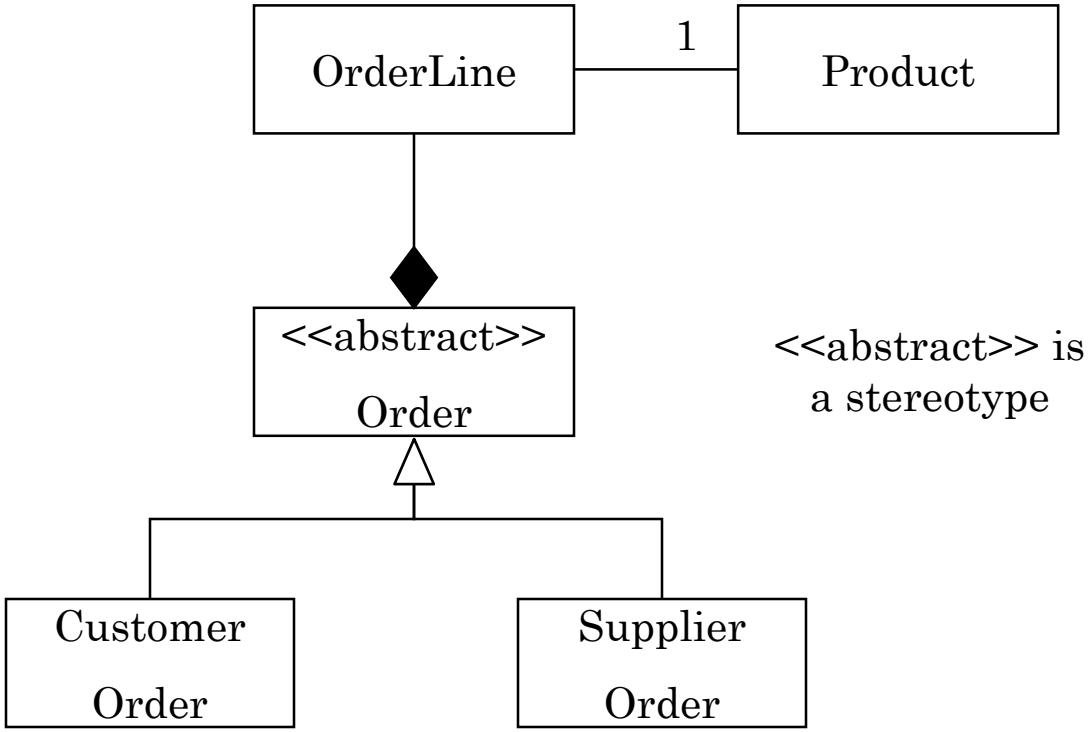
OrderLine: int n.

Composition

An orderLine may be part of a customerOrder or of a supplierOrder (but not of both).



or

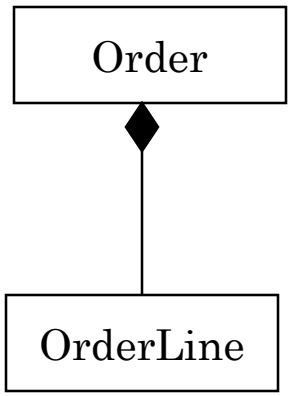


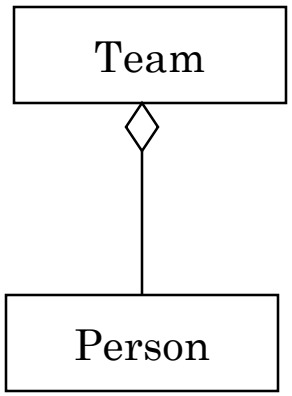
<<abstract>> is a stereotype

Aggregation

Composition is a strong link: a component belongs to only one container. The deletion of the container is cascaded to the parts.

