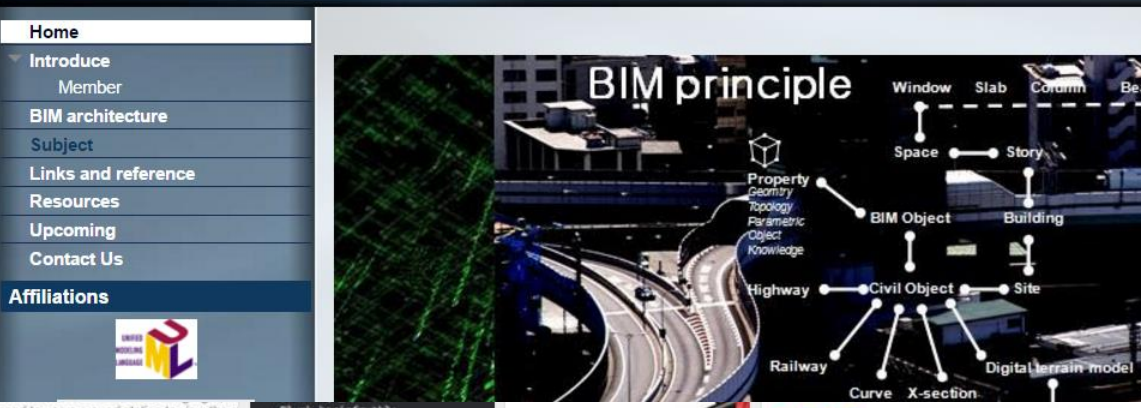
OOAD, UML, Method and Code overview

Kang Taewook. Ph.D
laputa99999@gmail.com
https://www.linkedin.com/in/tae-wook-kang-64a83917

BIM, Facility Management, DX, Scan to BIM
12 books publication

https://github.com/mac999



BIM digest

By noreply@blogger.com (Taewook Kang)

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

View in iTunes

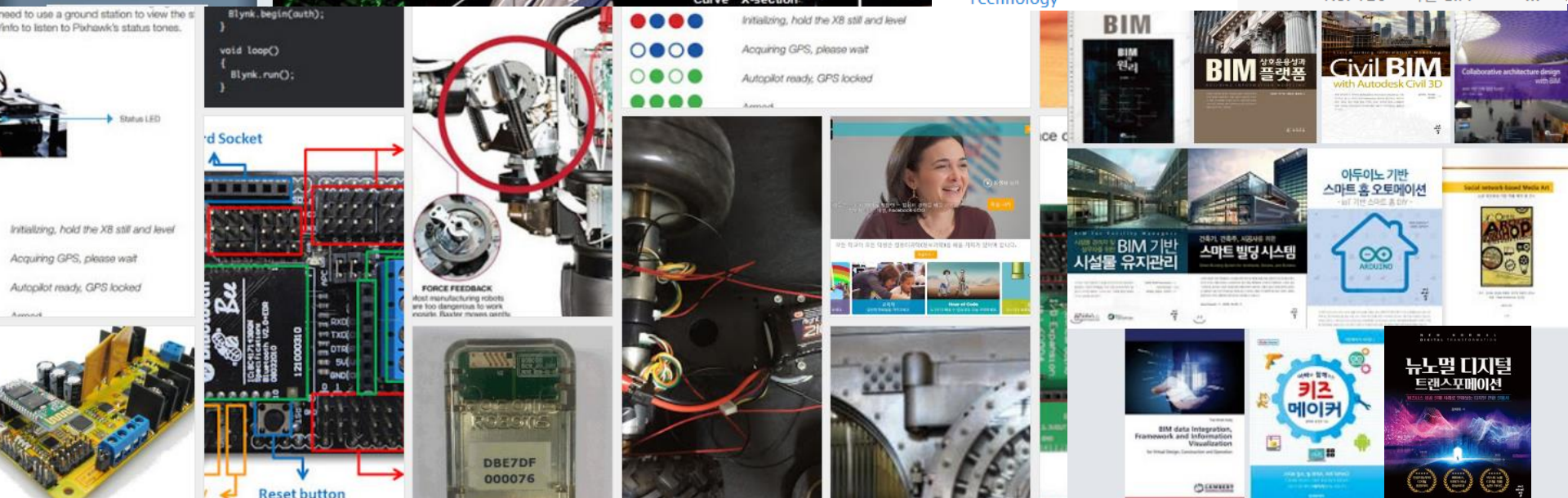
Free

Category: Educational Technology

Description

안녕하세요. Engineering을 쉽게 풀어서 d
하다가 podcast 에 대해 이야기를 나눌 페
Engineering digest는 현재 공학에서 화학
Software, Graphics 그리고 BIM 입니다. 잎
없는 방송을 편하고 유연하게 부담없이^^
기해주시길 바랍니다. 방송내용은 iPod에서
색하실수 있습니다. - Tae Wook, Kang. 강

	Name
1	No.122 – BIM 최신 솔류션 ...
2	No.121 – 건설분야 IoT 기...
3	No. 120 – 최근 BIM 소프트...



Contents

History
OOAD
UML with Design Pattern
Method
Code in Practice

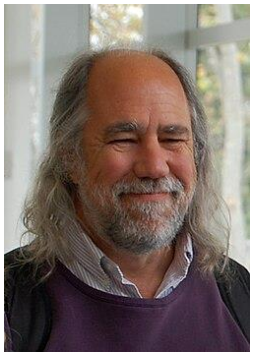
History

OOAD

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

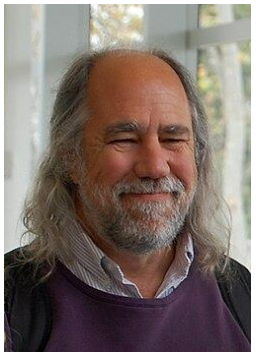
The UML offers a standard way to write a systems blueprints, including conceptual things like business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components."

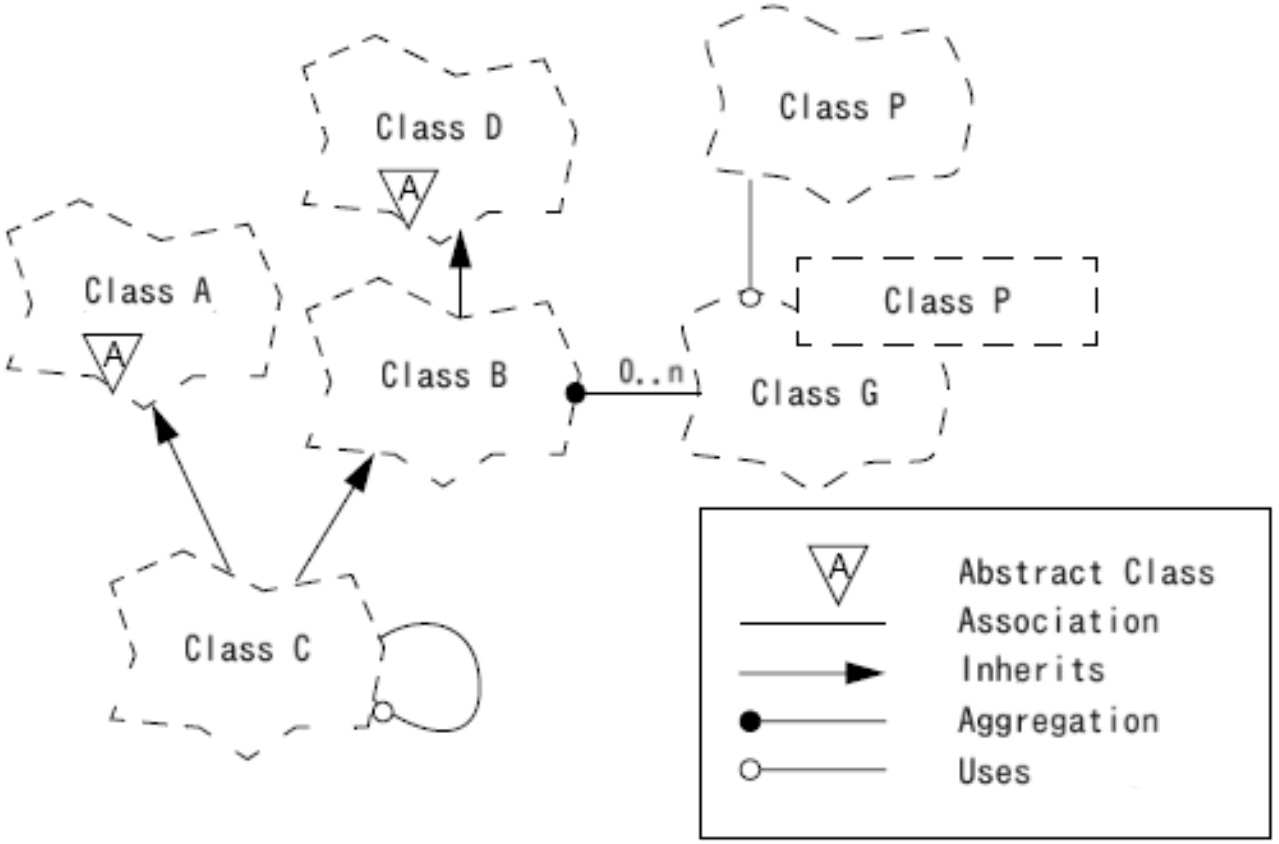
Grady Booch, James Rumbaugh. Ivar Jacobsen, Rational Software Corp
with Erich Gamma




OOAD

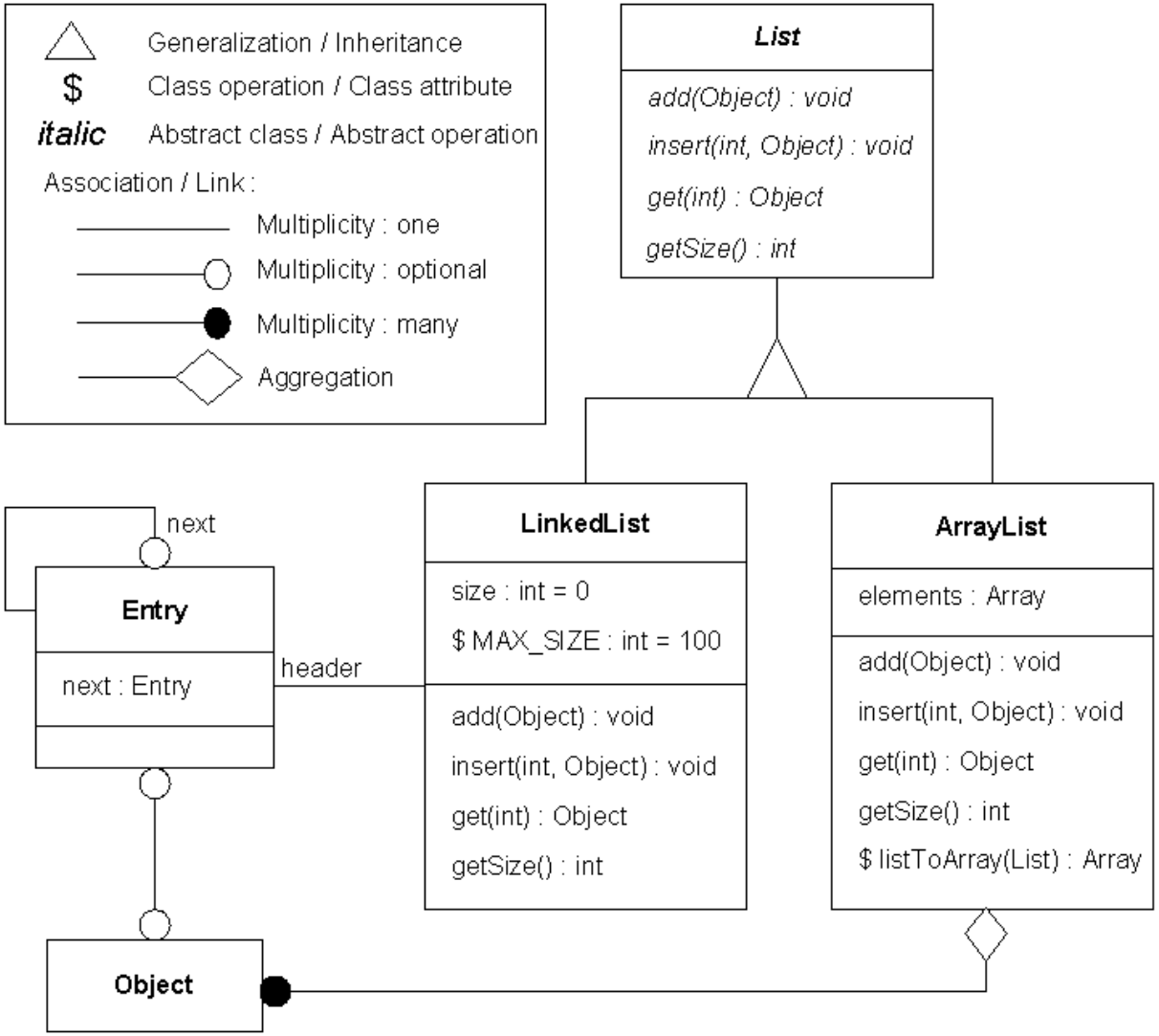
Grady Booch: Booch Method Class Diagram





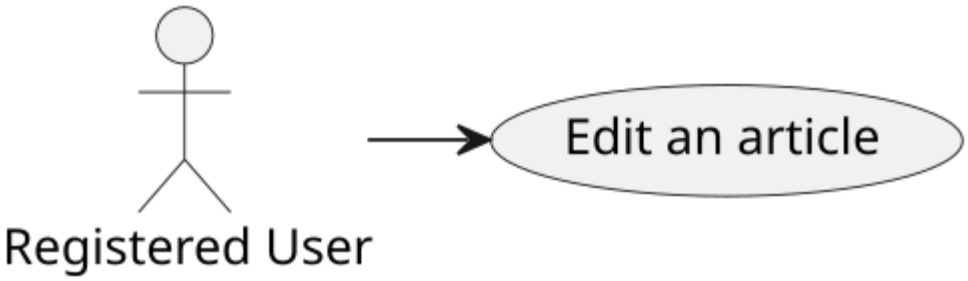
OOAD

James Rumbaugh: OMT



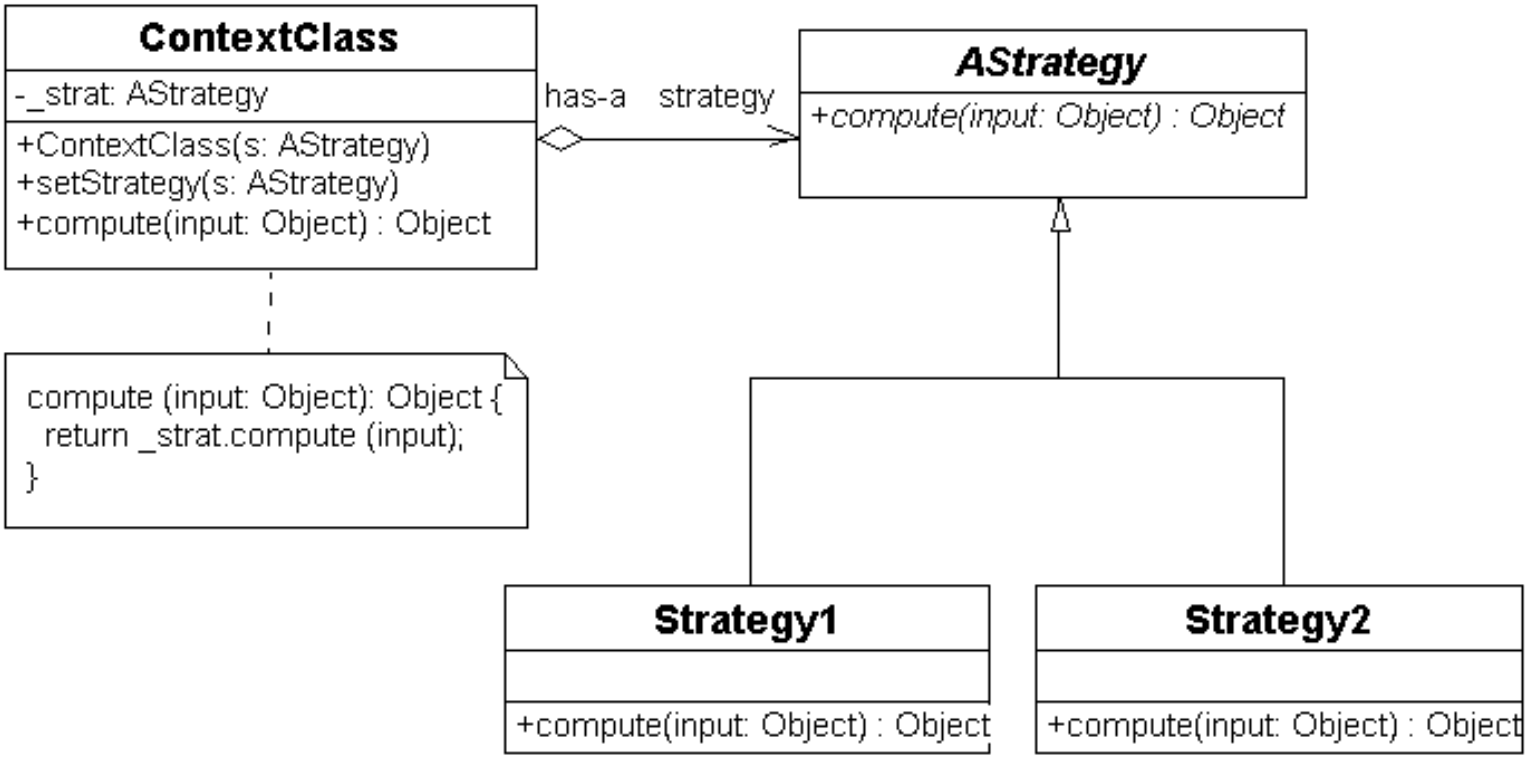
OOAD

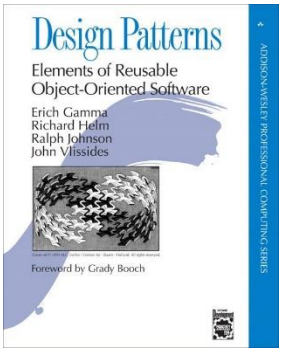
Ivar Jacobsen: Use case



OOAD

Erich Gamma: Design Pattern

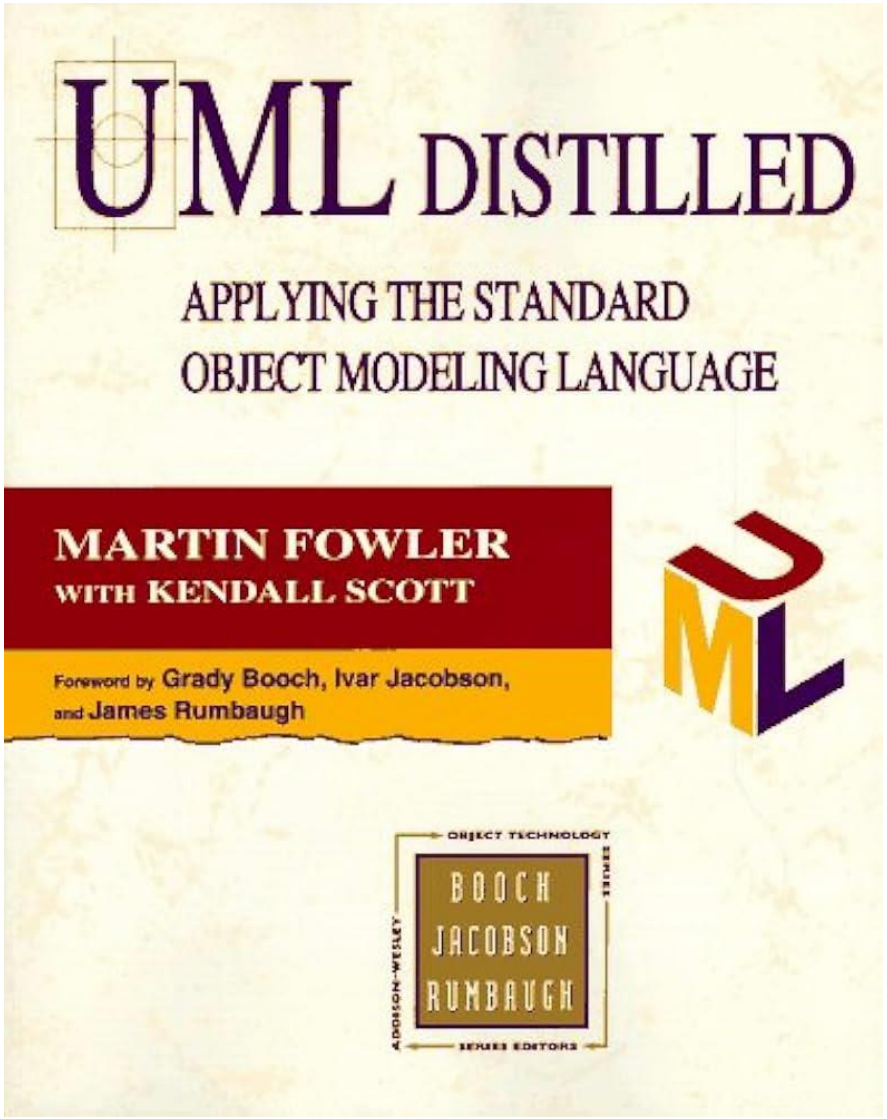




Ralph Johnson, Richard Helm, Erich Gamma, and John Vlissides

OOAD

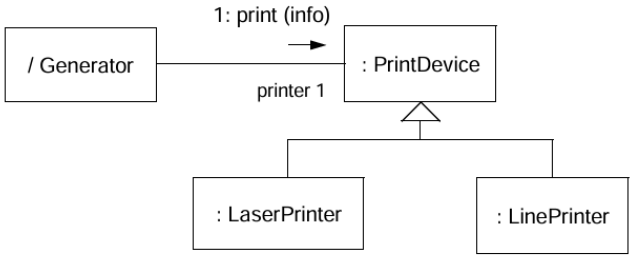
Object Management Group, 1989



Martin Fowler

OOAD

ISO/IEC 19501(2005): UML 1.0



Published
ISO/IEC 19501:2005

A standard is reviewed every 5 years
Stage: **90.93** (Confirmed)

