

OOAD, UML, Method and Code overview

Kang Taewook. Ph.D

laputa9999@gmail.com

<https://www.linkedin.com/in/tae-wook-kang-64a83917>

The BIM principle and philosophy

BIM principle diagram:

```
graph TD; BIMObject[BIM Object] --- Space[Space]; BIMObject --- Story[Story]; BIMObject --- Building[Building]; BIMObject --- CivilObject[Civil Object]; BIMObject --- Site[Site]; BIMObject --- Highway[Highway]; BIMObject --- Railway[Railway]; BIMObject --- Curve[Curve]; BIMObject --- Xsection[X-section]; BIMObject --- DigitalTerrainModel[Digital terrain model]; BIMObject --- Property[Property: Geometry, Topology, Parametric, Object, Knowledge]; BIMObject --- Slab[Slab]; BIMObject --- Column[Column]; BIMObject --- Window[Window]; BIMObject --- Beams[Beams];
```

BIM digest

By noreply@blogger.com (Taewook Kang)

To listen to an audio podcast, mouse over the title and click Play.

Home

- Introduce
- Member
- BIM architecture
- Subject
- Links and reference
- Resources
- Upcoming
- Contact Us

Affiliations

UNIVERSITY OF MICHIGAN LIBRARIES

need to use a ground station to view the status info to listen to Pixhawk's status tones.

Status LED

Initializing, hold the XB still and level

Acquiring GPS, please wait

Autopilot ready, GPS locked

Account

B'lynk.begin(auth);
}
void loop()
{
B'lynk.run();
}

'd Socket

FORCE FEEDBACK
Most manufacturing robots are too dangerous to work without a Baxter robot nearby.

Reset button

DBE7DF 000076

BIM digest

By noreply@blogger.com (Taewook Kang)

To listen to an audio podcast, mouse over the title and click Play.



[View in iTunes](#)

Free

Category: Educational Technology

Description

안녕하세요. Engineering을 쉽게 풀어서 듣는다가 podcast에 대해 이야기를 나눌 페드Engineering digest는 현재 공학에서 화학, Software, Graphics 그리고 BIM입니다. 오는 방송을 편하고 유연하게 부담없이 듣기 바랍니다. 방송내용은 iPod에서 듣색하실 수 있습니다. – Tae Wook, Kang. 김

Name

- No.122 - BIM 최신 솔루션 ...
- No.121 - 건설분야 IoT 기...
...
3 No. 120 - 최근 BIM 소프트...



Contents

History
OOAD

UML with Design Pattern
Method
Code in Practice

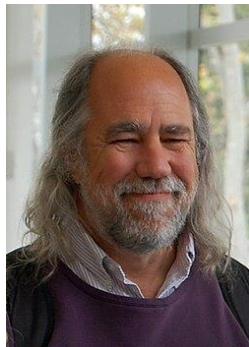
History

OOAD

“The Unified Modelling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software intensive system.

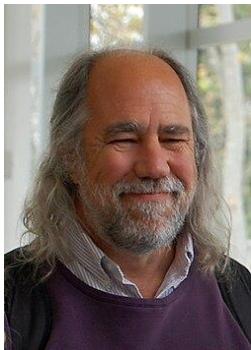
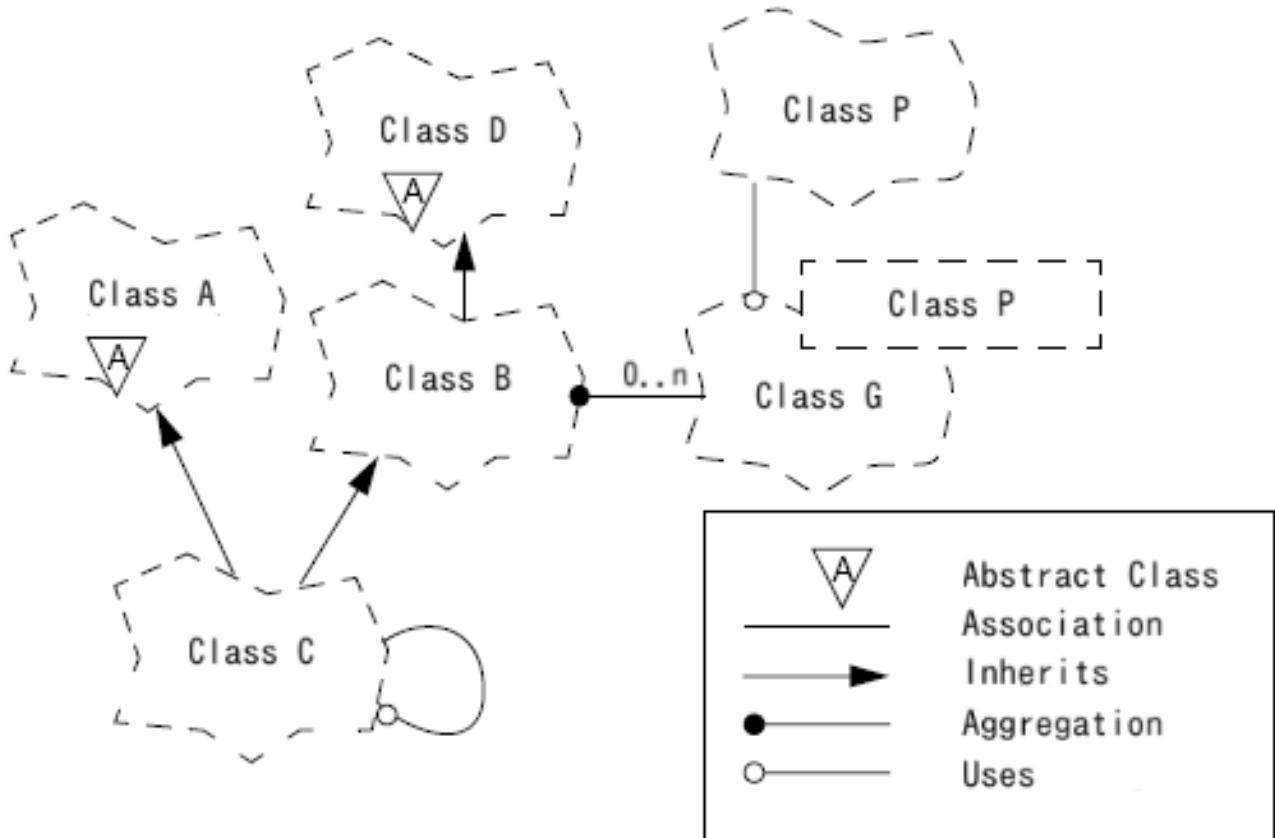
The UML offers a standard way to write a systems blueprints, including conceptual things like business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components.”

[Grady Booch](#), [James Rumbaugh](#), [Ivar Jacobsen](#), [Rational Software Corp](#)
with [Erich Gamma](#)



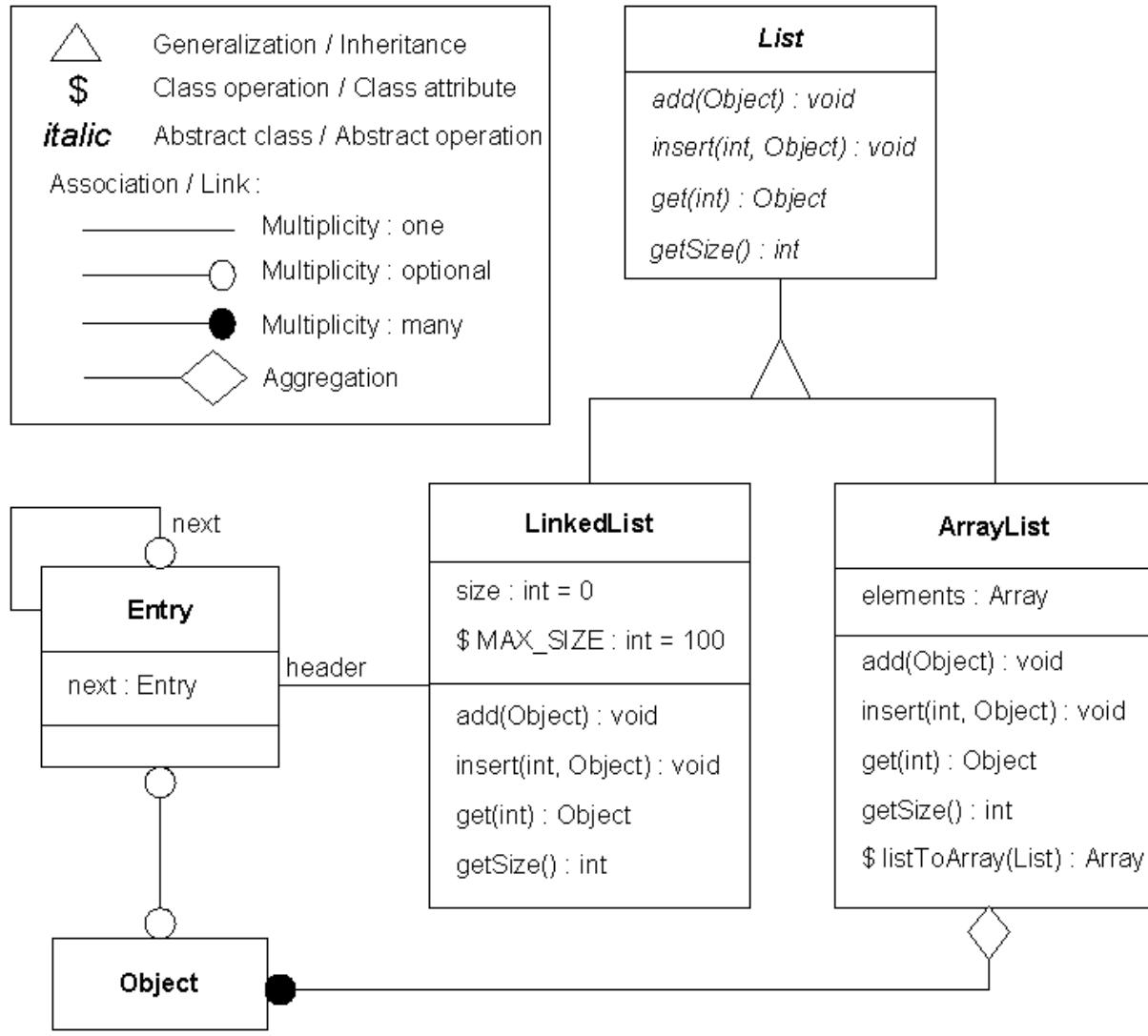
OOAD

Grady Booch: Booch Method Class Diagram



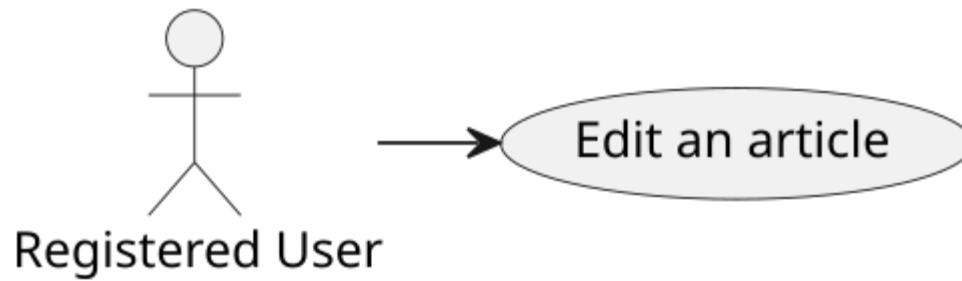
OOAD

[James Rumbaugh](#): OMT



OOAD

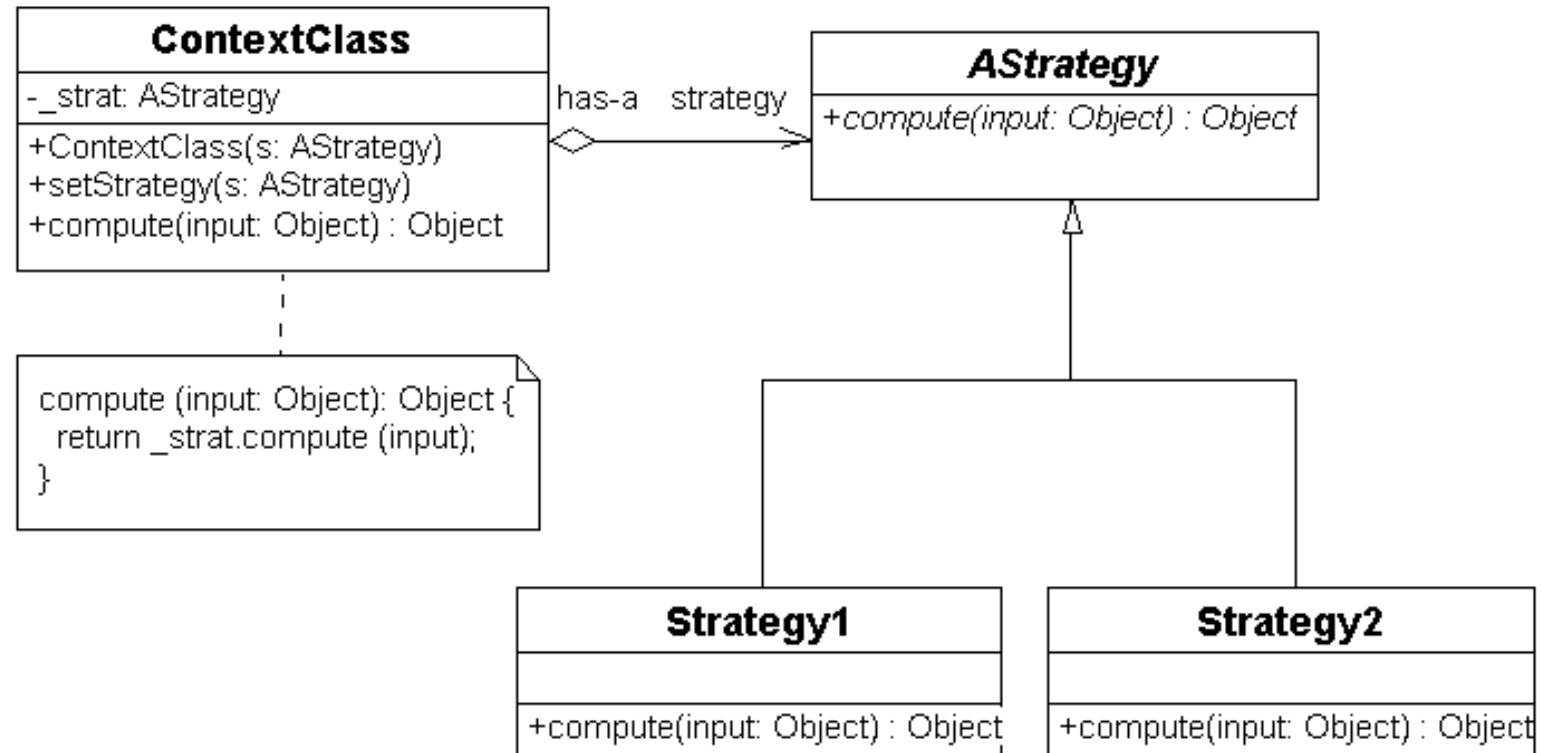
[Ivar Jacobson](#): Use case



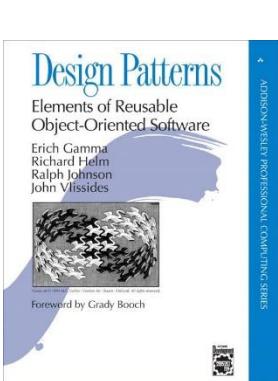
OOAD

[Erich Gamma:](#)