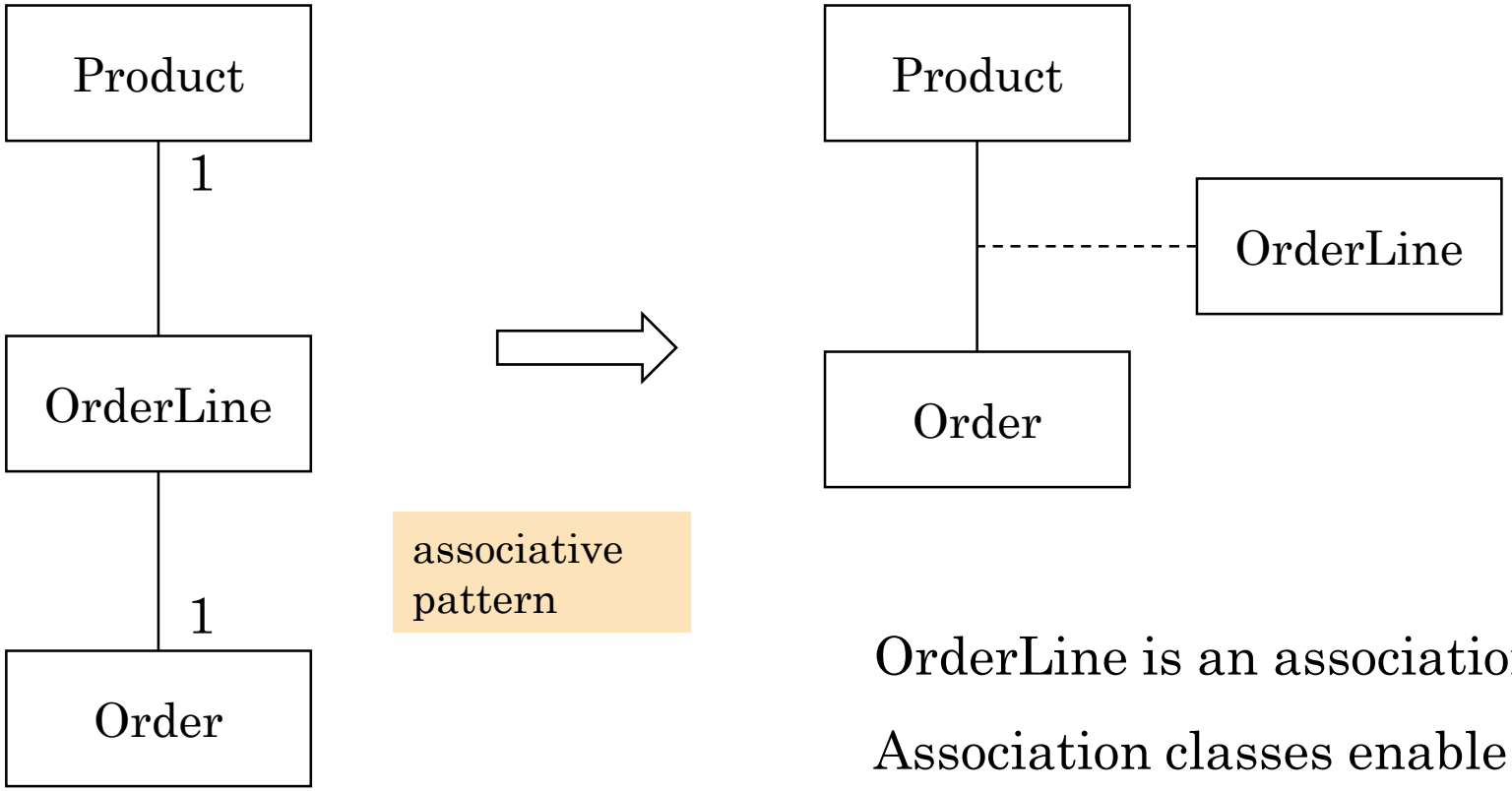
Aggregation denotes grouping and can be replaced by an associative relationship.





aggregation

Association class

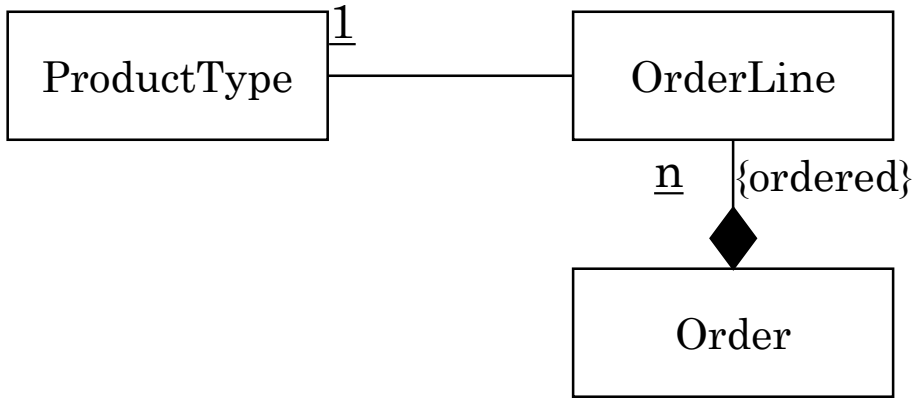


associative pattern

OrderLine is an association class.

Association classes enable attributes to be associated with relationships.

