→ what is UML ?

The unified Modelling Language is Standard graphical language for modelling object oriented software.

→ Class diagrams ?

describe classes and their relationships

→ intereaction diagrams ?

show the behaviour of systems in terms of how objects interact with each other

→ State - machine diagrams and activity diagrams ?

show how systems behave internally

* member = üye

booking = biletleme

passenger = yolcu

→ Classes ?

represent the types of data themselves

→ Associations ? Bağlantı

araba sınıfı ve insan sınıfı

Benim arabam

represent linkages between instances of classes

→ Attributes ?

are simple data found in classes and their instances / özellikler → specifications

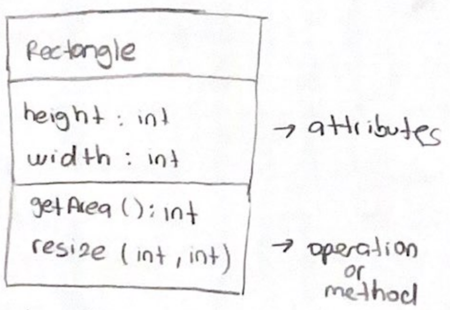
→ Operations ?

by the classes and their instances → sınıfın yaptığı işler

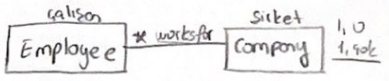
→ Generalizations ?

group classes into inheritance hierarchies

1)



! class



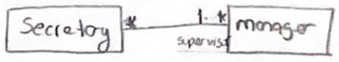
* → many

0, 3 - 8 → multiplicity

0 .. 1 → 0 to one

* hiç yada çok

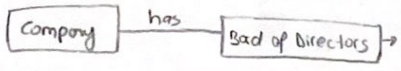
boş olan yer 1 kabul edilir.



many to many

Bir sekterin birden fazla menejeri olur

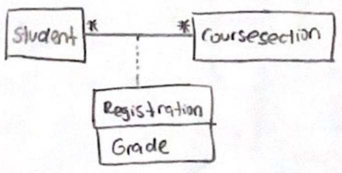
Bir menejerin 0 dan fazla bir çok sekreteri olur



→ yönetim kurulu

One to one

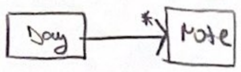
2) Associations classes

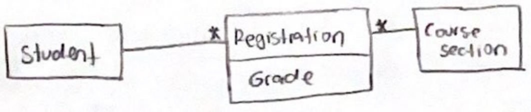


Directionality in associations

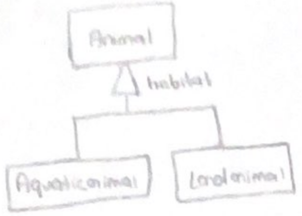
Kimin kimi ilgilendirdiği ⟶

Notun içinde gün var gün nota gider



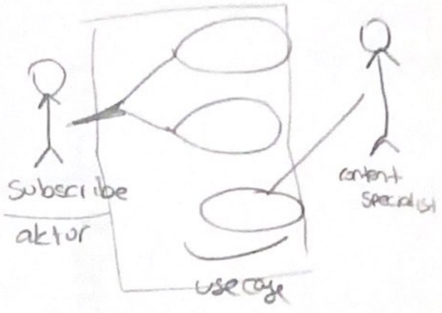


Generalization   ⇧ inheritance → miras   goals = hedefler



Scenarios   Use case

A scenario is an instance of a use case:
- a specific actor
- at a specific time
- with specific data.



Aspects of usability → divided → Bölünür

Learnability : kolay öğrenilebilmesi, bilgiye kolay ulaşılması

Efficiency of use : verimli kullanılması, How fast an expert user can do their work.

Error handling : The extent to which it prevents the user from making errors, and helps to correct errors.

(kabul edilebilirlik)
Acceptability : The extent to which users like the system.

