IMT2243 Systemutvikling

Tema : ObjektOrientert Design + intro til Testing

- Spørsmål / kommentarer til Arkitektur og prosjektlev. 3 ?

- ObjektOrientert Design
  - Bevissthet ved ansvarstildeling nede på softwareklasse-nivå
  - UML i Designfasen
    - Design KlasseDiagram i forhold til Domenemodellen
    - Sekvensdiagram i forhold til SystemSekvensdiagram
  - Gjennomgang av figurer fra kap. 7 og eksempler
  - Grunnleggende prinsipper og Patterns for OO-design

- Introduksjon til Testing (og Inspeksjoner)

Pensum : Kap.5.2.2., 7.1 og 7.2 Sommerville

Fra OOA til OOD

- Ved å benytte Objektorienterte metoder både i analyse og design i et programutviklingsprosjekt bygger man naturlig videre på de spesifikasjoner og modeller man allerede har laget.
- Nå skifter fokus fra «HVA» til «HVORDAN» for så i neste omgang og programmere dette i et objektorientert programmeringsspråk.
- Målet er å få utviklet en løsning preget av subsystemer, komponenter og også klasser med høy funksjonell styrke og lave koblinger.
- Det ligger viktige gevinster i at man legger til rette for programvare som er mer vedlikeholdbar, tilpasningsdyktig og fleksibel.
- Kjernen innen ObjektOrientert Design er å øke bevisstheten rundt plassering av ansvar på programvareløsningens klasser/objekter.

Objektorientert Design

ObjektOrientert Design handler altså om bevissthet i tildeling/fordeling av ansvar til softwareklasser for å ivareta systemoperasjonene (og derigjennom realisere brukernes krav til funksjonalitet gjerne uttrykt i en Use Case beskrivelse).

- Hva gjør dere pr. i dag ?
- Hva har dere lært om dette i Programmering ?
- Hva slags ansvar kan objekter ha ?

Ansvarsformer for et objekt : Handling og Kunnskap

Handling (Doing) :

- Doing something itself, such as creating an object or doing calculation
- Initiating action in other objects
- Controlling and coordination activities in other objects

Kunnskap (Knowing):

- Knowing about private encapsulated data
- Knowing about related objects
- Knowing about things it can derive or calculate

Craig Larman,

Det er objektene som ivaretar selve ansvaret. Ansvarsformene implementerer man i form av metoder som defineres av klassene.

OOD med UML

IMT2243 ser vi på bruk av 2 UML diagrammene innen OOD :

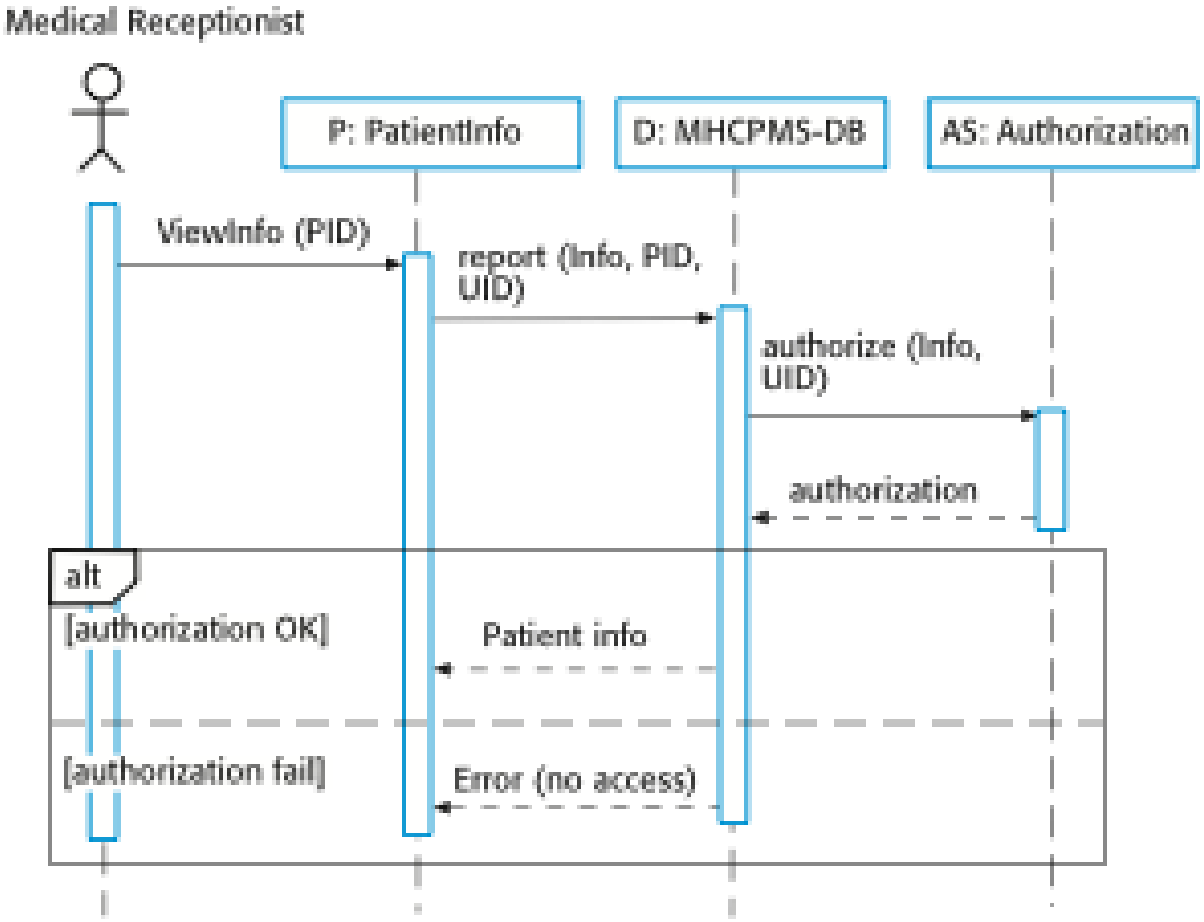
1. Design Klasse Diagrammet (Class Diagram - et diagram for ”hele” programvareløsningen). Merk dere kobling til Domenemodell !