Ordered associations

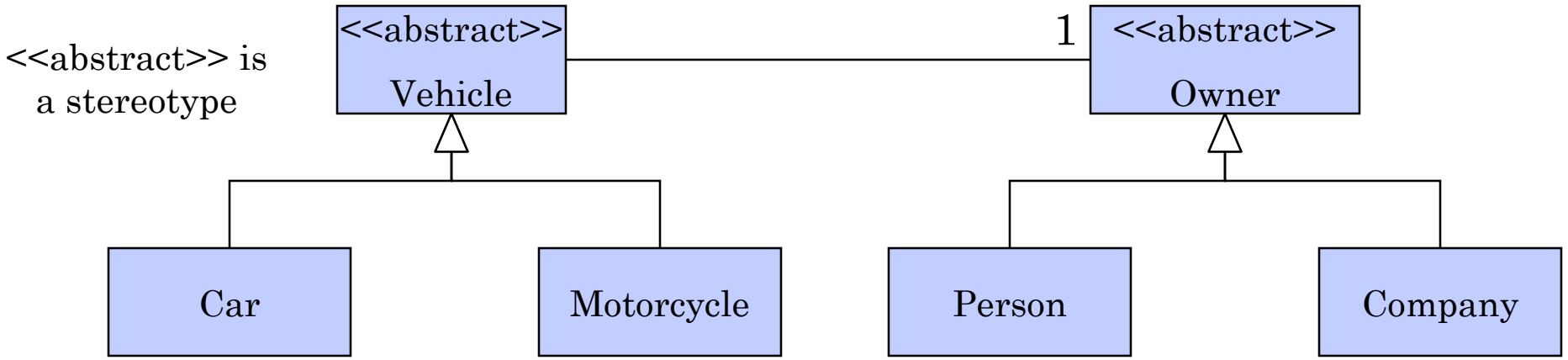


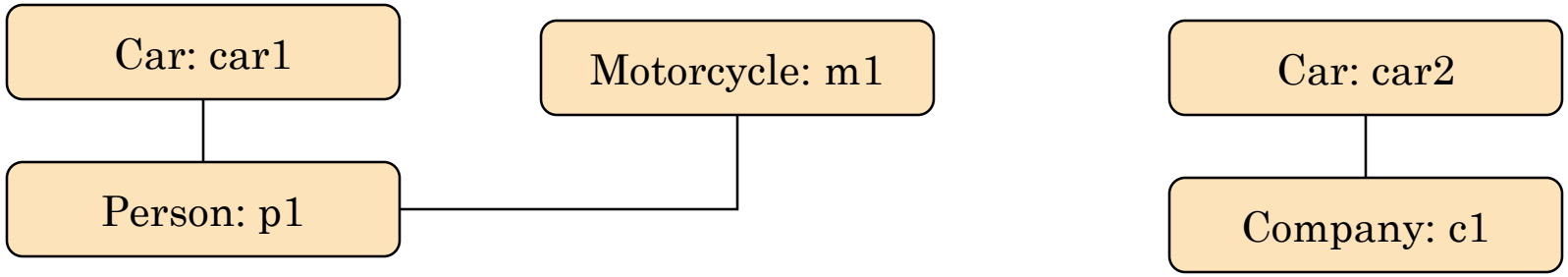
Lines are ordered.

Inheritance

A vehicle has exactly one owner that may be a person or a company. A vehicle may be a car or a motorcycle. Vehicle and Owner are abstract classes (they cannot be instantiated).

