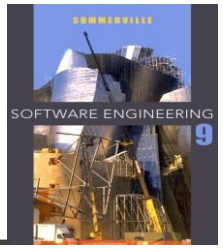
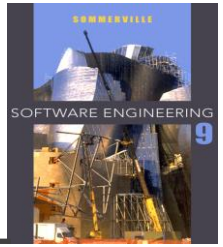


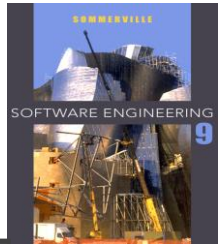
Poglavlje 21 – Aspektno-orijentisan Razvoj Softvera
I deo

Teme



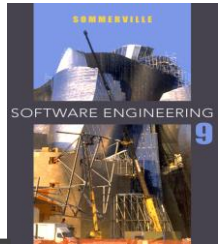
- Podela zahteva
- Aspekti, tačke spajanja, tačke priključivanja
- Razvoj softvera korišćenjem aspekata

Aspektno-orijentisan Razvoj Softvera



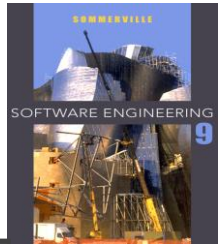
✧Pristup razvoju softvera koji se zasniva na relativno novom tipu apstrakcije – aspektu.
✧Koristi se u saradnji sa drugim pristupima – najčešće objektno-orijentisanim razvojem softvera.
✧Aspekt enkapsulira funkcionalnosti koje se preklapaju i sarađuju sa drugim funkcionalnostima.
✧Aspekti sadrže definicije gde treba da se nalaze u programu kao i implementacioni kod.

Podela zahteva



✧Princip podele zahteva : softver treba da bude tako organizovan da svaki element programa radi samo jednu stvar.
✧Svaki element programa treba da bude razumljiv bez referenciranja na druge elemente.
✧Apstrakcija programa se može izvršiti pomoću rutina, procedura, objekata.

Zahtevi



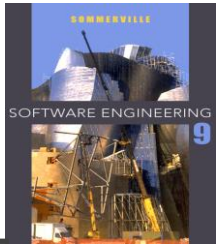
✧Zahtevi predstavljaju ciljeve i prioritete ulagača i naručioca sistema.

- Primeri zahteva mogu se odnositi na performanse, sigurnost, specifične funkcionalnosti.

✧Podelom zahteva može se pratiti tok implementacije.

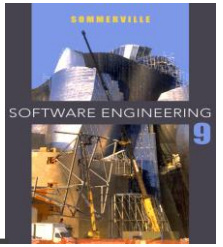
✧Glavni zahtevi su funkcionalni zahtevi koji se odnose na primarnu svrhu sistema, sporedni zahtevi su funkcionalni zahtevi koji se ne odnose na primarnu svrhu sistema.

Zahtevi ulagača



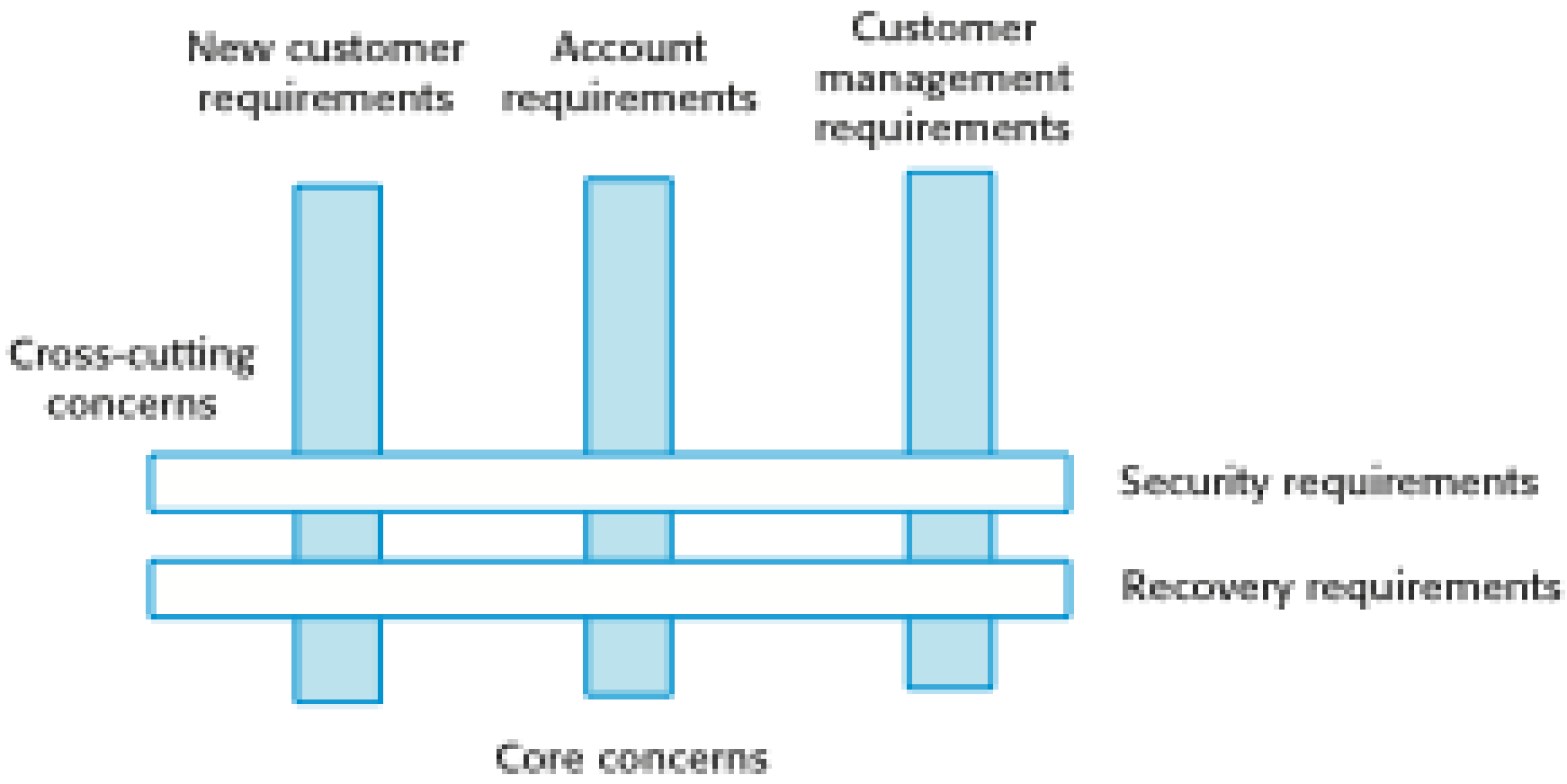
✧Definisanje funkcionalnih zahteva koji opisuju specifične funkcionalnosti koje treba da se nalaze u sistemu.
✧Definisanje zahteva koji opisuju ponašanje sistema
✧Politika korišćenja sistema
✧Definisanje sistemskih zahteva koji se odnose na atribute, održavanje i konfigurisanje sistema.
✧Definisanje organizacionih zahteva koji se odnose na ciljeve organizacije i prioritete kao što su finansiranje sistema, korišćenje postojećeg softvera i održavanje reputacije organizacije.