2. Sekvensdiagrammet (Sequence Diagram - ofte et diagram pr. Use Case eller sågar nede på et enkelt steg inne i et Use Case). Merk dere forskjeller/likheter med System Sekvensdiagram !

   Fokus flyttes fra våre modeller av ”den virkelige verden” til ”modeller av innholdet i selve programvaren”. I tillegg til «business-klassene» fra domenemodellen introduseres her en rekke programvare-spesifikke klasser som tildeles ansvar for å realisere kravene. Grensesnittklasser og Prosesskontrollklasser er eksempler på det.

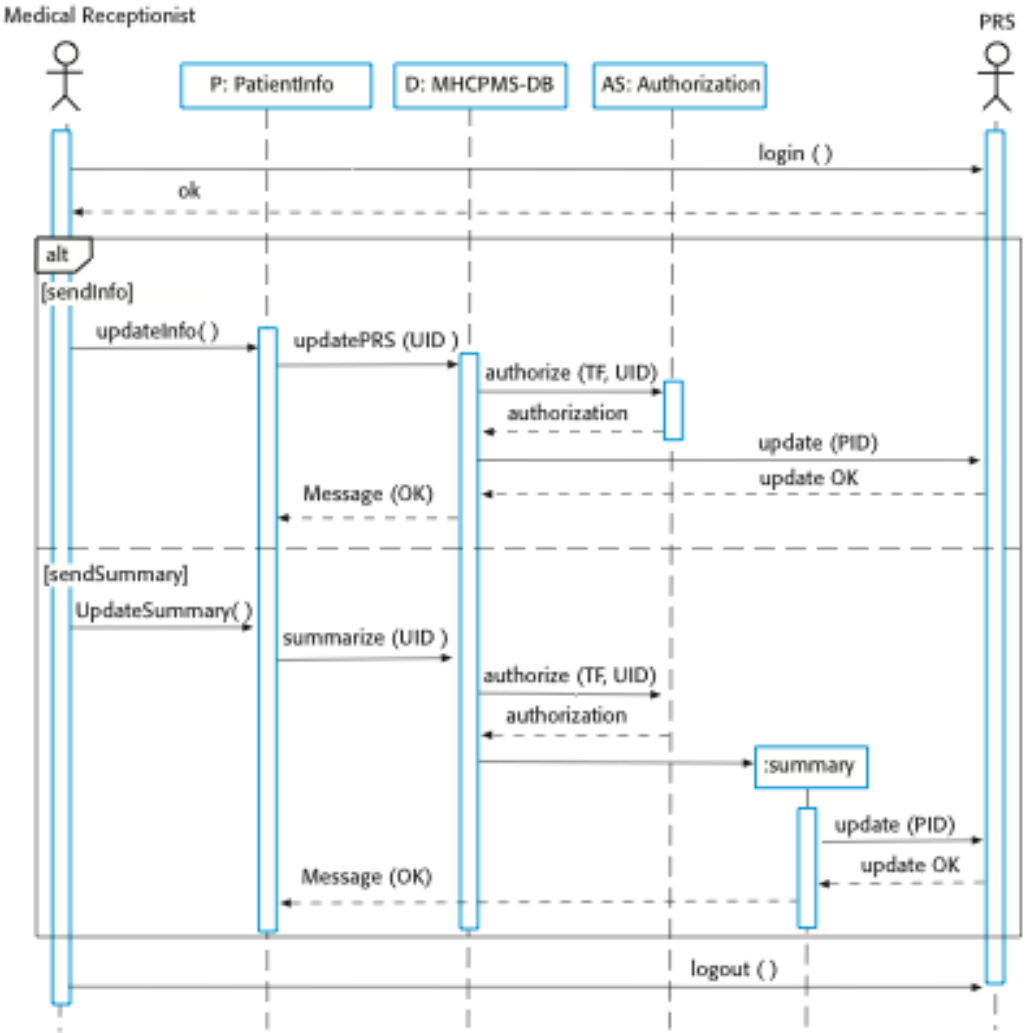
   Modellering av spesielt utfordrende deler av programvaren er alltid smart. For øvrig er det store forskjeller i hvor deltajert man velger å modellere.