Design
Pattern

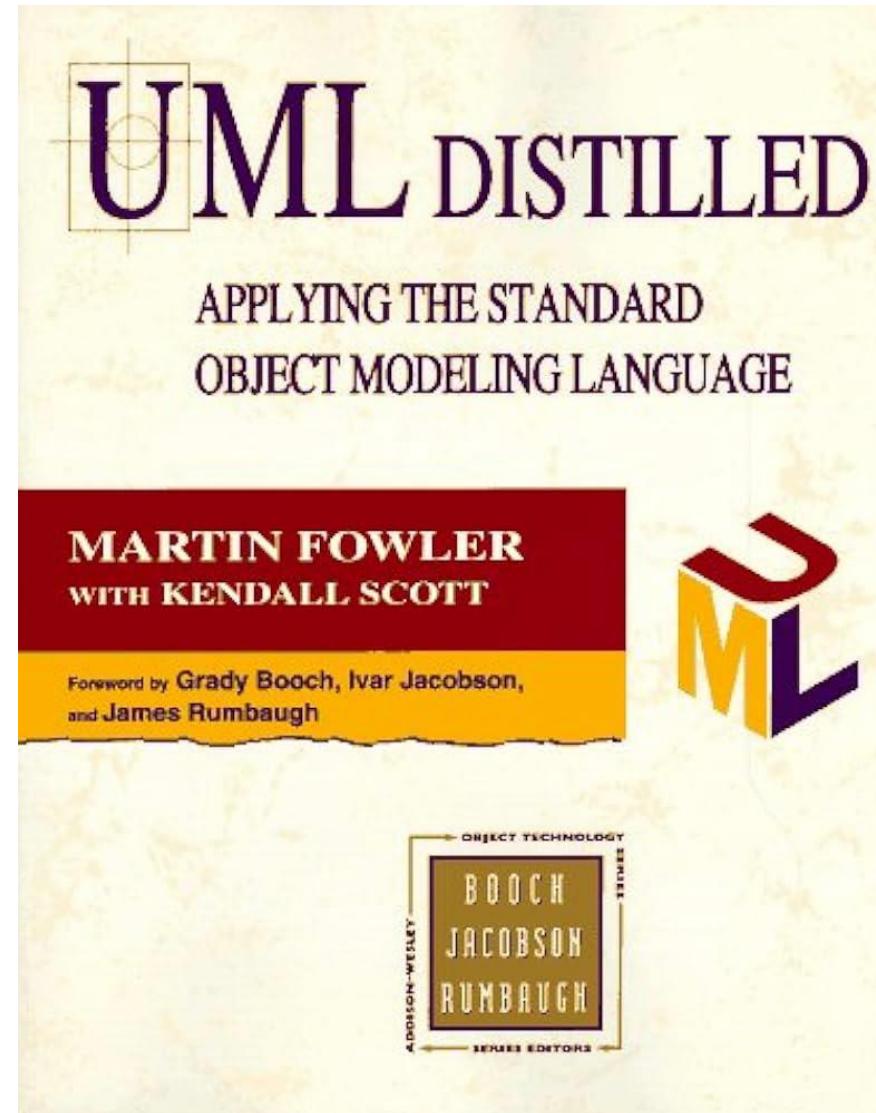


Ralph Johnson, Richard Helm, Erich Gamma,
and John Vlissides



OOAD

Object
Management
Group, 1989

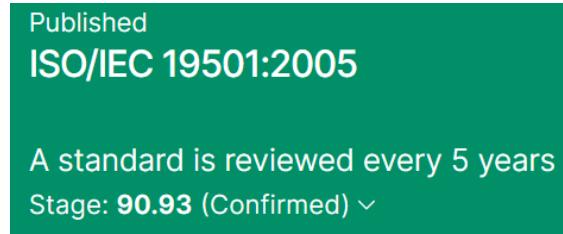
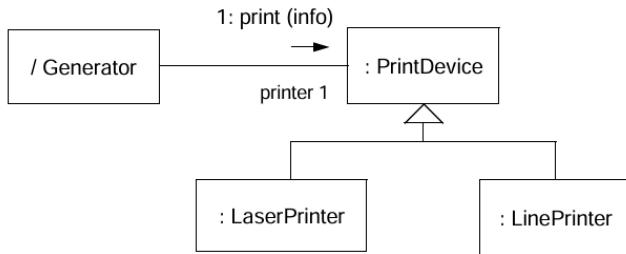


Martin Fowler



OOAD

ISO/IEC 19501(2005): UML 1.0



SUSTAINABLE GOALS

This standard contributes to the following **Sustainable Development Goal**

9

Table of Contents

ISO/IEC 19505-1:2012(E)
Date: April 2012

ISO/IEC 19505(2012): UML 2.0

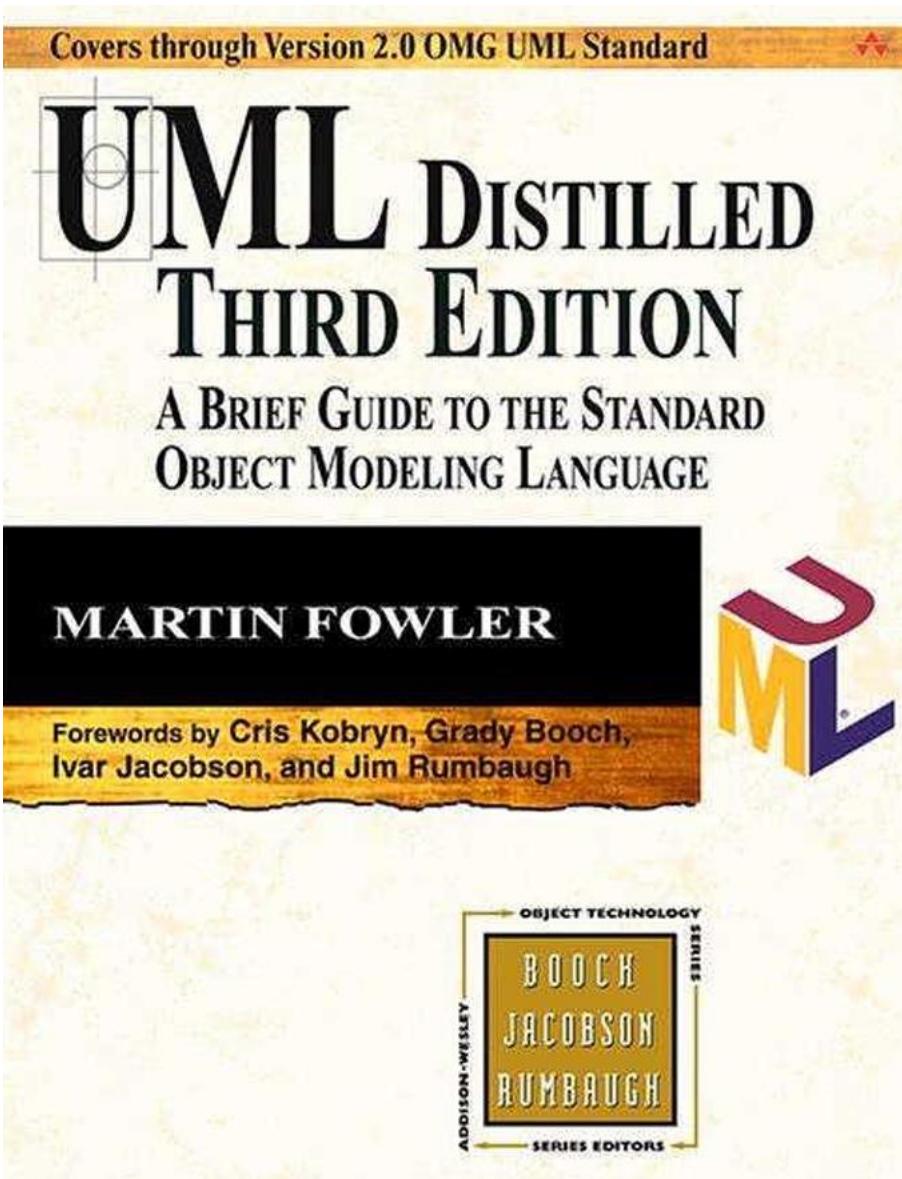
Information technology - Object Management Group
Unified Modeling Language (OMG UML),
Infrastructure

formal/2012-05-06

This version has been formally published by ISO as the 2012 edition standard: ISO/IEC 19505-1.

1. Scope	1
2. Conformance	1
2.1 General	1
2.2 Language Units	2
2.3 Compliance Levels	2
2.4 Meaning and Types of Compliance	3
2.5 Compliance Level Contents	5
3. Normative References	5
4. Terms and Definitions	6
5. Notational Conventions	6
6. Additional Information	6
6.1 Architectural Alignment and MDA Support	6
6.2 How to Proceed	6
6.2.1 Diagram format	7
7. Language Architecture	13
7.1 General	13
7.2 Design Principles	13
7.3 Infrastructure Architecture	13
7.4 Core	14
7.5 Profiles	16
7.6 Architectural Alignment between UML and MOF	16
7.7 Superstructure Architecture	17
7.8 Reusing Infrastructure	18
7.9 The Kernel Package	18
7.10 Metamodel Layering	18
7.11 The Four-layer Metamodel Hierarchy	19

UML



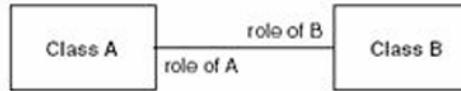
FOREWORD TO THE THIRD EDITION	4
FOREWORD TO THE FIRST EDITION	5
PREFACE	5
<i>Why Bother with the UML?</i>	6
<i>Structure of the Book</i>	7
<i>Changes for the Third Edition</i>	7
<i>Acknowledgments</i>	7
DIAGRAMS	10
CHAPTER 1. INTRODUCTION	14
<i>What Is the UML?</i>	14
<i>Where to Find Out More</i>	14
<i>Ways of Using the UML</i>	15
<i>How We Got to the UML</i>	18
<i>Notations and Meta-Models</i>	20
<i>UML Diagrams</i>	21
<i>What Is Legal UML?</i>	23
<i>The Meaning of UML</i>	24
<i>UML Is Not Enough</i>	24
<i>Where to Start with the UML</i>	25
CHAPTER 2. DEVELOPMENT PROCESS	26
<i>Iterative and Waterfall Processes</i>	26
<i>Predictive and Adaptive Planning</i>	28
<i>Agile Processes</i>	29
<i>Rational Unified Process</i>	30
<i>Fitting a Process to a Project</i>	30
<i>Fitting the UML into a Process</i>	32
<i>Choosing a Development Process</i>	35
<i>Where to Find Out More</i>	35
CHAPTER 3. CLASS DIAGRAMS: THE ESSENTIALS	35
<i>Properties</i>	36
<i>When to Use Class Diagrams</i>	38
<i>Where to Find Out More</i>	38
<i>Multiplicity</i>	38
<i>Programming Interpretation of Properties</i>	39
<i>Bidirectional Associations</i>	41
<i>Operations</i>	42
<i>Generalization</i>	43
<i>Notes and Comments</i>	44
<i>Dependency</i>	44
<i>Constraint Rules</i>	46
CHAPTER 4. SEQUENCE DIAGRAMS	47
<i>Creating and Deleting Participants</i>	47
<i>Loops, Conditionals, and the Like</i>	50
<i>Synchronous and Asynchronous Calls</i>	51
<i>When to Use Sequence Diagrams</i>	54
CHAPTER 5. CLASS DIAGRAMS: ADVANCED CONCEPTS	56
<i>Keywords</i>	56
<i>Classification and Generalization</i>	57

OOAD

Class



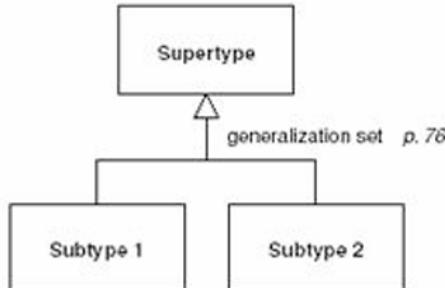
Association p. 37



Class Name

attribute:Type[0..1] = initialValue
operation(arg list) :return type
abstractOperation

Generalization p. 45



Constraint

{name: description} p. 49

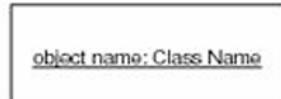
Keyword

«keyword» p. 65

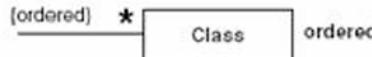
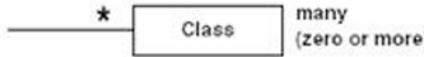
Note p. 46



Instance Specification p. 87



Multiplicities p. 38



Qualified Association p. 74



Navigability p. 42

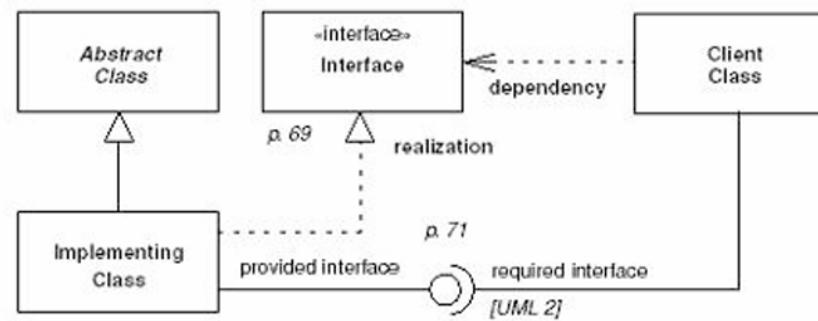


Dependency p. 47

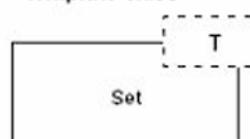


OOAD

Class Diagram



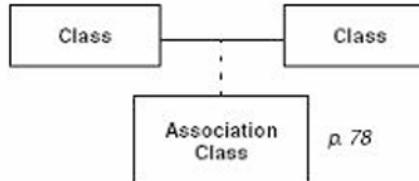
template class p. 81



bound element



Class Class

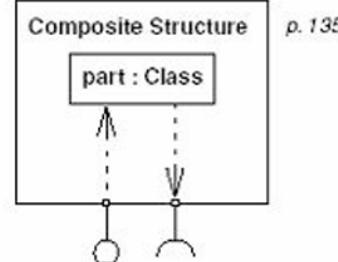


p. 78

p. 139

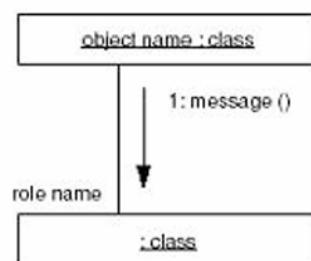


Composite Structure

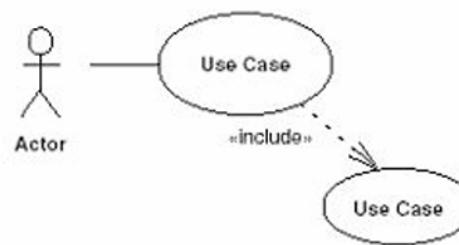


p. 135

Communication Diagram p. 131

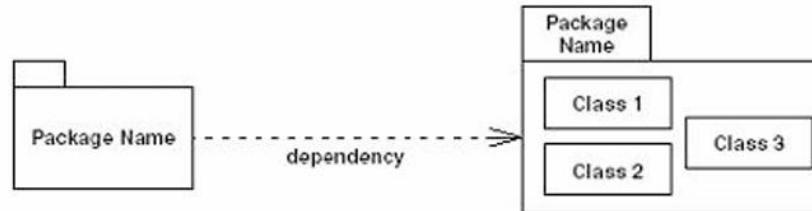


Use Case Diagram p. 99

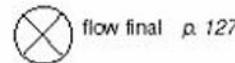
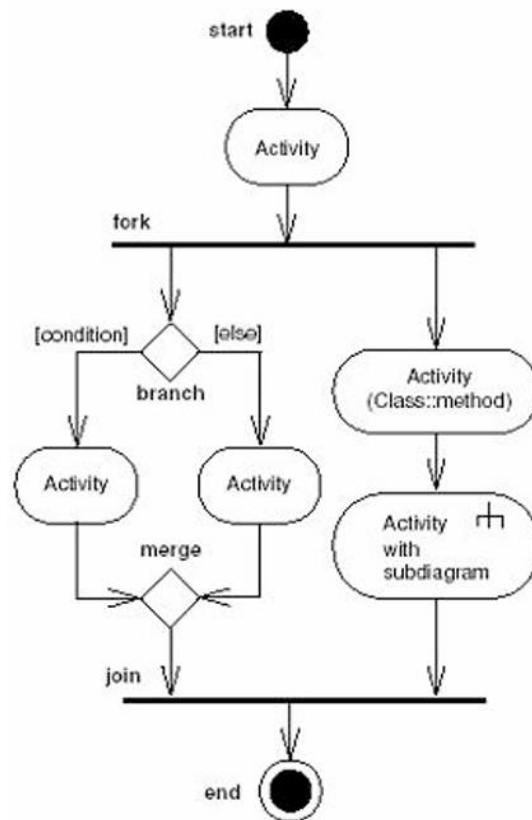