SUSTAINABLE DEVELOPMENT GOALS

This standard contributes to the following Sustainable Development Goal

9

ISO/IEC 19505(2012): UML 2.0

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

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

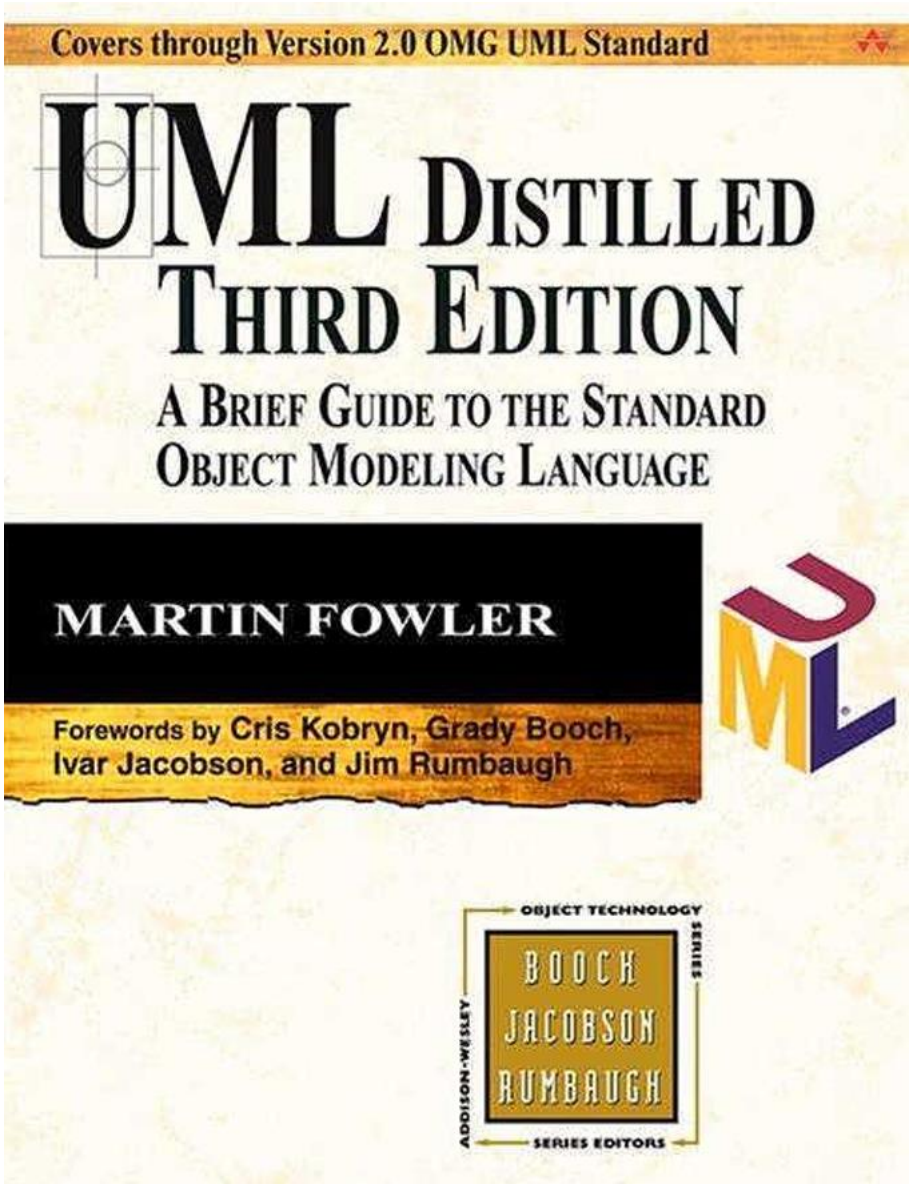
formal/2012-05-06

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

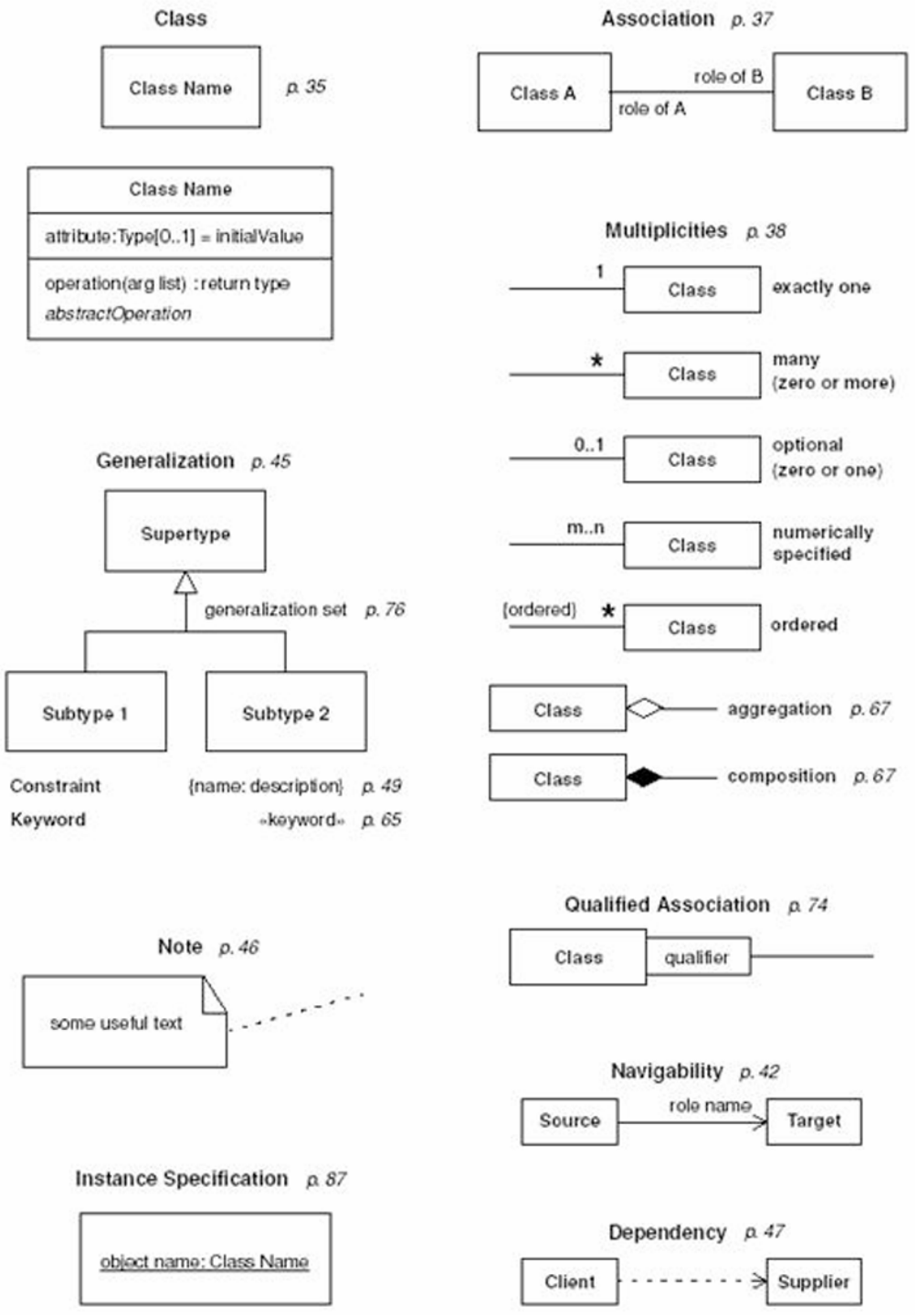
Table of Contents

UML

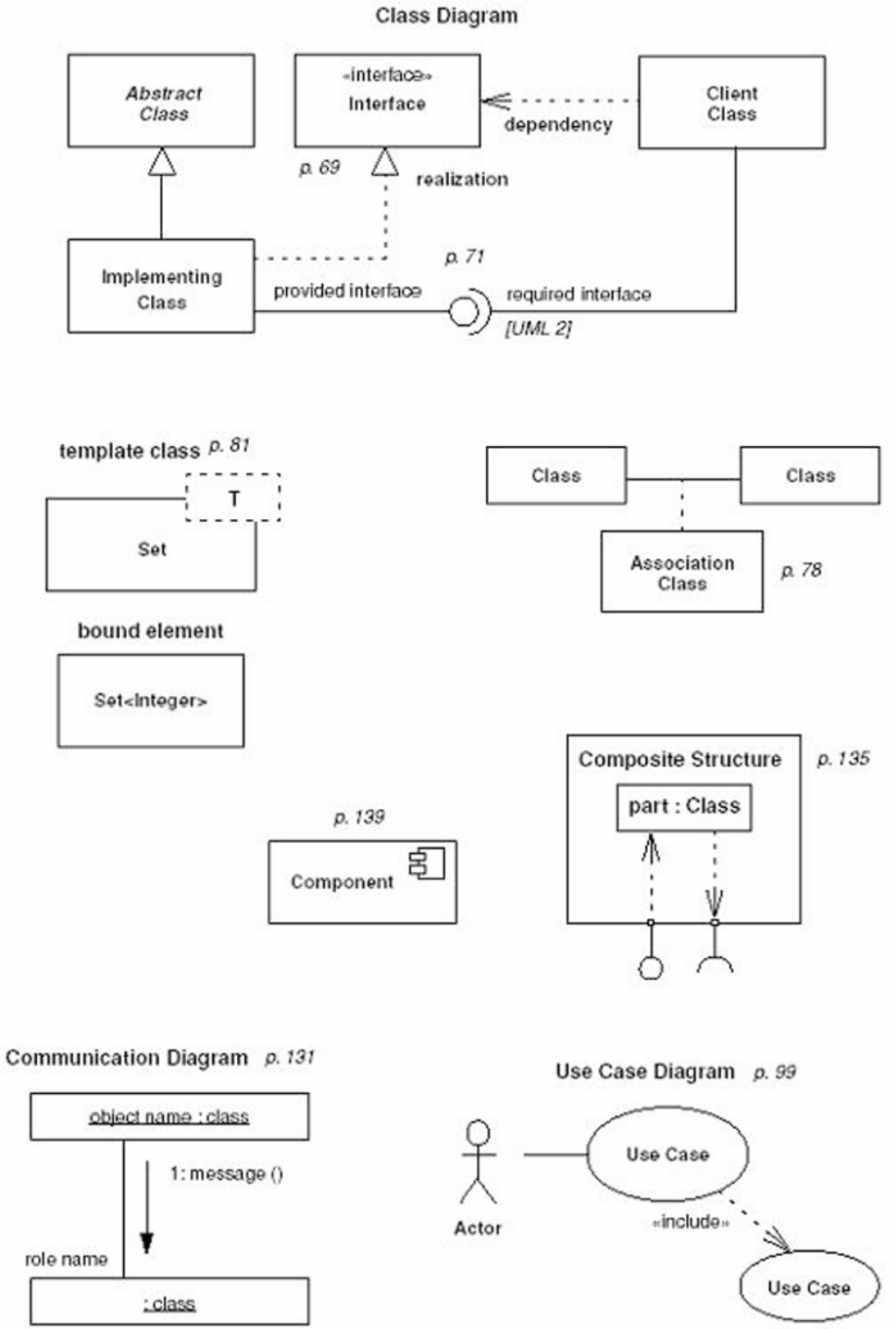
OOAD



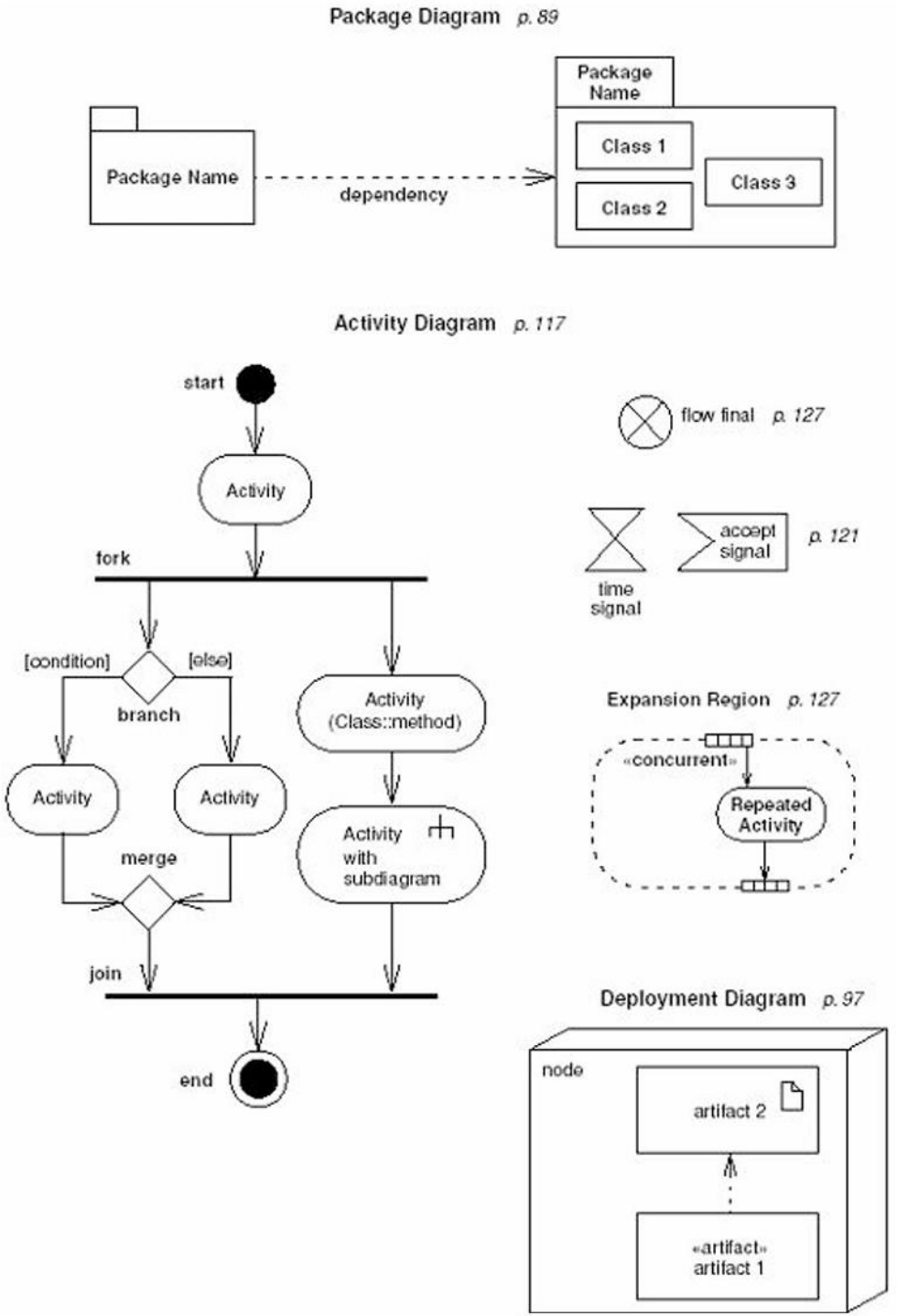
OOAD


Class
Class Name
p. 35
Class Name
attribute:Type[0..1] = initialValue
operation(arg list) :return type
abstractOperation
Generalization p. 45
Supertype
generalization set p. 76
Subtype 1
Subtype 2
Constraint
{name: description} p. 49
Keyword
«keyword» p. 65
Note p. 46
some useful text
Instance Specification p. 87
object name: Class Name
Association p. 37
Class A
role of B
role of A
Class B
Multiplicities p. 38
1
Class
exactly one
*
Class
many
(zero or more)
0..1
Class
optional
(zero or one)
m..n
Class
numerically
specified
{ordered}
*
Class
ordered
Class
aggregation p. 67
Class
composition p. 67
Qualified Association p. 74
Class
qualifier
Navigability p. 42
Source
role name
Target
Dependency p. 47
Client
Supplier

OOAD

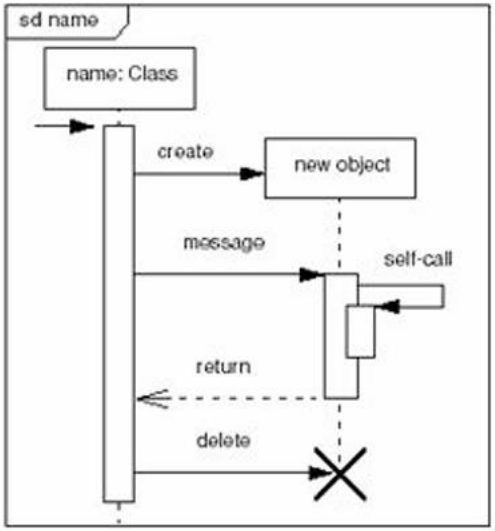

Class Diagram
Abstract Class
«interface» Interface
Client Class
dependency
p. 69
realization
Implementing Class
p. 71
provided interface
required interface
[UML 2]
template class p. 81
T
Set
bound element
Set<Integer>
Class
Class
Association Class
p. 78
p. 139
Component
Composite Structure
p. 135
part : Class
Communication Diagram p. 131
object name : class
1: message ()
role name
: class
Use Case Diagram p. 99
Actor
Use Case
«include»
Use Case