Sequence diagram for View patient information



Sommerville: Figure 5.6

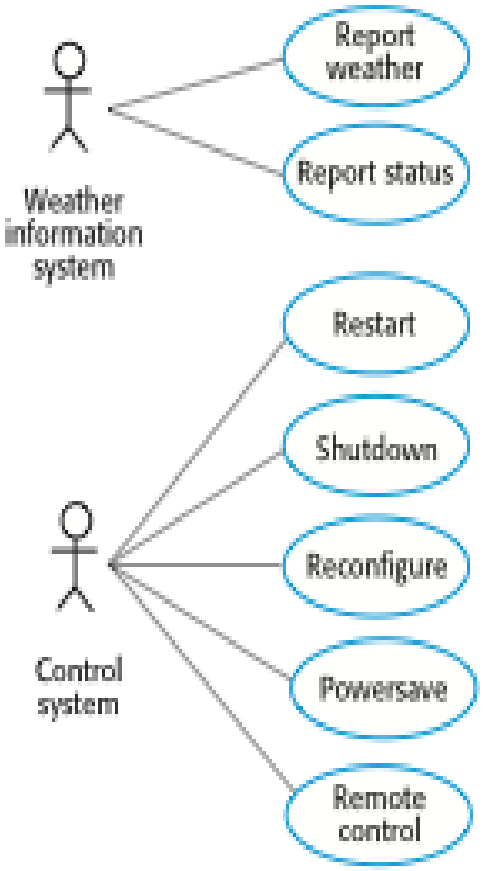
Sequence diagram for Transfer Data



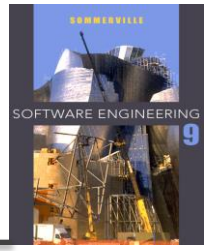
Sommerville: Figure 5.7

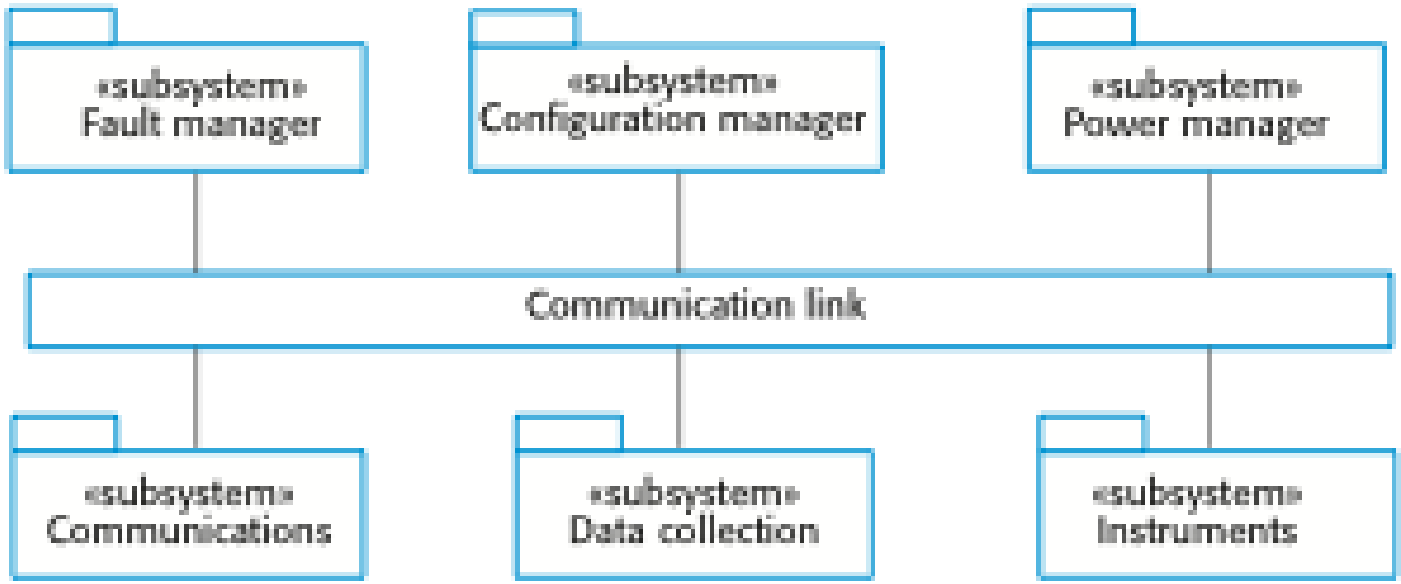
Værstasjonseksempel

(fig.7.2 hentet fra Software Engineering, Ian Sommerville)