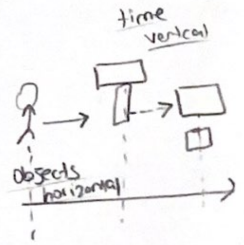
Use-case diagram

used during requirements elicitation and analysis stages
(gereksinim belirleme ve analiz aşamalarında kullanılır)

use-case diagrams → Boundary of the system → sistemin sınırı
use-case diagrams represent the functionality of system
use-case show the behavior of the system

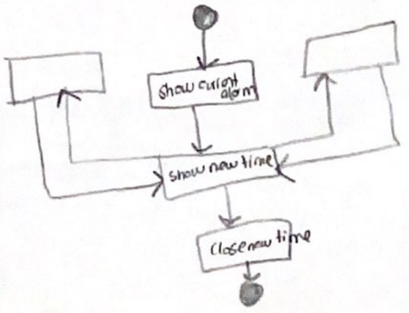
interaction diagram

Used to formalize the dynamic behavior of the system → dynamic model
(Davranış)



State-machine diagram

Describe the dynamic behavior of an individual object   Tek nesnenin..   , dynamic model
A number of states + Transitions between the states
Bir dizi durum + Durumlar arası geçişler.



Activity diagrams

Describe the behaviour of a system in terms of activities , dynamic model
The completion (Tamamla) of other activities
External events (Harici etkinlikler)
availability of object → kullanılabilirliği

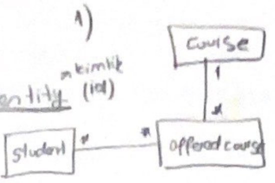
Dach finish

Class diagram , tasımlar

used to describe the structure of the system → sistemin yapısal anlatır

Object model Represent → temsil eder

(*) Each object has an identity (id) → kimlik

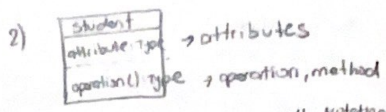


Object : instances of classes : sınıf örnekleri

nesne içermelidir

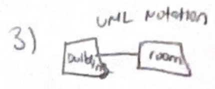
The values of its attributes → Niteliklerin değerleri

The links with other objects → Diğer nesnelerle olan bağlantılar



describe the system

object, class, attributes, operations, associations → bağlantı, ilişki



~~**~~!! Abstract class : Soyut sınıf

- A class that isn't meant to be instantiated → örneklenmesi amaçlanmayan
- generalized concepts represent
- Their names are italicized
- Not all generalizations are abstract classes   tüm genellemeler soyut sınıf değildir

interaction diagram

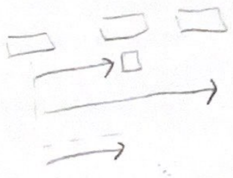
Sequence diagram : ardışıl

Object → Horizontal : yatay

time → vertical : dikey

- Tüm olası etkilesimler   Abstract sequence
- Bir olası etkilesim   Concrete sequence / somut

↳ All / one possible interaction



– fazla yer kaplar + Bulması kolay olur + sağa doğru eklenerek büyür

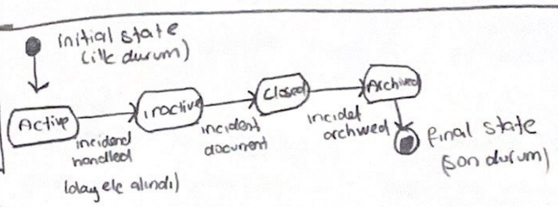
Örnek = Instance

Communication Diagram

+ same information

+ of messages by numbering   sequence

+ az dallanma vardır.



state-machine diagram

* ⬭

Kararları yuvarlatılmış dikdörtgenle gösterilir.