OOAD

Package Diagram p. 89



Activity Diagram p. 117



p. 127



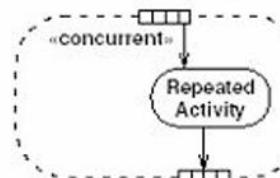
time signal



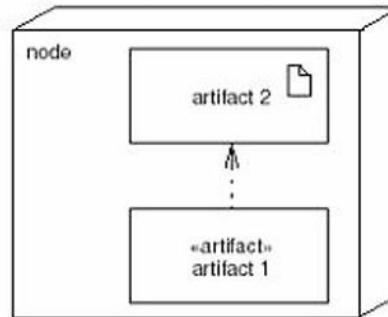
accept signal

p. 121

Expansion Region p. 127

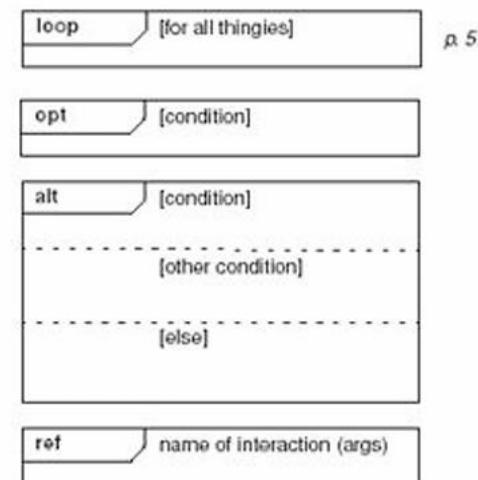
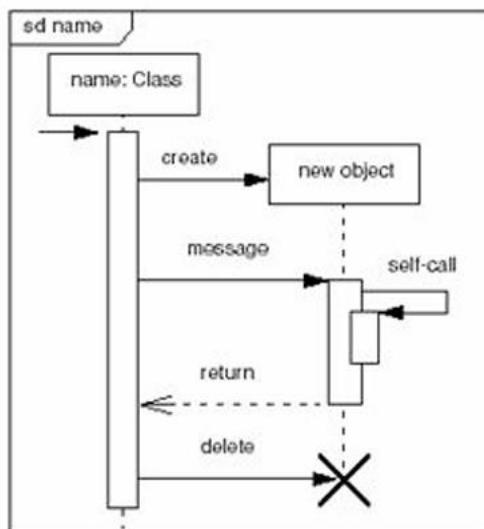


Deployment Diagram p. 97



OOAD

Sequence Diagram p. 53



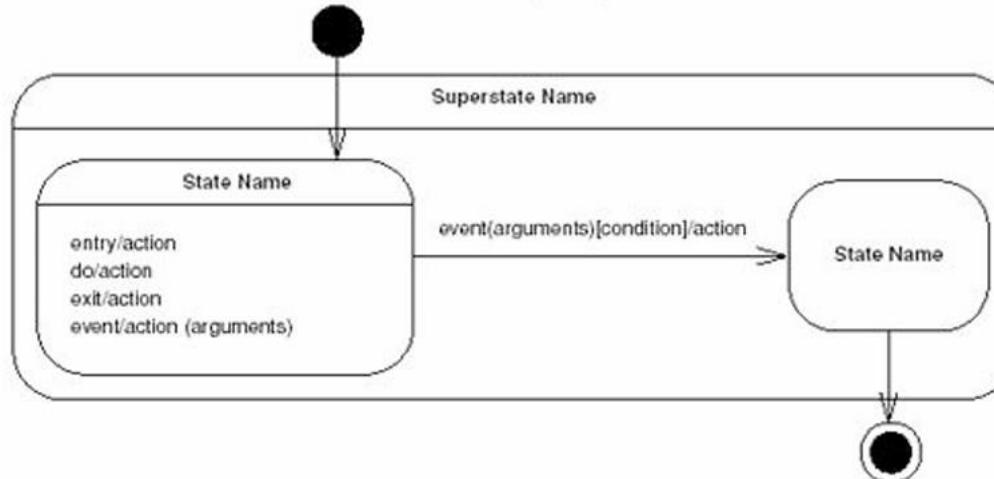
synchronous p. 61

asynchronous [UML >= 1.4]

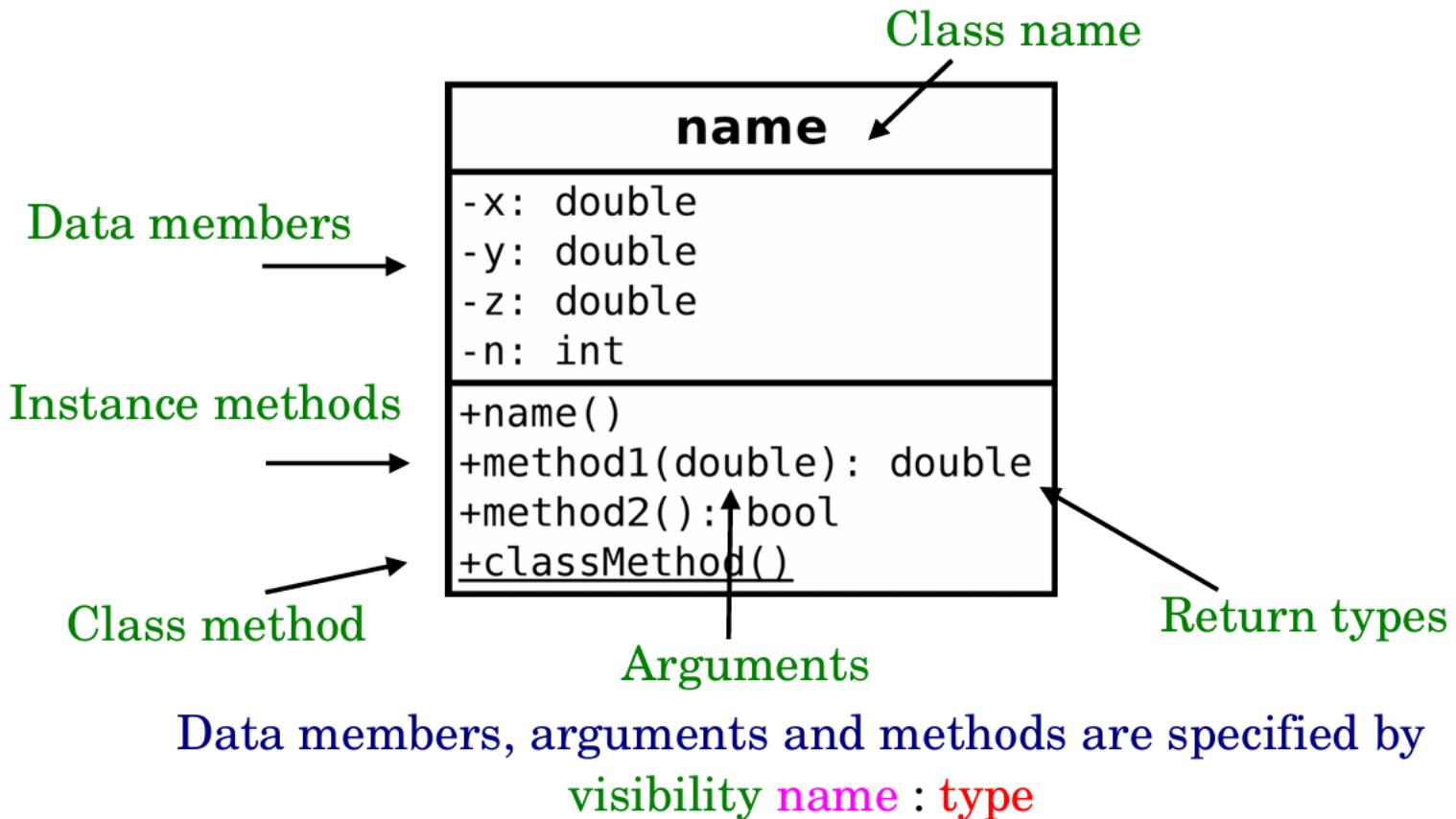
asynchronous [UML <= 1.3]

*: iteration message ()
[condition] message () [UML 1] p. 59

State Diagram p. 107

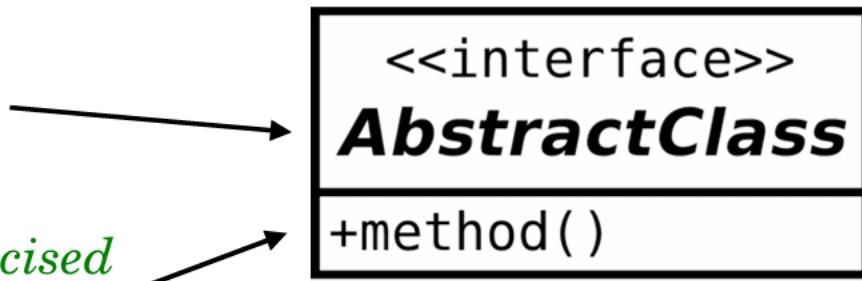


OOAD



OOAD

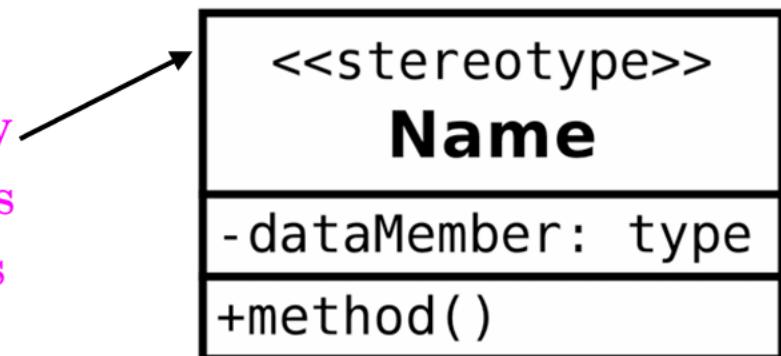
The top compartment contains the class name



Abstract classes have italicised names

Abstract methods also have italicised names

Stereotypes are used to identify groups of classes, e.g. interfaces or persistent (storeable) classes



OOAD

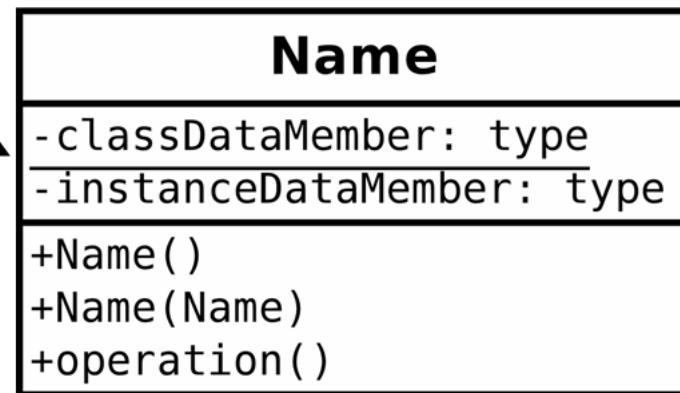
Attributes are the instance and class data members

Class data members (underlined) are shared between all instances (objects) of a given class

Data types shown after ":"

Visibility shown as
+ public
- private
protected

Attribute compartment



visibility name : type

OOAD

Operations are the class methods with their argument and return types

Public (+) operations define the class interface

Class methods (underlined) have only access to class data members, no need for a class instance (object)

visibility **name** : type

Name
<u>-classDataMember: type</u>
<u>-instanceDataMember: type</u>
+Name()
+Name(Name)
+instanceMethod()
<u>+classMethod()</u>



Operations compartment

+

public

-

private

#

protected

Anyone can access

No-one can access

Subclasses can access

Interface operations

Data members

Operations where subclasses collaborate

Not data members

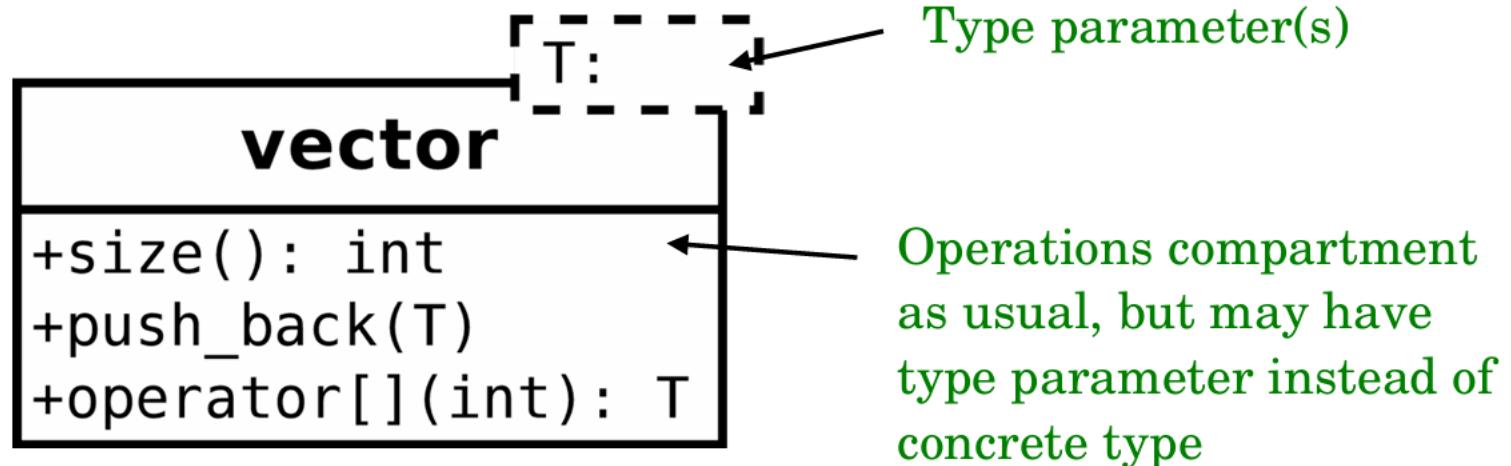
Helper functions

Not data members

“Friends” are allowed
in though

(creates dependency
of subclass on im-
plementation of parent)

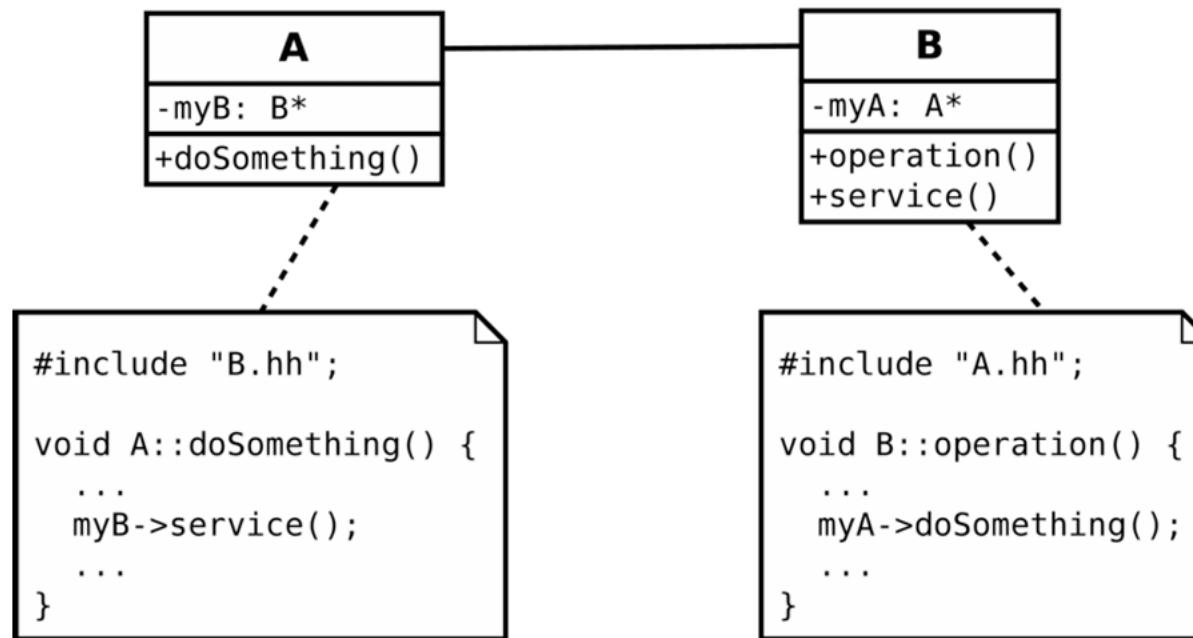
Generic classes depending on parametrised types