High-level architecture of the weather station



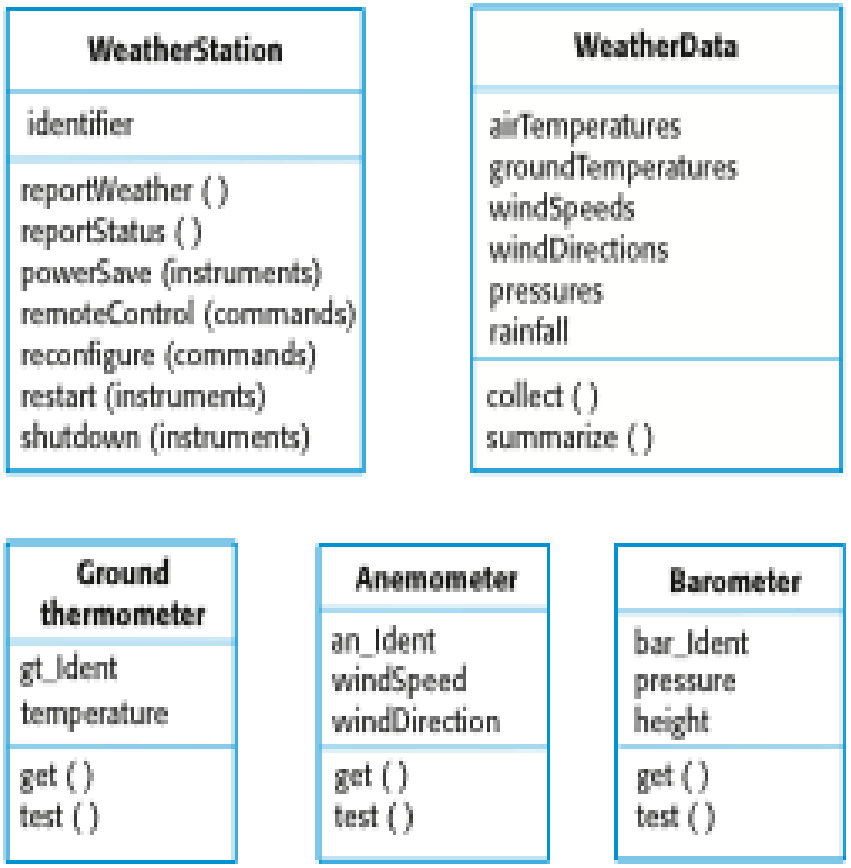


Use case - Report weather

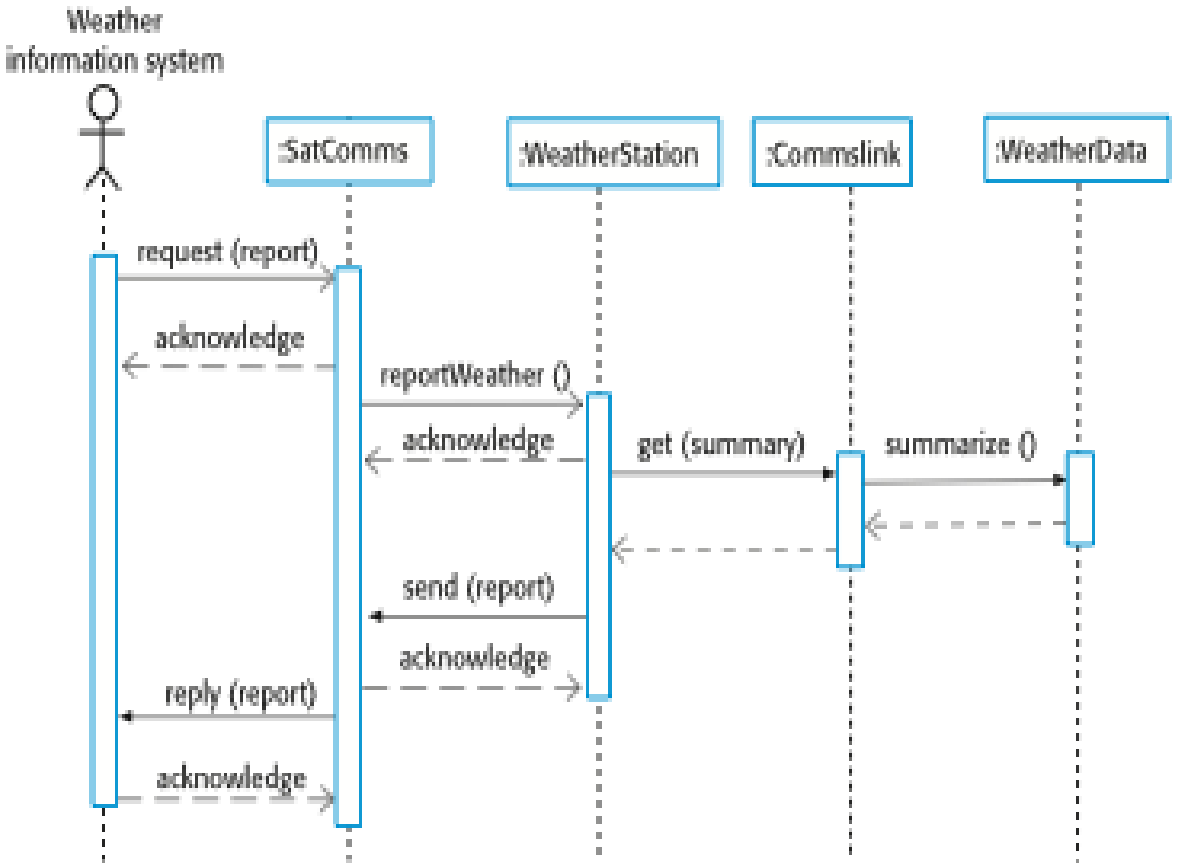
(fig. 7.3 Sommerville)