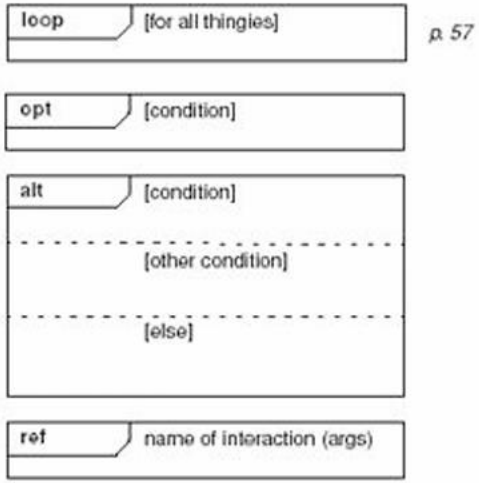
OOAD


Package Diagram p. 89
Package Name
Class 1
Class 3
Class 2
Package Name
dependency
Activity Diagram p. 117
start
Activity
fork
[condition]
[else]
branch
Activity
Activity
merge
Activity
(Class::method)
Activity
with
subdiagram
join
end
flow final p. 127
accept
signal
p. 121
time
signal
Expansion Region p. 127
«concurrent»
Repeated
Activity
Deployment Diagram p. 97
node
artifact 2
«artifact»
artifact 1

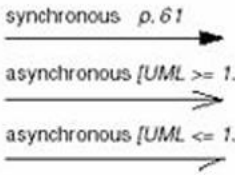
OOAD

Sequence Diagram *p. 53*





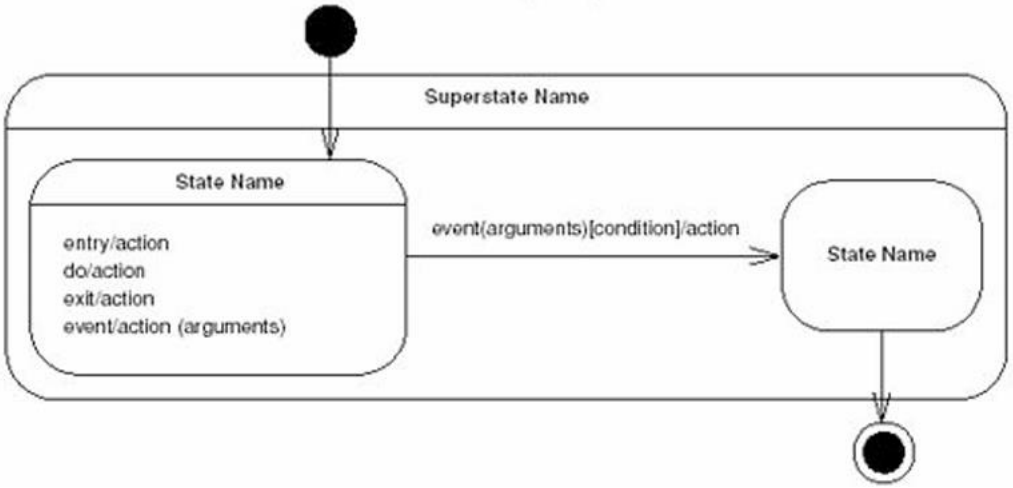
p. 57



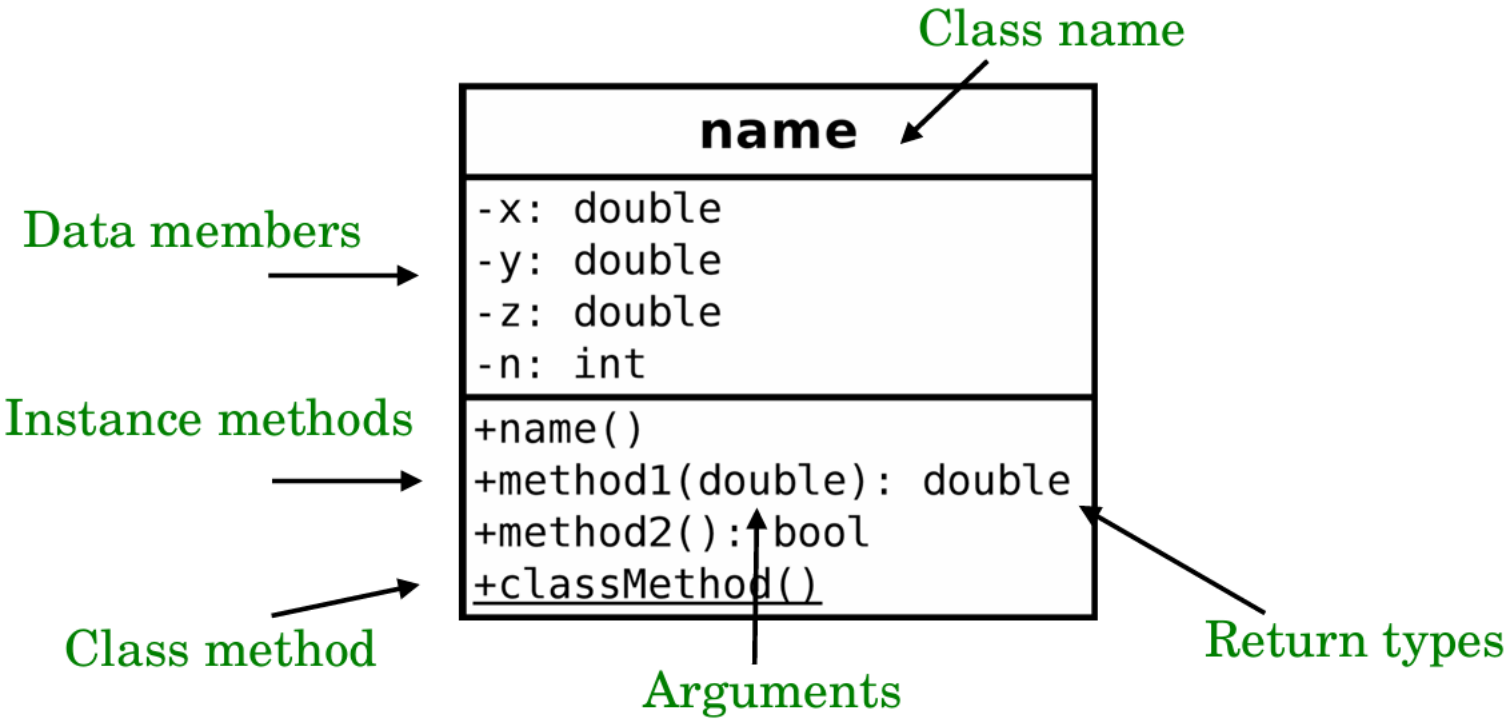
*: iteration message ()
[condition] message ()

[UML 1] p. 59

State Diagram *p. 107*



OOAD



Data members, arguments and methods are specified by visibility name : type

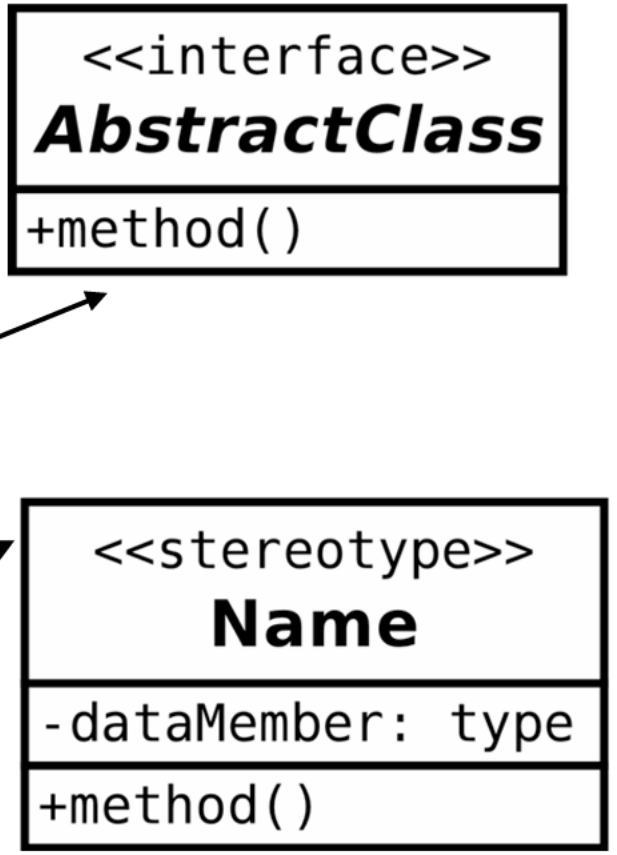
OOAD

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

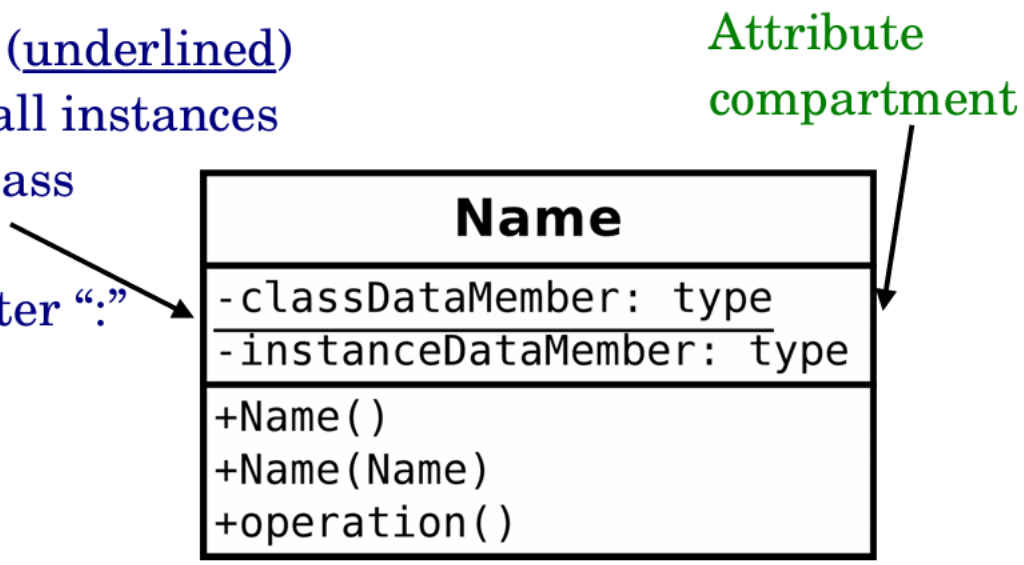
Attributes are the instance and class data members

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

Data types shown after ":"

Visibility shown as

- \+ public
- \- private
- \# protected



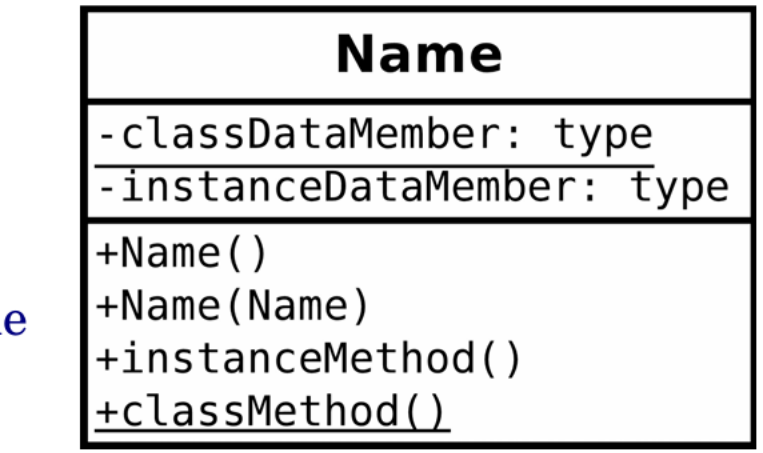
visibility name : type

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)



Operations compartment

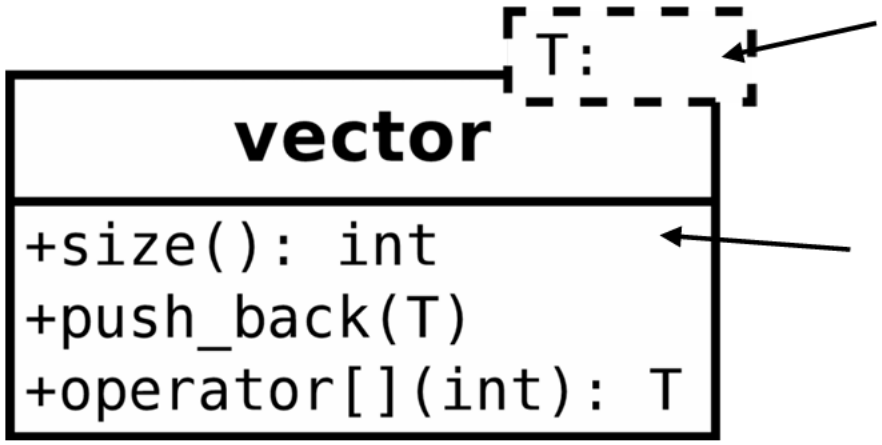
visibility name : type

OOAD

+	-	#
public	private	protected
Anyone can access	No-one can access	Subclasses can access
Interface operations	Data members	Operations where sub-classes collaborate
Not data members	Helper functions	Not data members (creates dependency of subclass on im-plementation of parent)
	"Friends" are allowd in though	

OOAD

Generic classes depending on parametrised types



Type parameter(s)

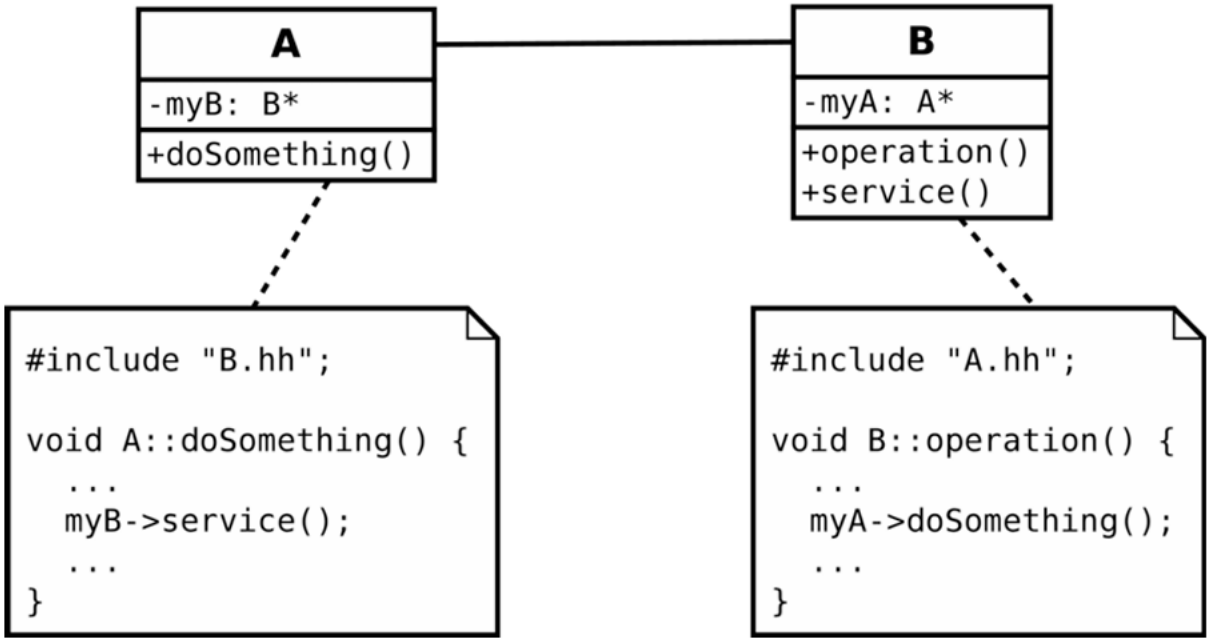
Operations compartment as usual, but may have type parameter instead of concrete type

OOAD

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

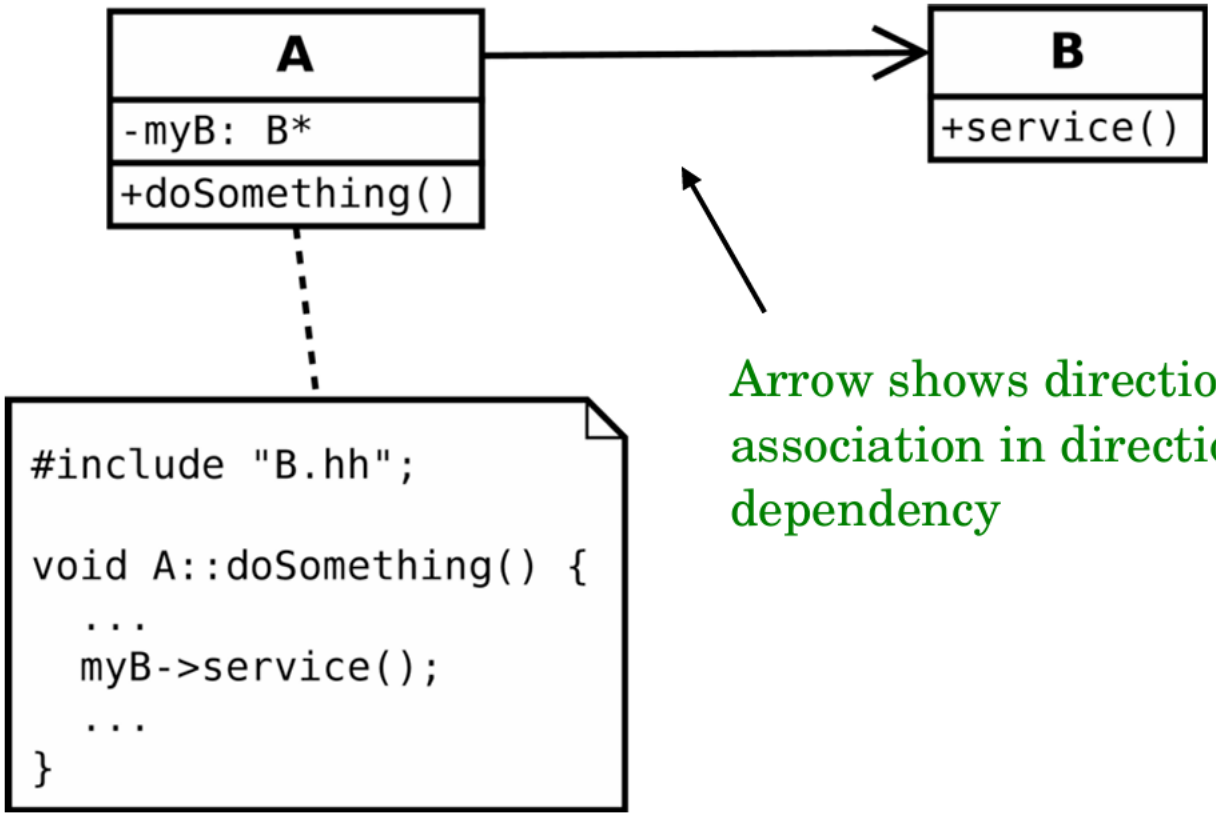
OOAD

Binary association: both classes know each other



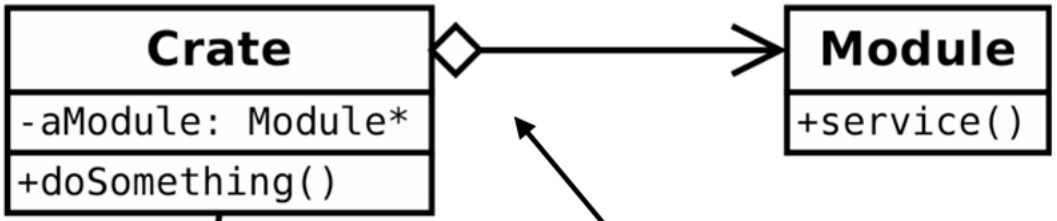
OOAD

A knows about B, but B knows nothing about A



OOAD

Aggregation = Association with “whole-part” relationship



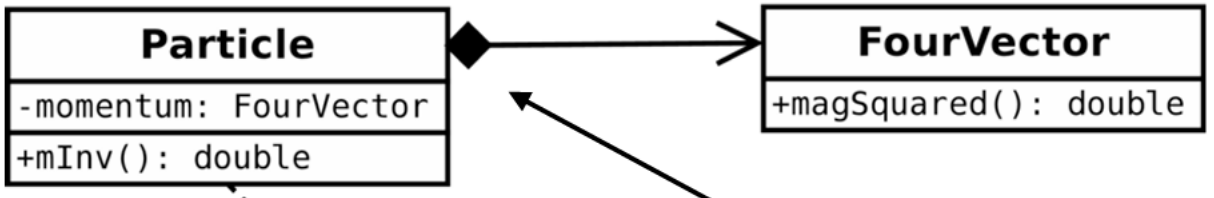

```
#include "Module.hh";

void Crate::doSomething() {
  ...
  aModule->service();
  ...
}
```