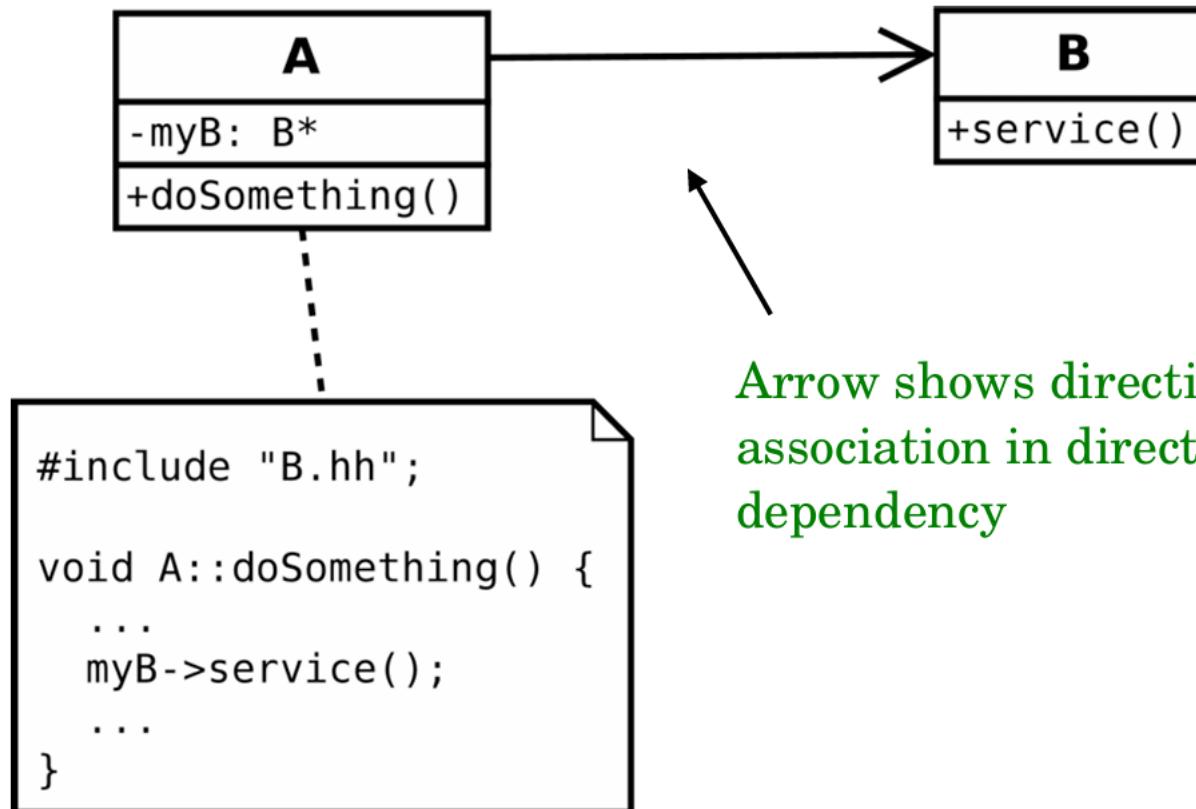
OOAD

- Association
- Aggregation
- Composition
- Parametric and Friendship
- Inheritance

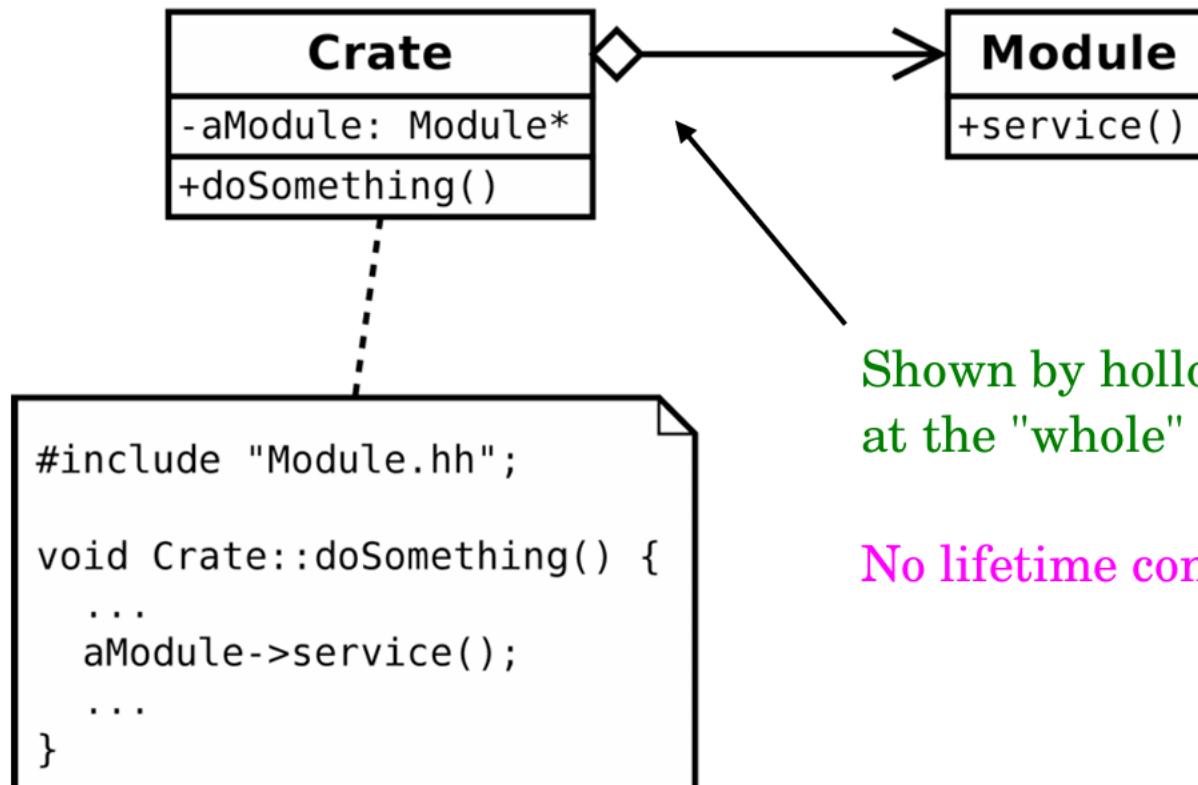
Binary association: both classes know each other



A knows about B, but B knows nothing about A



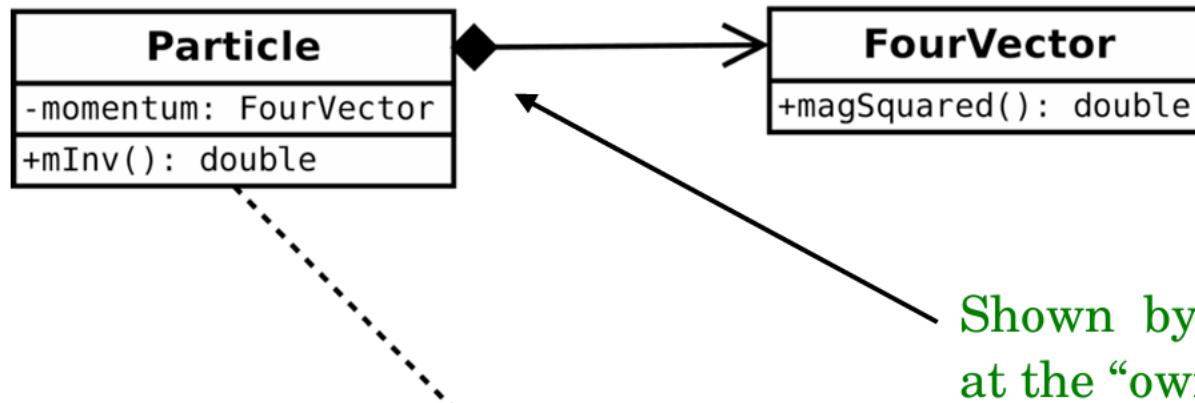
Aggregation = Association with “whole-part” relationship



Shown by hollow diamond
at the "whole" side

No lifetime control implied

Composition = Aggregation with lifetime control



Shown by filled diamond
at the “owner” side

```
double Particle::mInv() {  
    double minv2= momentum.magSquared();  
    return minv2<0?-sqrt(-minv2):sqrt(minv2);  
}
```

Lifetime control: construction and destruction controlled by “owner”

- call constructors and destructors (or have somebody else do it)

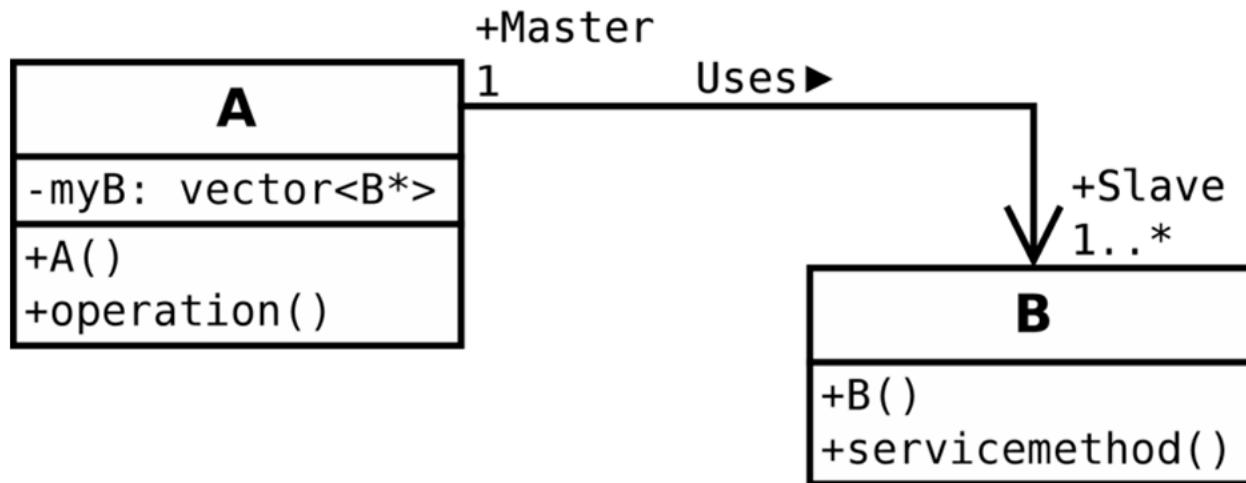
Lifetime control implied

Lifetime control can be transferred

OOAD

Name gives details of association

Name can be viewed as verb of a sentence

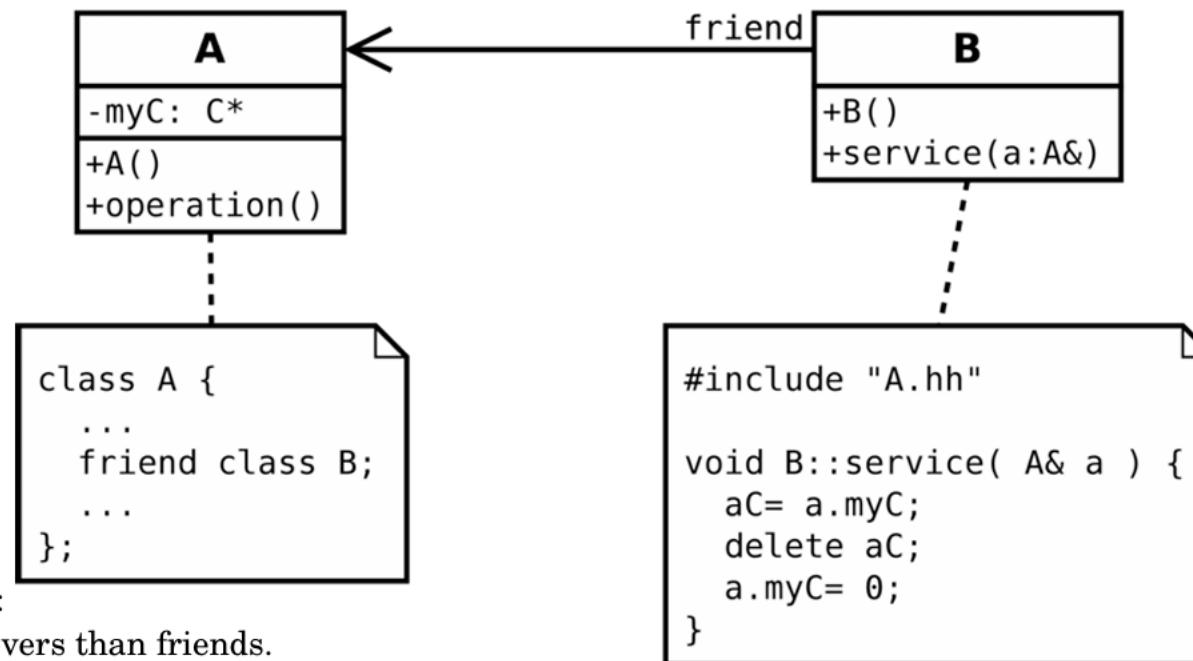


Notes at association ends
explain “roles” of classes (objects)

Multiplicities show number of
objects which participate in the
association

Friends are granted access to private data members and member functions

Friendship is given to other classes, never taken



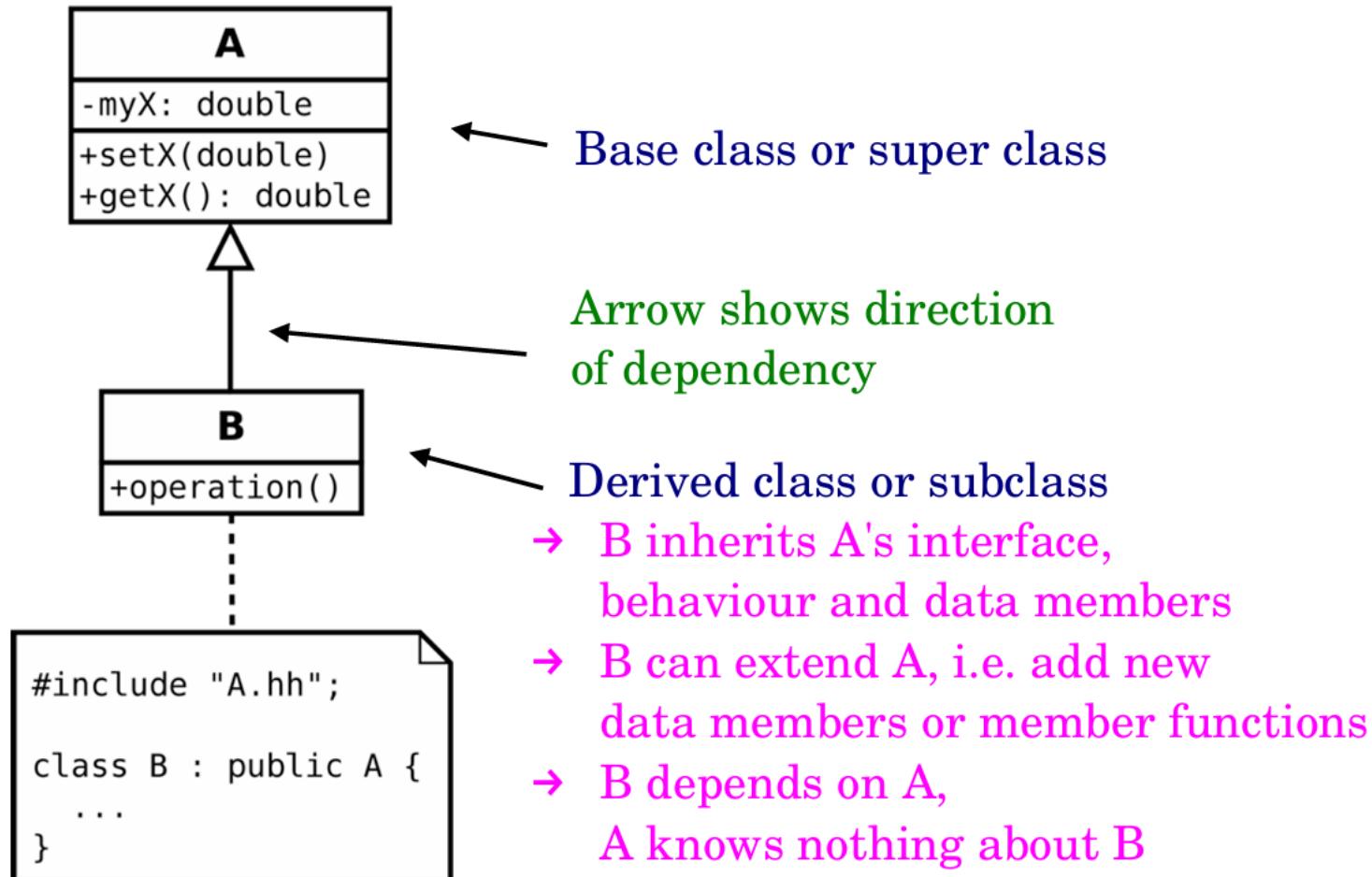
Bob Martin:

More like lovers than friends.