System	Weather station
Use case	Report weather
Actors	Weather information system, Weather station
Description	The weather station sends a summary of the weather data that has been collected from the instruments in the collection period to the weather information system. The data sent are the maximum, minimum, and average ground and air temperatures; the maximum, minimum, and average air pressures; the maximum, minimum, and average wind speeds; the total rainfall; and the wind direction as sampled at five-minute intervals.
Stimulus	The weather information system establishes a satellite communication link with the weather station and requests transmission of the data.
Response	The summarized data is sent to the weather information system.
Comments	Weather stations are usually asked to report once per hour but this frequency may differ from one station to another and may be modified in the future.

Weather station objects (fig 7.6)



Sekvensdiagram for datainnsamlingen (fig.7.7)



Hvordan øve på OOD ?

Bruk de eksemplene vi har tatt for oss i tilknytning til emnet gjennom semesteret : Biblioteksløsningen, Monopoleksemplet og OL-Veiviseren.

Tenk gjennom hvilke "brikker" må på plass i programvaren for å løse orn av de sentrale funksjonelle krav.

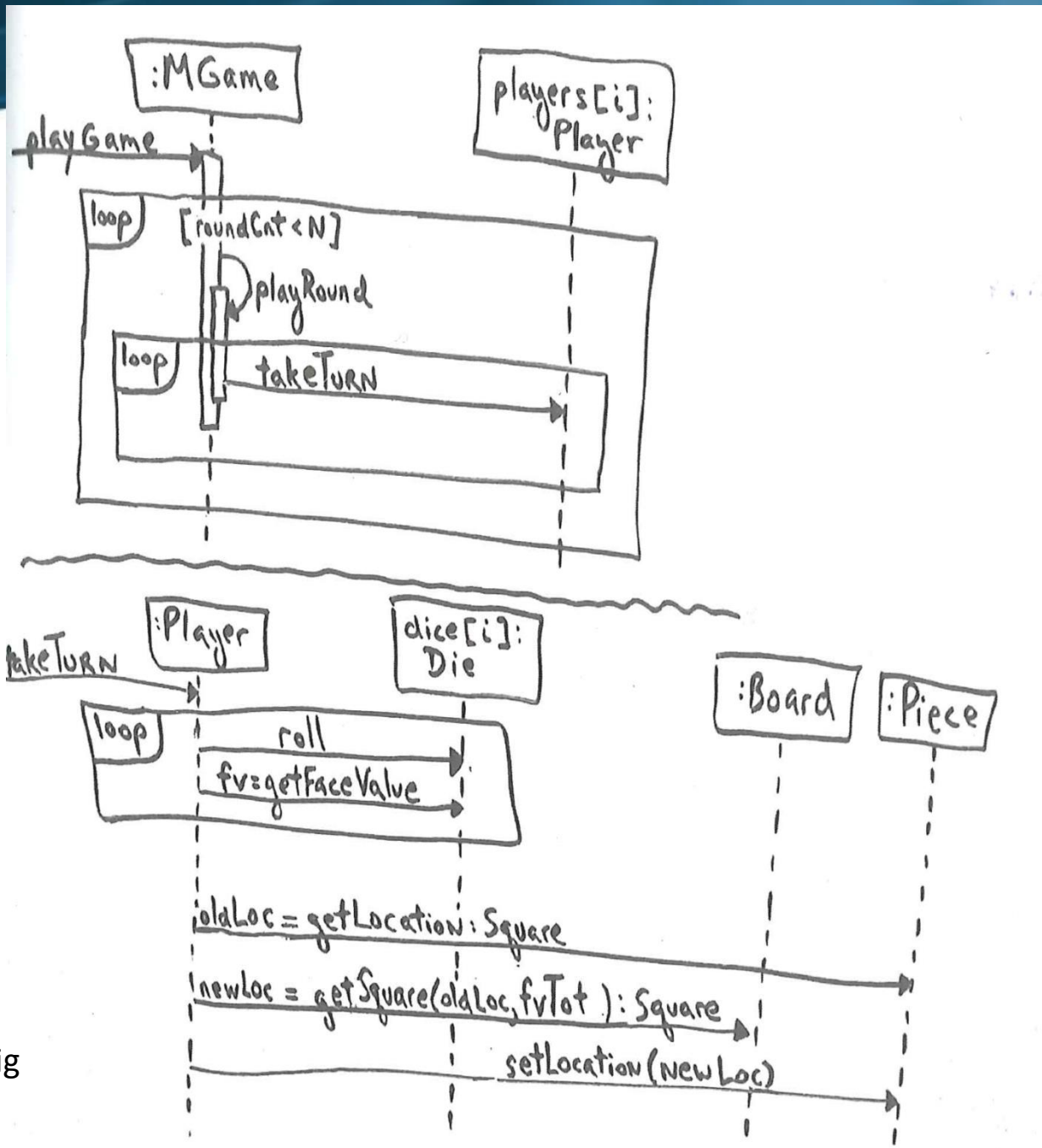
- skisser sekvensdiagram for sentrale Use Case