Shown by hollow diamond at the "whole" side

No lifetime control implied

OOAD

Composition = Aggregation with lifetime control




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

Shown by filled diamond at the "owner" side

Lifetime control implied

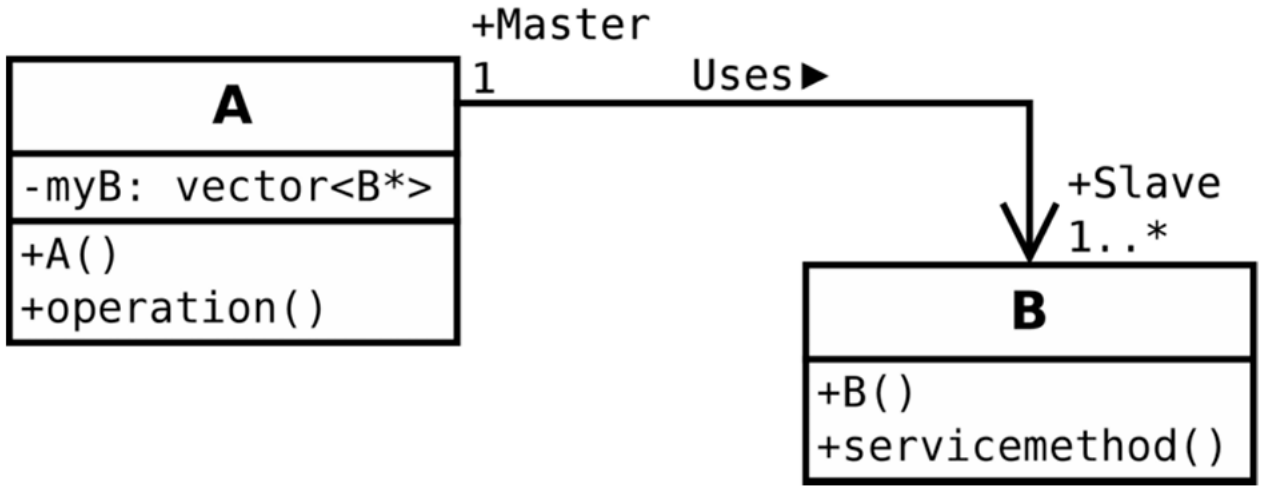
Lifetime control can be tranferred

Lifetime control: construction and destruction controlled by "owner"

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

OOAD

Name gives details of association
Name can be viewed as verb of a sentence



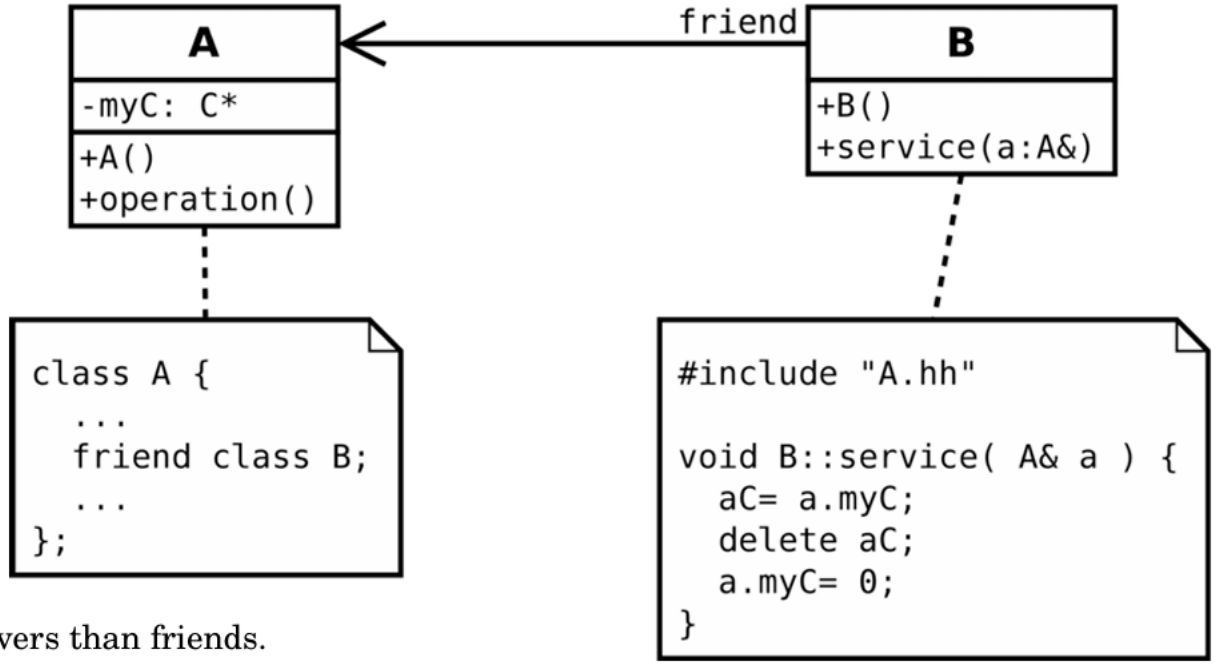
Notes at association ends explain "roles" of classes (objects)

Multiplicities show number of objects which participate in the association

OOAD

Friends are granted access to private data members and member functions

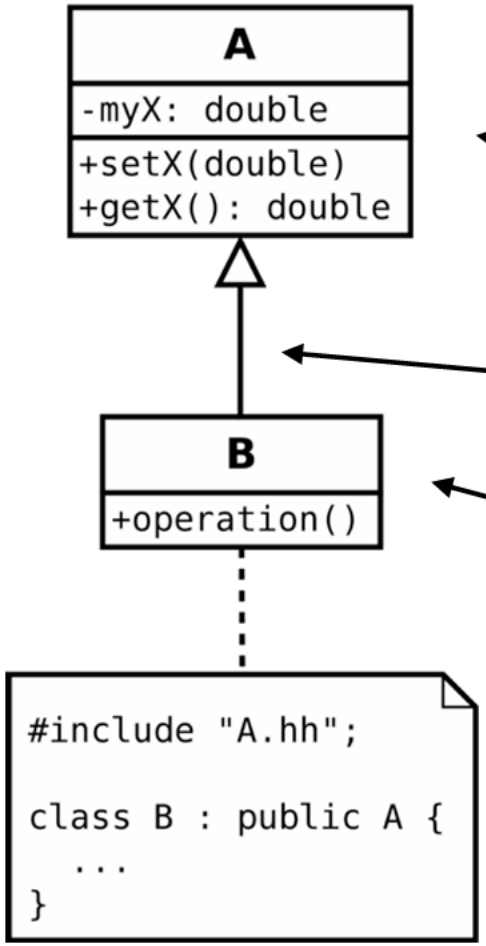
Friendship is given to other classes, never taken



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



Base class or super class

Arrow shows direction of dependency

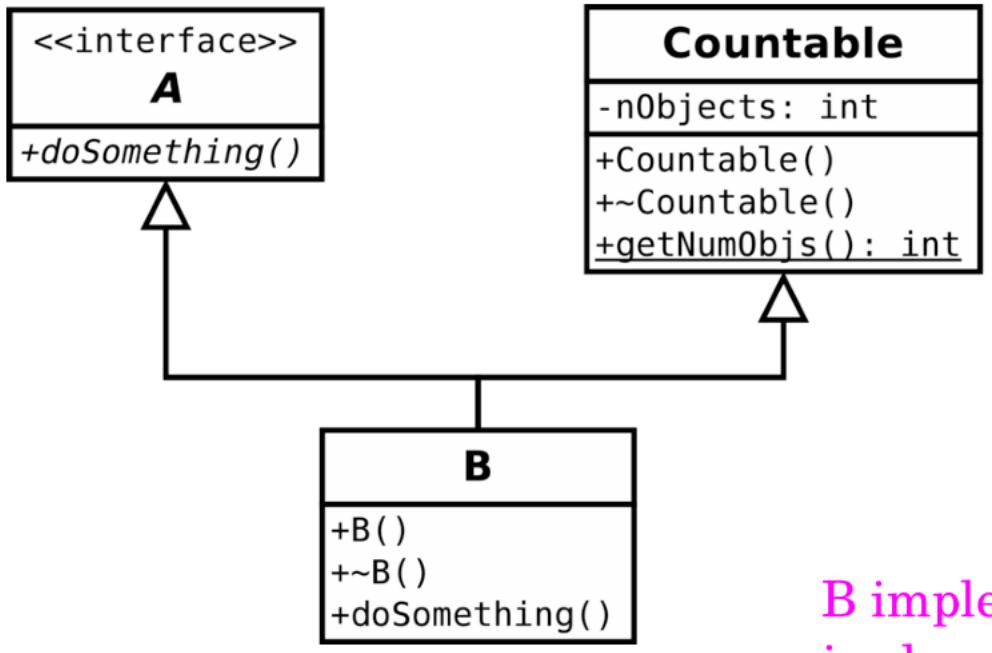
Derived class or subclass

→ B inherits A's interface, behaviour and data members

→ B can extend A, i.e. add new data members or member functions

→ B depends on A, A knows nothing about B

OOAD



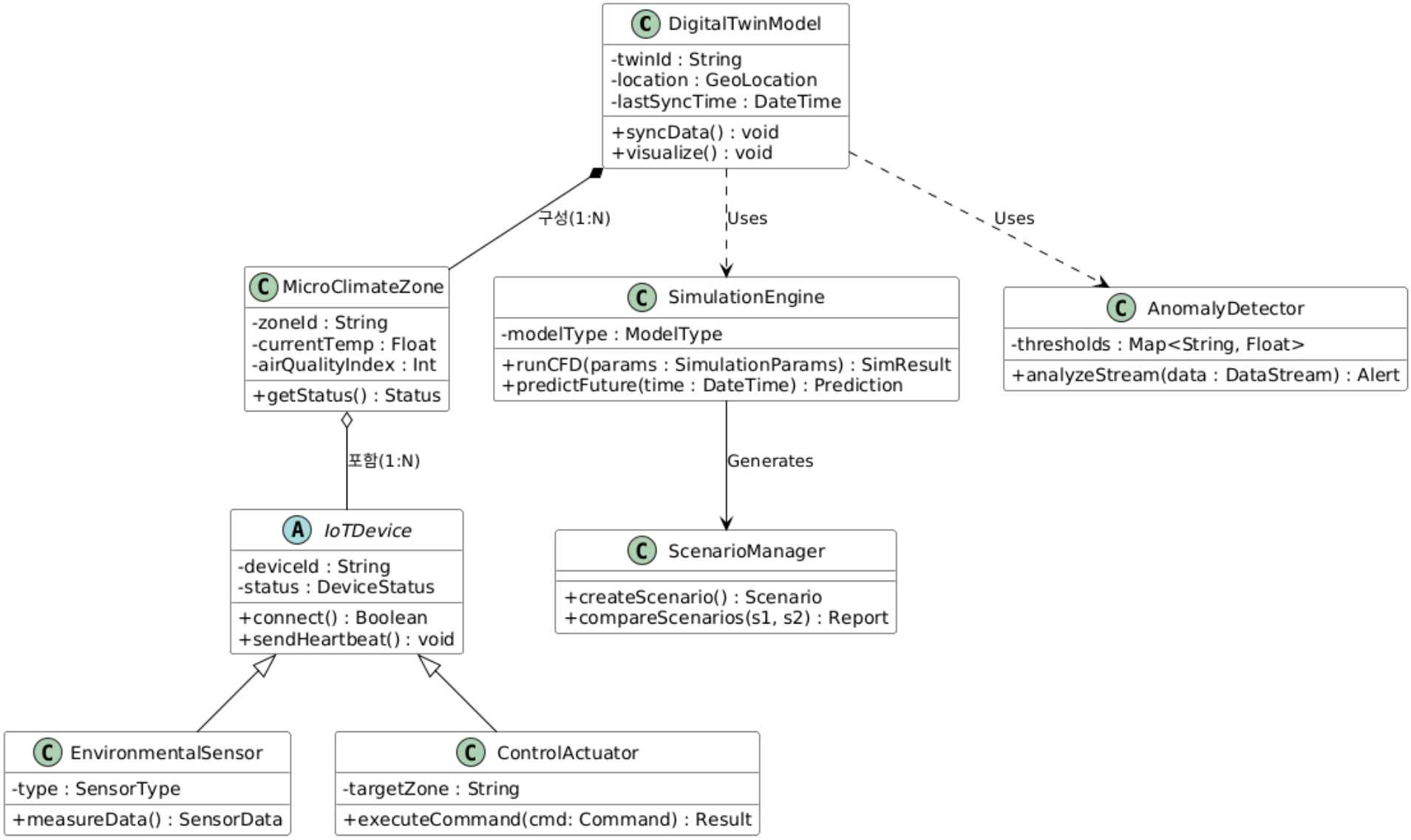
The derived class inherits interface, behaviour and data members of all its base classes

Extension and overriding works as before

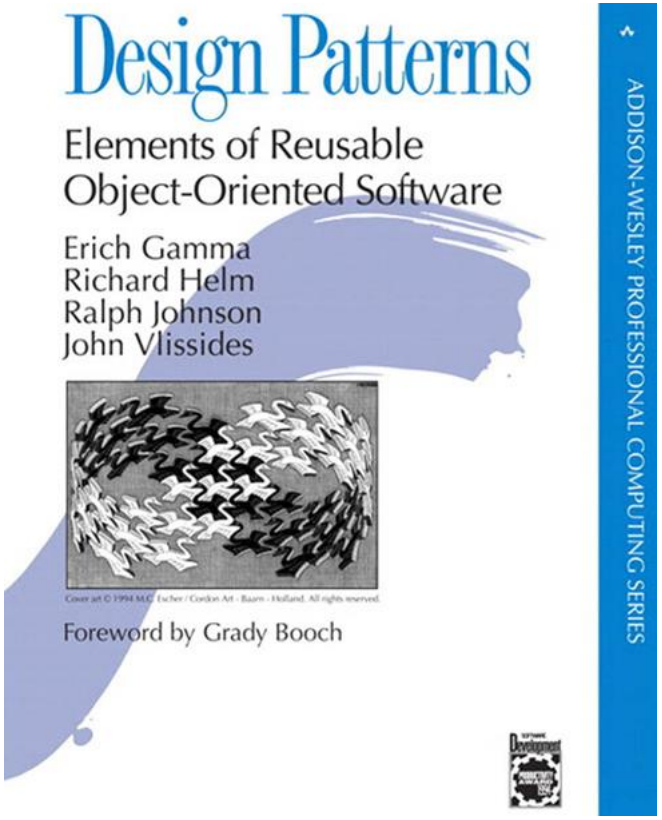
B implements the interface A and is also a “countable” class

Countable also called a ”Mixin class”

OOAD


DigitalTwinModel
-twinId : String
-location : GeoLocation
-lastSyncTime : DateTime
+syncData() : void
+visualize() : void
구성(1:N)
Uses
Uses
MicroClimateZone
-zoneId : String
-currentTemp : Float
-airQualityIndex : Int
+getStatus() : Status
SimulationEngine
-modelType : ModelType
+runCFD(params : SimulationParams) : SimResult
+predictFuture(time : DateTime) : Prediction
AnomalyDetector
-thresholds : Map<String, Float>
+analyzeStream(data : DataStream) : Alert
포함(1:N)
Generates
IoTDevice
-deviceId : String
-status : DeviceStatus
+connect() : Boolean
+sendHeartbeat() : void
ScenarioManager
+createScenario() : Scenario
+compareScenarios(s1, s2) : Report
EnvironmentalSensor
-type : SensorType
+measureData() : SensorData
ControlActuator
-targetZone : String
+executeCommand(cmd: Command) : Result

Design Pattern



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.

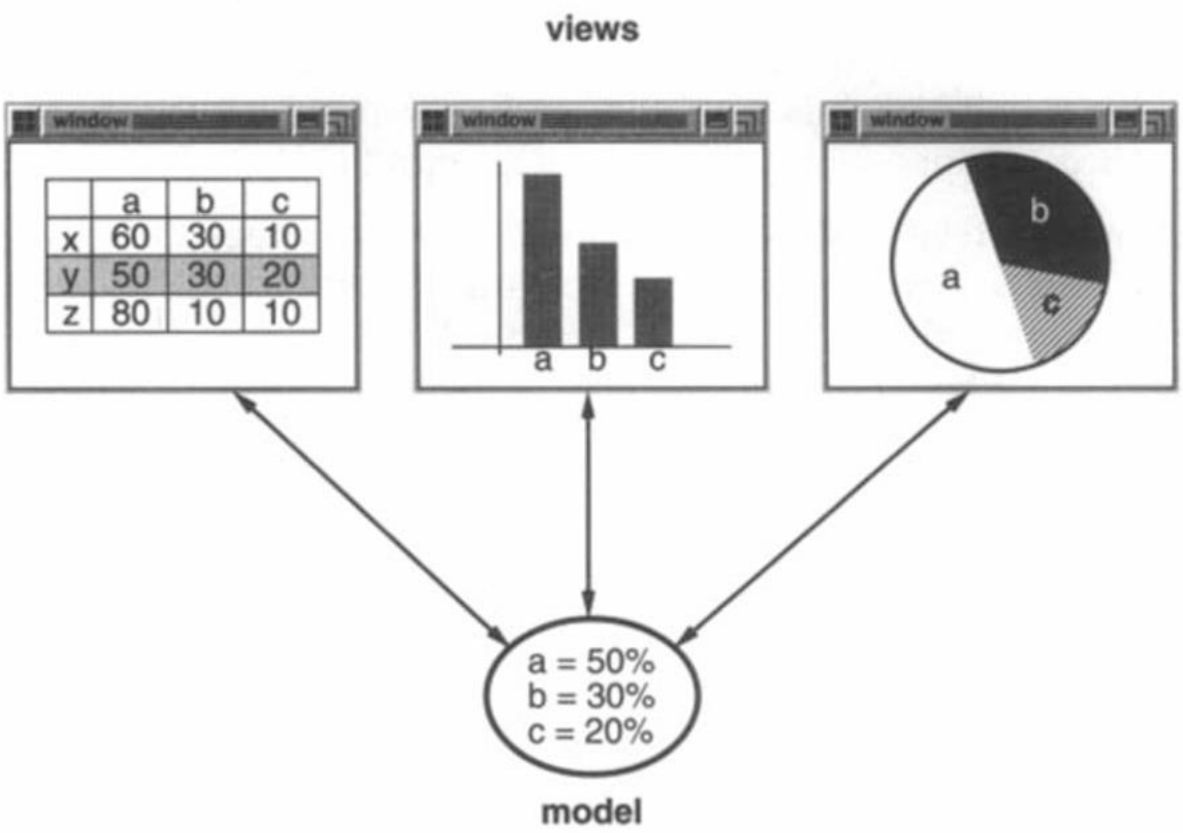
Decorator (175) Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

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.