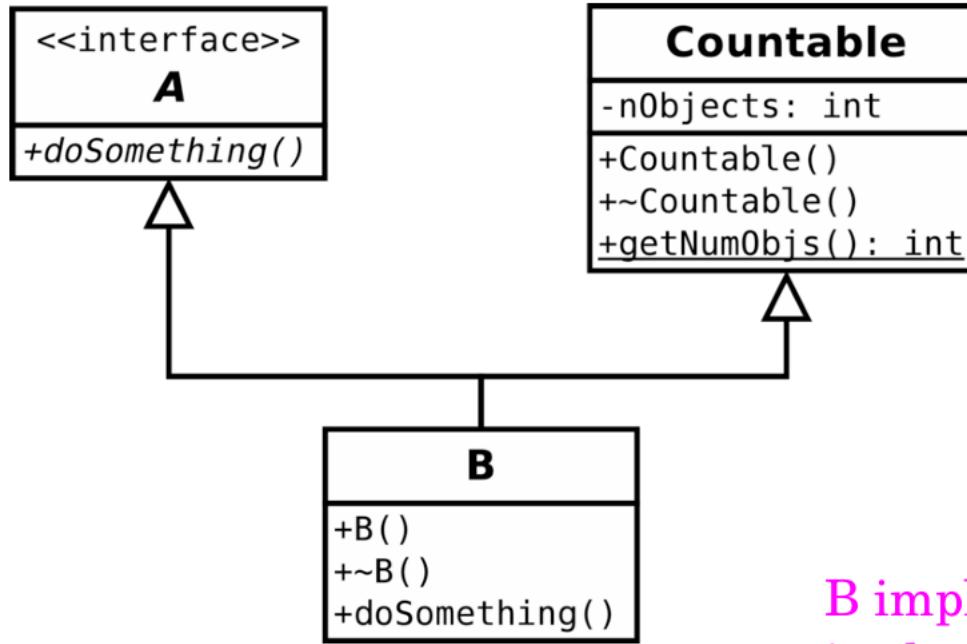
You can have many friends,
you should not have many lovers

Friendship breaks data hiding, use carefully

OOAD



OOAD



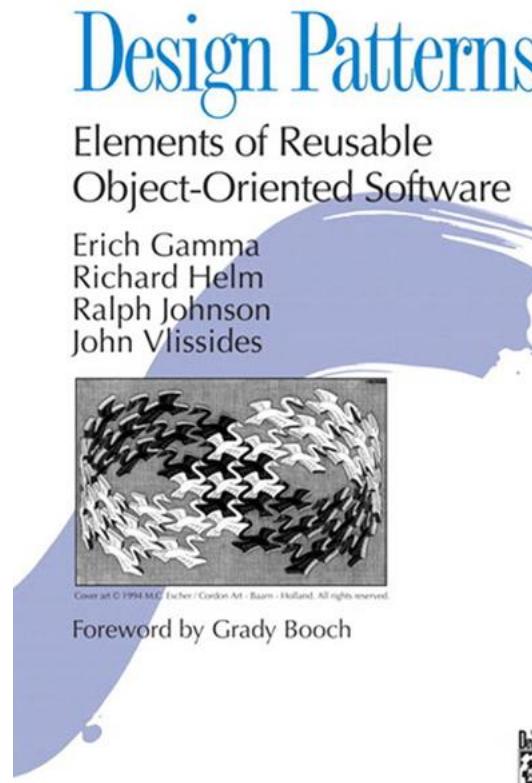
The derived class inherits interface, behaviour and data members of all its base classes

Extension and overriding works as before

B implements the interface A and is also a “countable” class

Countable also called a ”Mixin class”

Design Pattern



ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Creational Patterns

Abstract Factory (87) Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

Builder (97) Separate the construction of a complex object from its representation so that the same construction process can create different representations.

Factory Method (107) Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

Prototype (117) Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Singleton (127) Ensure a class only has one instance, and provide a global point of access to it.

Structural Patterns

Adapter (139) Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

Bridge (151) Decouple an abstraction from its implementation so that the two can vary independently.

Composite (163) Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

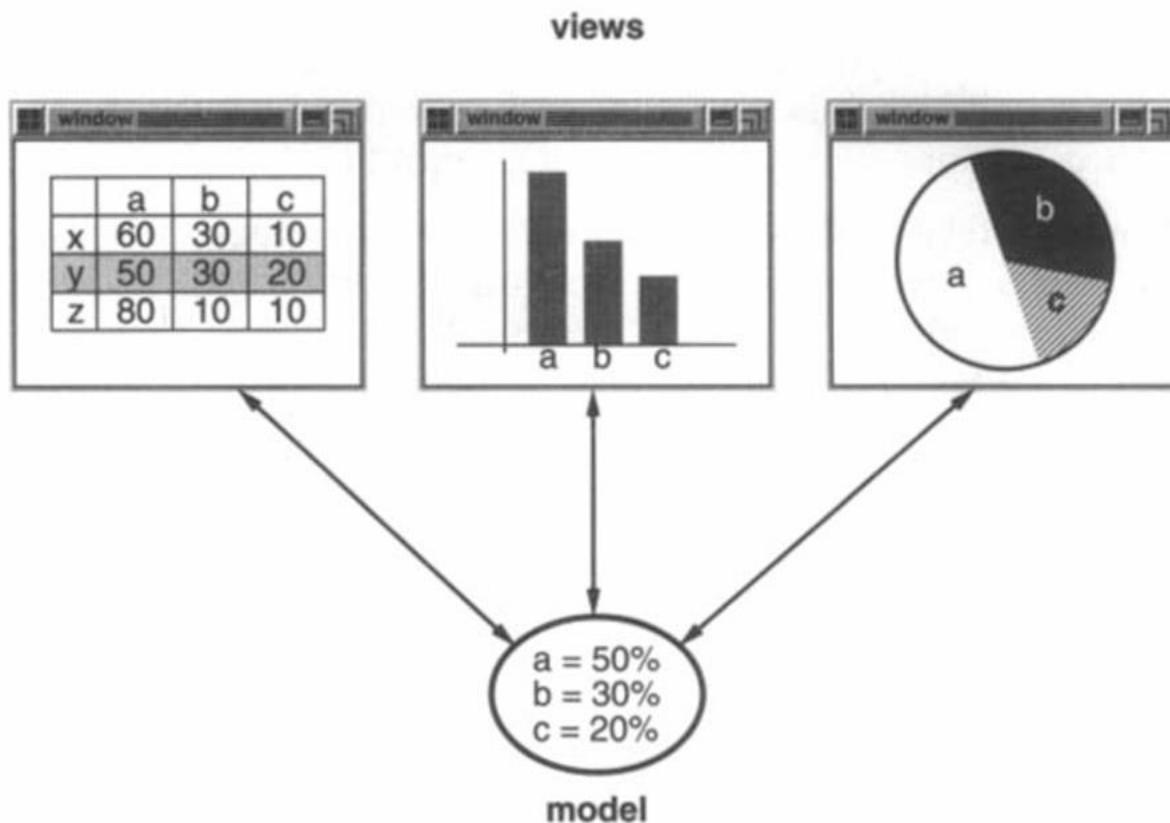
Decorator (175) Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

Facade (185) Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.

Flyweight (195) Use sharing to support large numbers of fine-grained objects efficiently.

Proxy (207) Provide a surrogate or placeholder for another object to control access to it.

Design Pattern



Design Pattern

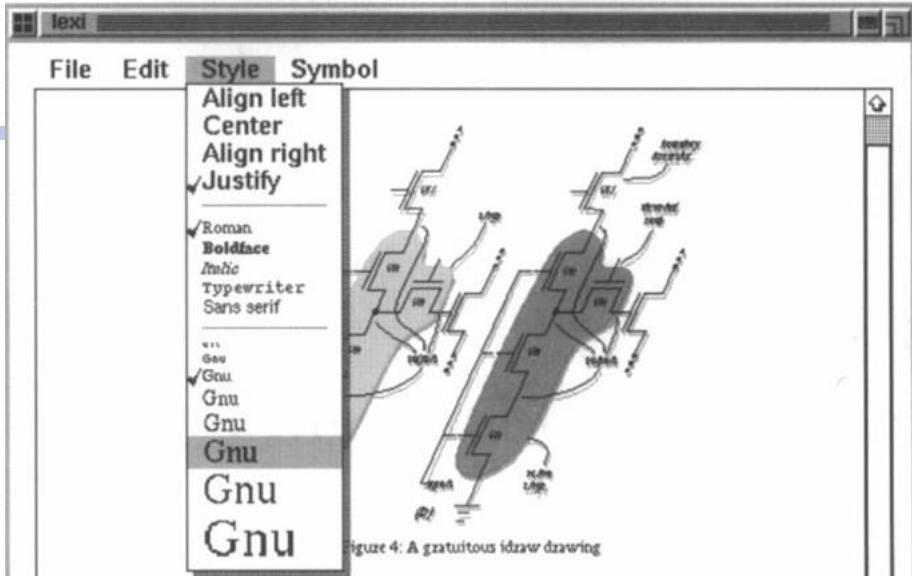


Figure 4: A gratuitous idraw drawing

The internal representation of the TextView. The draw operation (which is not shown) simply calls draw on the TBox.

The code that builds a TextView is similar to the original draw code, except that instead of calling functions to draw the characters, we build objects that will draw themselves whenever necessary. Using objects solves the zedraw problem because only those objects that lie within the damaged region will get draw calls. The programmer does not have to write the code that decides what objects to redraw—that code is in the toolkit (in this example, in the implementation of the Box draw operation). Indeed, the glyph-based implementation of TextView is even simpler than the original code because the programmer need only declare what objects he wants—he does not need to specify how the objects should interact.

?? Multiple fonts

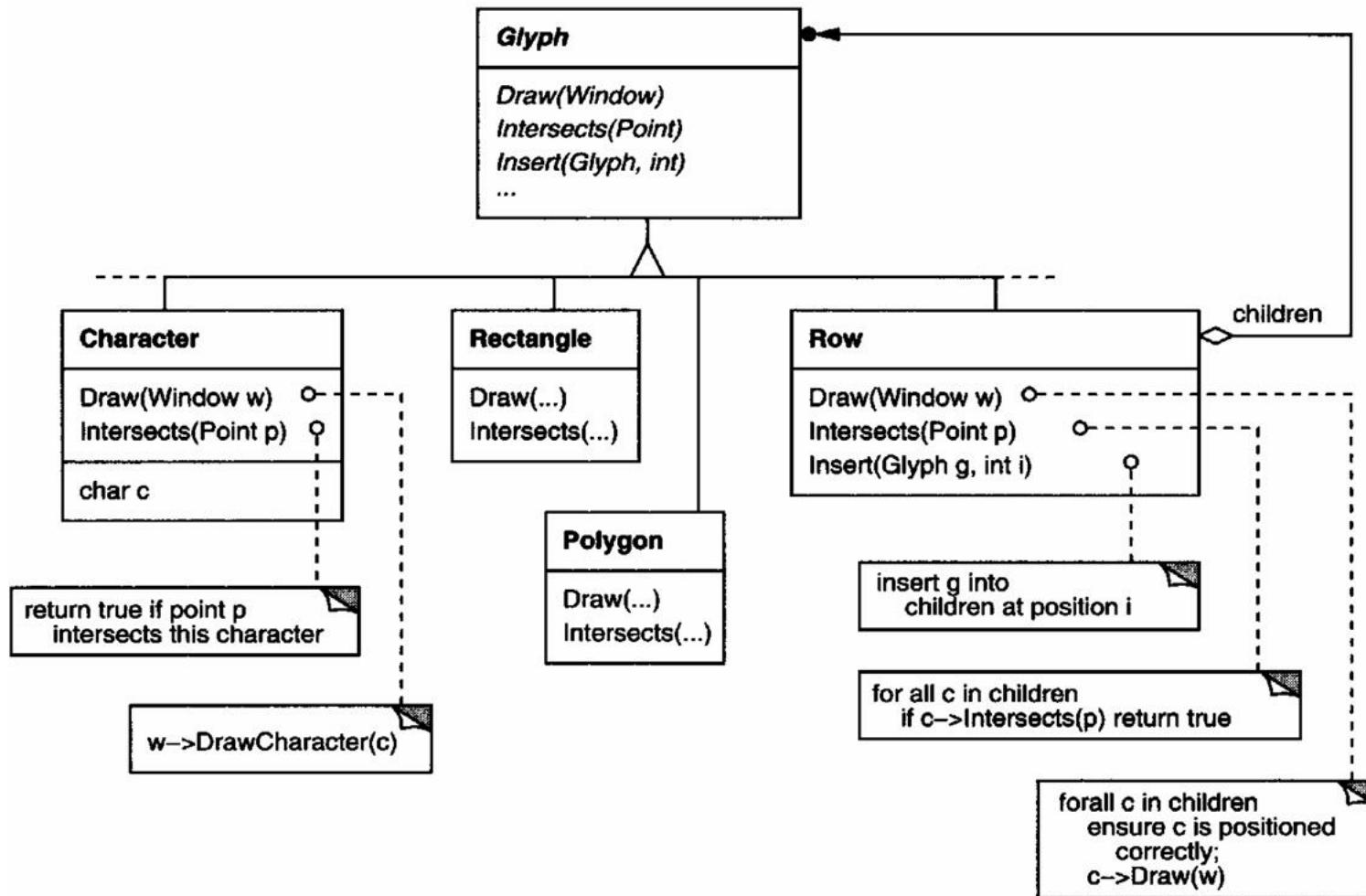
Because we built TextView with glyphs, we can easily extend it to add functionality that might otherwise be difficult to implement. For example, Figure 4 shows a screen dump of a version of TextView that displays EUC-encoded Japanese text. Adding this feature to a text view such as the Athena Text Widget would require a complete rewrite. Here we only add two lines of code. Figure 5 shows the change.