Preklapajući zahtevi

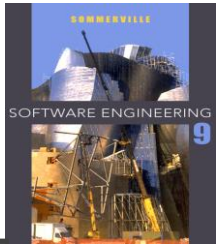


✧Preklapajući zahtevi su zahtevi čija se implementacija prostire kroz nekoliko komponenti sistema.
✧Problem nastaje kada dođe do promene zahteva. Kod koji mora biti promenjen nije lokalizovan već se nalazi na nekoliko mesta u sistemu.
✧Preklapanje vodi ka zapetljavanju i rasejavanju.

Preklapajući zahtevi



Raspetljavanje upravljanja baferom i sinhronizacija koda

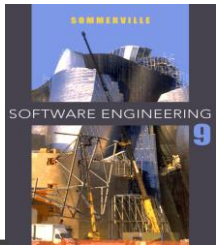


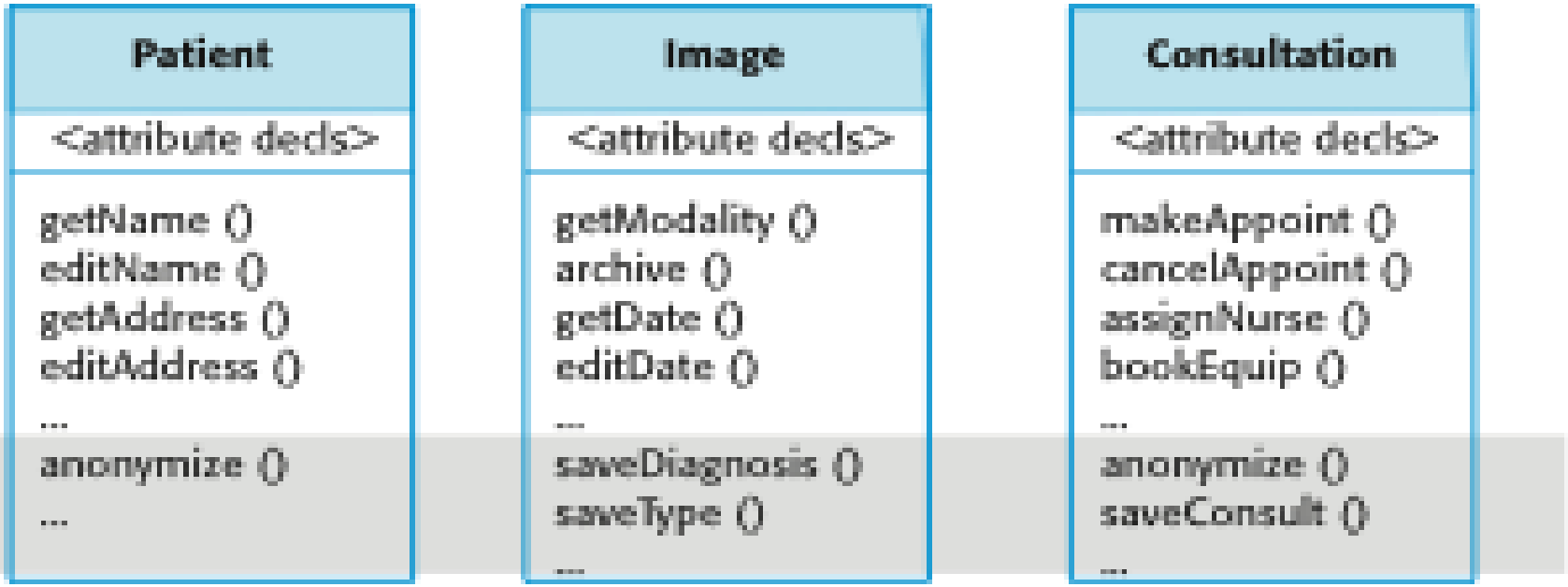

```
synchronized void put (SensorRecord rec )
{
// Check that there is space in the buffer; wait if not
```

```
// Add record at end of buffer
store [back] = new SensorRecord (rec.sensorId,
rec.sensorVal) ;
back = back + 1 ;
// If at end of buffer, next entry is at the beginning
if (back == bufsize)
back = 0 ;
numberOfEntries = numberOfEntries + 1 ;
// indicate that buffer is available
```

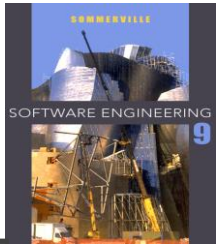
```
} // put
```

Rasejavanje metoda koji implementiraju sporedne zahteve



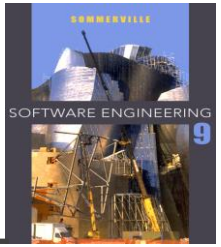


Aspekti, tačke spajanja, tačke priključivanja



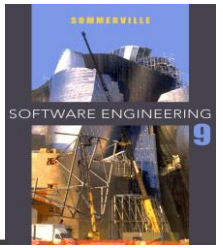
✧Aspekt je apstrakcija koja implementira zahtev. Sadrži informacije gde bi trebalo da se nalazi u programu.
✧Tačka spajanja je mesto u programu gde aspekt može biti uključen.
✧Tačke priključivanja definišu gde(na kojoj tački spajanja) aspekt može biti uključen.

Terminologija koja se koristi u aspektno-orijentisanom razvoju softvera



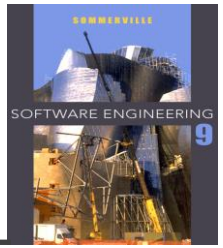
Term	Definition
advice	The code implementing a concern.
aspect	A program abstraction that defines a cross-cutting concern. It includes the definition of a pointcut and the advice associated with that concern.
join point	An event in an executing program where the advice associated with an aspect may be executed.
join point model	The set of events that may be referenced in a pointcut.
pointcut	A statement, included in an aspect, that defines the join points where the associated aspect advice should be executed.
weaving	The incorporation of advice code at the specified join points by an aspect weaver.

