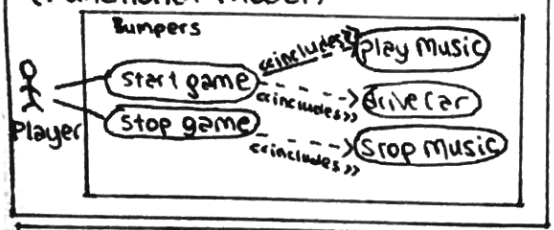
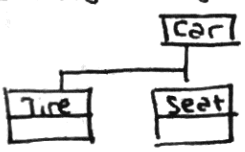
SOFTWARE CHEATSHEET

UML → Archival
→ Analysis & Design
→ Communication

UML USE CASE DIAGRAMM (Functional Model)

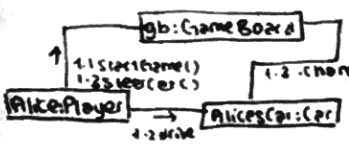


UML CLASS DIAGRAM (Analysis Object Model)



◇ Aggregation (has a)
◆ Composition (life depends on)
Δ inheritance

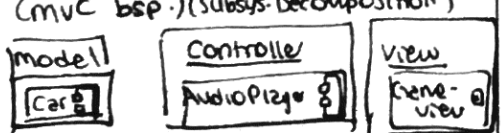
UML COMMUNICATION DIAGRAMM (Dynamic Model)



1.1 → Sire önemli
1.0 → " " değil

(dynamic model → comm.diagramm, activity " , state chart)

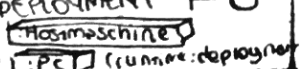
UML COMPONENT DIAGRAMM (MVC bsp.) (Subsys-Decomposition)



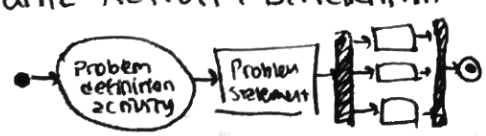
also Data

entity — control — boundary

DEPLOYMENT (HW/SW Mapping)



UML ACTIVITY DIAGRAMM



3. Req. Eli

Nonfunctional Req

Usability: need 5 click to... (learnability, efficiency, memorability, error handling, satisfaction)

Reliability: prepared against any virus

Performance: 99.9% of time in a year

Supportability: if any change occurs, it should adapt (Adaptability & maintainability)

Pseudo Req. (constraints)

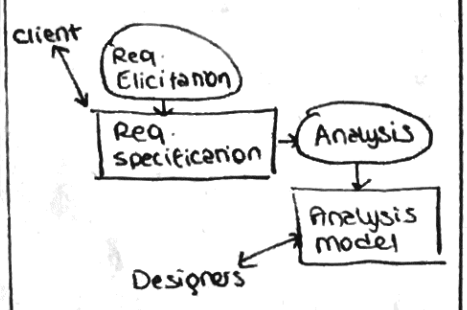
- implementation tools, lang.
- operation req
- legal req
- packaging req (delivery) (ios...)
- interface req (word format)
- verification
- validation

Scenario

- Scenario name:
- Participating Actors: Bob: Player
- Flow of Events:
  1. Actor Step
  2. System Step
  3. Actor Step
  4. Exit Cond: (Exit cond: dive yeamly osun)

Use Cases

1) Name:
2) Participating Actors: Player
3) Flow of Events
1. Tyhe player starts the game
⋮
4) Entry Exit Conditions: The game window is open
5) Exit Conditions: The player plays the game
6) Special requirements: device has enough power

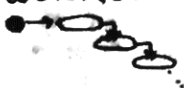


09. SW. Lifecycle modeling

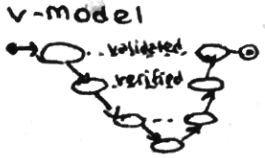
→ incremental - "add onto sth"
→ iterative - "re-do or re-work sth"
→ agile

INCREMENTAL

→ waterfall : activity oriented
: if " completed, the next one starts



→ V-model



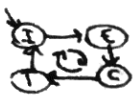
ITERATIVE

→ Spiral

1. Determine objectives, alternatives, constraints
2. identify risks, assign priorities to them
3. develop prototype with waterfall model
4. if risk resolved → evaluate results of iteration and plan next one
If not → project is terminated.