Nesnenin veya nesnelerin × anındaki durumunu ifade etmek için kullanılır

Transition (geçiş) : Nesnenin bir durumdan diğer durumuna geçişini ifade eder. → change of state by events, conditions, time
⟶

State : Bir nesnenin nitelikleri tarafından sağlanan koşul → by the attributes of an object

Class   Person ⟶ student, Professor

generalization   Genel → superclass = Person

Specialized   Özel → subclass = student, Professor

$$\begin{tikzcd} & \text{video} & \\ \text{movie} \arrow[ur] & & \text{TV} \arrow[ul] \end{tikzcd}$$

specialization

* Each object belongs to exactly one class → Her nesne tam olarak bir sınıfa aittir

Object oriented Modelling   * solution Domain → çözüm alanı

The solution domain model is much richer and more volatile than the application domain model.
Değişken

OOA → Modelling the application domain

OOD → Modeling the solution domain
↓
Design

Falsification : sahtecilik

- The model doesn't correspond to the reality it is supposed to represent.
  (gerçek)

Model temsil etmesi gereken gerçekliğe uymuyor

Prototype :

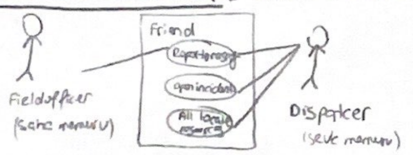
- used to help applying falsification in software system → sahteciliğe yardım eder
- Modify subsequently (Daha sonra değiştirin
  ||
  Change it later

Use case Diagrams

Relationship

include for types:
communication
Inclusion → Dahil etme
Extension → uzantı
Inheritence → Bir sınıfın bilgisini başka bir sınıfa aktarır ve işlevini arttırır.

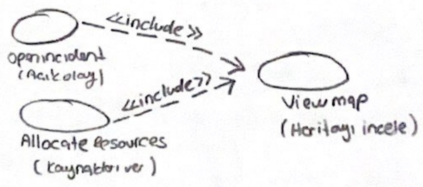
1) communication Relationship: used to denote (→ belirtmek) access (→ erişim) to functionality



2) Include relationship : (Tanımlar) complexity and redundancy (fazlalık) reduce ↓ <<include>> dahil et

Include: Benzer olan bir davranış için kullanılır. O davranışın aynen diğer use case kopyalanmasını sağlar.

indentifying commonalities in different use cases. Farklı use case lerde ortak noktaları tanımlar.



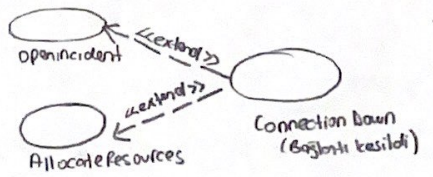
∗ Behavior shared
paylaşılan davranış

<<extend>> genişlet

3) Extend Relationship : complexity Reduce ↓

extend: Genişletilmiş use case temel use case'e davranış eklenebilir. Temel sınıf extension pointleri tanımlar.

Extend another use case by adding events. Olaylar ekleyerek başka bir kullanım durumunu genişletin.

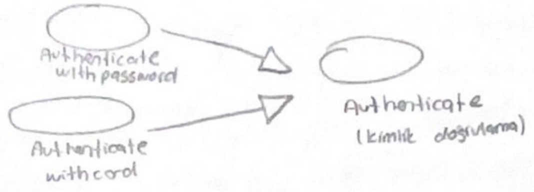


∗ Exception istisna
Help, error, unexpected condition
Beklenmeyen durum

∗ Include beetwen extend differance;
The location of a dependency → Bağımlılığın konumu

inheritance relationship : complexity Reduce ↓

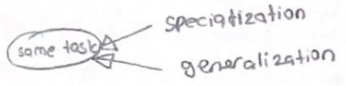
One use case can specialize another more general one by adding more detail.
(Daha çok ayrıntı ekleyip daha genel bir kullanım senaryosunu özelleştirir).