Aspekt provere identiteta



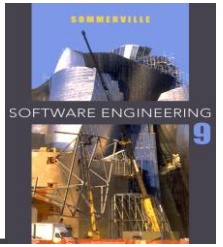

```
aspect authentication
{
        before: call (public void update* (..))   // this is a pointcut
        {
                // this is the advice that should be executed when woven into
                // the executing system
                int tries = 0 ;
                string userPassword = Password.Get ( tries ) ;
                while (tries < 3 && userPassword != thisUser.password ( ) )
                {
                        // allow 3 tries to get the password right
                        tries = tries + 1 ;
                        userPassword = Password.Get ( tries ) ;
                }
                if (userPassword != thisUser.password ( )) then
                        //if password wrong, assume user has forgotten to logout
                        System.Logout (thisUser.uid) ;
        }
} // authentication
```

AspectJ – model tačke spajanja



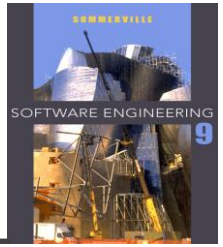
- Call events
  - Poziv metoda ili konstruktora
- Execution events
  - Izvršavanje metoda ili kontruktora
- Initialisation events
  - Inicijalizacija klase ili objekta
- Data events
  - Pristup ili ažuriranje polja
- Exception events
  - Rukovanje izuzecima

Tačke priključivanja(Pointcuts)



- Identifikuje događaje i tačke u kojima može biti spojen sa programom.
- Primeri gde događaj može biti povezan sa programom:
  - Pre izvršenja specifičnog metoda
  - Nakon normalnog izlaska iz metode ili nakon izuzetka
  - Kada je polje u objektu modifikovano

Povezivanje aspekata

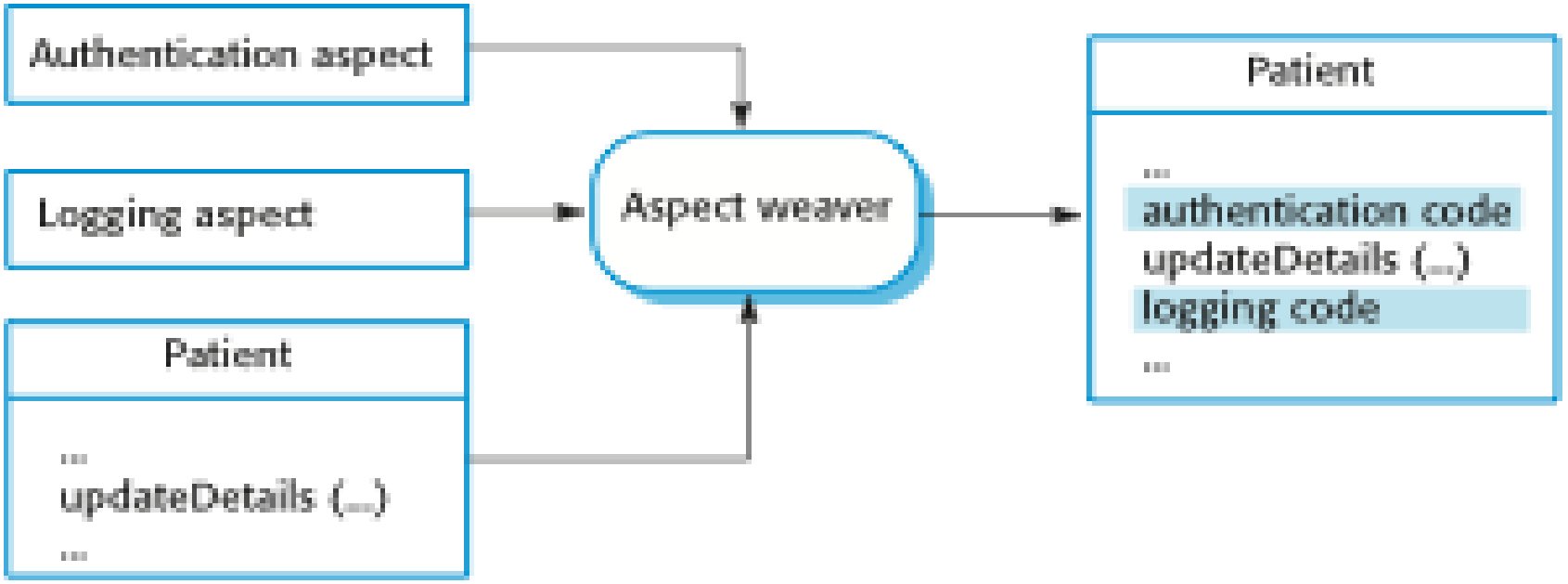


✧Aspekt se povezuje sa postojećim kodom na definisanim tačkama priključivanja.

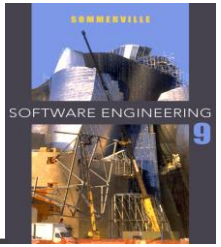
✧Tri pristupa povezivanju aspekata

- Pre-procesiranje izvornog koda
- Vremensko povezivanje
- Dinamičko, povezivanje u toku izvršavanja