Character glyphs take an optional second constructor parameter that specifies the font to use when drawing. For ASCII-encoded text we create Characters that use the 8-bit ASCII-encoded “a14” font; for JIS-encoded

```
while ((c = getc(file)) != EOF) {
    if (c == '\n') {
        line = new LRBox();
    } else if (!isascii(c)) {
        line->append(
            new Character(
                tojis(c, getc(file)), k14
            )
        );
    } else {
        line->append(
            new character(c, a14)
        );
    }
}
```

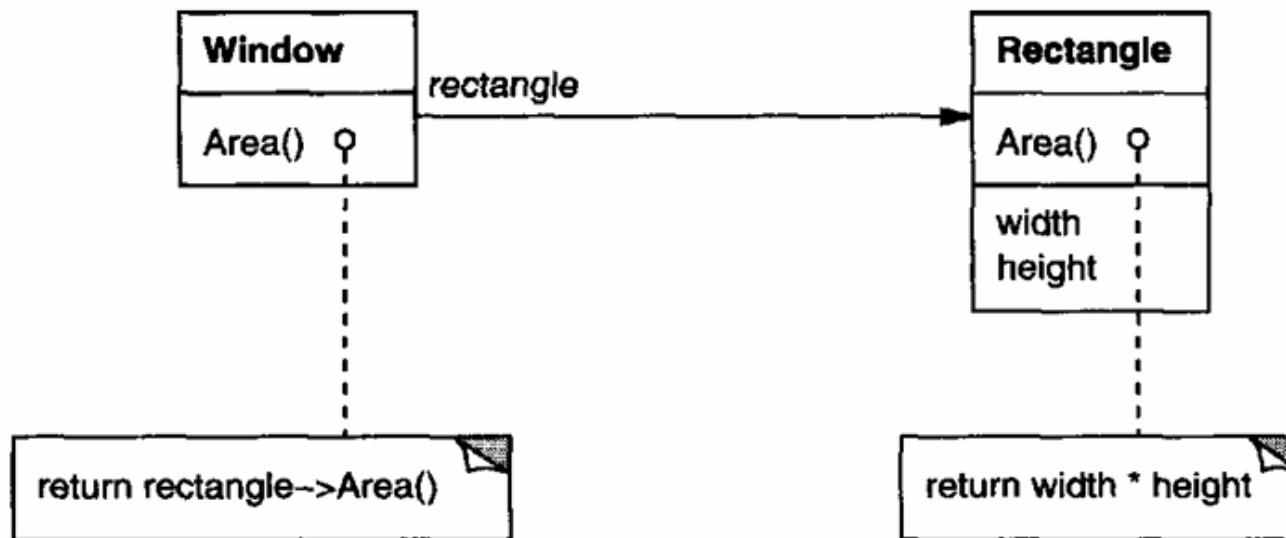
Figure 5: Modified TextView that displays Japanese text

Design Pattern



Design Pattern

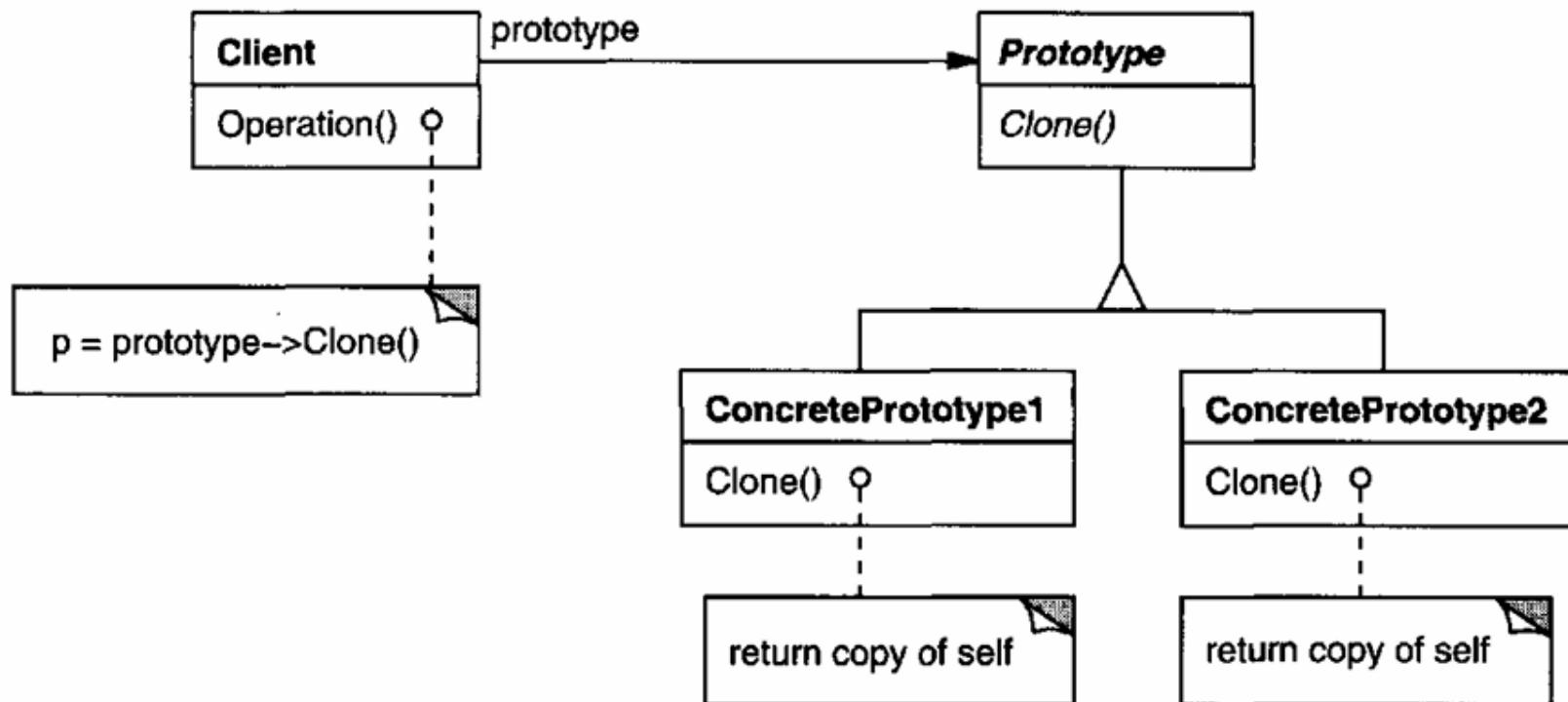
Delegation



Design Pattern

Prototype

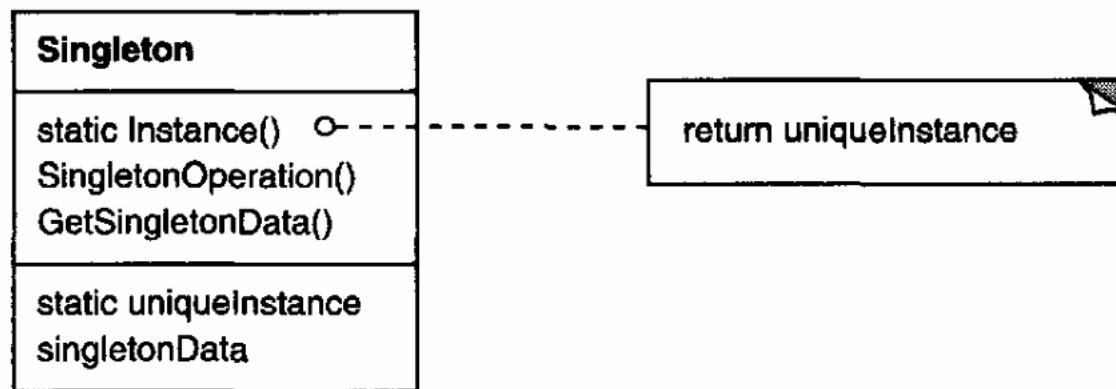
Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype



Design Pattern

Singleton

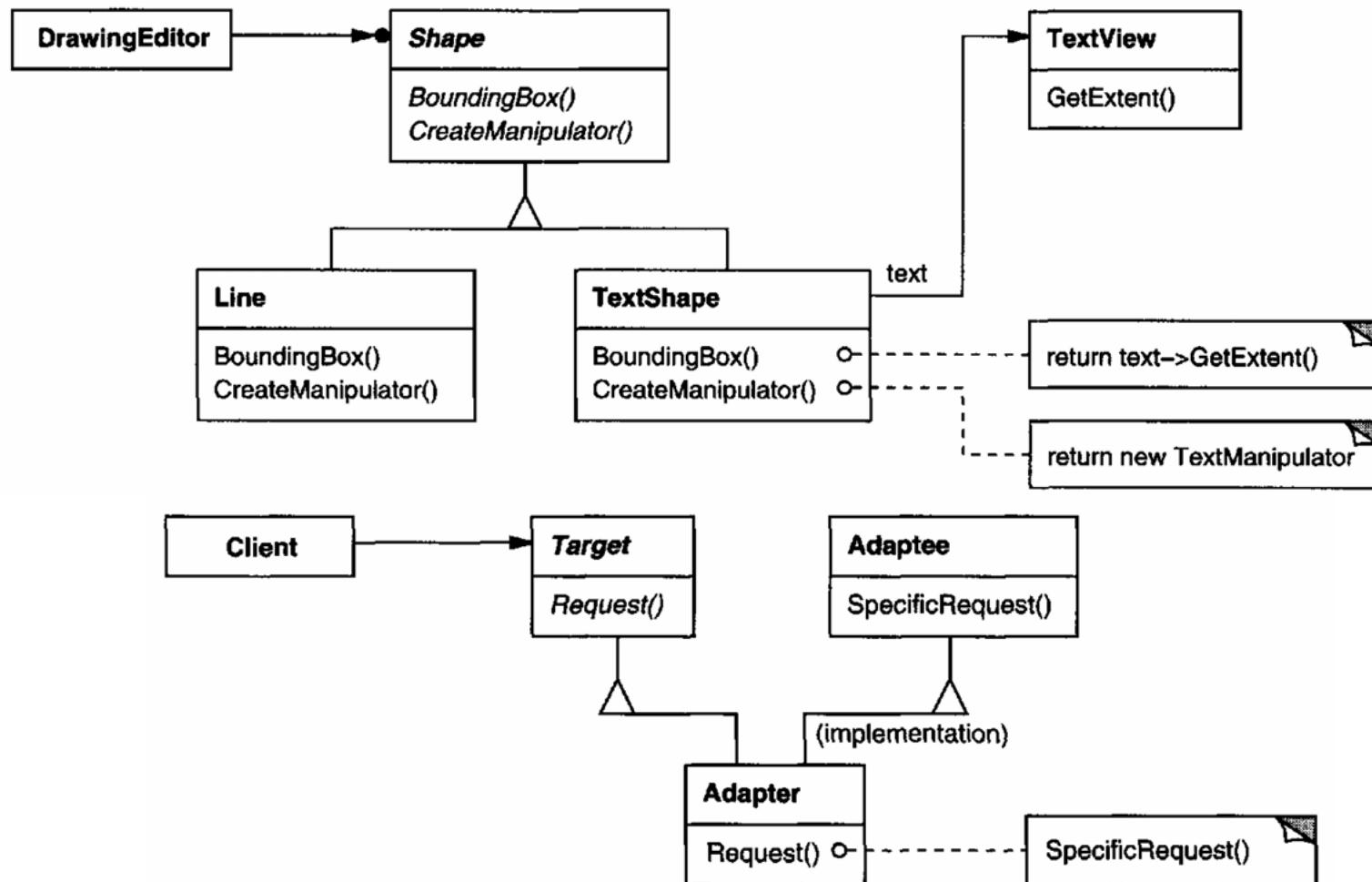
Ensure a class only has one instance, and provide a global point of access to it



Design Pattern

Adapter

Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.



Planning a Project with the Rational Unified Process

Author: David West

Rational Software White paper
TP 151, 08/02

Method

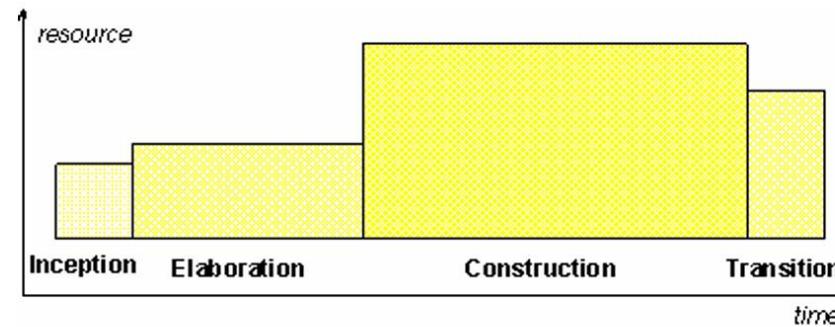
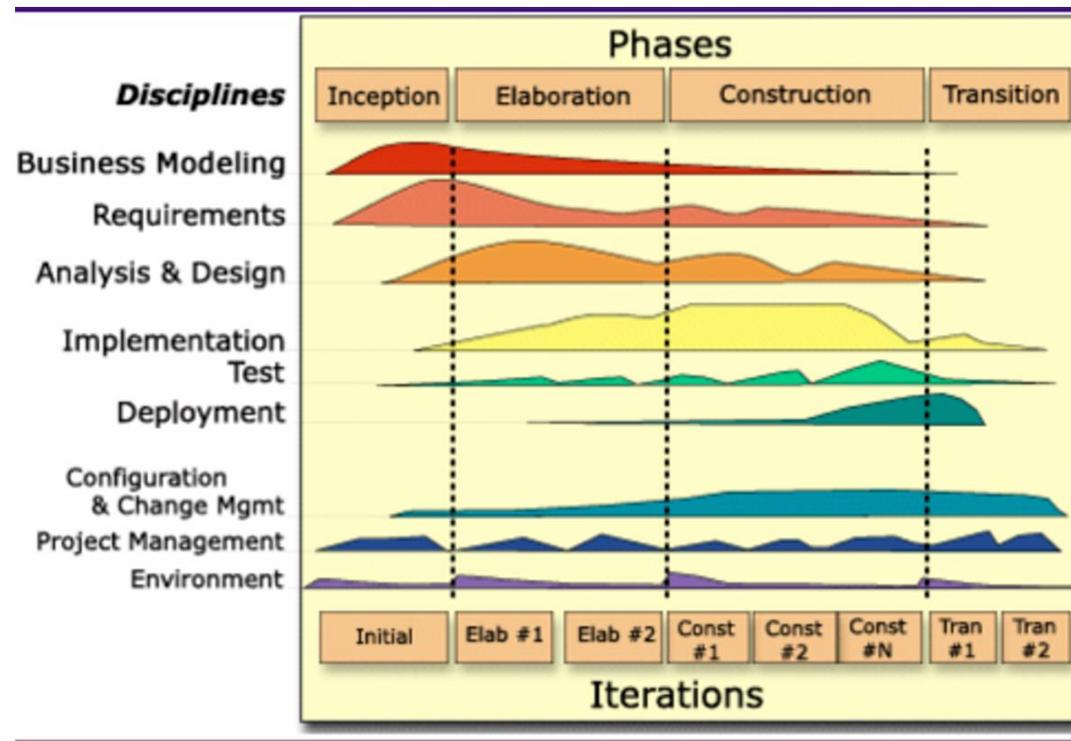
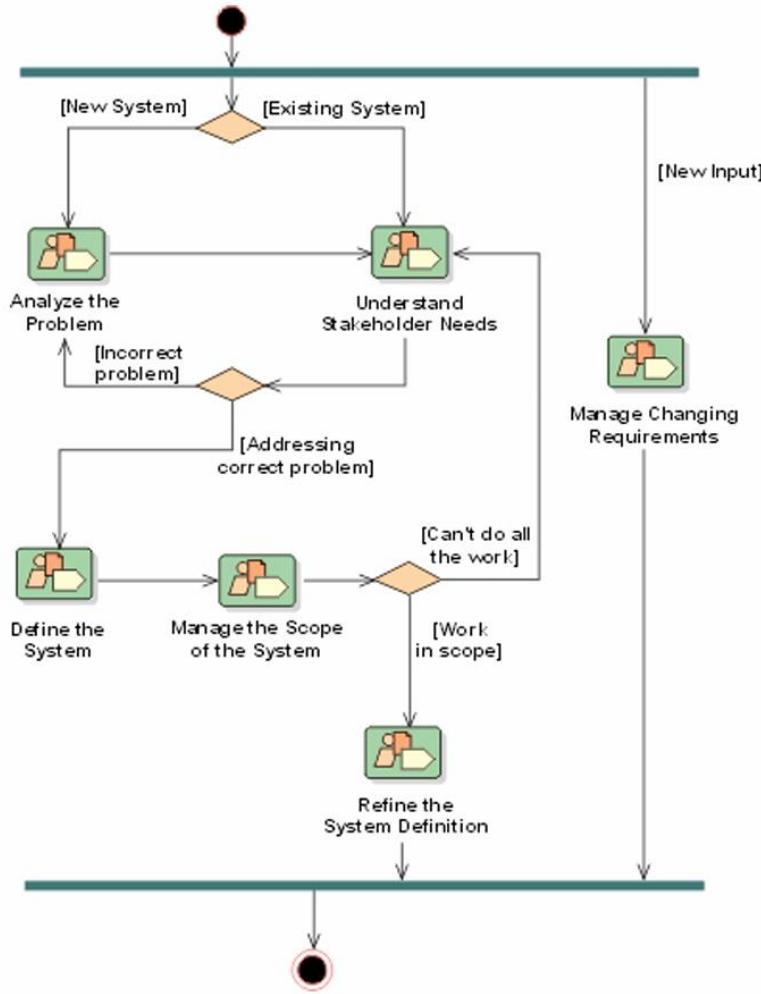