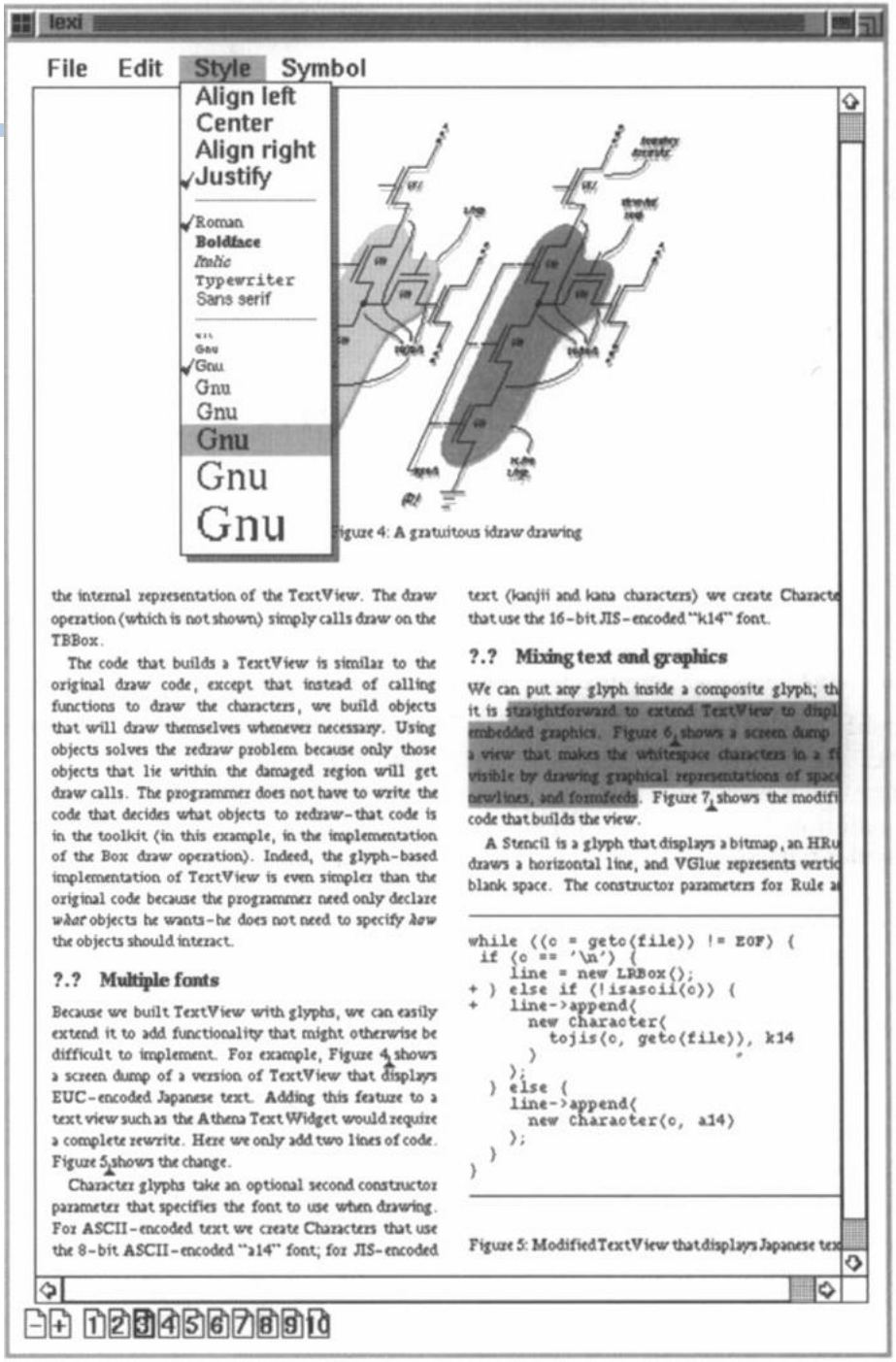
Flyweight (195) Use sharing to support large numbers of fine-grained objects efficiently.

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

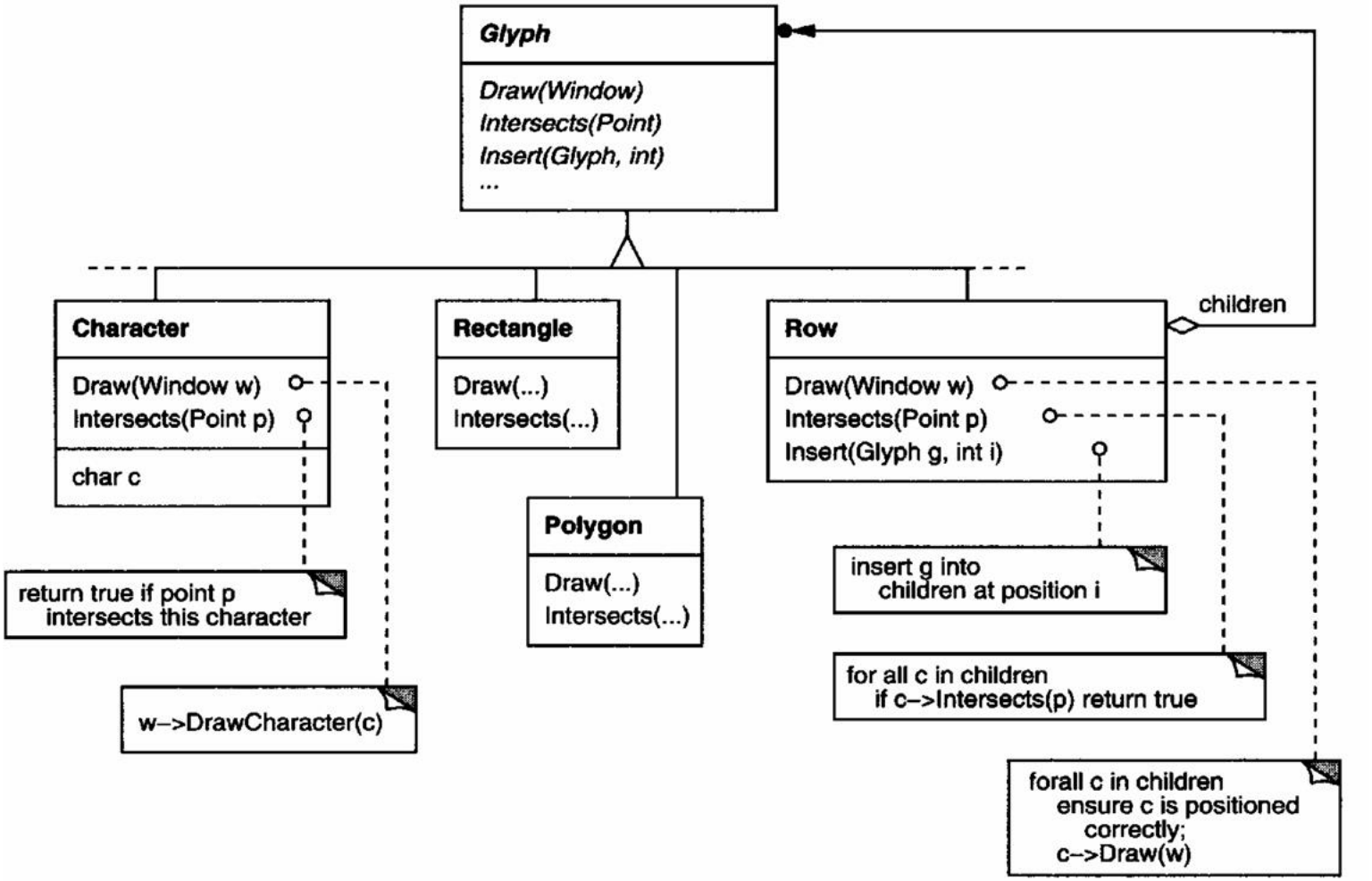
Design Pattern



Design Pattern

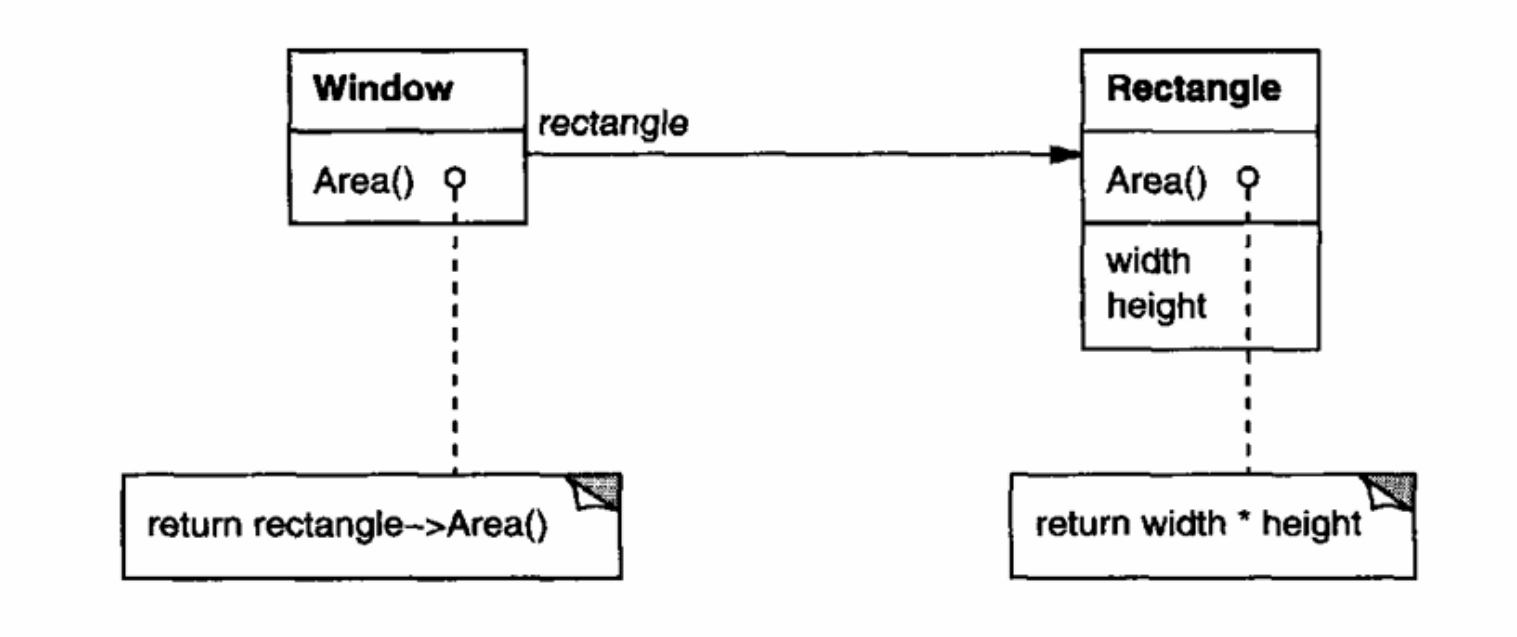


Design Pattern



Design Pattern

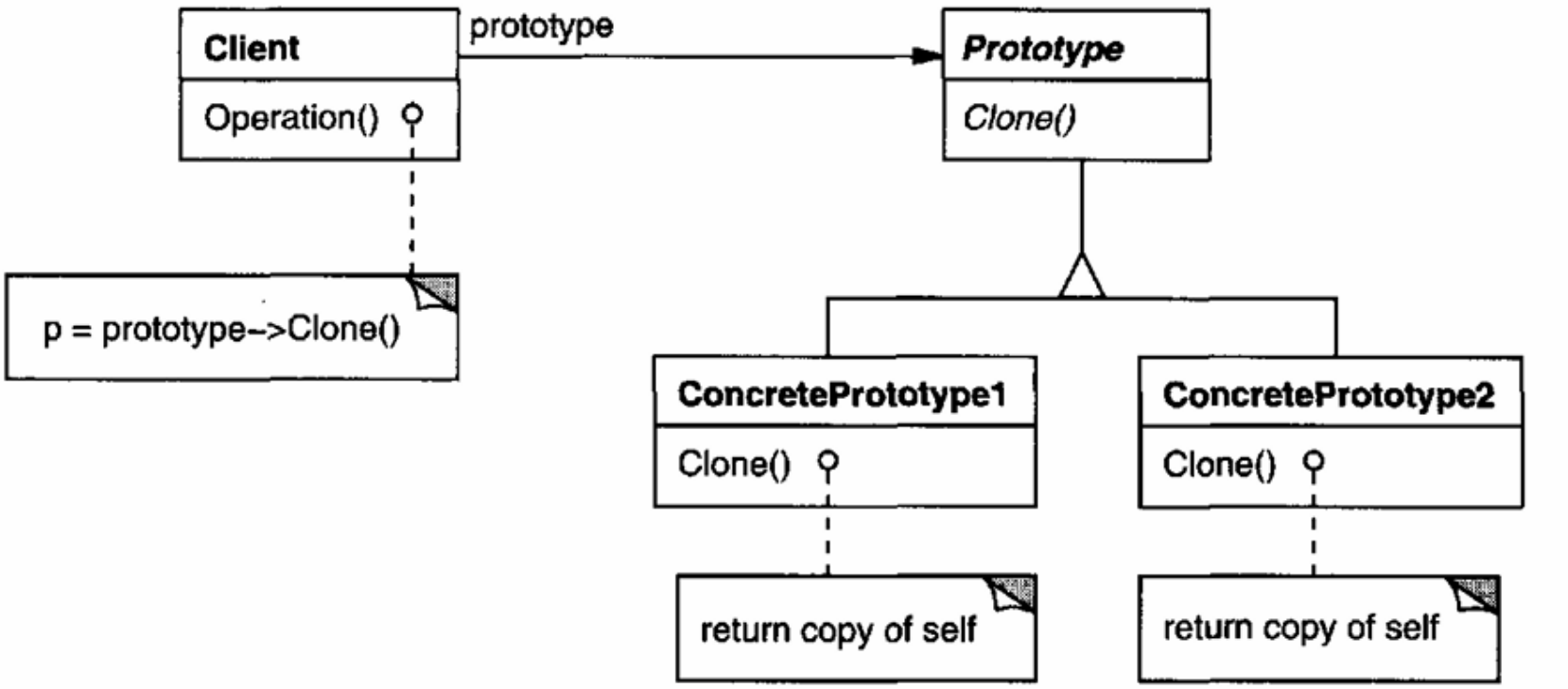
Delegation



Design Pattern

Prototype

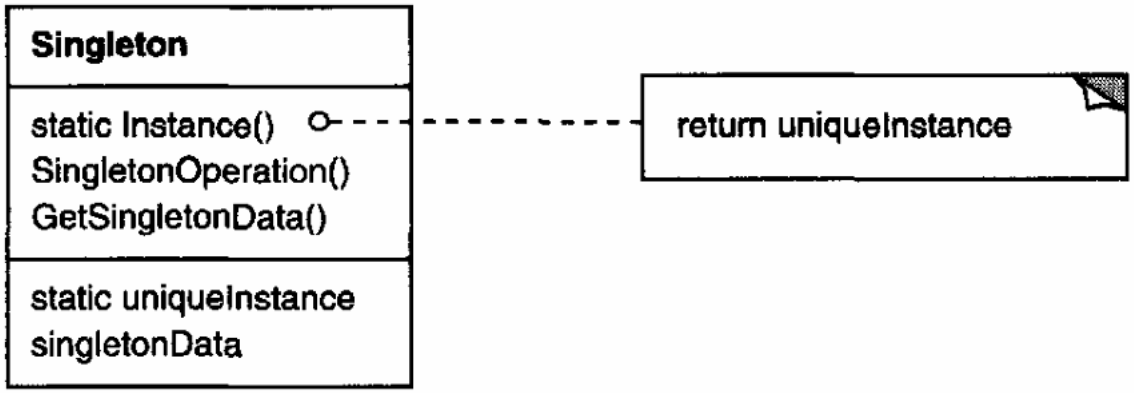
Specify the kinds of objects to createusing a prototypical instance, and create new objects by copying this prototype



Design Pattern

Singleton

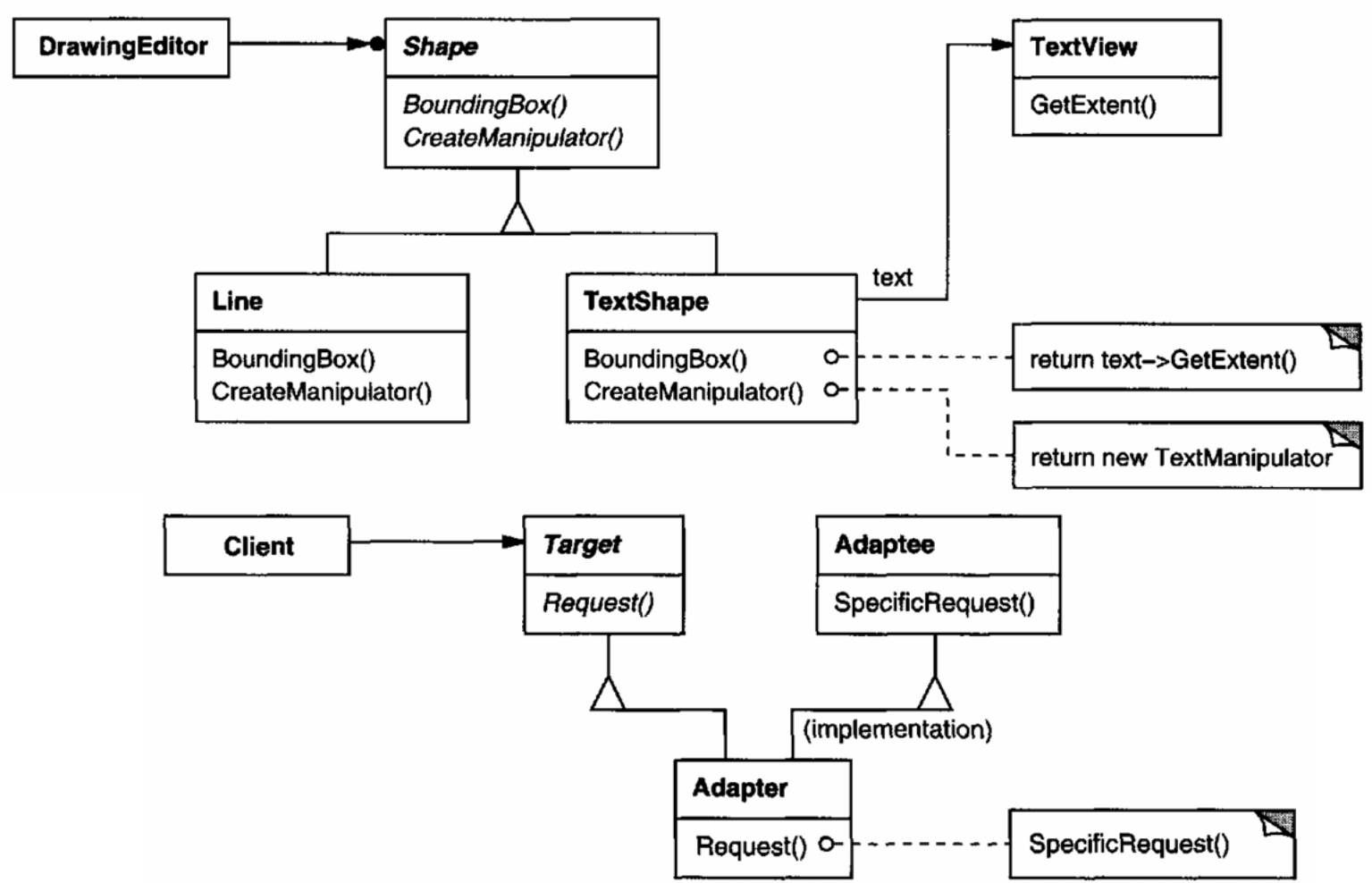
Ensure a class only has one instance, and provide a global point of access to it



