Razvoj softvera zasnovan na komponentama

Kocić Ognjen 1142/2013

Motivacija

- Čist objektno-orijentisani pristup je promašaj
  - Softver treba pisati onako kako doživljavamo problem – kroz objekte bliske ljudima
  - Iz enkapsulacije sledi modularnost
  - Nasleđivanje omogućava da se znanje prenosi na podklase (efikasno rešenje?)
  - Komentar
- Potraga za pristupom koji će doneti traženu ponovnu iskoristivost izvornog koda
  - Prodaja velikih klasa kao nezavisnih komponenti je bila nemoguća
  - Potreba za apstraktnijim jedinicima koda od objekata
  - Ideja nezavisnih interfejsa

Šta je razvoj softvera zasnovan na komponentama?

- Razvoj softvera zasnovan na komponentama je proces definisanja, implementacije i integracije ili uklapanja nezavisnih komponenti u kompletan sistem.
- Lakše ga je shvatiti kroz osnovne koncepte razvoja:
  - Nezavisne komponente koje su u potpunosti određene javim intefejsom
  - Postoji standardizacija interfejsa u cilju jednostavnije integracije komponenti
  - Postoji posrednički sloj zadužen za komunikaciju
  - Proces razvijanja mora da bude prilagođen ovoj vrsti razvoja softvera

Šta direktno sledi iz osnovnih koncepata?

- Nezavisne komponente koje su u potpunosti određene javim intefejsom
  - To znači da možemo imati više implementacija iste komponente pri čemu će se svaka jednako dobro uklopiti u sistem (zašto je ovo super?).
- Postoji standardizacija interfejsa u cilju jednostavnije integracije komponenti
  - Ovo se odnosi na to da će sve komponente definisati jedan zajednički deo interfejsa koji će služiti za lakšu komunikaciju , održavanje i korišćenje.
- Postoji posrednički sloj zadužen za komunikaciju
  - Postoji jako puno komunikacionih problema niskog nivoa o kojima ne brinemo.
- Proces razvijanja mora da bude prilagođen ovoj vrsti razvoja softvera
  - Zavisi od toga da li razvijamo komponente za druge aplikacije ili razvijamo aplikaciju koristeći postojeće komponente.

Poznatiji modeli komponenti

- Enterprise Java Beans (Sun)
  - Originalna specifikacija razvijena od strane IBM-a 1997.
  - Prvu verziju razvio Sun 1999., proširenja kroz Java Community Process
- COM (Microsoft)
  - Razvijen od strane Microsoft-a 1993., predstavlja osnovu za Microsoft Runtime
  - Interfejs na nivou mašinskog koda koji omogućava komunikaciju između procesa i dinamičko kreiranje objekata iz različitih programskih jezika
- .NET (Microsoft)
  - Predstavlja na određeni način omotač oko najkorišćenijih COM funkcionalnosti
- CORBA Component Model (Wang, C. Schmidt, and O'Ryan)
  - Definisan od strane Object Management Group 1991. Različita težina mapiranja.

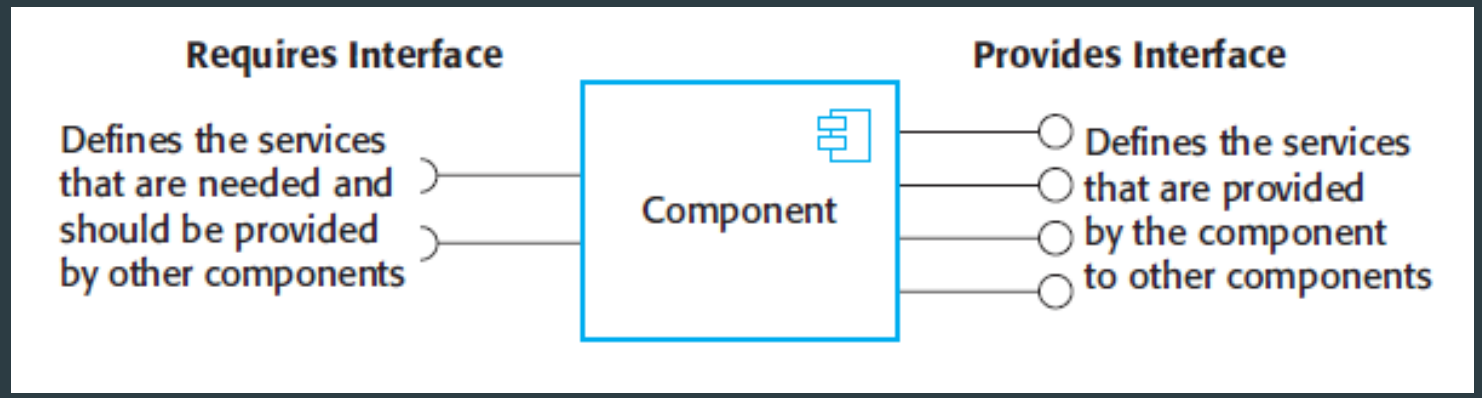
Šta je komponenta?