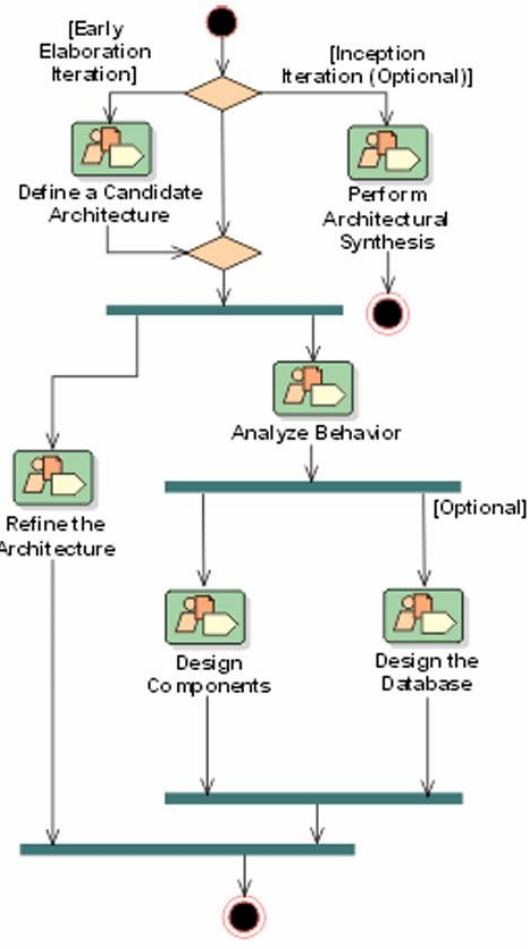
Figure 1: Typical time allocation for the four phases of a project

Method

Requirements

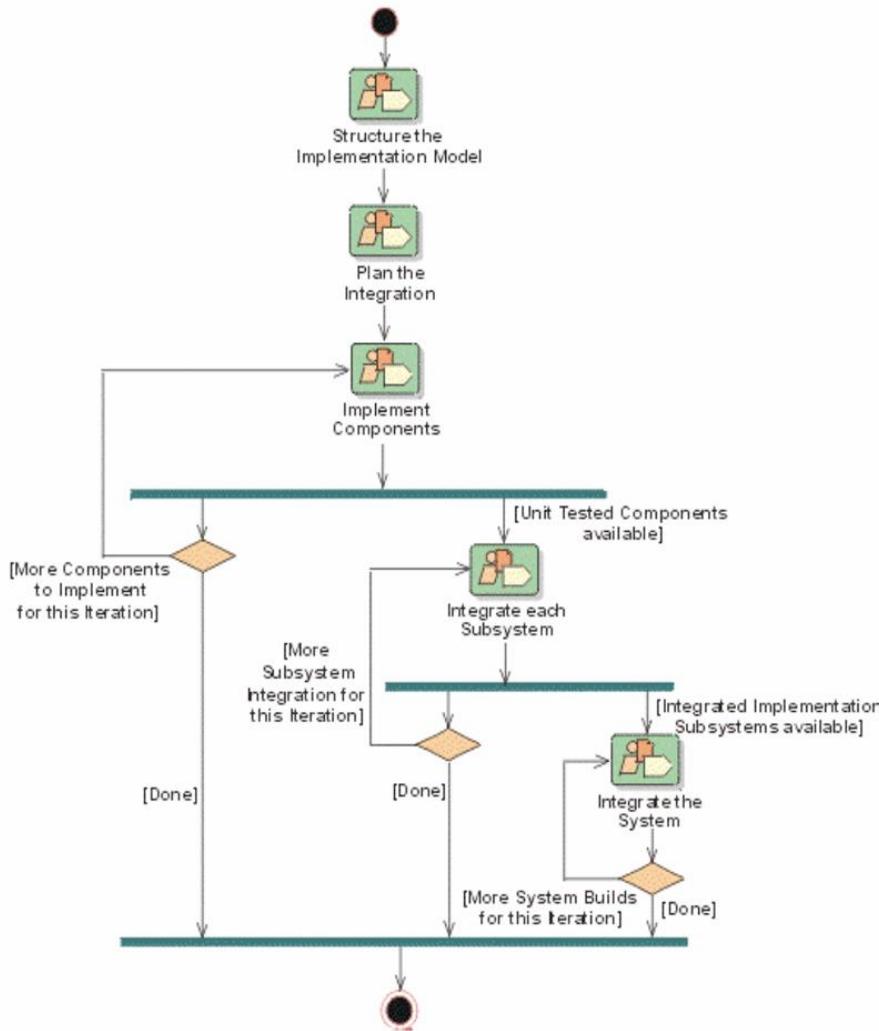


Analysis and Design

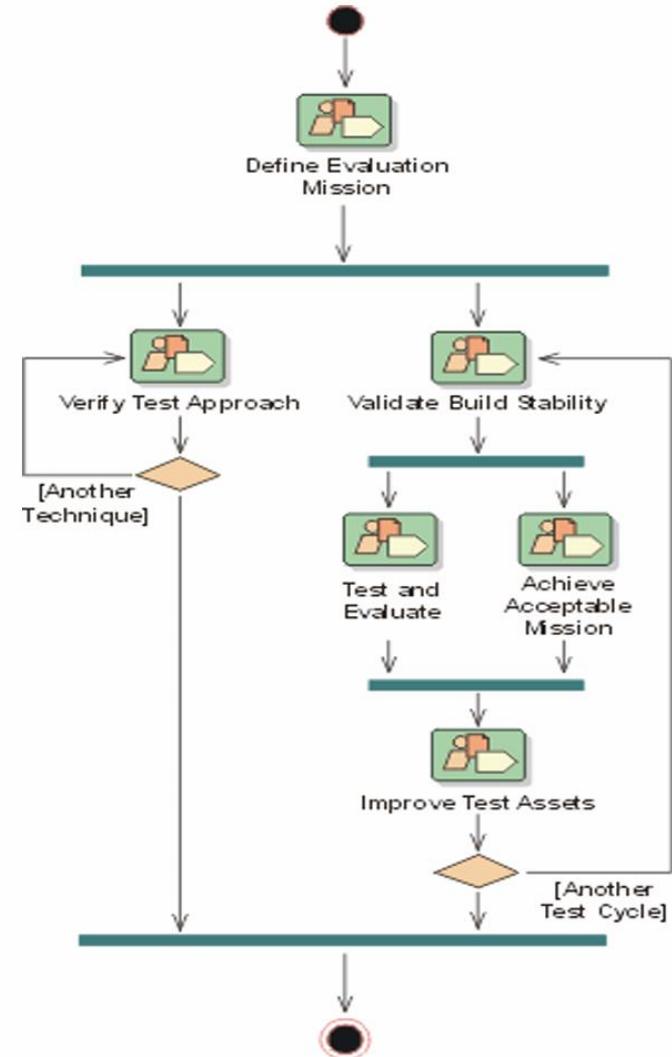


Method

Implementation



Testing



Method

Kent Beck

eXtreme Programming An Overview

Methoden und Werkzeuge der Softwareproduktion WS 1999/2000

Author Thomas Dudziak



Method

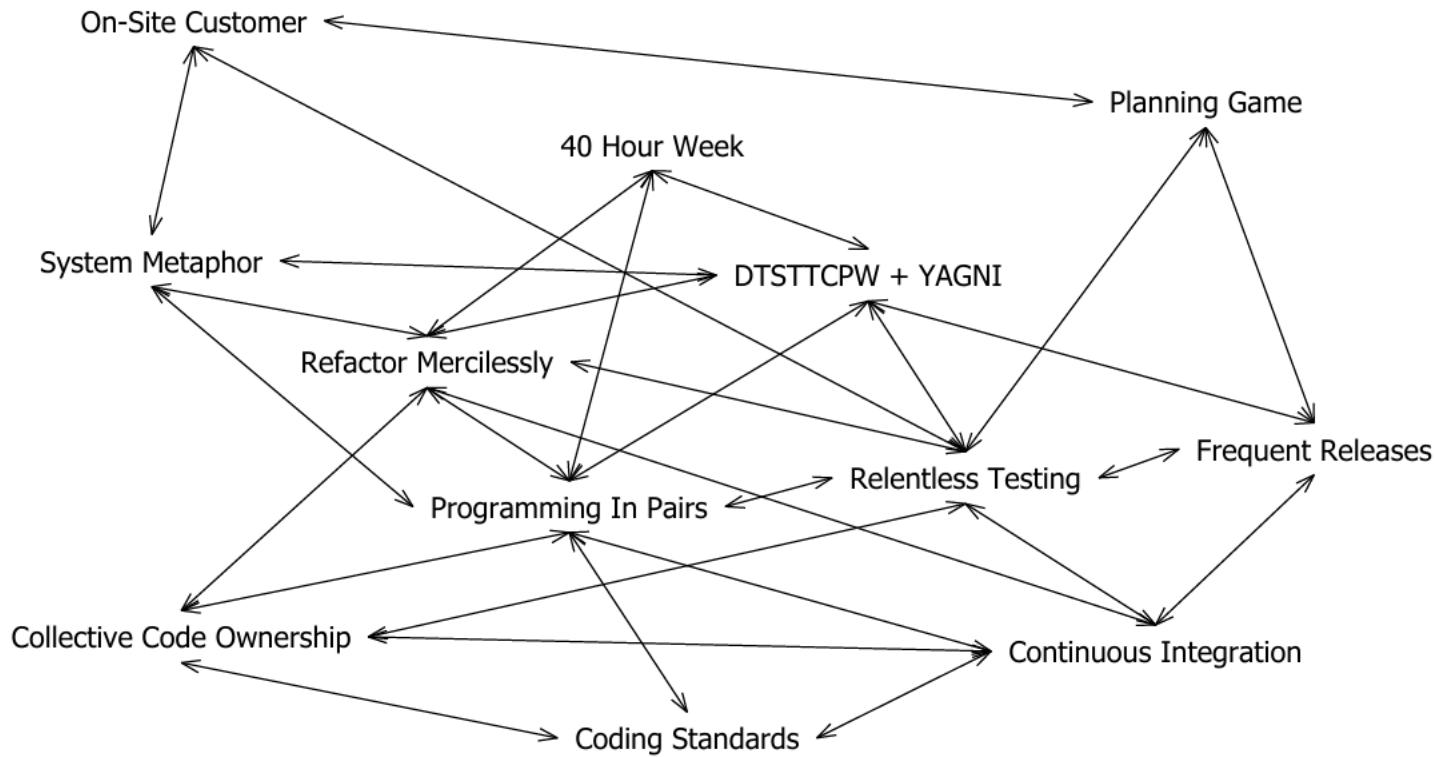


Figure 3 : Support relationships between the rules and practices

DTSTTCPW: Do The Simplest Thing That Could Possibly Work
YAGNI: You Ain't Gonna Need It