! inheritance and extend relationship different :

Extend R. : Her kullanım durumu different flow of event (akış, olay) different task (görev)

inheritance R : each at different abstraction level



Scenerio : concrete set of actions → somut bir dizi eylem

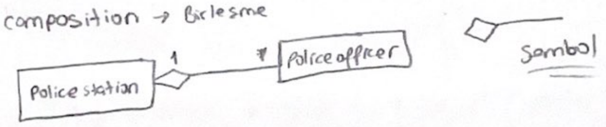
Class diagram

Aggregation : Toplama A special case of an association → Bir ilişkinin özel durumu

composition → Birleşme

Yaşam döngüsü farklı olan iki şey
Birbirine bağlı olmak zorunda değil

PC ve Çanta



Inheritance = (Genel) Generalization → superclass üst sınıf
Specialization → subclass alt sınıf (özel)

operation vs method : Specificiation of behavior → Operation (çalışma) (Özelliğin belirtilmesi)
Implementation of behavior → Method (Yöntem) (Davranışın uygulanması)
UML distinguishes operations from methods. işlemleri yöntemlerden ayırır.

State machine Diagrams

UML 2 46

Actions : Fundamental units of processing → Temel işleme birimleri
Can take a set of inputs → bir dizi girdi alabilir
Produce a set of puts → bir dizi çıktı üretir.

3 yerde meydana gelir
occur ↓ transition is taken (yapmak)
state is entered
state is exited

Activity : A coordinated set of actions.
with state using the do label

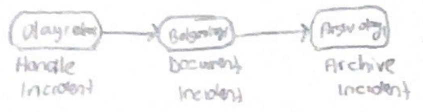
Internal Transition : iç geçiş
not leave the state Durumu terk etmez
of any exit or entry actions

Nested-State Machines : iç içe
Reduce complexity ↓
* set time → UML (50)
(zaman kurma)
* Blink → yanıp sönen ışık

Activity Diagrams

Sequencing and coordination of lower level behaviors.
(Sıralanması ve koordinasyonu Alt düzey davranışların)

- one or several sequences of activities. (Bir veya birkaç etkinlik dizisi)
- The object flows needed for coordinating the activities.
  (Faaliyetleri koordine etmek için ihtiyaç duyulan nesne akışları)

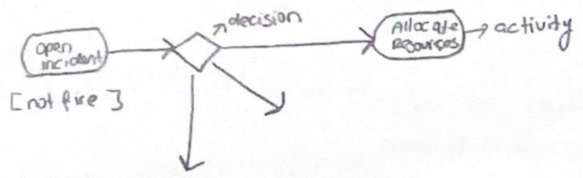


Control nodes : kontrol düğümleri

Main control nodes : Decision — Fork nodes — Join nodes
(Kararlar) (Çatal düğümler) (Düğümlere katılın)

Decision : Decisions are branches in the control flow. ↗akış (Kararlar kontrol akışındaki dallardır.)
Denote alternatives based on a condition of the state of an object or a set of object.
(Bir nesnenin veya, bir dizi nesnenin durumunun bir koşuluna dayalı olarak alternatifler)

[ ] : köşeli parantezde açıklanır.

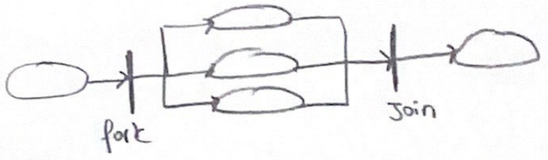


Çatal — Birleştirme
Fork nodes - Join nodes : ⟶ represent concurrency = eş zamanlılığı temsil eder

paralel aktivitelerin başlangıcını ve bitişini göstermek için.

| ilk çizgiye fork => girerken bir, çıkarken çok

| ikinci çizgiye join => girerken çok, çıkarken bir



Swim - Lane : used to group activities
kulvarlar
Transition may cross swim-lanes
çapraz

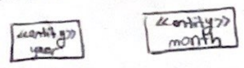
Amaç object or subsystem denote

Diagram extensions

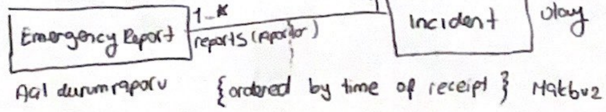
uzantı

Sometimes fixed notations may not be sufficient → (Bazen sabit gösterimler yeterli olmayabilir)
Enabling the modeler to extend the language → (Modelleyicinin dili genişletmesini sağlamak)

Stereotypes <<string>> : kalıp yargılar - klişe
Geliştiricilerin UML'deki model öğelerini sınıflandırmak ↗classify için kullanılır