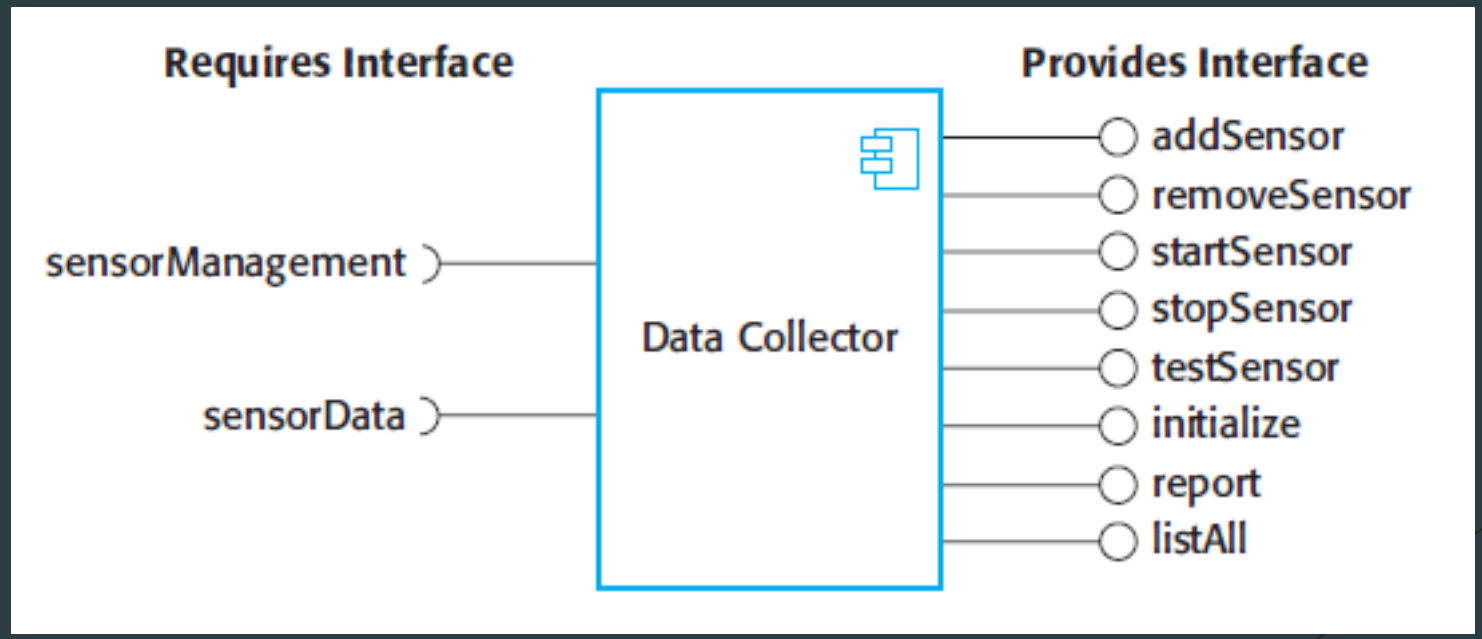
- Koliko autora toliko definicija, bolje definisanje kroz poželjne osobine.
- Ono što je zajedničko svim definicijama je da je komponenta nezavisna jedinica softvera koju je moguće ukolpiti sa drugim komponentama da bi se proizveo jedan funkcionalan softverski sistem.
- Osobine:
  - Standardizovanost
  - Nezavisnost
  - Jednostavna za uklapanje
  - Zaokružena celina (u smislu da je deployable eng.)
  - Dokumentovanost