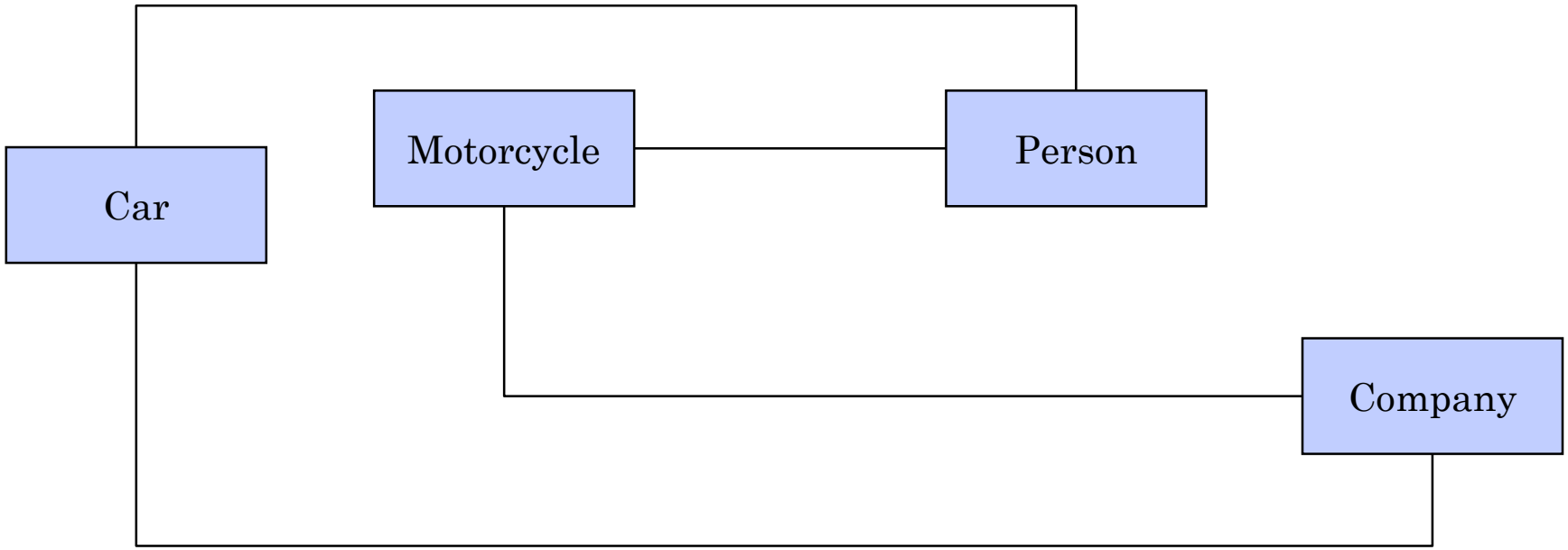
class model





object model

Without inheritance

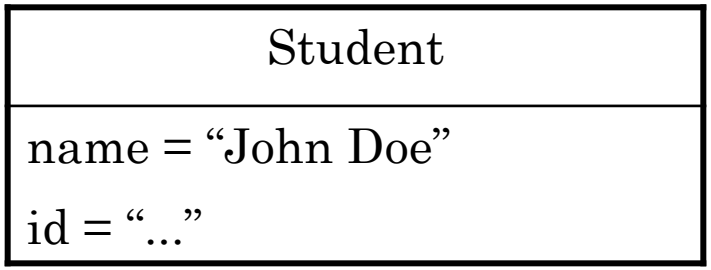


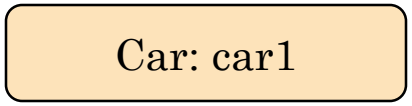
What are the problems?

Instances

Classes represent individuals, also called instances or objects.

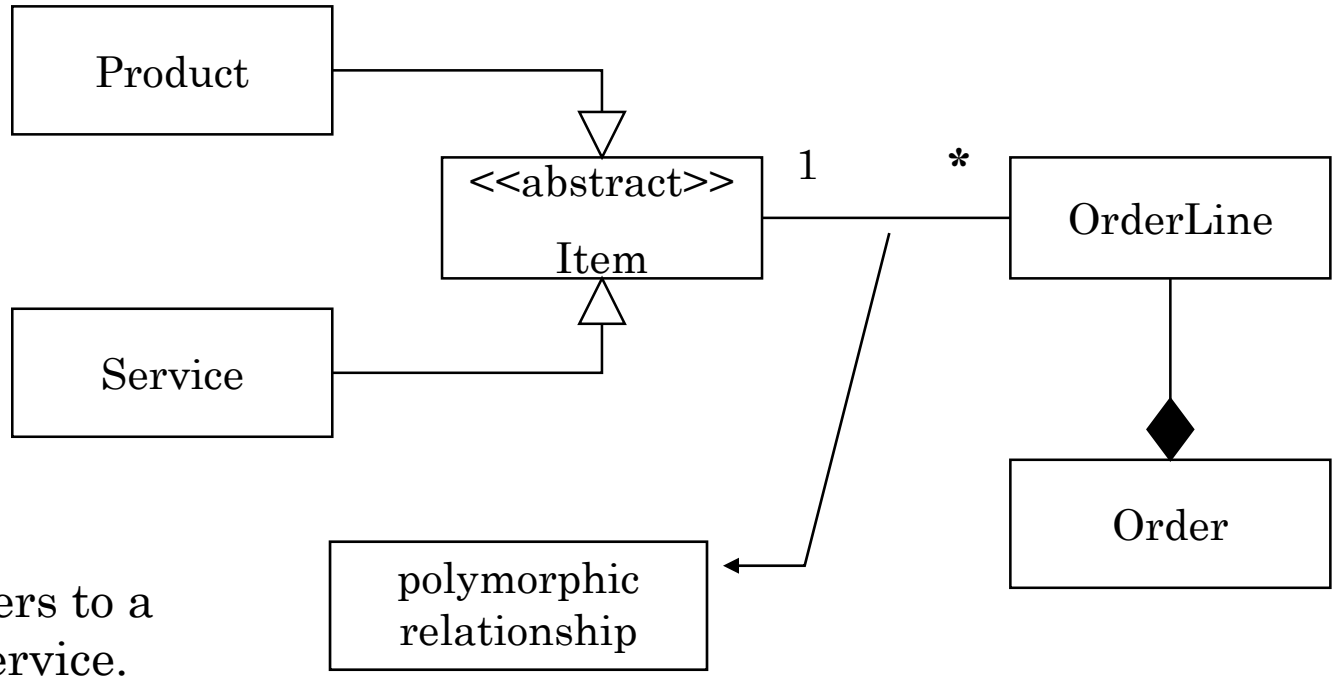
An object model shows a number of objects along with their attributes and associations.