→ Unified Process → the scope is strictly defined
- cycles
- each phase can have several iterations

1. Engineering Stage: Inception, Elaboration — less predictable, smaller teams — design/synthesis activities
2. Production Stage: Construction, Transition — test and deployment



AGILE

- empirical
- frequent change

→ Extreme Programming:
- Avoid over planning
- improve sw quality
- " responsiveness to changing customer req

→ Kanban
- Existing process, Leadership ...

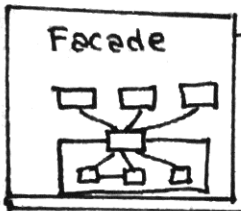
→ Scrum
- change!
- risk management, communication and delivery improved

5 meetings...
3 Roles:
- Product Owner: defines product, responsible for results
- Scrum Master: resolves impediments, respons. for process
- Dev. Team: self-organizing, cross-func., realizes the product increment

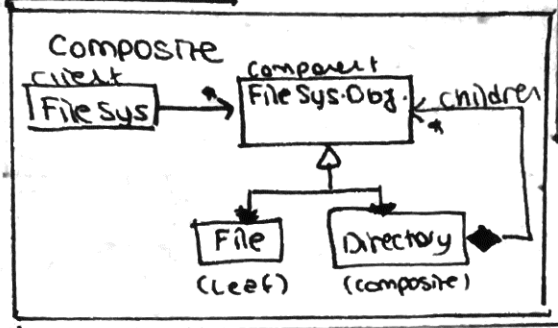
PATTERNS (Obj Design I: Reuse)