Realizacija interfejsa komponente

- Podsećanje: Osnovna karakteristika komponente je njen interfejs!
- Čest pogled na komponente je da one pružaju određene vrste usluga
  - Ne moramo znati njen izvorni kod, već je pozivamo kao spoljašnju uslugu ili direktno uključujemo (linkujemo) u program
  - Bitan je samo interfejs, ne zanima nas unutrašnje stanje komponente
- Sledi logična podela interfejsa na dva dela:
  - Usluge koje komponenta pruža
    - To je praktično API komponente
  - Usluge koje komponenta potražuje da bi pružila tražene usluge
    - Ne utiče na nezavisnost i celovitost komponente jer je akcenat na tome šta se potražuje a ne kako će to biti pruženo

Opšta šema i primer





Šta je model komponenti?

- Model komponenti ukratko predstavlja definiciju standarda za implementaciju komponenti, njihovu dokumentaciju i puštanje u rad.
- Iako je poznato koji su osnovni elementi dobrog modela komponenti, ne postoji jedinstveni model komponenti koji svi koriste (podsetnik na slajdu 5).
- Postoje 3 osnovna elementa i više podelemenata koje dobar model komponenti mora da definiše:
  - Interfejse
  - Način korišćenja
  - Postupak za puštanje komponente u rad

Nastavak…

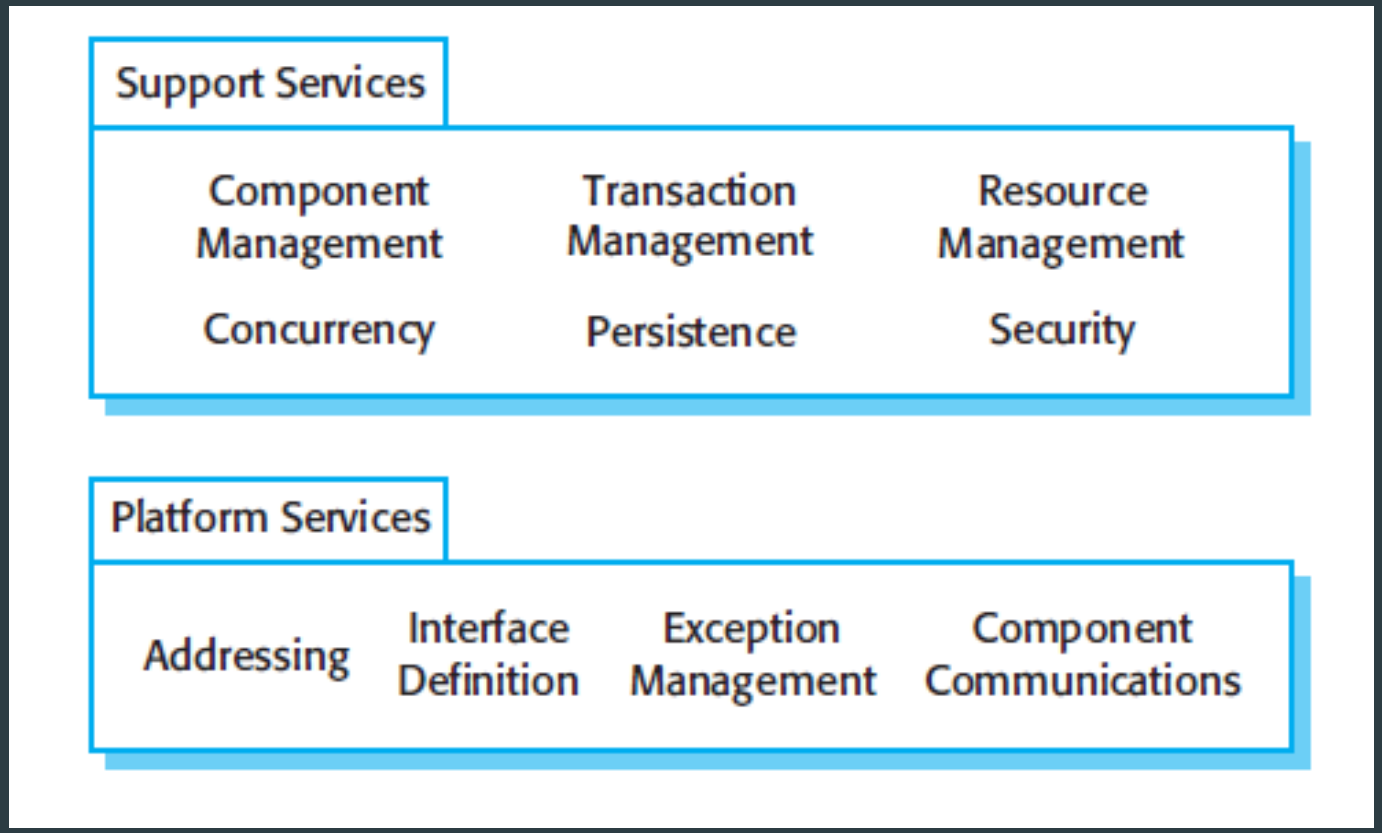
- Interfejsi
  - Definicija interfejsa
  - Kompozicija interfejsa
  - Specifični interfejsi
- Način korišćenja
  - Standardizovanost imenovanja komponenti
  - Meta podaci o samom interfejsu komponente
  - Prilagođavanje generičke komponente tačno određenom okruženju
- Postupak za puštanje komponente u rad
  - Informacije o pripremi komponente za puštanje u rad (packaging)
  - Način i obim dokumentovanja komponente
  - Opis postupka zamene komponente (sistem će sigurno evoluirati)