Povezivanje aspekata

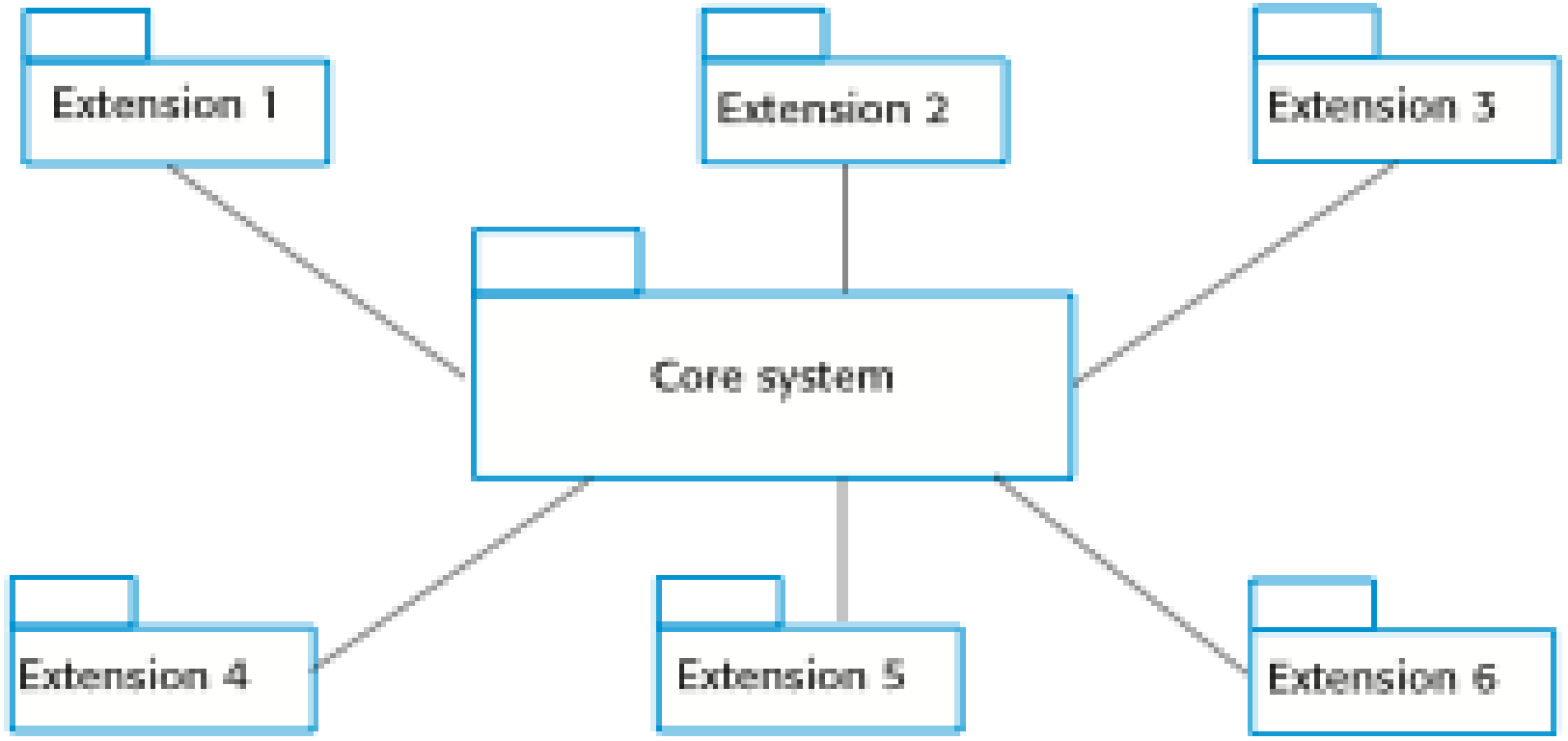


Razvoj softvera korišćenjem aspekata

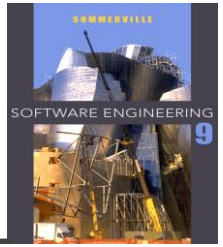


✧Aspekti su predstavljeni kao koncept programiranja, zato što se formiraju na osnovu zahteva, aspektno-orijentisani pristup može biti usvojen u svim fazama razvoja sistema.
✧Arhitektura aspektno-orijentisanog sistema je zasnovana na jezgru sistema plus ekstenzije.
✧Jezgro sistema implementira primarne zahteve.
✧Ekstenzija implementira sporedne i preklapajuće zahteve.

Jezgro sistema sa ekstenzijama

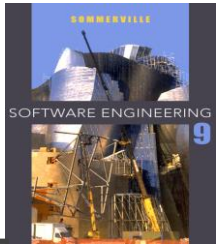


Tipovi ekstenzija

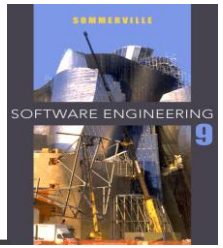


- Sporedne funkcionalne ekstenzije
  - Dodatne funkcionalnosti sistema
- Političke ekstenzije
  - Dodaje funkcionalnosti koje su u skladu sa organizacionom politikom, npr. sigurnost
- Infrastrukturne ekstenzije
  - Dodaje funkcionalnosti za podršku implementacije sistema na različitim platformama.

Zaključak

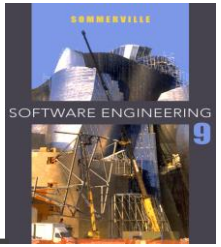


✧Aspektno-orijentisan pristup razvoju softvera podržava podelu zahteva. Predstavljajući preklapajuće zahteve kao aspekte, pojedinačni zahtevi mogu bit razumljivi, ponovo korišćeni i modifikovani bez promena u ostatku sistema.
✧Rasejavanje se javlja kada modul u sistemu sadrži kod koji se proteže kroz nekoliko komponenti. Zapetljavanje se javlja kada kod implementira više zahteva.
✧Aspekti sadrže tačke priključivanja koje definišu gde aspekti mogu biti sadržani u programu. Tačke spajanja mogu biti referencirane pomoću priključnih tačaka.
✧Da bi se osigurala podela zahteva, sistem može biti dizajniran tako da implementira glavne zahteve ulagača, i skup ekstenzija koje implementiraju sporedne zahteve.



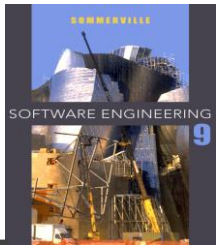
Poglavlje 21 – Aspektno-orijentisan Razvoj Softvera
II deo