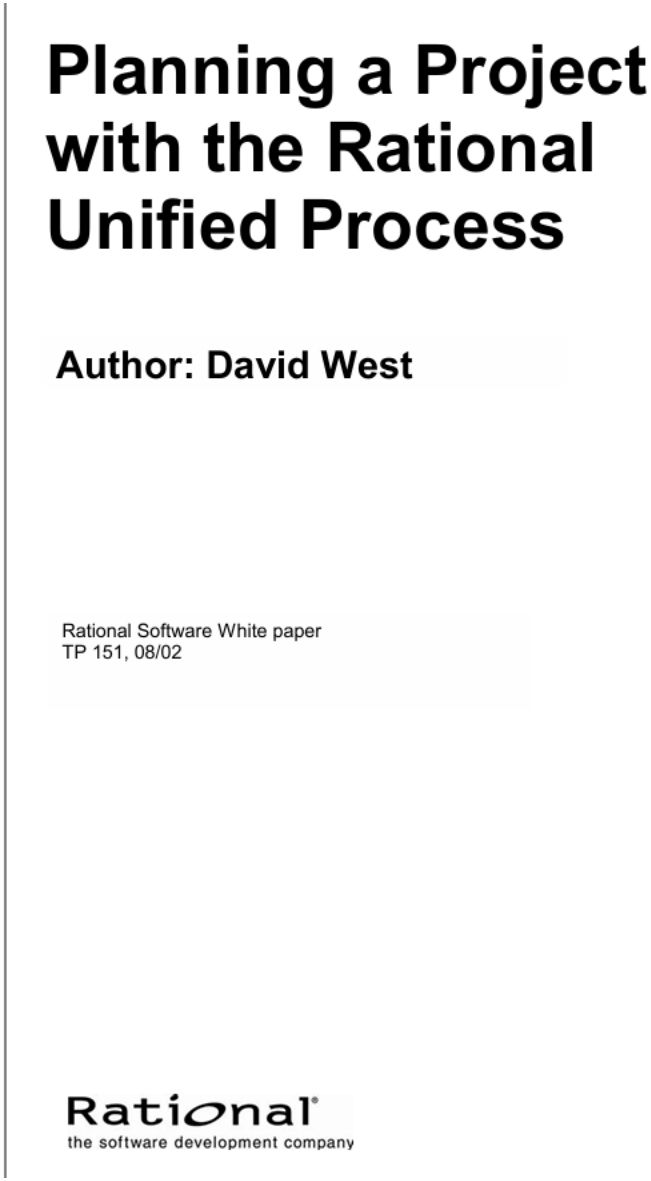
Design Pattern

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.



Method



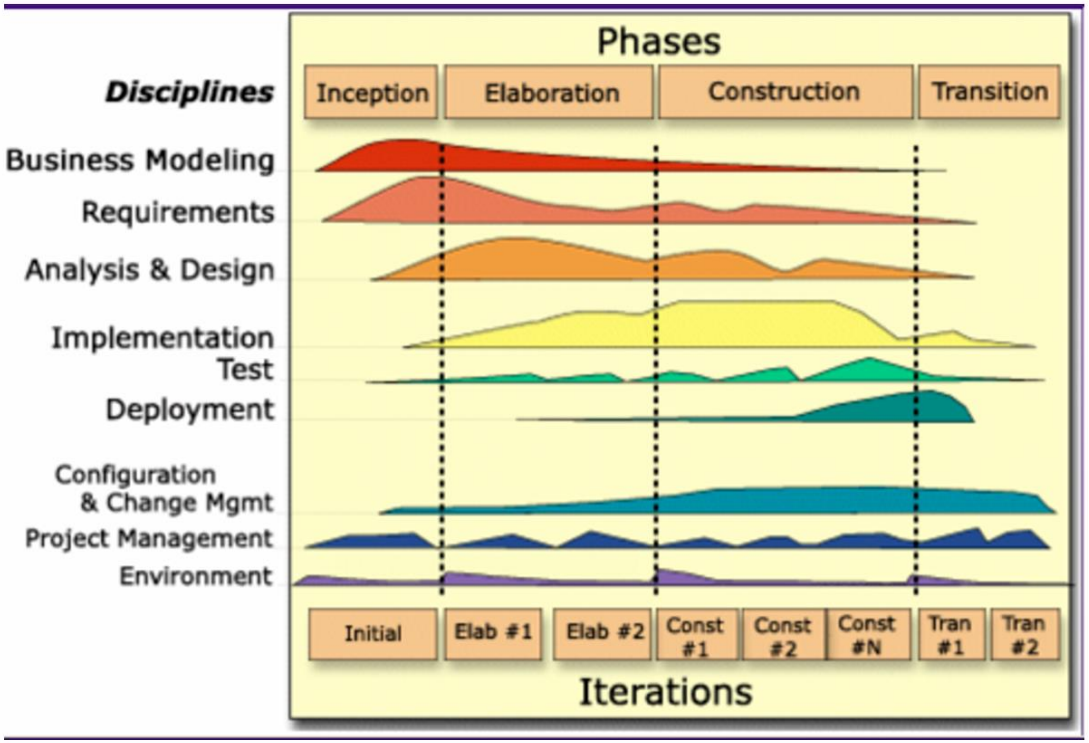
Planning a Project with the Rational Unified Process

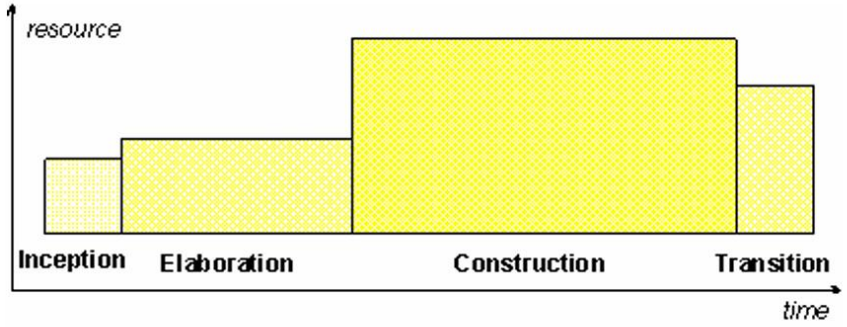
Author: David West

Rational Software White paper
TP 151, 08/02

Rational®
the software development company

Method

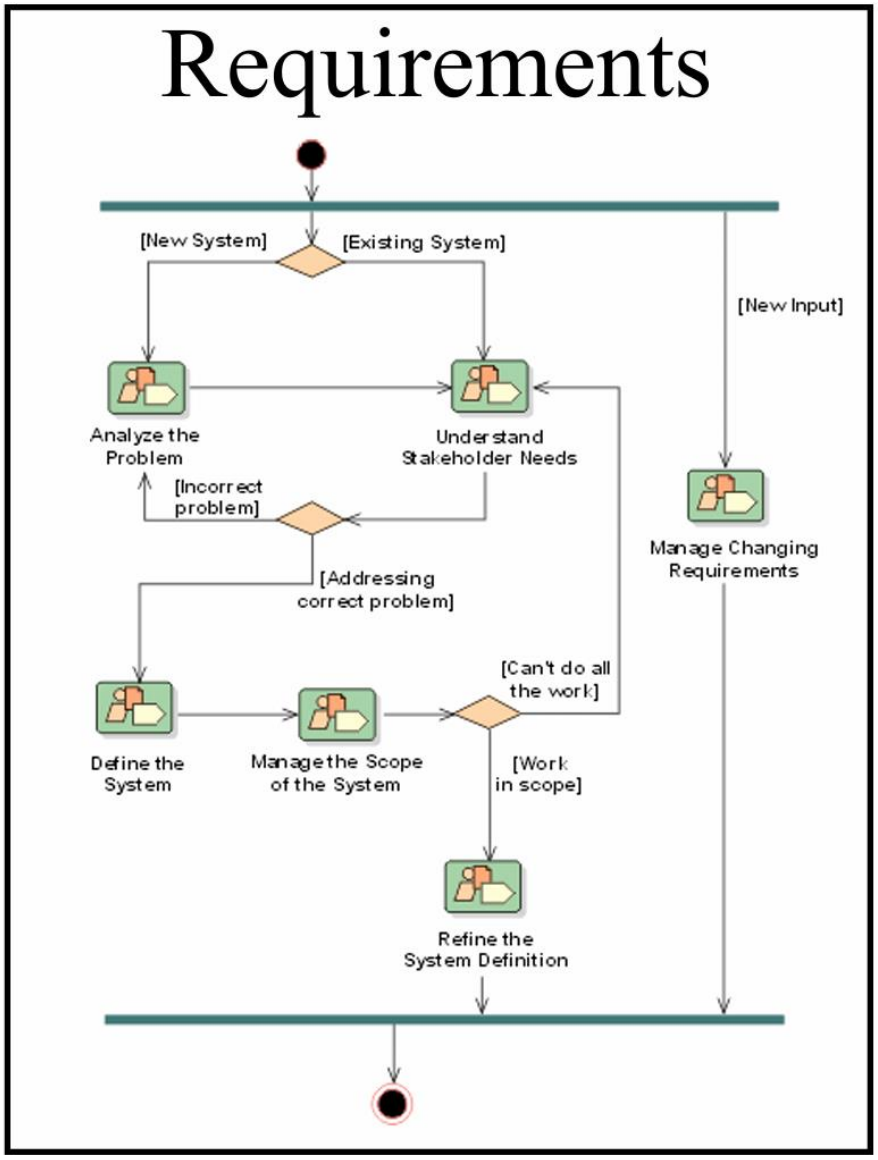


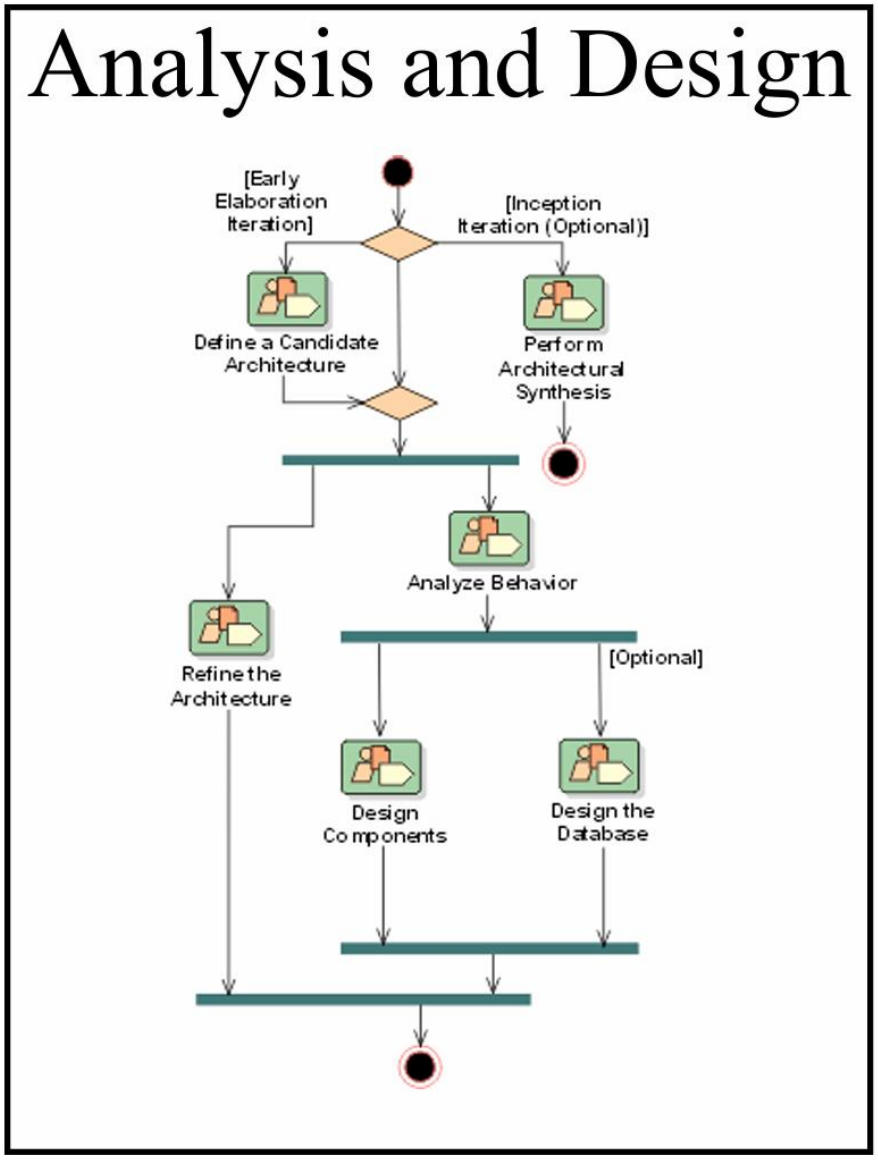


Effort	~5%	20%	65%	10%
Schedule	10%	30%	50%	10%

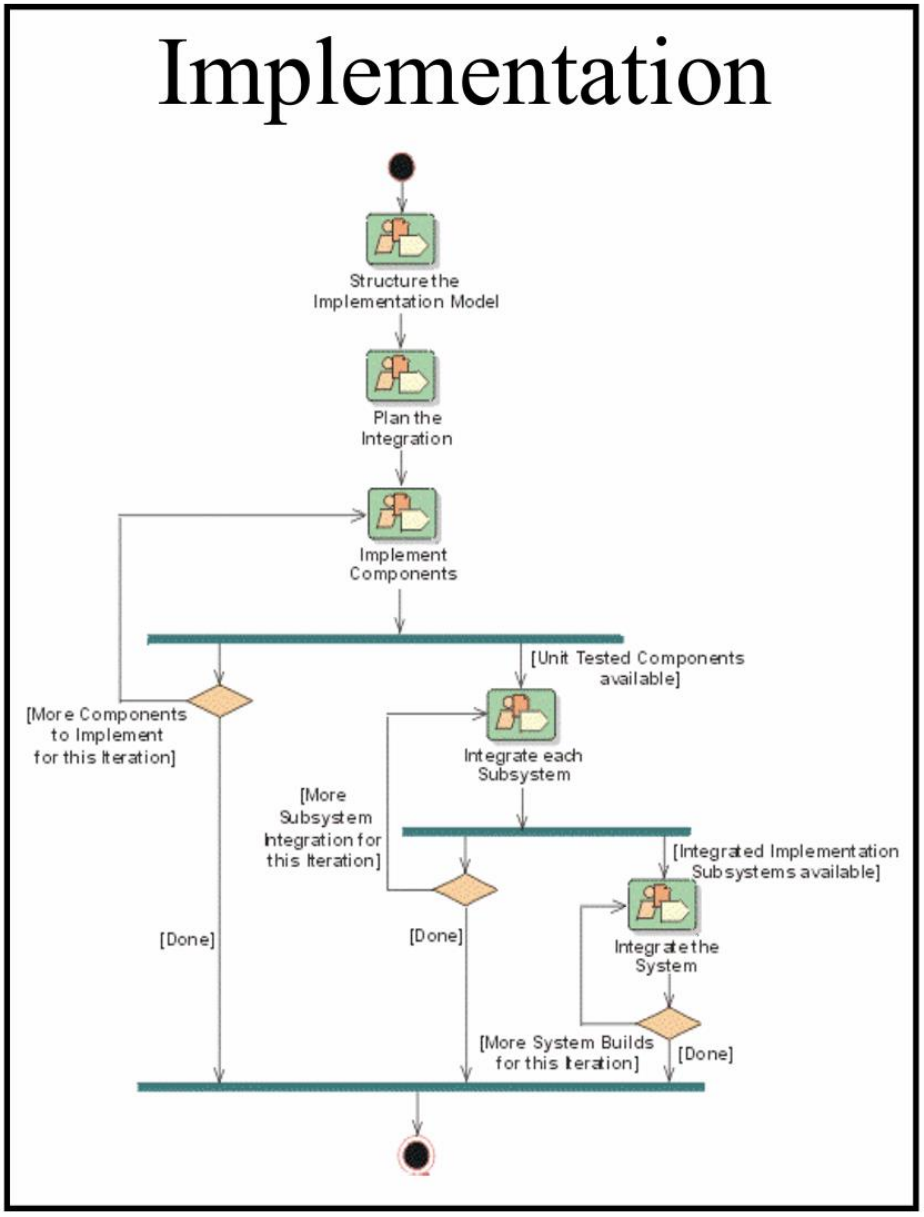
Figure 1: Typical time allocation for the four phases of a project

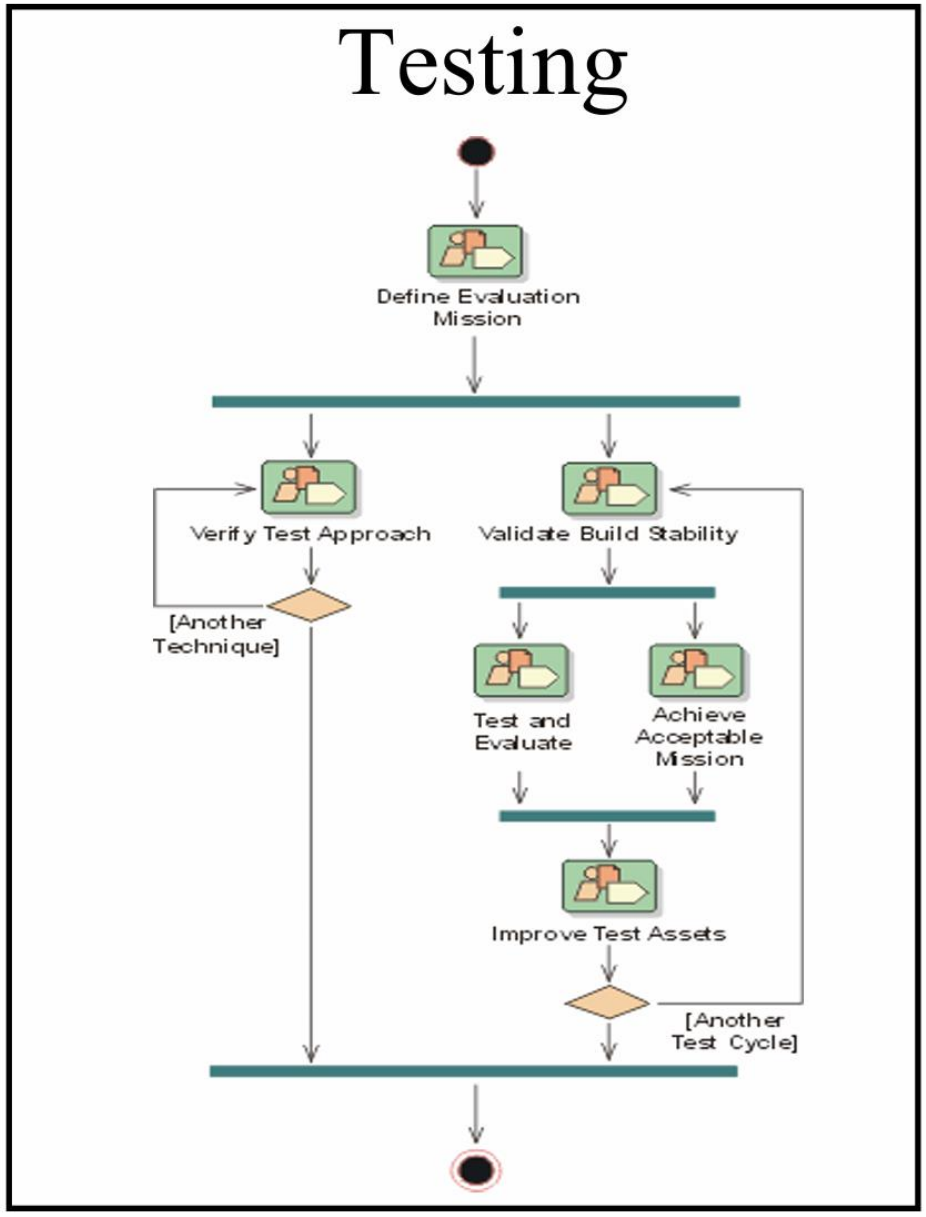
Method


Requirements
[New System]
[Existing System]
[New Input]
Analyze the Problem
Understand Stakeholder Needs
[Incorrect problem]
Manage Changing Requirements
[Addressing correct problem]
[Can't do all the work]
Define the System
Manage the Scope of the System
[Work in scope]
Refine the System Definition