A simplified representation: car1 may be the value of an attribute or simply a label.

Inheritance



An order line refers to a product or to a service.

Product and Service inherit from Item; Item is an abstract class and hence it cannot be instantiated.

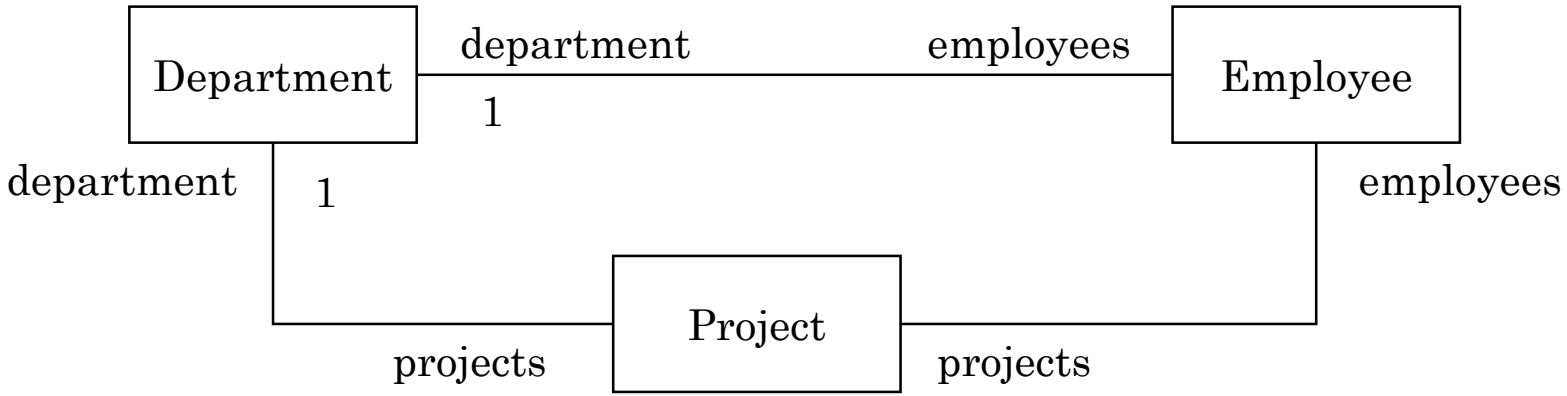
Associative attributes

A company is organized into departments. Each of its employees works in a department and may take part in a number of projects. Each project is managed by one department.

Each entity (department, employee and project) has a name; an employee also has a position.

Associative attributes

Associative (or role) attributes are references to other objects and are used in navigational constructs.



Attributes:

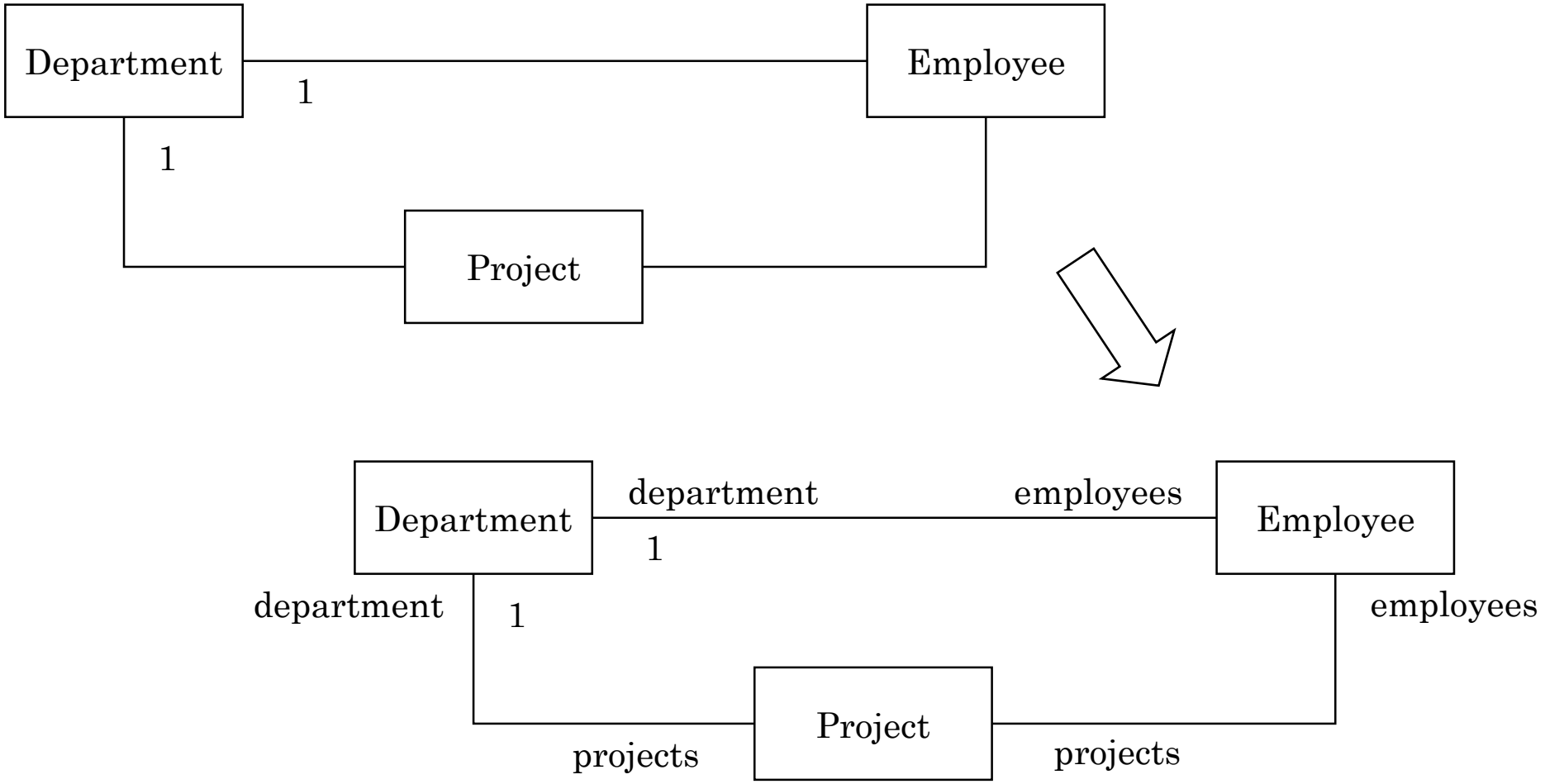
Department: String name.

Project: String name.

Employee: String name, String position.

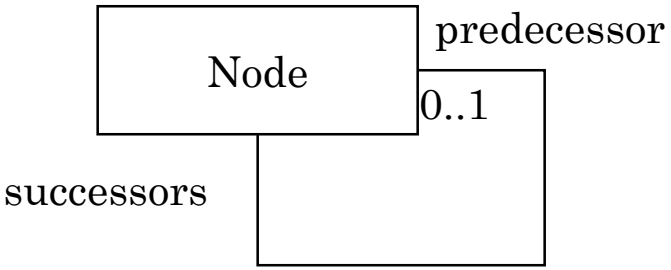
Example: if p is a reference to a project, p.employees denotes the collection of the employees participating in the project.

Associative attributes (default names)

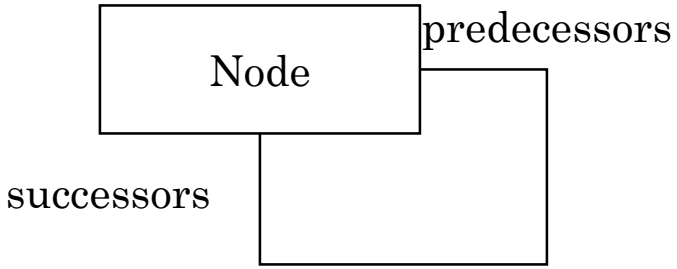


When the associative attribute denotes a collection of partner entities, the plural of their class name is used, e.g. employees; otherwise the class name with the initial in lower case is used, e.g. department.

Recursive relationships



tree



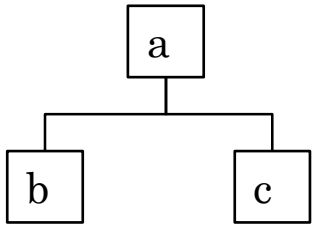
graph

Remark: predecessor, predecessors and successors are associative attributes.