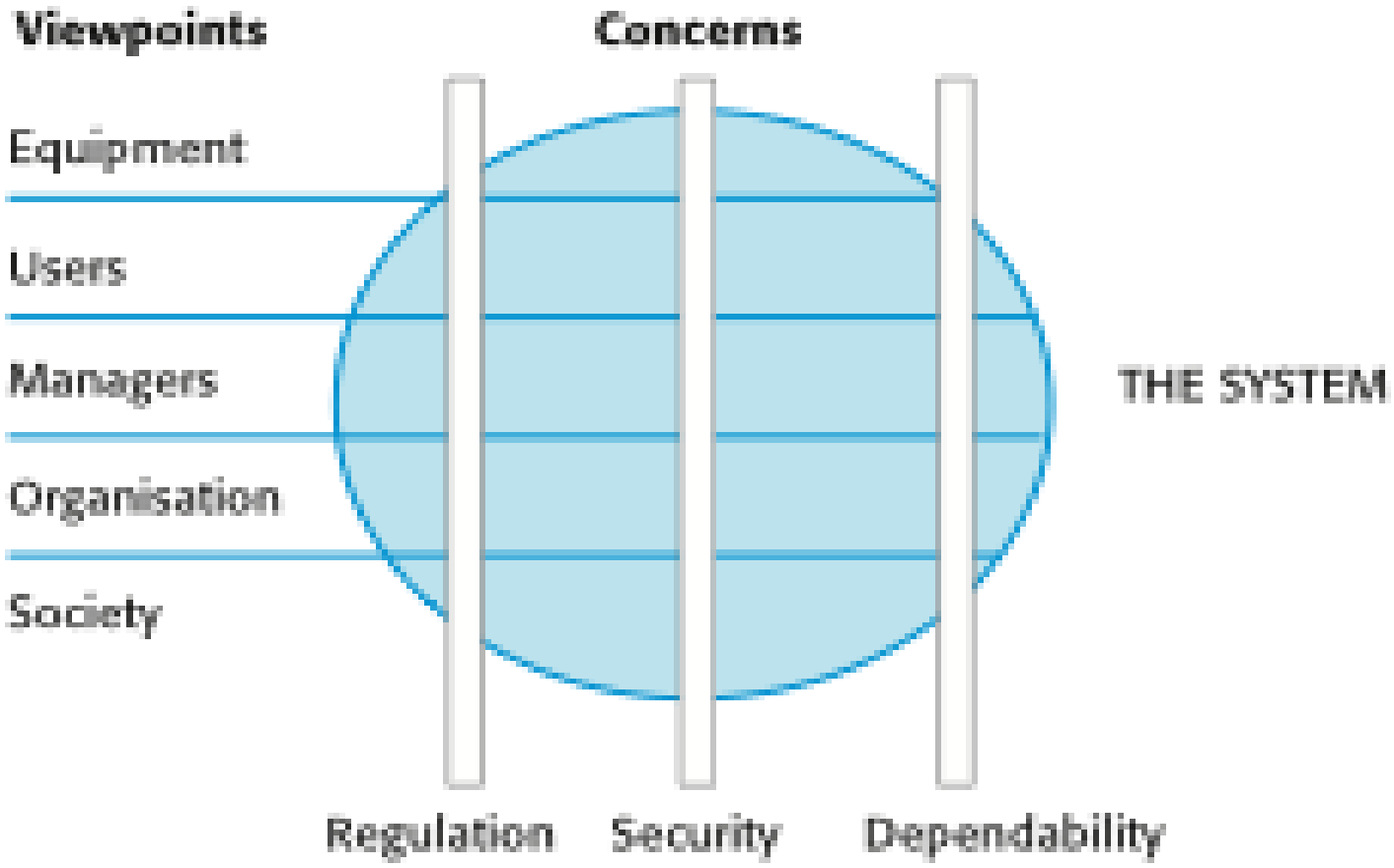
Razvoj softvera orijentisan zahtevima



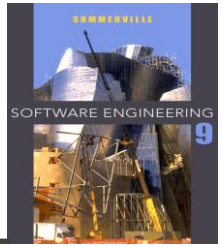
✧Pristup razvoju softvera koji se fokusira na zahteve korisnika i koristi se u sklopu aspektno-orijentisanog razvoja.
✧Pogledi su način za razdvajanje zahteva od različitih korisnika i ulagača
✧Pogledi predstavljaju zahteve od različitih grupa korisnika i ulagača
✧Preklapajući zahtevi mogu biti identifikovani iz svih pogleda.

Odnos pogleda i zahteva





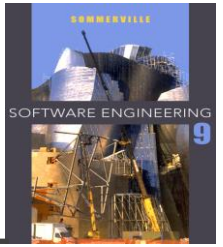
Pogledi na sistem za praćenje inventara



1. **Emergency service users**
1.1 Find a specified type of equipment (e.g., heavy lifting gear)
1.2 View equipment available in a specified store
1.3 Check-out equipment
1.4 Check-in equipment
1.5 Arrange equipment to be transported to emergency
1.6 Submit damage report
1.7 Find store close to emergency

2. **Emergency planners**
2.1 Find a specified type of equipment
2.2 View equipment available in a specified location
2.3 Check-in/cCheck out equipment from a store
2.4 Move equipment from one store to another
2.6 Order new equipment

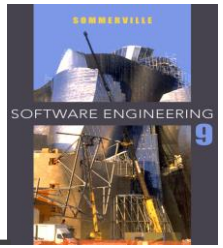
Pogledi na sistem za praćenje inventara



3. **Maintenance staff**

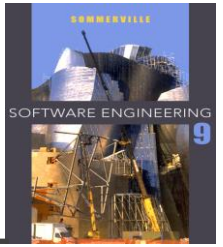
3.1	Check -in/cCheck -out equipment for maintenance
3.2	View equipment available at each store
3.3	Find a specified type of equipment
3.4	View maintenance schedule for an equipment item
3.5	Complete maintenance record for an equipment item
3.6	Show all items in a store requiring maintenance

Zahtevi za načine pristupa sistemu za praćenje inventara



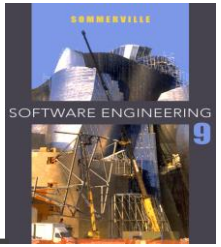
AV.1 There shall be a ‘hot standby’ system available in a location that is geographically well-separated from the principal system.
Rationale: The emergency may affect the principal location of the system.
AV.1.1 All transactions shall be logged at the site of the principal system and at the remote standby site.
Rationale: This allows these transactions to be replayed and the system databases made consistent.
AV.1.2 The system shall send status information to the emergency control room system every five minutes.
Rationale: The operators of the control room system can switch to the hot standby if the principal system is unavailable.

Sistem za praćenje inventara – glavni zahtevi



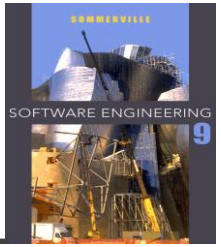
✧C.1 Sistem treba da dozvoli autorizovanim korisnicima da vide opis bilo kog dela inventara.
✧C.2 Sistem treba da ima opciju pretrage kompletnog inventara za autorizovane korisnike.

Sistem za praćenje inventara – sporedni zahtevi



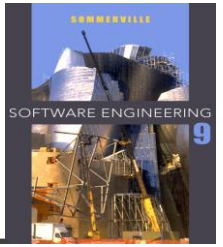
✧E1.1 Treba omogućiti autorizovanim korisnicima da prave narudžbenice za rezervne delove od akreditovanih nabavljača.
✧E1.1.1 Kada je neki deo naručen, treba da bude zaveden u specifičnom inventaru i obeležen kao „naručen“.