Constraint kısıtlama : Anlamını kısıtlayan bir model öğesine eklenen bir kural



Acil durum raporu {ordered by time of receipt} Makbuz zamanına göre sıralanmıştır

Three more properties of requirements specification : 5

Realistic (Gercekci) :

The system can be iplemented within constraints. kısıtlama

Sistem kısıtlamalar dahilinde uygulanabilir

Verifiable (Doğrulanabilir, Onaylanabilir) :

Repeatabe tests to demonstrate that the system fullfills the Requirements specification

Sistemin gereksinim belirtimini karşıladığını göstermek için tekrarlanabilir testler.

Traceable (izlenebilir) :

Each requirement can be traced

Her gereksinim izlenebilir

Requirements Elicitation Activities

* problem statement → Sorun bildirimi
* preparing Glossary → Sözlük hazırlama

Benefits

- Help distinguishing parts, objects of the system
  Sistem parçalarını nesneleri ayırt etmeye yardımcı olur
- Eliminates repetition in diagrams, models,... tekrarları ortadan kaldırır
- Eliminates ambiguity → Belirsizliği ortadan kaldırr

* Dispatcher = gönderici
* implementation = uygulama
* Scratch = 0 dan
* Target Environment : Hedef ortam
* Incident = olay
* Emergency Report = Acil durum raporu
* boundary = sınır

* Refining = iyileştirme
* Access = Erişim
* constraint = kısıtlama
* imposed = uygulan
* current = Mevcut
* Training = Eğitim

Indentifying Actor Aktörlerin belirlenmesi

which user groups ?

Actors → boundary system → Actors system boundary outside
external = harici

Subsystems and objects are inside the systen boundary. They are internal
Dahili

Indentifying scenarios

what ve actor ? Senaryo anlatılacak

extend = genişlet
include = birleştir
Collaboration = işbirliği

Indentifying use cases

Generalize scenarios, high-level use cases

flow of events → olayların akışı

quality requirements = kalite gereksinimi

Which actions, by the actor / system ?

- Reliability : Güvenilirlik
  including robustness, safety, and security
- supportability : Desteklenebilirlik
  including maintability and portability
  içermeli

Heuristics (Bulusal) :

validate functionality → işlevselliği doğrula
adopting specific user interface conventions → Belirli kullanıcı arabirimi kurallarını benimsemek
not detail
validate with user
different alternatives.

- implementation : Uygulama
  constraints imposed
- interface : Arayüz
  system
- operation → running system
- Packaging = ambalaj
  install, installations
- legal = yasal
  licensed

Requirement : Gereksinim

Collaboration : ortak çalışanlar

Requirement elicitation = Gereksinim tespiti

output → Requirement specification = Gereksinim özellikleri

problem statement = sorun bildirimi

indentifying Nonfunctional Requirements
(işlevsel olmayan gereksinimlerin belirlenmesi)

implementation = uygulama

interface = arayüz

Requirements Elicitation concepts – Nonfunctional Requirements ✱

– constraints (Pseudo Requirements) kısıtlamalar, sözde gereksinimler

+ usability – kullanılabilirlik
+ Dependability – Güvenilebilirlik
  - Reliability, robustness, safety.
    Güvenilirlik, sağlamlık, emniyet
+ Performance
  - Response time, throughput, availability
    Tepki süresi, çıktı, kullanılabilirlik
+ Maintainability sürdürülebilirlik
+ Portability Taşınabilirlik

→ Requirements (gereksinim) are continuously validated (Doğrulama) with the client (müşteri) and the user.

→ Requirement validation → checking that the specification is;

- complete (Tamamlamak)
- Consistent (Tutarlı)
- Clear (Açık)
- Correct (Doğru)

✱ These are also (properties) requirements specification
gereksinim belirtimi özellikleri ile aynı

Complete (Tamamlamak)

- All features of interest are described by requirements.
  ilgilen tüm özellikler gereksinimlere göre tanımlanmıştır

Consistent (Tutarlı)

- No two requirements of the specification contradict each other
  Gereksinim belirtimi özelliklerinde iki şart birbiri ile çelişmez.