Posrednički sloj

- Ako su komponente implementirane kao udaljeni servisi ovaj sloj je od jako malog značaja, međutim ako su komponente implementirane kao programske jedinice onda postaje jako bitan.
- Servisi koje model komponente obezbeđuje u posredničkom sloju se dele u dve veće grupe:
  - Servisi za povezivanje platformi (Platform services)
    - Omogućavaju komponentama da komuniciraju u distribuiranom okruženju
    - Prevazilazi problem različitih platformi, obavezno se mora definisati modelom komponenti
  - Servisi podrške (Support services)
    - Skup servisa koje koristi većina komponenti, na primer logovanje
    - Izbegava se ponavljanje koda i moguće nekompatibilnosti različitih realizacije istog servisa

Nastavak...

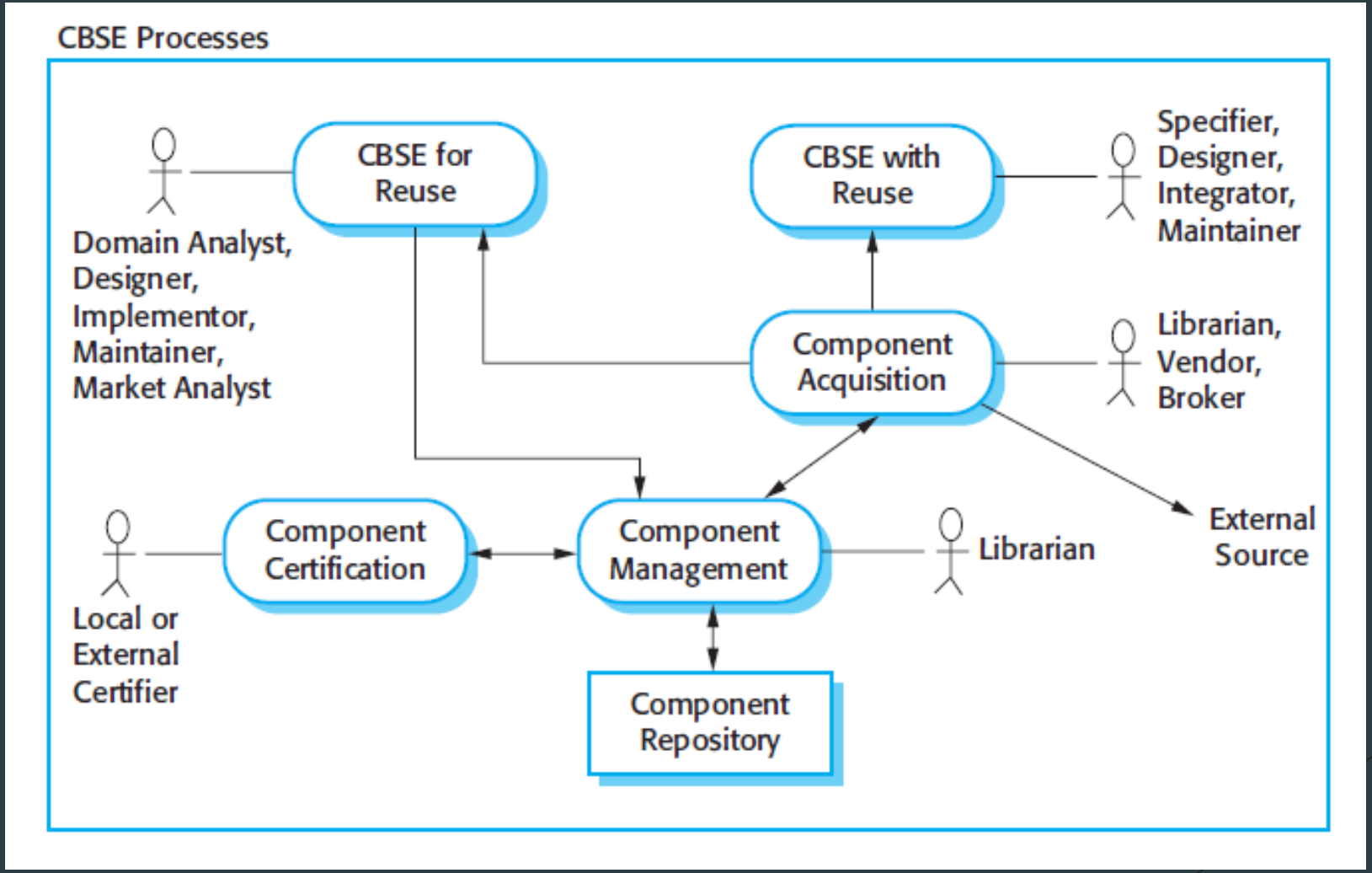
Neki predloženi servisi podeljeni u grupe:



Razvojni procesi za CBSE

- Razlikujemo dva osnovna razvojna procesa koja odgovaraju razvoju zasnovanom na komponentama:
  - Razvoj softvera za ponovnu upotrebu (Development for reuse)
    - Odnosi se na razvoj komponenti zbog njih samih ne zbog konkretne aplikacije
    - Prave se što generalizovanije komponente da bi se bile upotrebljive u širokom spektru aplikacija
  - Razvoj softvera sa postojećim komponentama (Development with reuse)
    - Bavimo se razvojem konkretne aplikacije
    - Postojeće komponente i servise uklapamo u našu aplikaciju
  - Ove dve vrste procesa imaju očigledno različite ciljeve pa se zato i dosta razlikuju