Aspektno-orijentisan dizajn/programiranje



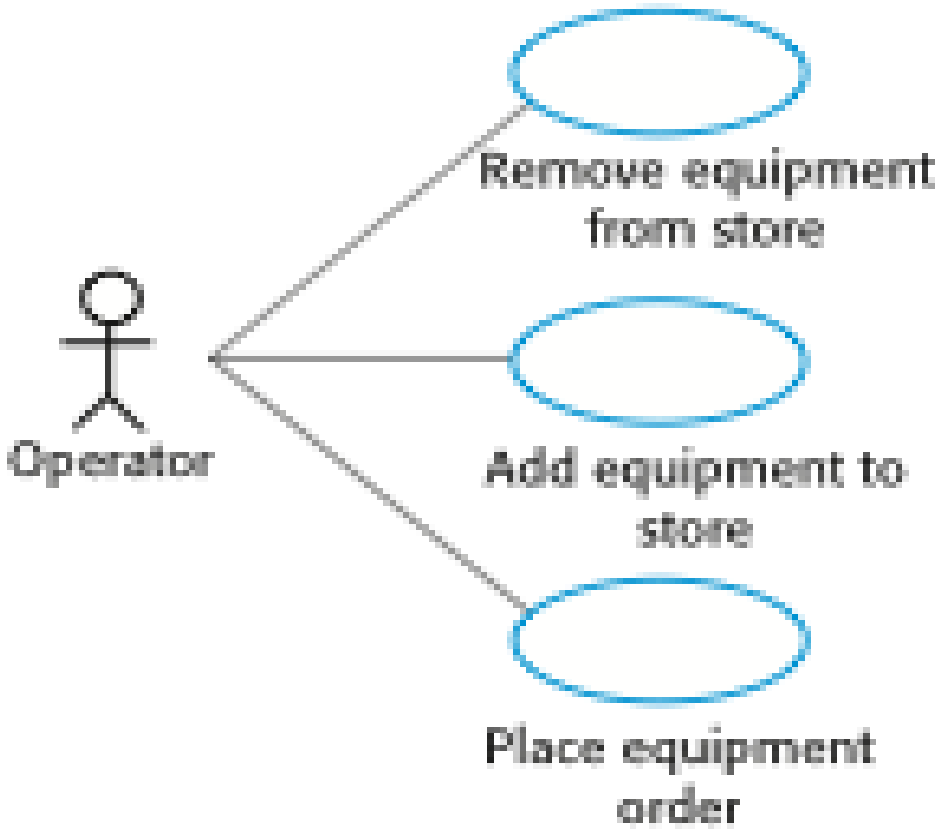
- Aspektno-orijentisan dizajn
  - Proces dizajniranja sistema koji koristi aspekte za implementaciju preklapajućih zahteva i ekstenzija, koji se identifikuju tokom analize zahteva.
- Aspektno-orijentisano programiranje
  - Za implementaciju aspektno-orijentisanog dizajna koristi se aspektno-orijentisani jezik kao što je AspectJ.

Slučajevi upotrebe

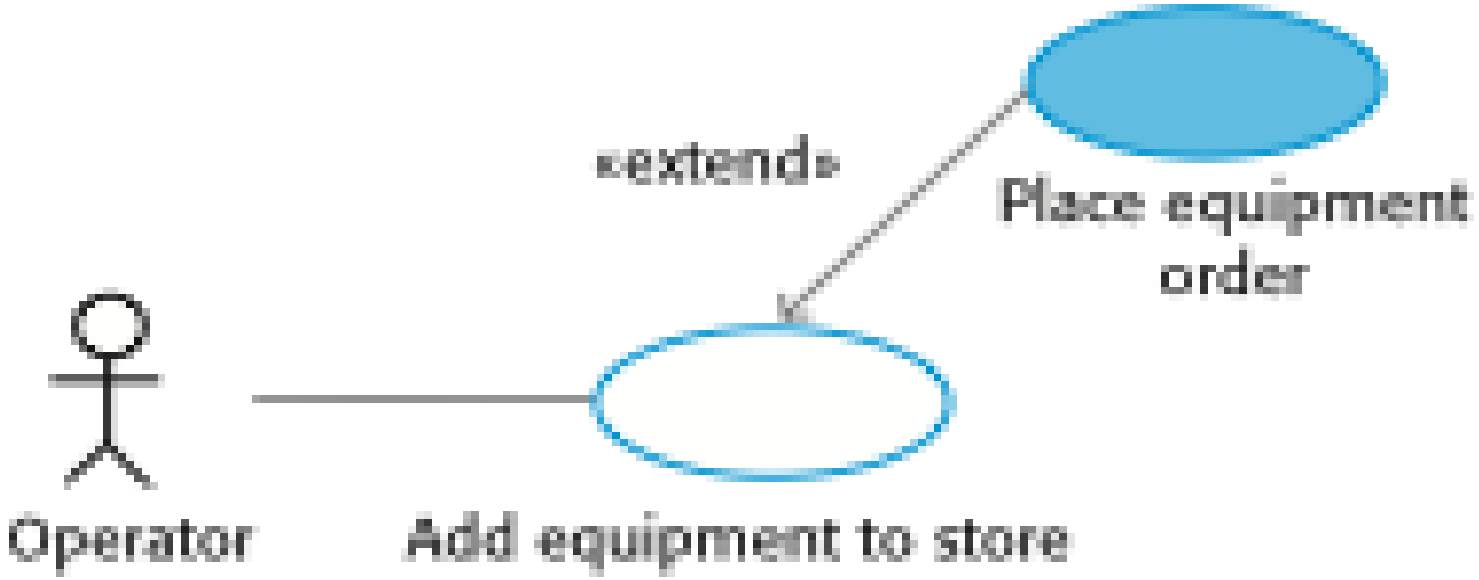


- Slučajevi upotrebe mogu poslužiti kao osnova za aspektno-orijentisani razvoj.
- Svaki slučaj upotrebe predstavlja jedan aspekt.
  - Slučajevi upotrebe ekstenzija se prirodno uklapaju u model sistema.

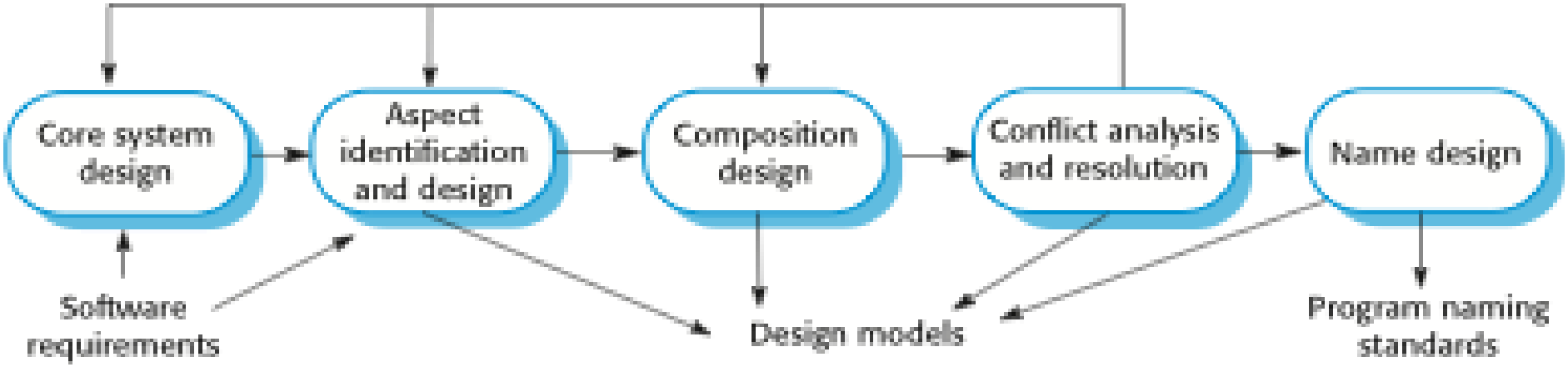
Slučajevi upotrebe sistema za upravljanje inventarom