Method

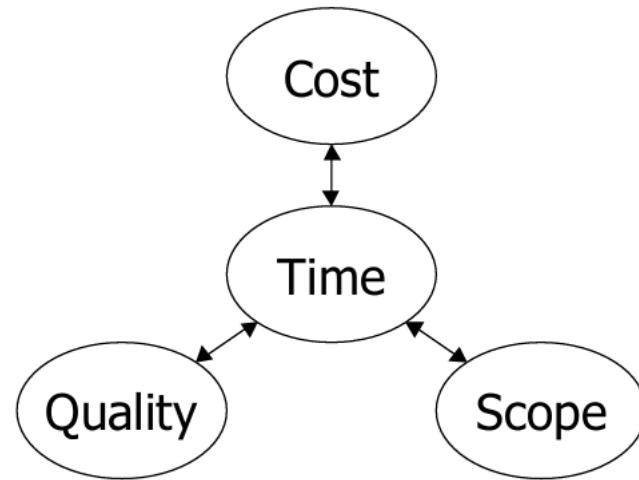


Figure 1 : The relationships between the variables

Method

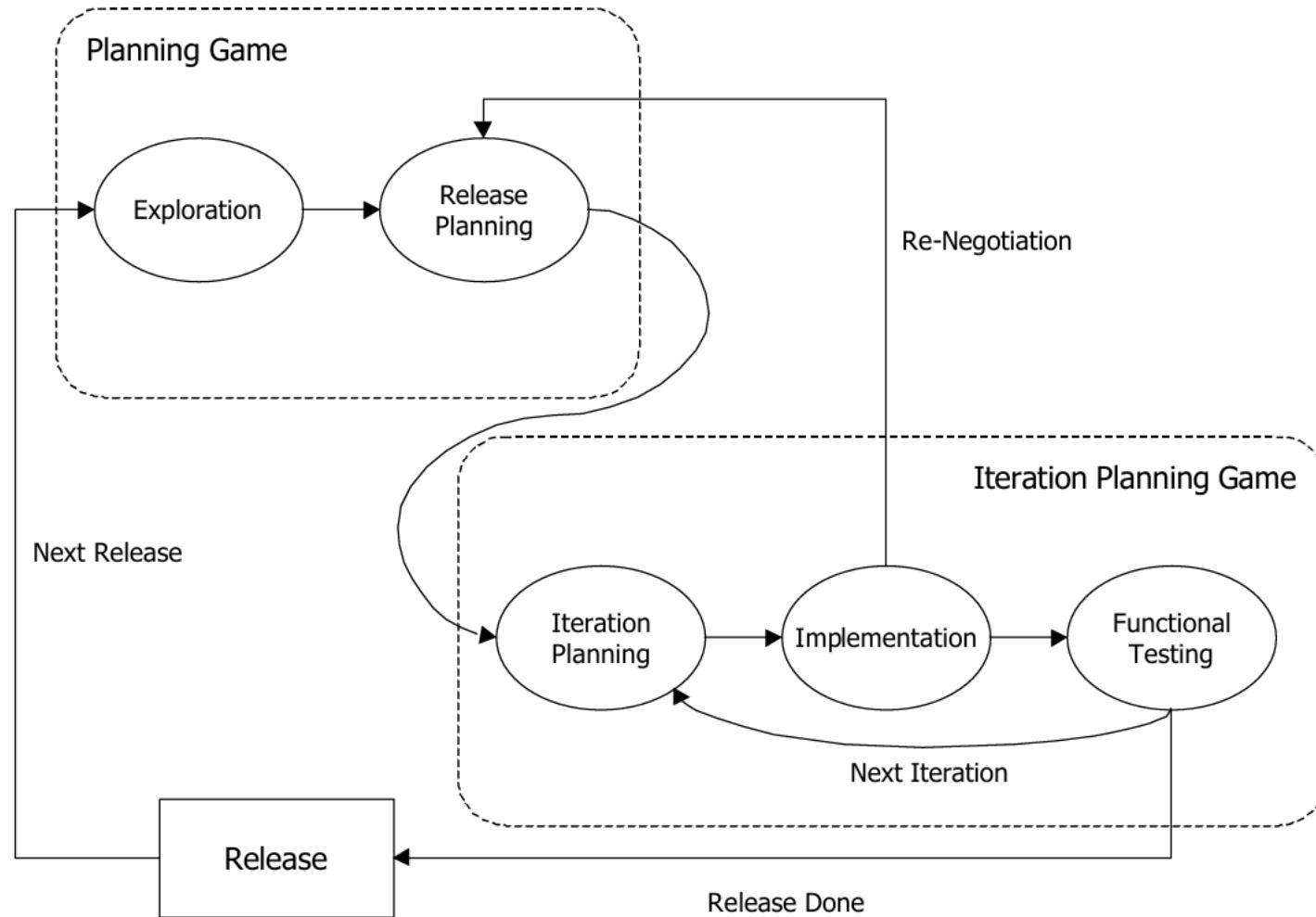


Figure 2 : Simplified process structure

Method

Picture 1 : A User Story card from the C3 project

Method

Picture 2 : A Engineering Task card from the C3 project

Method

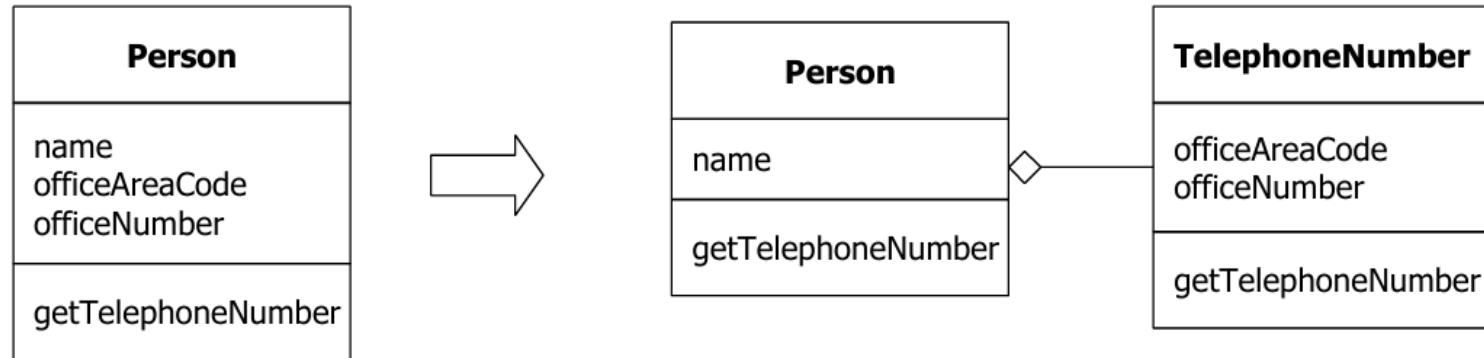


Figure 5 : Extract Class Refactoring



Picture 5 : A pair

Method

Class Responsibility Collaborator

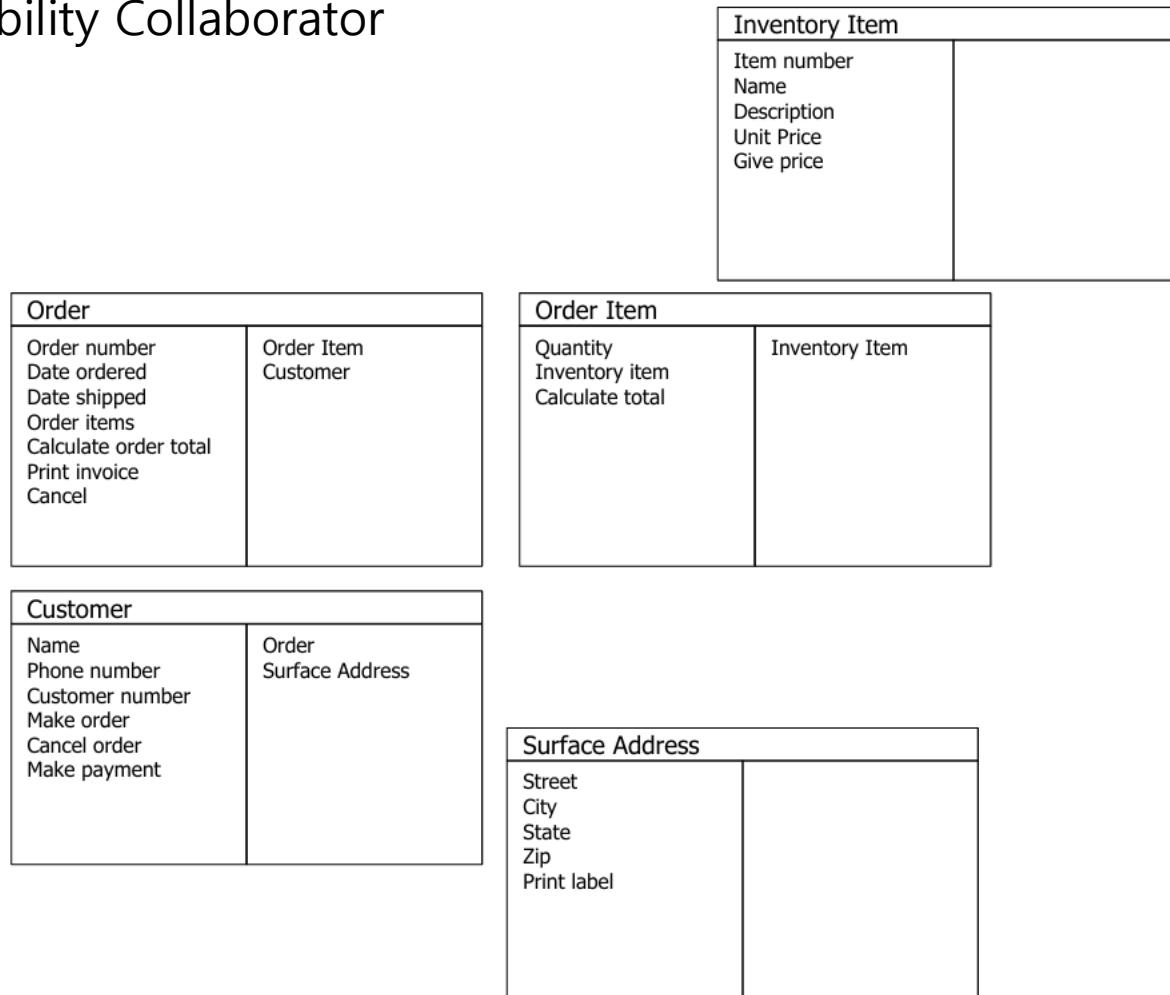


Figure 6 : Example of a CRC layout

Method

Planning

Co-locate the project with the client, write user stories with the client, frequent small releases (1-2 months), create schedule with release planning, kick off an iteration with iteration planning, create programmer pairs, allow rotation of pairs

Modeling

Select the simplest design that addresses the current story; Use a system metaphor to model difficult concepts; Use CRC cards for the initial object identification; Write code that adheres to standards; Refactor whenever possible

Process

Code unit test first, do not release before all unit tests pass, write a unit test for each uncovered bug, integrate one pair at the time

Control

Code is owned collectively. Adjust schedule, Rotate pairs, Daily status stand-up meeting, Run acceptance tests often and publish the results.

Design Patterns In Python

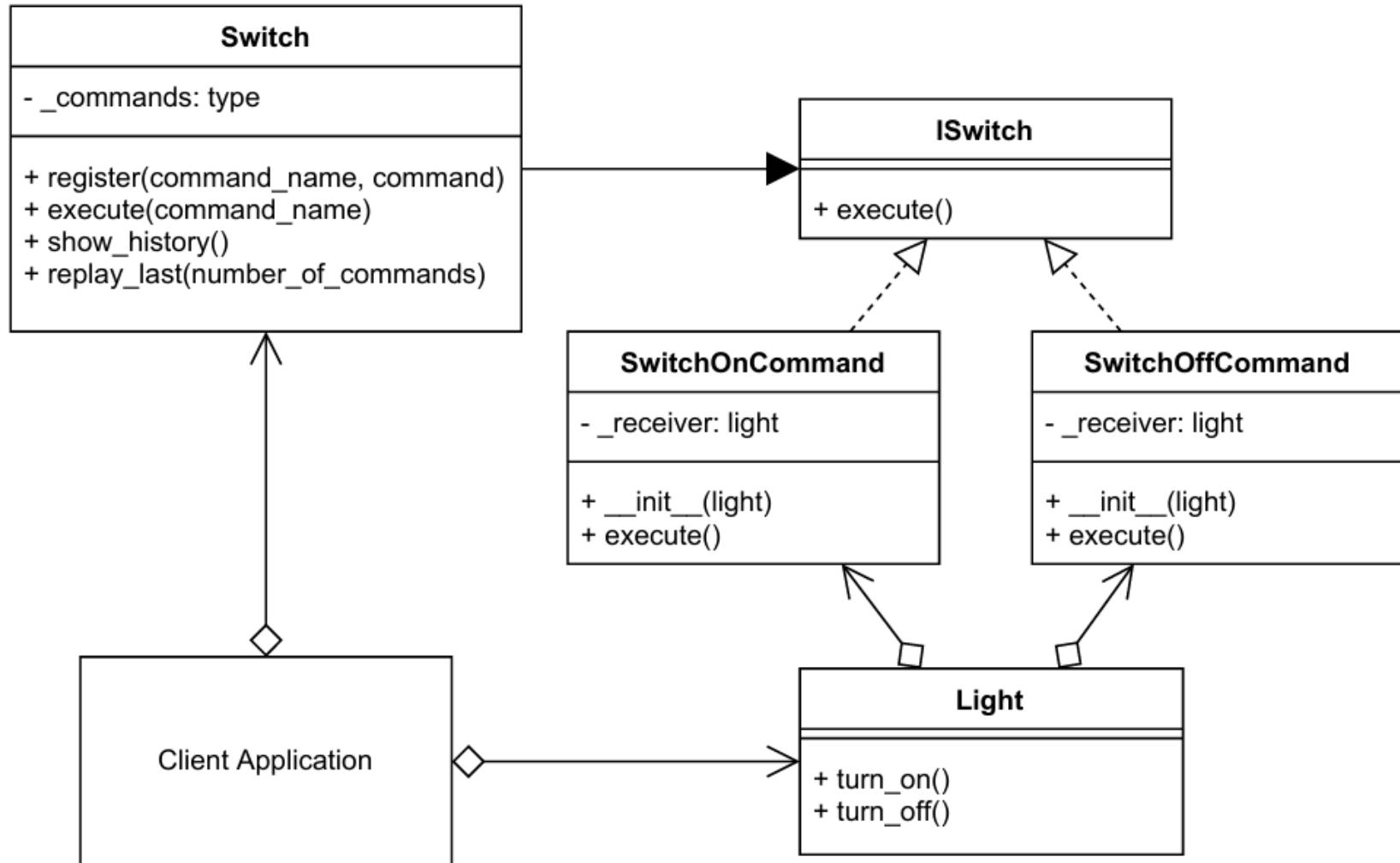
Common GoF (Gang of Four) Design Patterns

Implemented In Python

Sean Bradley

Copyright © 2019-2021 Sean Bradley

Practice



Practice

```
class Switch:  
    "The Invoker Class."  
    def __init__(self):  
        self.commands = {}  
        self.history = []  
  
    def show_history(self):  
        "Print the history of each time a command was invoked"  
        for row in self.history:  
            print(  
                f"{{datetime.fromtimestamp(row[0]).strftime('%H:%M:%S')}}"
```

Practice

Practice

[CityGML 3.0 \(Python version\) parser for reading, writing, and converting into JSON using Python.](#)

The screenshot shows a GitHub repository page for 'citygml_parser'. The repository is public and has 9 stars, 2 forks, and 19166 issues. It includes sections for 'About', 'Readme', 'Activity', '9 stars', '2 watching', and '2 forks'. The 'About' section describes the project as a 'CityGML 3.0 (Python version) parser for reading, writing, and converting CityGML files into JSON using Python.' The repository has 7 commits from 'mac999' and 8 files added via upload. The 'Code' tab is selected.

mac999 / citygml_parser

Code Issues Pull requests Actions Projects Wiki Security Insights

citygml_parser Public Unpin Unwatch 2 Fork 2 Starred 9

main Go to file + Code About

mac999 Update README.md 79664d5 · 6 months ago

docs Add files via upload 8 months ago

mesh Add files via upload 11 months ago

sample Add files via upload 11 months ago

README.md Update README.md 6 months ago

citygml_converter.py Add files via upload 11 months ago

citygml_json.py Add files via upload 11 months ago

citygml_parser3.py Add files via upload 11 months ago

citygml_parser_exampl... Add files via upload 11 months ago

About

CityGML 3.0 (Python version) parser for reading, writing, and converting CityGML files into JSON using Python.

python parser json digital
iso mesh city smart
citygml twin 19166

Readme Activity 9 stars 2 watching 2 forks

Practice

[landxml parser, civil model server and civil model web viewer](#)

The screenshot shows a GitHub repository page for 'landxml_parser'. The repository is public and has 7 stars, 3 watching, 2 forks, and 7 code contributions. It contains files like doc, javascript, sample, web_app, README.md, block_model_output.js..., and civil_geo_engine.py. The repository was last updated 3 months ago.

Code contributions:

- mac999 Update section headers in README... 64e5302 · 3 months ago
- doc Add files via upload 10 months ago
- javascript Add files via upload 11 months ago
- sample Add files via upload 11 months ago
- web_app Add files via upload 11 months ago
- README.md Update section headers ... 3 months ago
- block_model_output.js... Add files via upload 3 months ago
- civil_geo_engine.py Add files via upload 3 months ago

About:

- LandXML parser
- Readme
- Activity
- 7 stars
- 3 watching
- 2 forks

Releases:

No releases published
[Create a new release](#)

Packages:

Appendix: StarUML

Chapter 1. StarUML Overview

This chapter contains a general overview of StarUML™. Included are brief overviews of StarUML™ and UML, and outlines of StarUML™'s new features and overall organization.

- What is StarUML
- Key Features
- System Requirements

What is StarUML

StarUML™ is a software modeling platform that supports UML (Unified Modeling Language). It is based on UML version 1.4 and provides eleven different types of diagram, and it accepts UML 2.0 notation. It actively supports the MDA (Model Driven Architecture) approach by supporting the UML profile concept. StarUML™ excels in customizability to the user's environment and has a high extensibility in its functionality. Using StarUML™, one of the top leading software modeling tools, will guarantee to maximize the productivity and quality of your software projects.

UML Tool That Adapts to the User

StarUML™ provides maximum customization to the user's environment by offering customizing variables that can be applied in the user's software development methodology, project platform, and language.

True MDA Support

Software architecture is a critical process that can reach 10 years or more into the future. The intention of the OMG (Object Management Group) is to use MDA (Model Driven Architecture) technology to create platform independent models and allow automatic acquisition of platform dependent models or codes from platform independent models. StarUML™ truly complies with UML 1.4 standards, UML 2.0 notation and provides the UML Profile concept, allowing creation of platform independent models. Users can easily obtain their end products through simple template document.

Excellent Extensibility and Flexibility

StarUML™ provides excellent extensibility and flexibility. It provides Add-In frameworks for extending the functionality of the tool. It is designed to allow access to all functions of the model/meta-model and tool through COM Automation, and it provides extension of menu and option items. Also, users can create their own approaches and frameworks according to their methodologies. The tool can also be integrated with any external tools.

Chapter 3. Managing Project

This chapter describes in detail the procedures for project management: creating a new project, making a part of the project into a unit, creating and importing model fragments, importing frameworks, and including and excluding UML profiles.

- Managing a Project
- Managing Units
- Working with Model Fragments
- Importing a Framework
- Working with UML Profiles

Managing a Project

CREATING NEW PROJECT

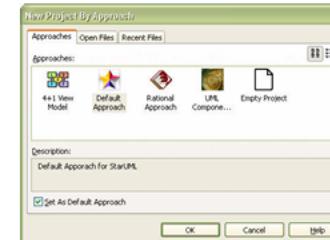
In order to work on a new software development, a new project must be created. You may start with a completely empty project or with a new project that has been initialized according to a specific approach.

Procedure for Creating New Project #1 – New Project:

1. Select the [File] -> [New Project] menu.
2. A new project is created with the default approach selected by the user. Depending on the approach, profiles and/or frameworks may be included/loaded

Procedure for Creating New Project #2 – Select New Project Dialog Box:

1. Select the [File] -> [Select New Project...] menu.
2. A list of the available approaches will be displayed in the Select New Project dialog box. Select one from the list and click the [OK] button.



3. A new project is created and initialized according to the selected approach. Depending on the approach, profiles and/or frameworks may be included/loaded.

Note

- The list of the available approaches may differ depending on the user's installation environment.
- To change the default approach, open the Select New Project dialog box, select an approach, and then check the option "Set As Default Approach"

Appendix: PlantUML

Drawing UML with PlantUML



PlantUML Language Reference Guide (Version 1.2025.0)

PlantUML is a component that allows to quickly write :

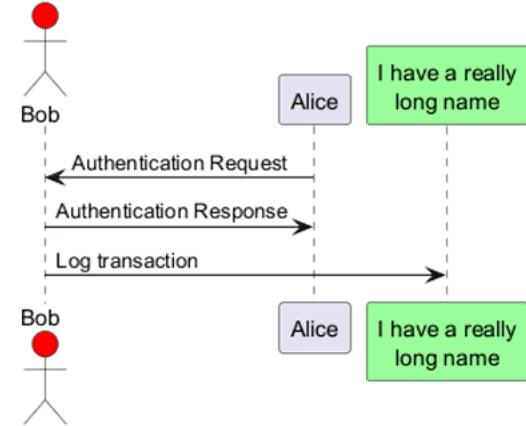
- Sequence diagram
- Usecase diagram
- Class diagram
- Object diagram
- Activity diagram
- Component diagram
- Deployment diagram
- State diagram
- Timing diagram

The following non-UML diagrams are also supported:

- JSON Data
- YAML Data
- Network diagram (nwdiag)
- Wireframe graphical interface

```
@startuml
actor Bob #red
' The only difference between actor
' and participant is the drawing
participant Alice
participant "I have a really\nlong name" as L #99FF99
' You can also declare:
participant L as "I have a really\nlong name" #99FF99
'/
```

```
Alice->Bob: Authentication Request
Bob->Alice: Authentication Response
Bob->L: Log transaction
@enduml
```



You can use the `order` keyword to customize the display order of participants.

```
@startuml
participant Last order 30
participant Middle order 20
participant First order 10
```

Q&A

laputa9999@gmail.com

<https://www.linkedin.com/in/tae-wook-kang-64a83917>