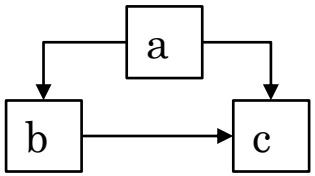
In the tree model, given a node n, *n.predecessor* returns the predecessor node (which may be null) of n and *n.successors* returns the collection (which may be empty) of the successor nodes of n.

Object models

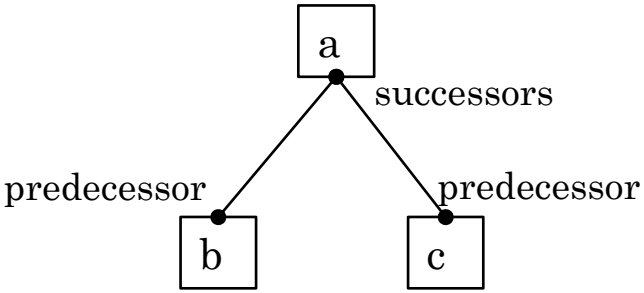
tree diagram



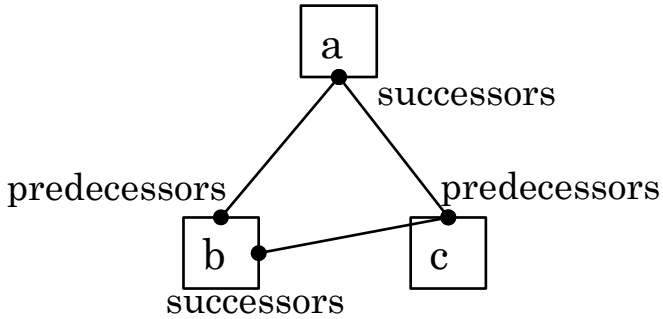
graph diagram



object model



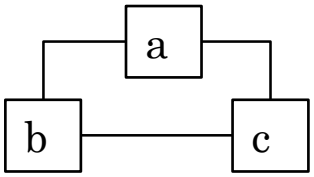
object model

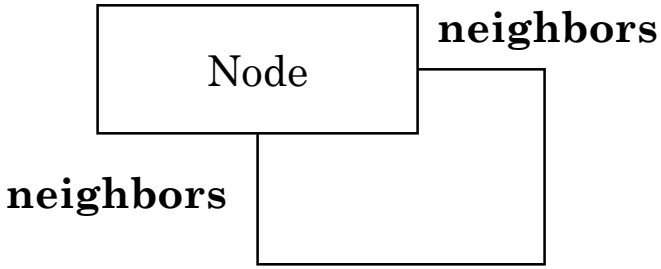


Symmetrical recursive relationships

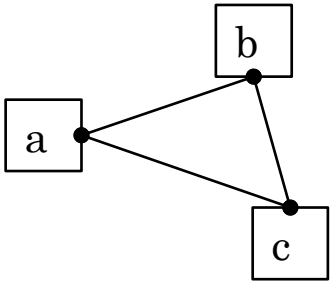
The associative attribute is the same for both ends.

undirected graph





object model



- neighbors

Interpretations of class models

1. Programming-oriented perspective

The class model represents an object-oriented program (e.g. a java program): the model classes are mapped to java classes, the attributes of the model classes are mapped to the attributes of the java classes, inheritance relationships are mapped to “extend” relationships between java classes, and the other relationships are mapped to single references or collections in the java classes.

Interpretations of class models

2. Information-oriented perspective

According to the approach called *object-relational*, the class model represents persistent information that is stored in a relational database and accessed through java objects. Therefore two major outputs can be obtained from the class model: the definition of the relational tables and the collection of java classes whose objects are the representatives of the database records.

These objects are called *entities* and their classes are called *entity classes;* the class model (from this perspective) is called *entity model*.

A well known framework is **Hibernate**: it maps java classes to database tables, java data types to SQL data types, and provides an object-oriented query language.

Esempi di analisi dei requisiti

Home banking

Requisiti per bonifici

Un utente può ordinare un bonifico scegliendo il conto corrente di appoggio e indicando il beneficiario, il codice IBAN, l'importo (<= 5000,00 euro) e la causale. Se il saldo del conto è >= importo, il sistema genera un bonifico pendente (togliendo l'importo dal saldo).

Un utente può revocare un bonifico pendente e il sistema aggiunge l'importo al saldo del conto.