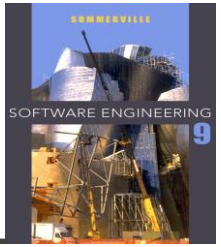
Slučajevi upotrebe ekstenzija



Proces aspektno-orijentisanog dizajniranja

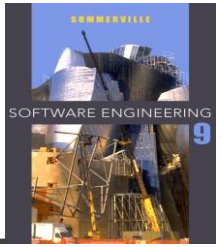


Dizajniranje



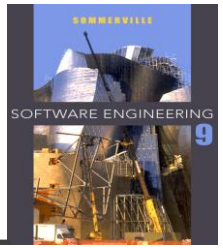
✧*Dizajn jezgra sistema* je mesto gde je dizajnirana arhitektura sistema tako da podržava glavne funkcionalnosti sistema.
✧*Identifikacija i dizajn aspekta,* nakon identifikovanja ekstenzija u zahtevima, treba ih dodatno analizirati da bi se ustanovilo da li su i ekstenzije aspekti ili ih treba podeliti u nekoliko aspekata.
✧*Dizajn sastavljanja,* u ovoj fazi neophodno je analizirati jezgro sistema i aspekte dizajna da bi se utvrdilo koji od aspekata treba da uđe u sastav jezgra sistema. Identifikuju se tačke priključivanja u programu na koje će određeni aspekt biti priključen.

Dizajniranje



✧*Analiza i razrešavanje konflikta,* do konflikta dolazi kada nekoliko preklapajućih aspekata ulazi u koliziju sa drugim aspektima, kada treba da budu priključeni na istim tačkama spajanja.
✧*Dizajniranje imena,* je izuzetno važno da bi se izbegao problem slučajnih tačaka priključivanja.Do toga dolazi kada u programu neka tačka spajanja ima isti naziv kao i tačka priključivanja.