Clear (Açık) → unambiguous → belirsiz

- A requirement cannot be interpreted in two mutually exclusive ways.
  Bir gereksinim birbirinden farklı iki şekilde yorumlanamaz

Correct (Doğru)

- Requirements specificiation represents accurately (doğru şekilde) the system that the client needs and needs and that the developers intend to build. edin

  Gereksinim belirtimi, müşterinin ihtiyaç duyduğu ve geliştiricilerin oluşturmayı amaçladığı sistemi doğru bir şekilde temsil eder.

The result of the requirements elicitation → Requirements specification

RAD → Requirement Analysis Document   clients — users

system analysts
system designers

! stable after written
development process
baseline

Analysis activity → Analysis model

object-oriented analysis → application domain → alanı } Modeling the
object-oriented design → solution domain

Requirement specification → understandable) Difference
Analysis model → may not be understandable

Analysis model composed of

Functional model → use cases and scenarios
Analysis object model → class and object diagrams
Dynamic model → state machine and sequence diagrams

* Analysis → Refine the functional model
iyileştir.

* More precise and complete specification
keskin   eksiksiz   özellikler

* Derive the object and the dynamic model
Türet

Diyagramında
Analysis
- indentifying objects
belirle
- indentifying inheritance relationships

analysis object model
focus on structure of the system
Depicted with class diagrams

Dynamic model
focus on the behavior of the system
davranış
Depicted with sequence and state machine diagrams.

} Represent user-level concepts
Subscriber e-mail vs user ID

GENERALIZATION
Indentify abstract concepts from lower-lever ones → soyut kavramları alt düzey kavramlardan ayırt edin

SPECIALIZATION
Indentify specific concepts from high-level one

find inheritance
inheritance → Relationship

* specifation → belirtimi
Elicitation → ortaya çıkarılması

* flow of event → olay akışı

indentifying Entity objects

* persistent information = kalıcı bilgi
  Entity object of <u>Watch Video</u> ★

entity
subscriber
video
viewer

indentifying boundary object

actor and system
<u>videolist button</u> ★

indentifying control object

<u>WatchVideoControl</u> ← video list button clicks
subscriber
after close

Indentifying Interactions

Sequence Diagrams

CRC → Class, Responsibilities, Collaborators
→ sorumluluklar → ortak çalışanlar
↓
Uygulaması kolay, güvenliği güçlü bir hata bulma yöntemi

● approach and <u>noun pharese</u> approach are used to identify → classes

« create » new
oluştur
« detroy »
yok et

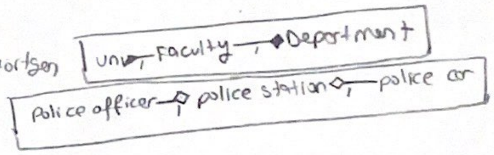
indentifying Associatiations

Association → Relationship between <u>two</u> or <u>more clasess</u>
properties of associations → Name, role, multiplicity
isim rol çokluk

- Indentifying Aggregates (Birleştirme tanımlama)

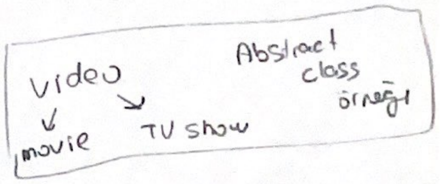
<u>whole</u> - <u>part</u> relationship
Bütün parça

◆— composition (Birleştirme) → Solid diamond dolu dikdörtgen
◇— shared → Hallow diamond içi boş dikdörtgen
(paylaşılan)



indentifying state-Dependent Behaviors ; Duruma bağlı davranışların tanımlanması

- single object
- more formal description → Resmi tanım
- missing use cases → Eksik kullanım durumları
- new behavior → Yeni davranış



★ The requirements are <u>realistic</u> and <u>verifiable</u>
doğrulanabilir.