Structural

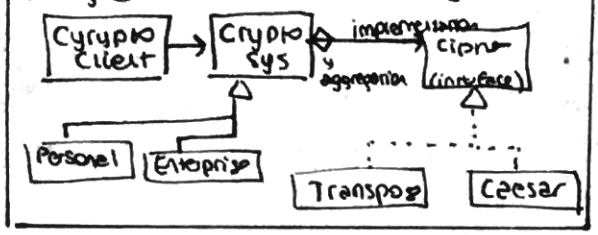
Facade



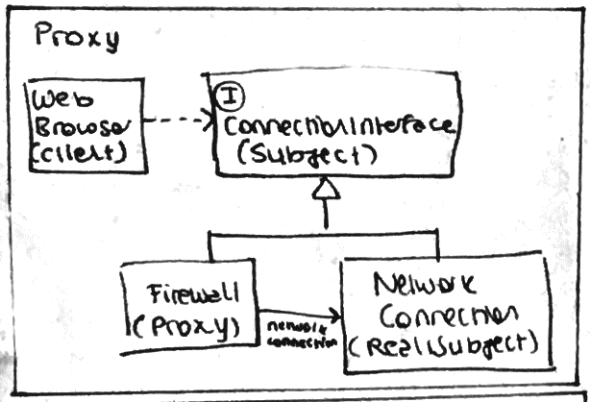
Composite



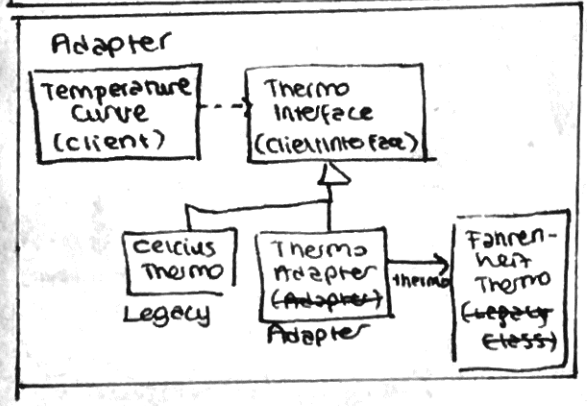
Bridge → runtime vs. load-time delay



Proxy

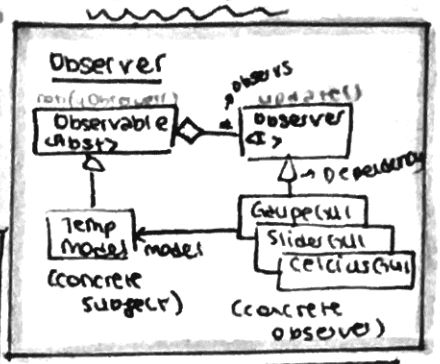


Adapter

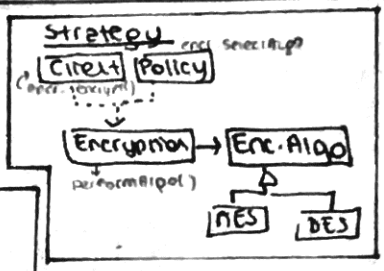


Behavioral

Observer



Strategy



(Subsyst. Decomposition : Sys Design)

ARCHITECTURE STYLES → pattern for subsys decomp

- Layered
- Client/Server
- MVC

a sw archi is an instance of an archi style.

TESTING

- Unit Testing
- Integration "
- System "
- ...> Acceptance "

internal / external

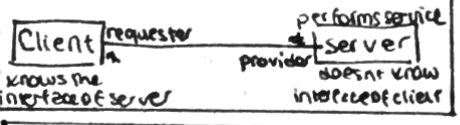
LAYERED

=) Closed: access to below layer
=) Open: " " each " "
=) 3 layered: WebBrowser: UI
WebServer: serves requests from UI
Database
=) 4 layered: + Application Server
=) 7 layered: Application
↓ Presentation
↓ Session
↓ Transport
↓ Network
↓ Data Ling
↓ Physical

CLIENT-SERVER

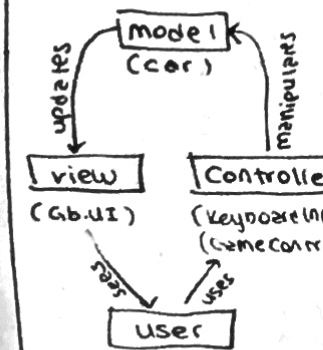
(data-base systems) } clients can be servers and vice-versa.
(peer to peer archi)

- one or more servers provide services to clients
- each client calls a service offers by the server



MVC

model: processes and stores app domain data (entity)
view: display (boundary)
controller: interact with user & update the model (control)



- 3 tier Archi (tier: instance)
- monkies.

UNIT → development environments (OBJ. DESIGN)

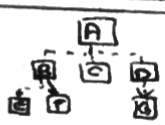
- Component is correct carries out the intended func.

```
expected = ...
observed = ...
assertTrue(expected.equals(observed))
```

```
@Test(expected=IllegalArgumentException.class)
public void ...
:
@Test(timeout=1000)
(Determinism de)
```

INTEGRATION (System Design)

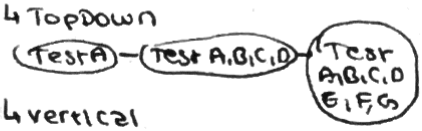
↳ BigBang



Test A,B,C,D,E,F,G

↳ Bottom up → real time sys

↳ TopDown



↳ vertical
always an executable version

SYSTEM (REQ. ANALYSIS)

↳ Functional t.: black box functionality of sys.
↳ Structure t.: white box cover all paths in sys.
↳ Performance t.: try to violate non-func. req.

ACCEPTANCE (CLIENT EXPECTATIONS)

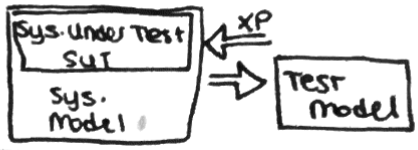
is sys. ready for operational use
↳ Alpha: at developer site by client & end user
↳ Beta: without " " " "

STATIC VS DYNAMIC ANALYSIS

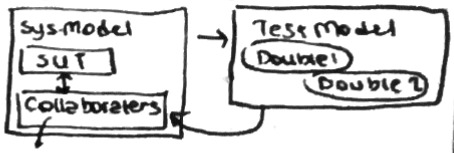
Static: hand execution
dyn.: black box → test the input/output behaviour
white box → test the implementation
↳ statement testing
loop "
Path "
Branch "