Analysis and Design
[Early Elaboration Iteration]
[Inception Iteration (Optional)]
Define a Candidate Architecture
Perform Architectural Synthesis
Analyze Behavior
Refine the Architecture
[Optional]
Design Components
Design the Database

Method


Implementation
Structure the Implementation Model
Plan the Integration
Implement Components
[More Components to Implement for this Iteration]
[Unit Tested Components available]
Integrate each Subsystem
[More Subsystem Integration for this Iteration]
[Integrated Implementation Subsystems available]
[Done]
[Done]
Integrate the System
[More System Builds for this Iteration]
[Done]


Testing
Define Evaluation Mission
Verify Test Approach
Validate Build Stability
[Another Technique]
Test and Evaluate
Achieve Acceptable Mission
Improve Test Assets
[Another Test Cycle]

Method

Kent Beck

eXtreme Programming

An Overview

Methoden und Werkzeuge der Softwareproduktion WS 1999/2000

Author Thomas Dudziak

Method

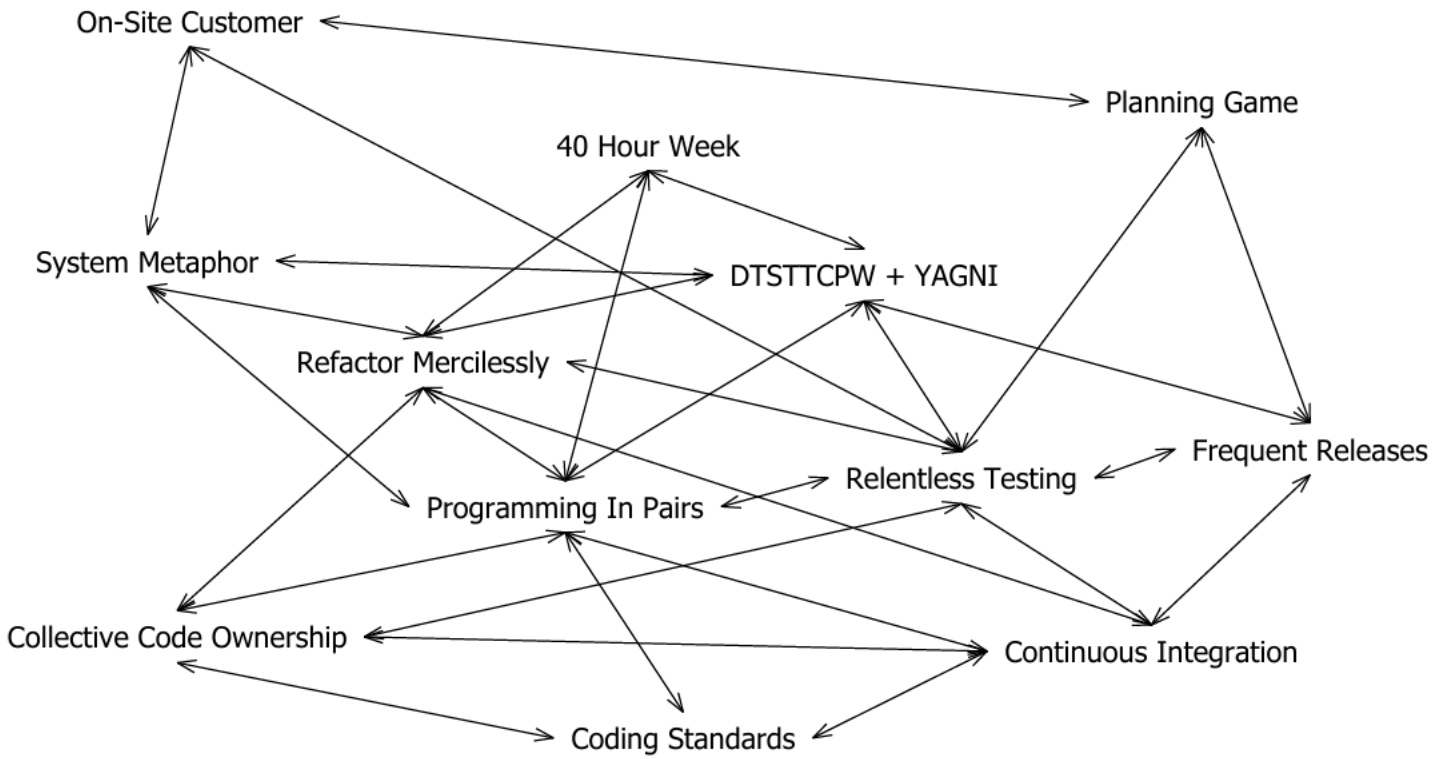


Figure 3 : Support relationships between the rules and practices

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

Method

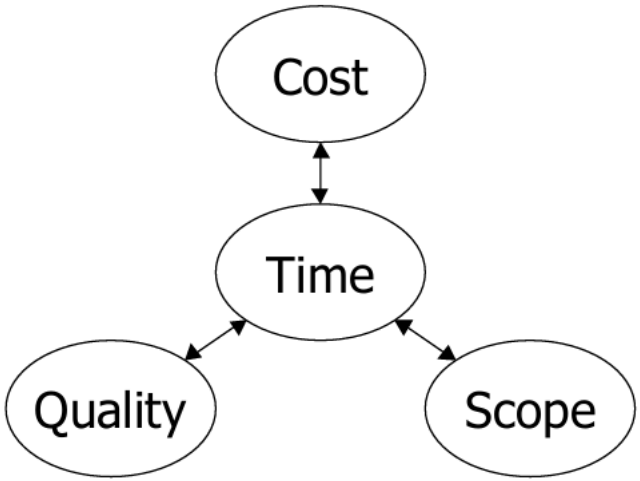


Figure 1 : The relationships between the variables

Method

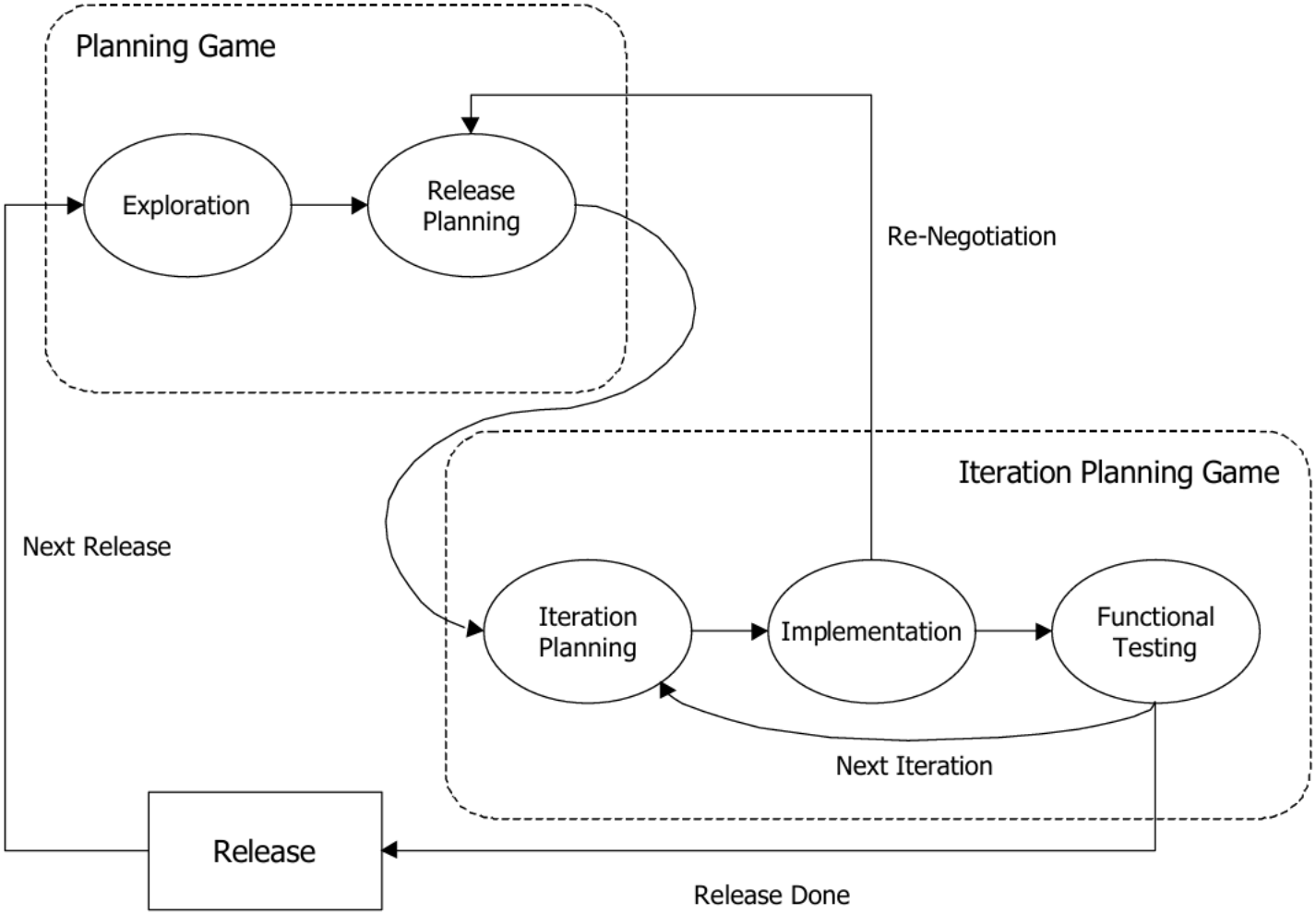
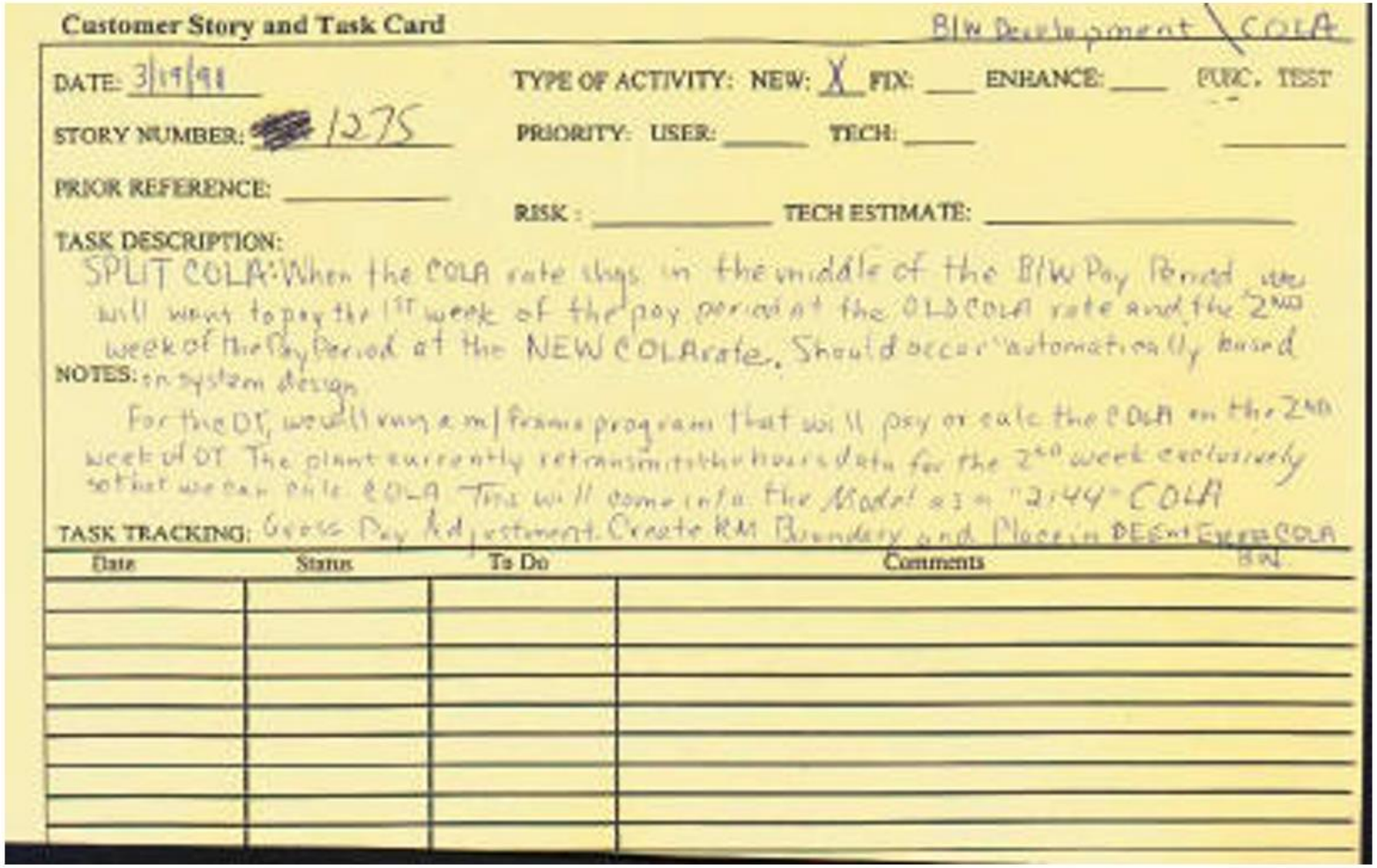


Figure 2 : Simplified process structure

Method



Customer Story and Task Card

B/W Development \ COLA

DATE: 3/19/98

TYPE OF ACTIVITY: NEW: X FIX: ___ ENHANCE: ___ PERF. TEST

STORY NUMBER: 1275

PRIORITY: USER: ___ TECH: ___

PRIOR REFERENCE: ___

RISK: ___ TECH ESTIMATE: ___

TASK DESCRIPTION:

SPLIT COLA: When the COLA rate slips in the middle of the B/W Pay Period we will want to pay the 1st week of the pay period at the OLD COLA rate and the 2nd week of the Pay Period at the NEW COLA rate. Should occur automatically based

NOTES: on system design

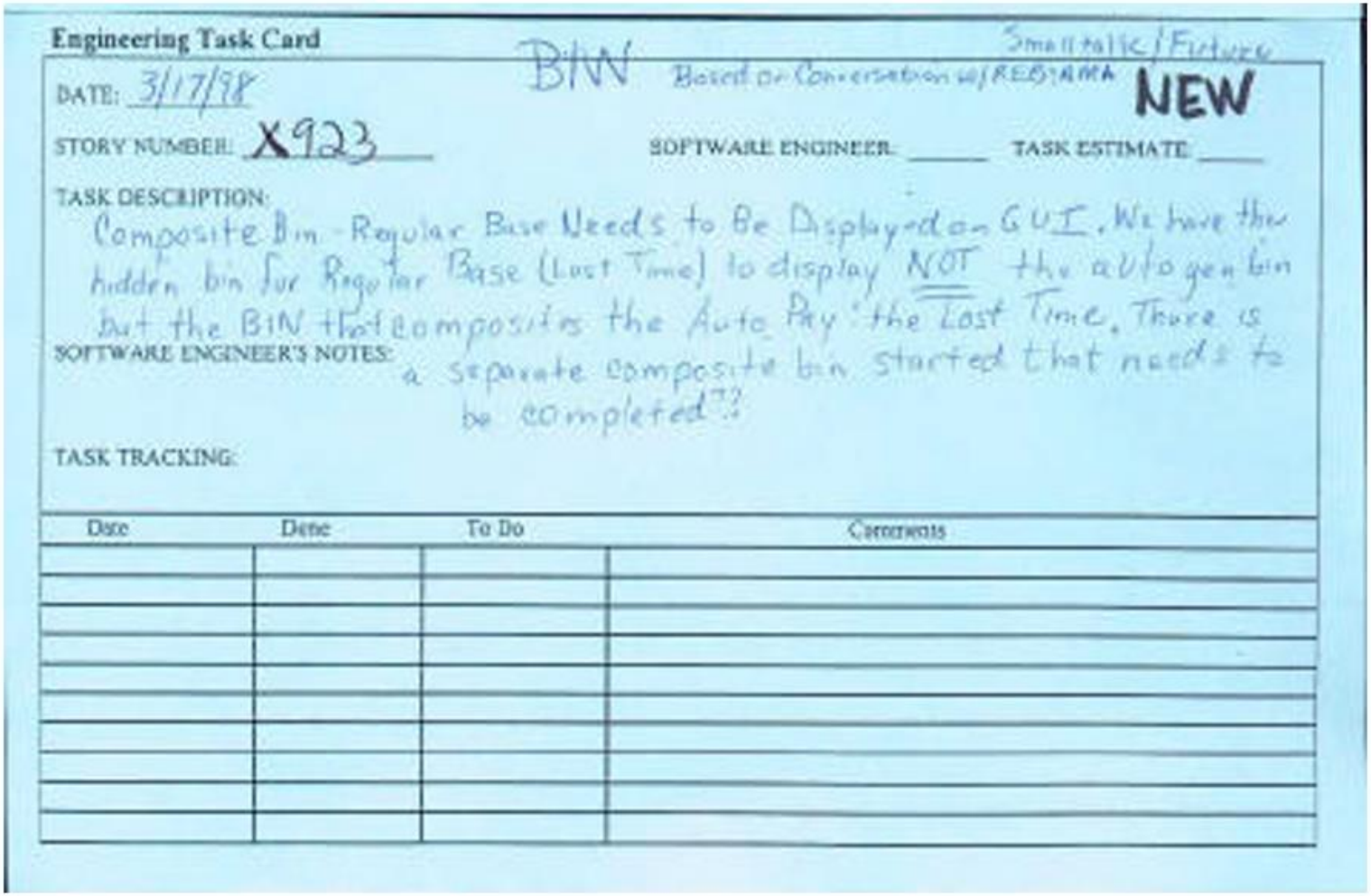
For the DT, we will write a m/frame program that will pay or calc the COLA on the 2nd week of OT. The plant currently retransmits the hours data for the 2nd week exclusively so that we can calc COLA. This will come into the Model as a "2:44" COLA

TASK TRACKING: Gross Pay Adjustment. Create RM Boundary and Place in DEPT EXEMPT COLA B/W

Date	Status	To Do	Comments

Picture 1 : A User Story card from the C3 project

Method



Engineering Task Card

BIN — Based on Conversation w/ REBECCA — Smalltalk / Future — NEW

DATE: 3/17/98

STORY NUMBER: X923

SOFTWARE ENGINEER: _____ TASK ESTIMATE: _____

TASK DESCRIPTION:
Composite Bin - Regular Base Needs to Be Displayed on GUI. We have the hidden bin for Regular Base (Last Time) to display NOT the auto gen bin but the BIN that composites the Auto Pay the Last Time. There is a separate composite bin started that needs to be completed??