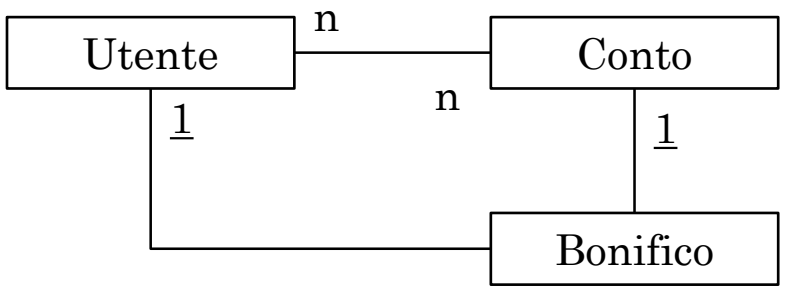
Alle 18.00 il sistema esegue tutti i bonifici pendenti.

Note

Un conto corrente ha uno o più intestatari, un agenzia di riferimento, un iban, un saldo >= 0. L'IBAN è un codice di 27 caratteri che identifica il conto, l'agenzia e la banca.

L'intestatario di un conto è un utente della banca; un utente ha vari attributi.

Modello informativo



Attributi (UML esteso):

Conto: String iban, float saldo (>= 0).

Bonifico: String beneficiario, String iban, float importo (> 0 and <= 5000,00), String causale, stato (pendente, revocato, eseguito).

Note

Gli attributi sottolineati sono obbligatori.

(>= 0) e (> 0 and <= 5000,00) sono invarianti locali.

I bonifici hanno uno stato che può assumere 3 valori, pendente, revocato, eseguito. Quello iniziale è sottolineato.

Online job applications

ThisCompany intends to develop a web site enabling applicants to apply online for the available positions. Applicants (name, email) are supposed to have already registered with ThisCompany.

An available position (name, description) is associated with a number of skills (name, description).

The applicant fills in the online application form: they select one of the available positions and give scores (1 to 10) to their ability levels for the skills required. The score is registered in attribute aScore of the ability.

Online job applications

ThisCompany intends to develop a web site enabling applicants to apply online for the available positions. Applicants (name, email) are supposed to have already registered with ThisCompany.

An available position (name, description) is associated with a number of skills (name, description).

The applicant fills in the online application form: they select one of the available positions and give scores (1 to 10) to their ability levels for the skills required. The score is registered in attribute aScore of the ability.

Analysis

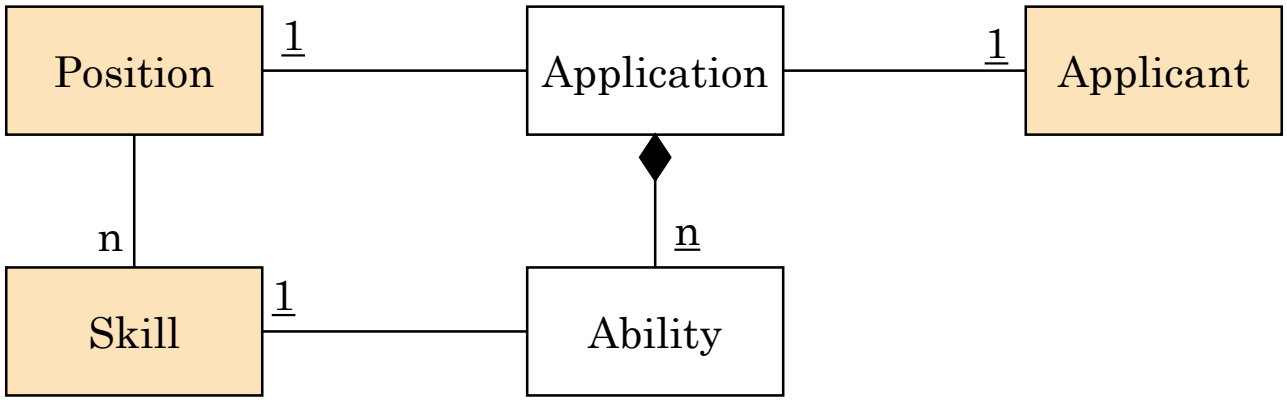
ThisCompany intends to develop a web site enabling applicants to apply online for the available positions. *Applicants* (name, email) are supposed to have already registered with ThisCompany.

An available position (name, description) *is associated with* a number of skills (name, description).

The *applicant fills in the online application form*: *they select one of the available positions* and give scores (1 to 10) to *their ability levels for the skills required*. The score is registered in attribute aScore of the ability.

Information model?

Identify classes, attributes and relationships.



Attributes:

Applicant: String name, String email. Position: String name, String description.

Skill: String name, String description. Ability: int aScore (1..10). // (1..10) is a local inv.