MODEL BASED TESTING

- not their absence can be tested

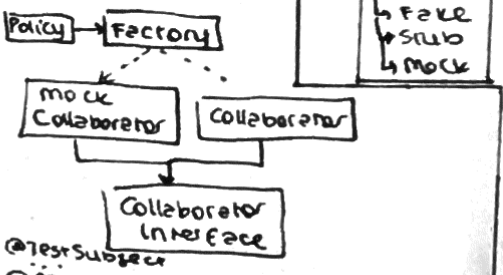


OBJECT ORIENTED TESTING



not yet implemented

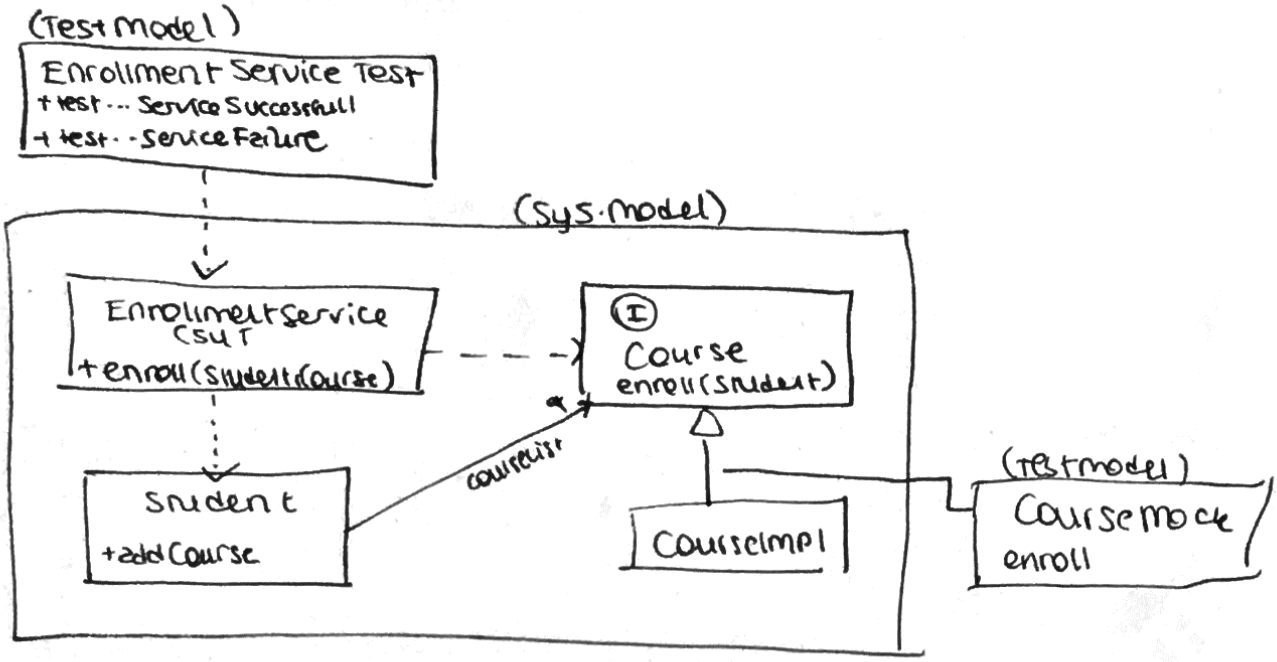
MOCK OBJ PATTERN



Double
↳ Dummy
↳ Fake
↳ Stub
↳ Mock

```
@TestSubject
...
@Mock
private ...
@Test
public ... { ... mock = mock(...class)
  expect(...mock.....).andReturn(..)
  replay(...mock)
  ... executeSUT...
  ...compare expected & observed
```

Subsys. decomp: identification of subsys, services and relationships




```
@TestSubject
  private RoomReservation roomres = new RoomReservation()

@Mock
  private Room roomMock;

@Test
  public void roomReservationSuccessfullTest() {

     Course course = new Course;
     String roomName = "Lecture Hall"

     expect(roomMock.getName()).andReturn(roomName);
     replay(roomMock);
     roomres.reserve(roomMock, course);

     String assignedCourseName = course.getRoom().getName();
     assertEquals(assignedCourseName, roomName);

  }
```