(Lån bok, Send purring)

(Sett opp spill, Spill omgang)

(Vis Ti-på-topp, Generer reiserute)

- ta utgangspunkt i Konseptuelt klassediagram og legg på operasjoner på klassene + legg inn nye programvarespesifikke klasser (grensesnittklasser og kontrollerklasser)



Craig Larman, Applyig
UML And Patterns

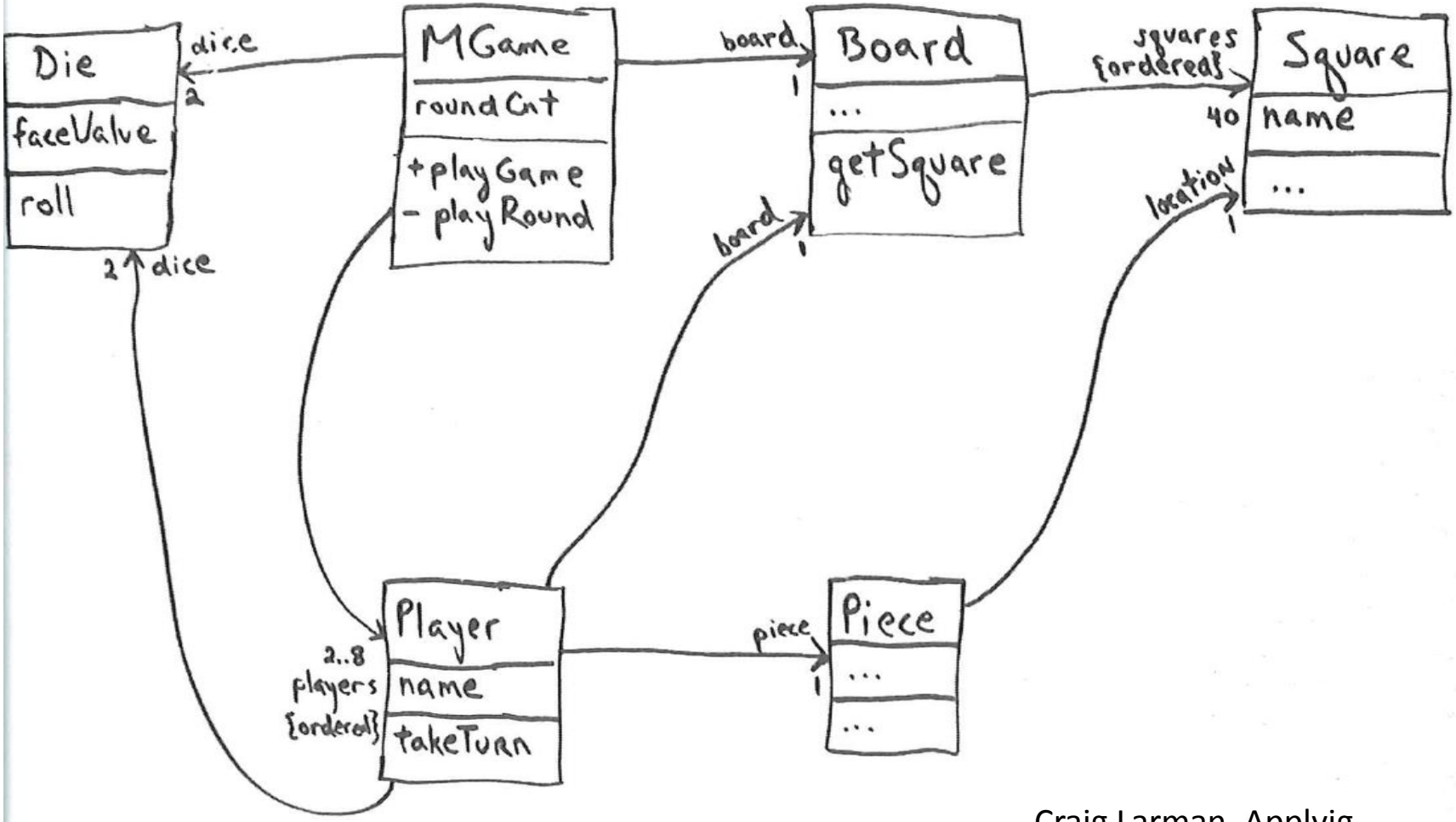


Figure 18.26 Static design for *playGame*.