SOFTWARE ENGINEER'S NOTES:

TASK TRACKING:

Date	Done	To Do	Comments

Picture 2 : A Engineering Task card from the C3 project

Method

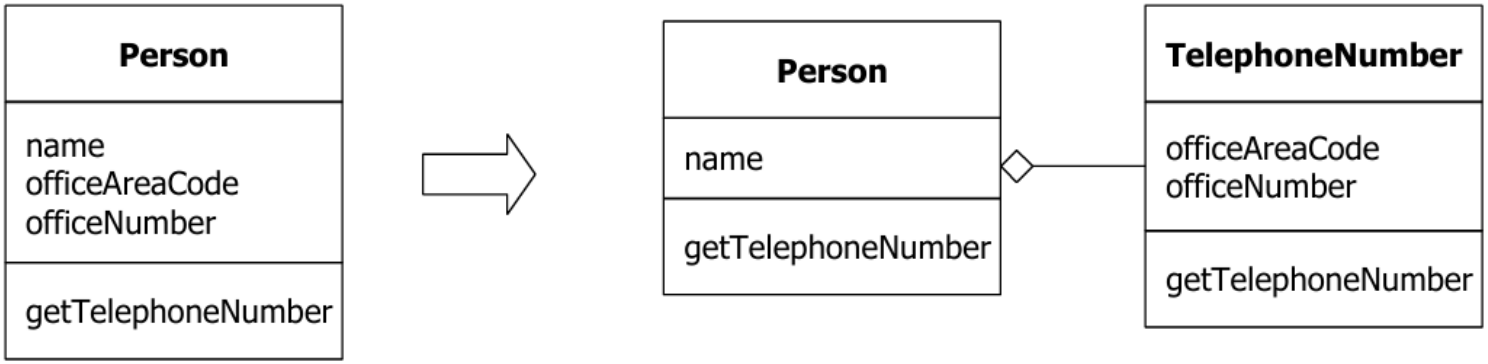


Figure 5 : Extract Class Refactoring

Picture 5 : A pair

Method

Class Responsibility Collaborator

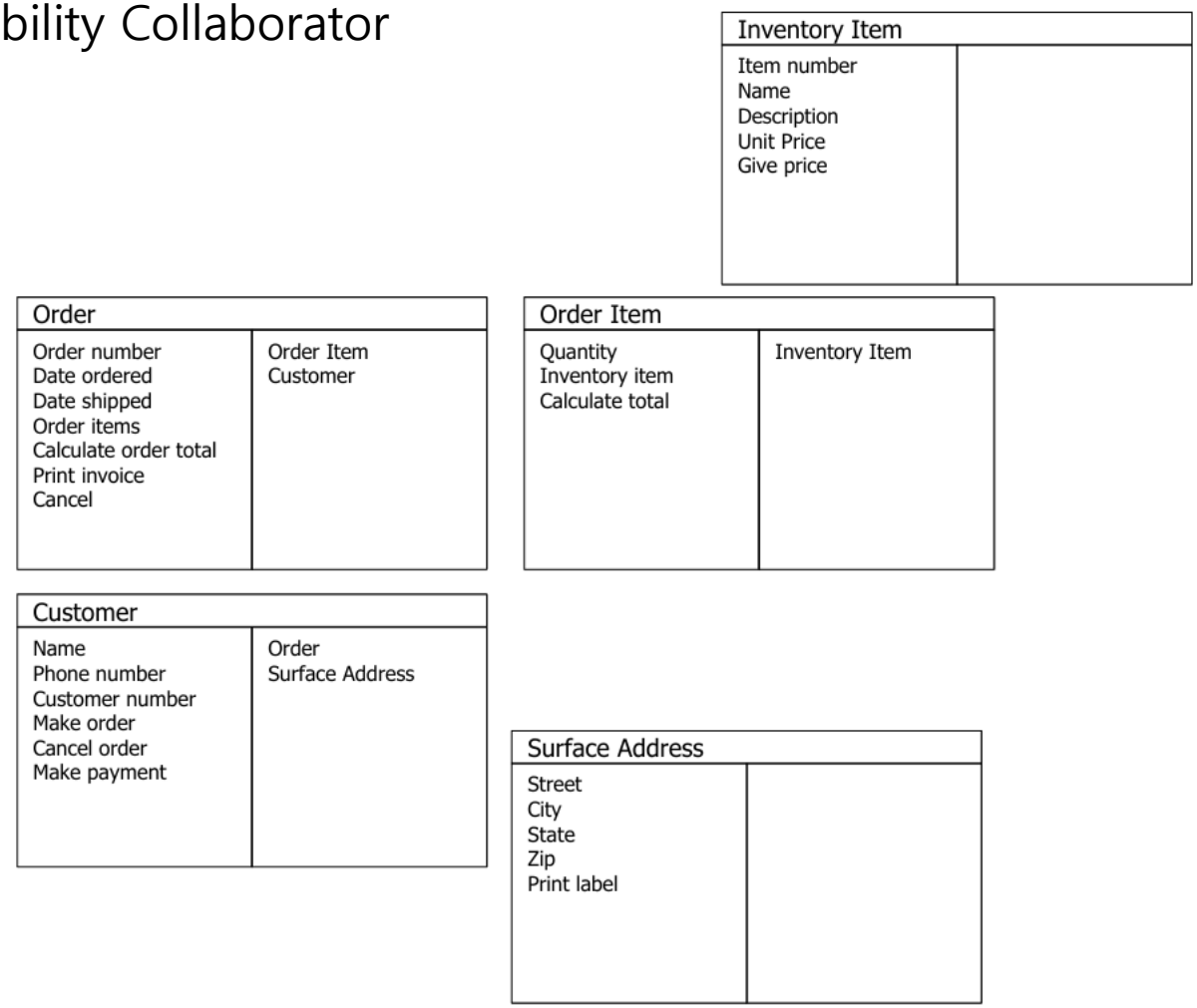


Figure 6 : Example of a CRC layout

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.

Practice

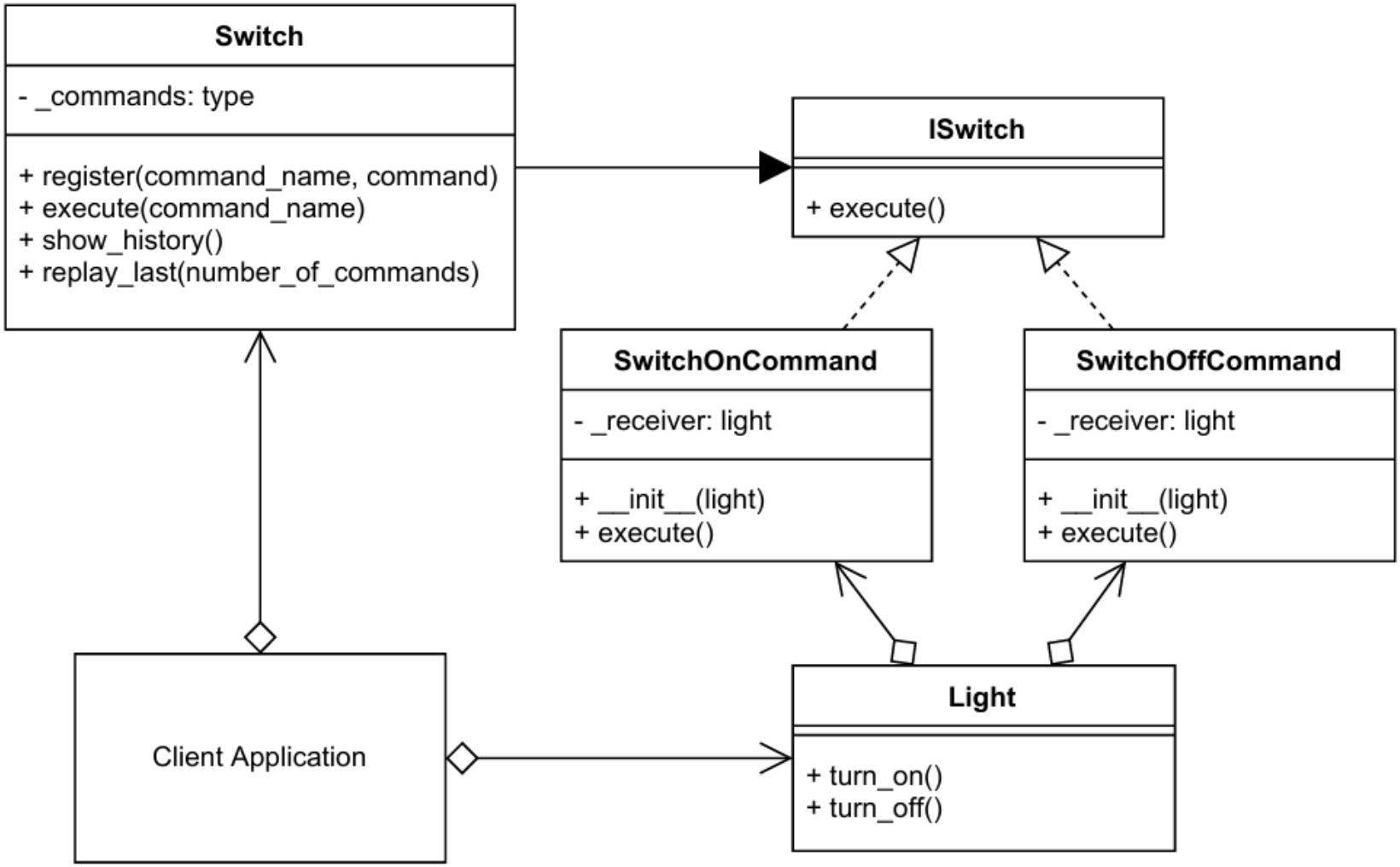
Design Patterns In Python

Common GoF (Gang of Four) Design Patterns

Implemented In Python

Sean Bradley

Practice


Switch
- _commands: type
+ register(command_name, command)
+ execute(command_name)
+ show_history()
+ replay_last(number_of_commands)
ISwitch
+ execute()
SwitchOnCommand
- _receiver: light
+ __init__(light)
+ execute()
SwitchOffCommand
- _receiver: light
+ __init__(light)
+ execute()
Client Application
Light
+ turn_on()
+ turn_off()

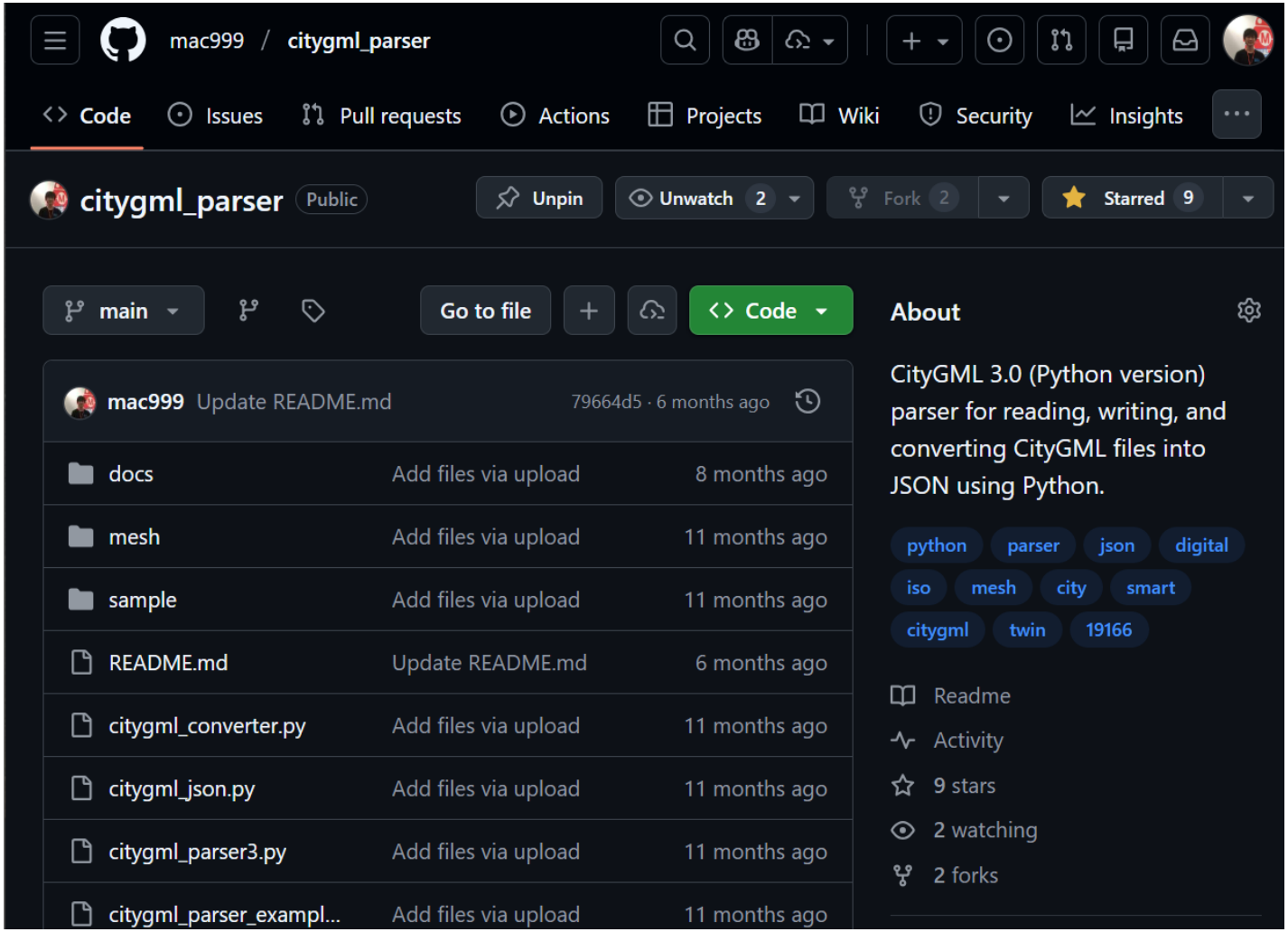
Practice

```
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

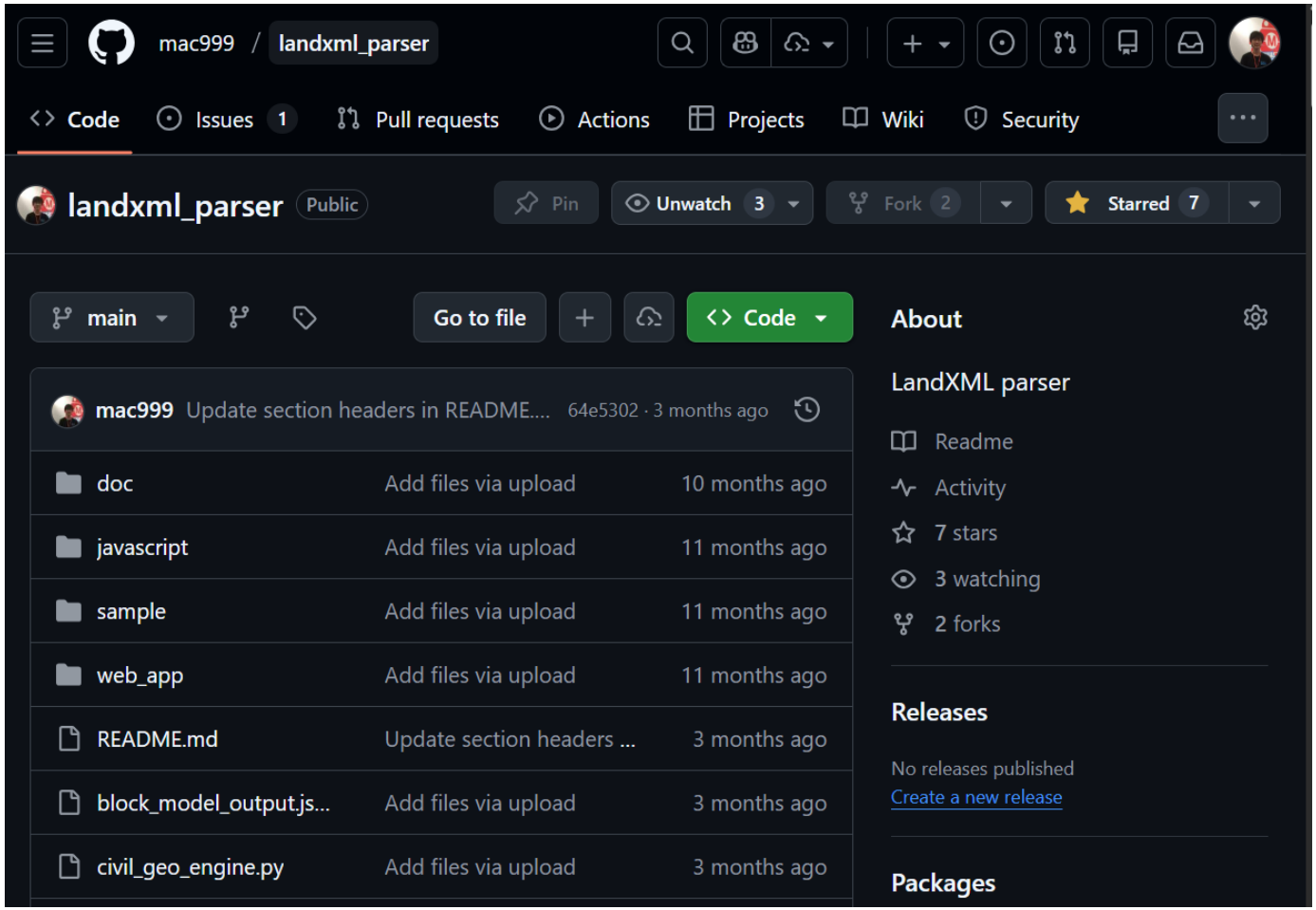
Practice

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



Practice

[landxml parser, civil model server and civil model web viewer]



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 Frameworkdk
- Working with UML Profiles

Managing a Project

Creating New Project

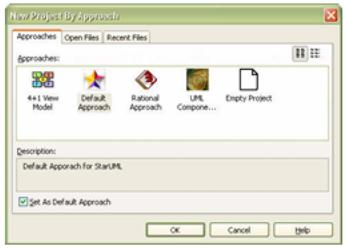
In order to work on a new software development, a new project must be created. You may start with a completely empty project or with a new project that has been initialized according to a specific approach.

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.



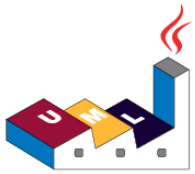
3. A new project is created and initialized according to the selected approach. Depending on the approach, profiles and/or frameworks may be included/loaded.

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)

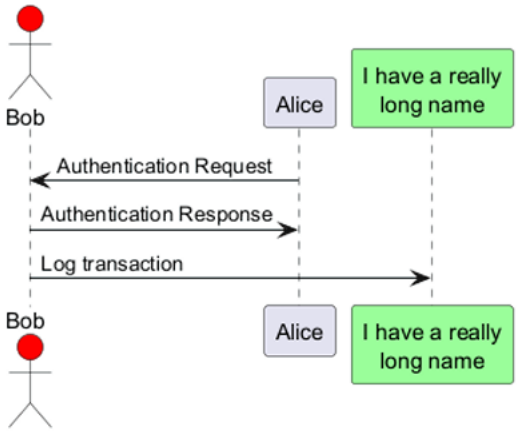
PlantUML is a component that allows to quickly write :

- 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

laputa99999@gmail.com

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