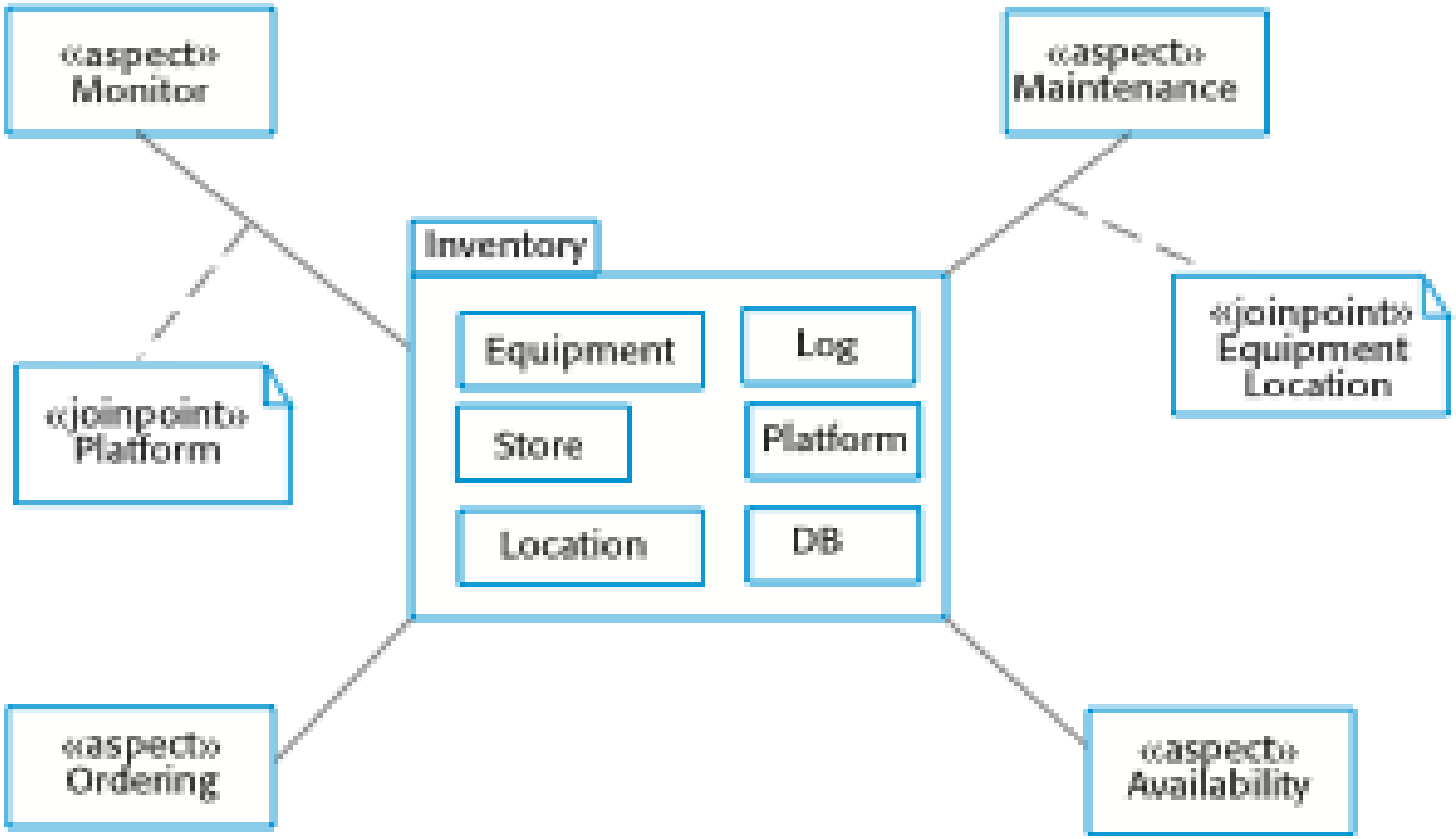
UML ekstenzije



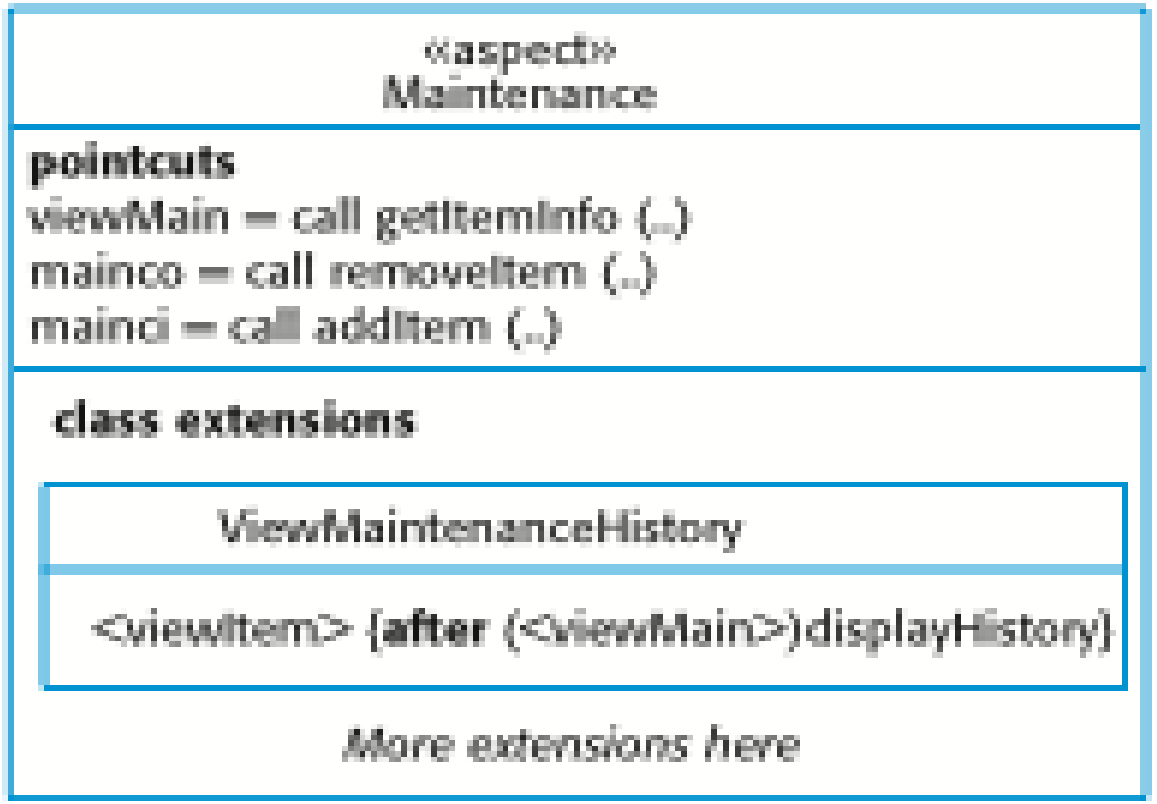
✧Predstavljanje aspektno-orijentisanog dizajna korišćenjem UML-a:

- Modelovanje aspekata korišćenjem UML tipova.
- Definisanje tačaka spajanja sa jezgrom sistema.

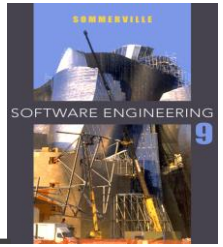
Dizajn aspektno-orijentisanog modela



Deo modela jednog aspekta

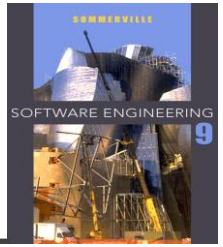


Ekstenzija



✧U metodu viewItem, nakon poziva metode getItemInfo, poziv metode displayHistory treba da ispiše istorijat promena.

Verifikacija i validacija



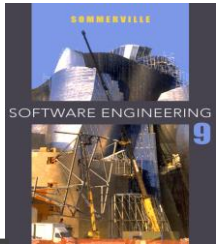
✧Proces provere da program ispunjava zadatu specifikaciju(verifikacija) i da ispunjava realne potrebe ulagača i korisnika(validacija)

✧Kao i svi ostali sistemi, aspektno-orijentisani sistemi mogu biti testirani kao crne-kutije, korišćenjem specifikacije za formiranje testova.

✧Inspekcija programa i 'white-box' testiranje koje se oslanja na izvorni kod je problematično.

✧Aspekti uvode dodatne probleme prilikom testiranja.

Problemi prilikom inspekcije programa

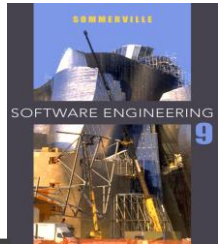


✧Za efektivnu proveru programa (pisanog u nekom od konvencionalnih jezika) mora se čitati s leva na desno, i odozgo na dole.

✧Aspekti čine ovo nemogućim, zato što je program pre web nego sekvencijalni dokument. Iz izvornog koda se ne može zaključiti gde će aspekt biti priključen ili izvršen.

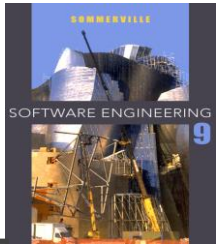
✧Ispravljanje aspektno-orijentisanog programa da bude pogodan za čitanje je praktično nemoguće.

White box testiranje



- Cilj white box testiranja je upotreba izvornog koda za dizajniranje testova.
- Aspektni problemi
  - Kako se izvorni kod može iskoristiti za formiranje testova?
  - Kakva je uloga testiranja?

Aspektni problemi

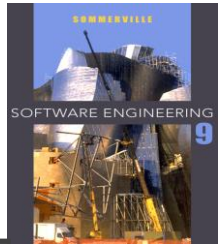


✧Pravljenje dijagrama toka za program baziran na aspektima je nemoguće. Zbog toga je teško sistematično dizajnirati testove koji će pokriti sve kombinacije izvršavanja aspekata.

✧Šta znači pokrivanje testovima?

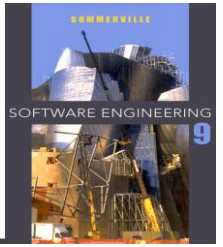
- Kod svakog aspekta je izvršen barem jednom?
- Kod svakog aspekta izvršen jednom na svakoj tački spajanja?
- ???

Testiranje problema sa aspektima



✧Kako aspekti treba da budu definisani da bi bilo moguće formirati testove?
✧Kako aspekti mogu biti testirani nezavisno od sistema?
✧Kako smetnje aspekata mogu biti testirane?
✧Kako mogu biti dizajnirani testovi tako da se ispitaju sve tačke spajanja i da je primenjen na odgovarajućim aspektima.

Zaključak



✧Za prepoznavanje zahteva ulagača i preklapajućih zahteva moguće je koristiti poglede

✧Prelaz sa zahteva na dizajniranje može biti izvršen definisanjem slučajeva upotrebe, gde svaki slučaj upotrebe predstavlja jedan zahtev korisnika ili ulagača.

✧Problem inspekcije i pravljenja testova za aspektno-orijentisane programe je značajna barijera za prihvatanje aspektno-orijentisanog razvoja softvera u velikim projektima.