Craig Larman, Applyig UML And Patterns

GRASP # 1 : Creator (hentet fra Craig Larman)

Problem : Hvem skal ha ansvaret for å kreere nye instanser av en klasse

Løsning : Gi klassen B ansvaret for å kreere instanser av klassen A hvis :
B aggregerer / inneholder A, B arkiverer A objekter,
B hyppig bruker A, B har initialiseringsdata for A

Eksempel : Salg aggregerer Salgslinje, og bør ha ansvar for at det kreeres instanser av Salgslinje

Diskusjon : Legger ansvaret for å kreere instanser til et objekt som uansett skal være tett knyttet til den aktuelle objektet.

Motforestillinger: Dette er en ”enkel” og rimelig ”lokal” løsning. Omstendighetene rundt kreering kan ofte være kompleks, og en delegering av ansvaret for kreering til spesialiserte klasser håndteres av ulike Design Patterns.

Fordeler : Ivaretar lave koblinger og dermed vedlikeholdbarhet og robusthet

GRASP # 2 : Information Expert (Craig Larman)

Problem : Hva bør være det generelle prinsipp når det gjelder å knytte ansvar til objekter

Løsning : Plasser ansvaret på den klassen som har nok informasjon til å oppfylle ansvaret. (Gjennom “doing” og “knowing”)

Eksempel : Salg er informasjonsekspert for Totalsum, Salgslinjen er informasjonsekspert for delsummene. Varen er ekspert på prisen.

Diskusjon : I stor grad intuisjonsbasert OO

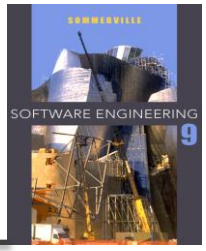
Motforestillinger: Kommer enkelte ganger i konflikt med idealet om Lave Koblinger og Høy Styrke

Fordeler : Vedlikeholdbarhet og robusthet i løsningen

Beslektede patterns / prinsipper : Low Coupling og High Cohesion

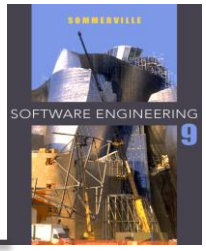
Kjent som : ”Do It Myself” , ”Place responsibilities with data”

DesignPattern : The Observer pattern (1)



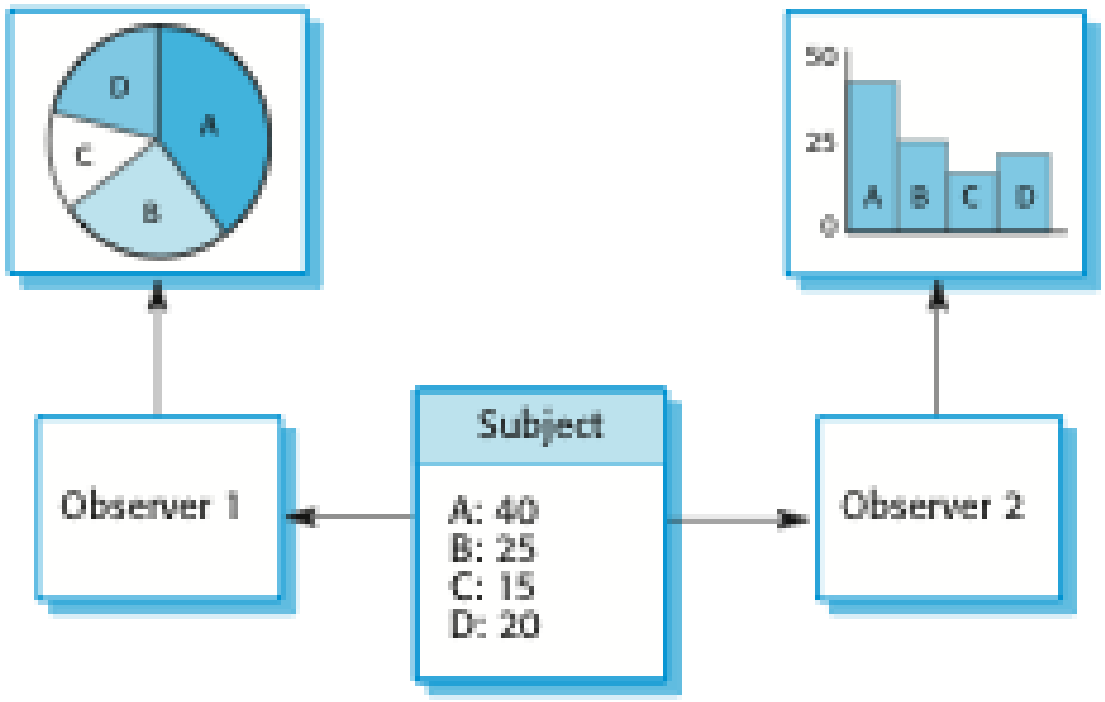
Pattern name	Observer
Description	Separates the display of the state of an object from the object itself and allows alternative displays to be provided. When the object state changes, all displays are automatically notified and updated to reflect the change.
Problem description	In many situations, you have to provide multiple displays of state information, such as a graphical display and a tabular display. Not all of these may be known when the information is specified. All alternative presentations should support interaction and, when the state is changed, all displays must be updated. This pattern may be used in all situations where more than one display format for state information is required and where it is not necessary for the object that maintains the state information to know about the specific display formats used.

The Observer pattern (2)

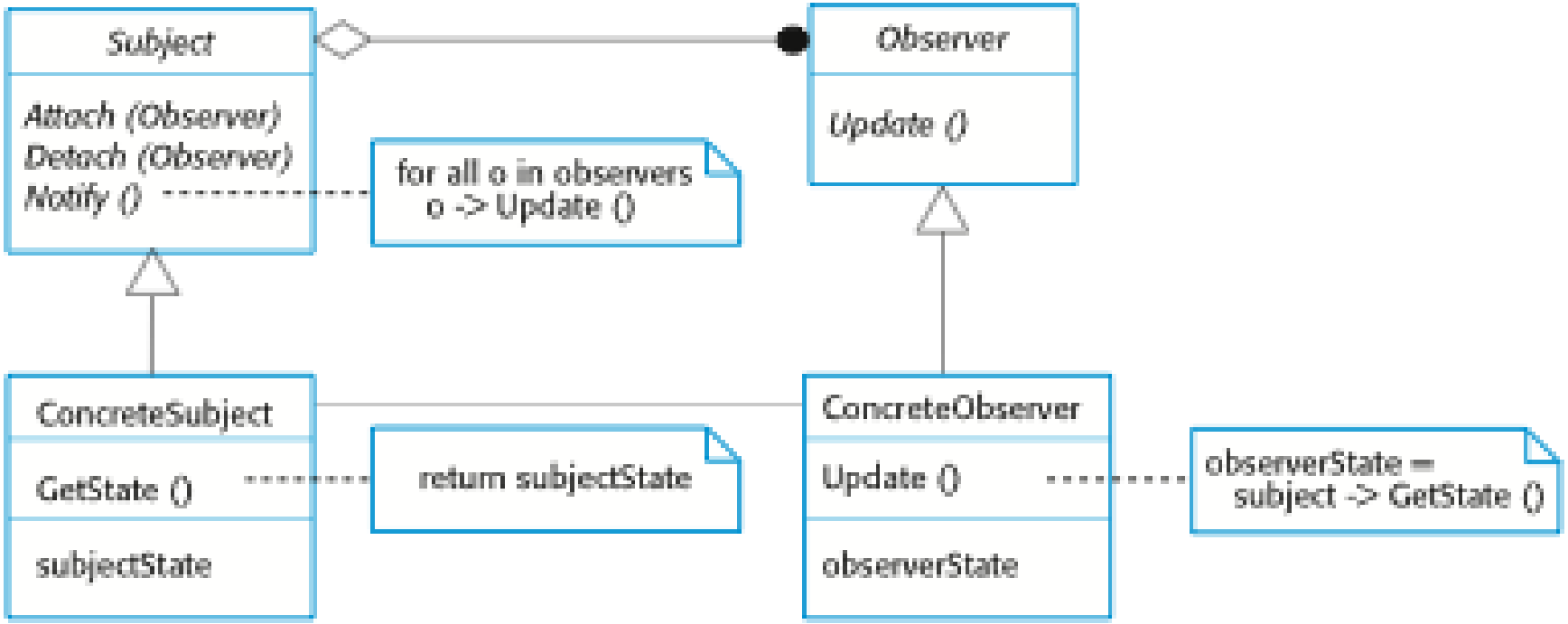


Pattern name	Observer
Solution description	This involves two abstract objects, Subject and Observer, and two concrete objects, ConcreteSubject and ConcreteObject, which inherit the attributes of the related abstract objects. The abstract objects include general operations that are applicable in all situations. The state to be displayed is maintained in ConcreteSubject, which inherits operations from Subject allowing it to add and remove Observers (each observer corresponds to a display) and to issue a notification when the state has changed. The ConcreteObserver maintains a copy of the state of ConcreteSubject and implements the Update() interface of Observer that allows these copies to be kept in step. The ConcreteObserver automatically displays the state and reflects changes whenever the state is updated.
Consequences	The subject only knows the abstract Observer and does not know details of the concrete class. Therefore there is minimal coupling between these objects. Because of this lack of knowledge, optimizations that enhance display performance are impractical. Changes to the subject may cause a set of linked updates to observers to be generated, some of which may not be necessary.

Multiple displays using the Observer pattern



A UML model of the Observer pattern



IMT2243 Systemutvikling

Tema : Testing

- Hvorfor tester vi i programvareutviklingsprosjekter ?
- Hva vil det si å teste ?
- Hvordan tester dere det dere utvikler pr idag ?
- Testaktiviteten i ulike omgivelser
  - (SU-modell, type applikasjon, risikobildet)
- Hvem skal teste ? Og når bør vi teste ?
- Hva er inspeksjoner ? Sammenheng med testing ?
- Hvordan skal vi legge opp testingen ?
  - V-modellen, strategier, metoder og teknikker for testing

Pensum : Sommerville kap 2.2.3, kap 8 kap 24.3 + wikipedia om V-modellen
http://en.wikipedia.org/wiki/V-Model_(software_development)