Povezanost razvojnih procesa u praksi

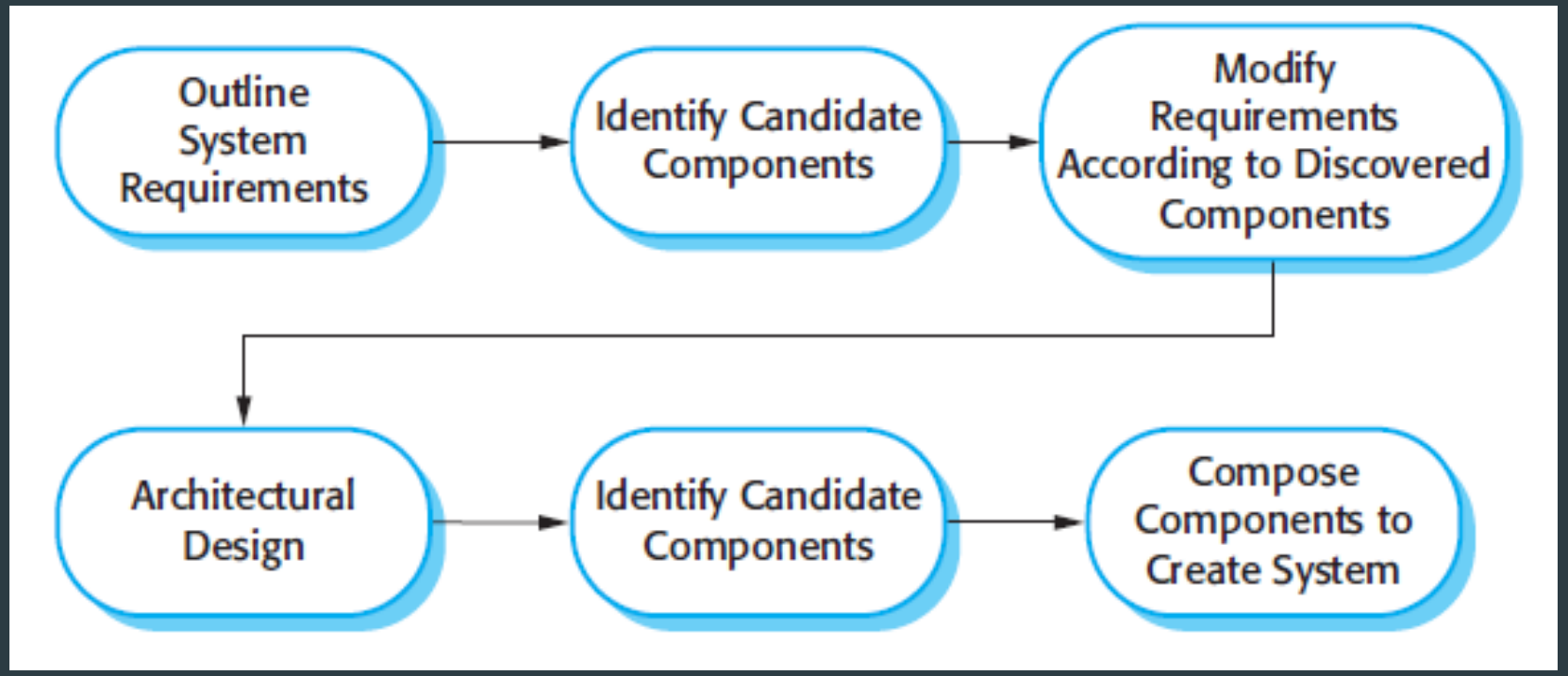


Razvoj softvera za ponovnu upotrebu

- Globalni repozitorijum komponenti (zašto nije uspelo?)
- Skoncentrisanost i izolovanost na nivou kompanija
- Odnos dobiti zbog buduće ponovne upotrebe prema ceni razvoja takve komponente
- Najčešće promene da bi se postigla ponovna upotrebljivost komponente su:
  - Brisanje metoda specifičnih za konkretnu aplikaciju
  - Davanje opštijih imena
  - Dodavanje metoda koji upotpunjuju funkcionalnost
  - Standardizacija izuzetaka (postaju deo interfejsa)
  - Pravljenje interfejsa za prilagođavanje komponente (konfigurisanje)
- Pitanje ispravnosti napravljene komponente (razvijaoci kao testeri ?)

Razvoj softvera sa postojećim komponentama

- Osnovni proces razvoja teče ovako:

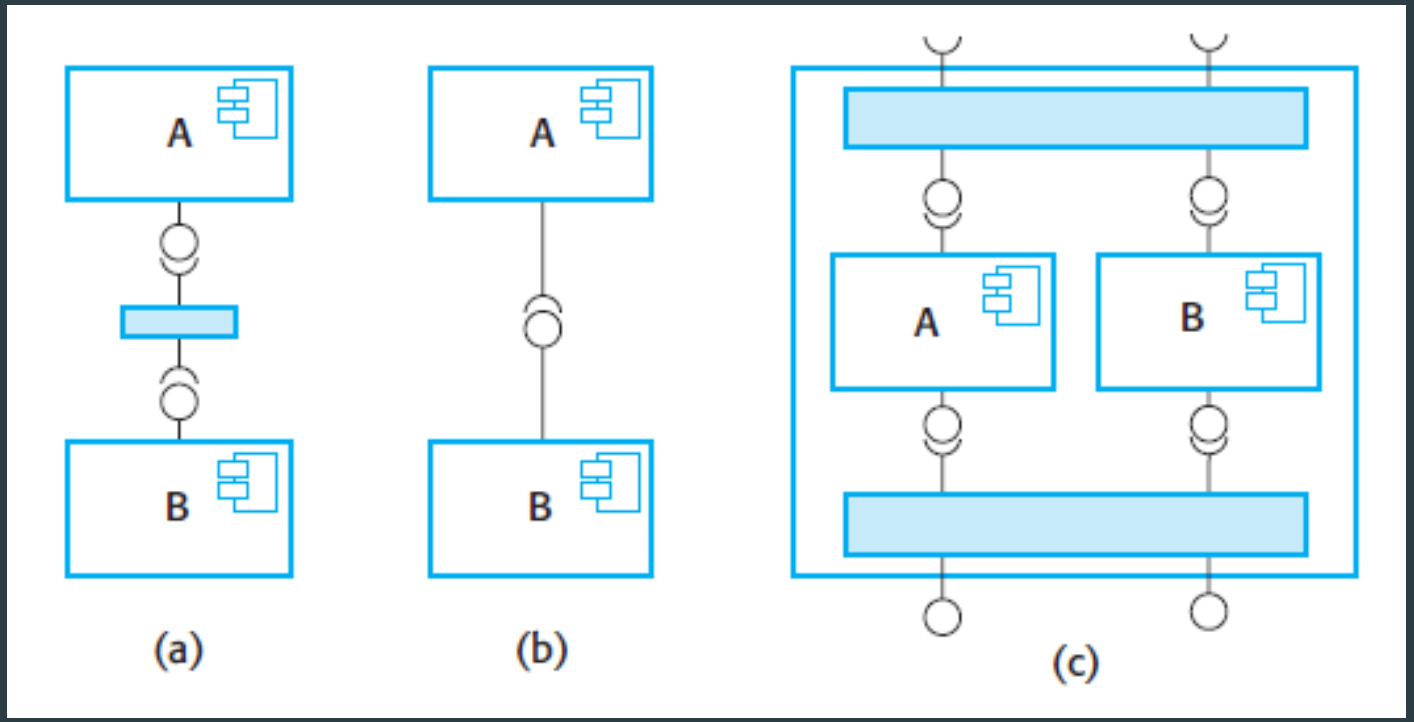


Promene koje komponente donose u razvoju

- Osnovne promene koje ovakav razvoj donosi su:
  - Korisnički zahtevi se prihvataju u kratkim crtama, ne ulazi se u detalje. Poslovni menadžment projekta bi trebao da bude što fleksibilniji po pitanju specifičnih zahteva kako se ne bi previše suzio izbor komponenata. Za razliku od inkrementalnog razvoja svi zahtevi moraju biti unapred poznati (velika mana).
  - Nalaženje najboljih komponenti zatim prilagođavanje zahteva
  - Provera uklopljenih i odabranih komponenti nakon dizajna arhitekture sistema
  - Postoji dodatna faza u razvoju, faza integracije komponenti i dopunjavanja funkcionalnosti i interfejsa. Veoma često će se desiti da komponente nemaju definisane neke funkcionalnosti koje su specifične za našu aplikaciju.
- Da li vredi gledati izvorni kod komponente ako je dostupan (Ariane 5)?

Integracija komponenti

- Svodi se na uklapanje komponenti i pisanje koda koji će te komponente povezati u celinu (glue code).
- Načelno postoje 3 pristupa kod uklapanja komponenti:



Nastavak…

a) Sekvencijalna kompozicija
   - Komponente se pozivaju jedna za drugom
   - Izlaz iz prve komponente se koristi kao ulaz za poziv druge komponente
   - Treba napisati kod koji ih poziva i povezuje (npr. vrši prilagođavanja izlaza u ulaz)

b) Hijerarhijska kompozicija
   - Naš cilj je najčešće pozivanje usluga komponente B
   - “Provides” interfejs komponente A mora odgovarati “requires” interfejsu komponente B
   - Postoje slučajevi kada su potrebno napisati kod koji “doteruje” ove interfejse

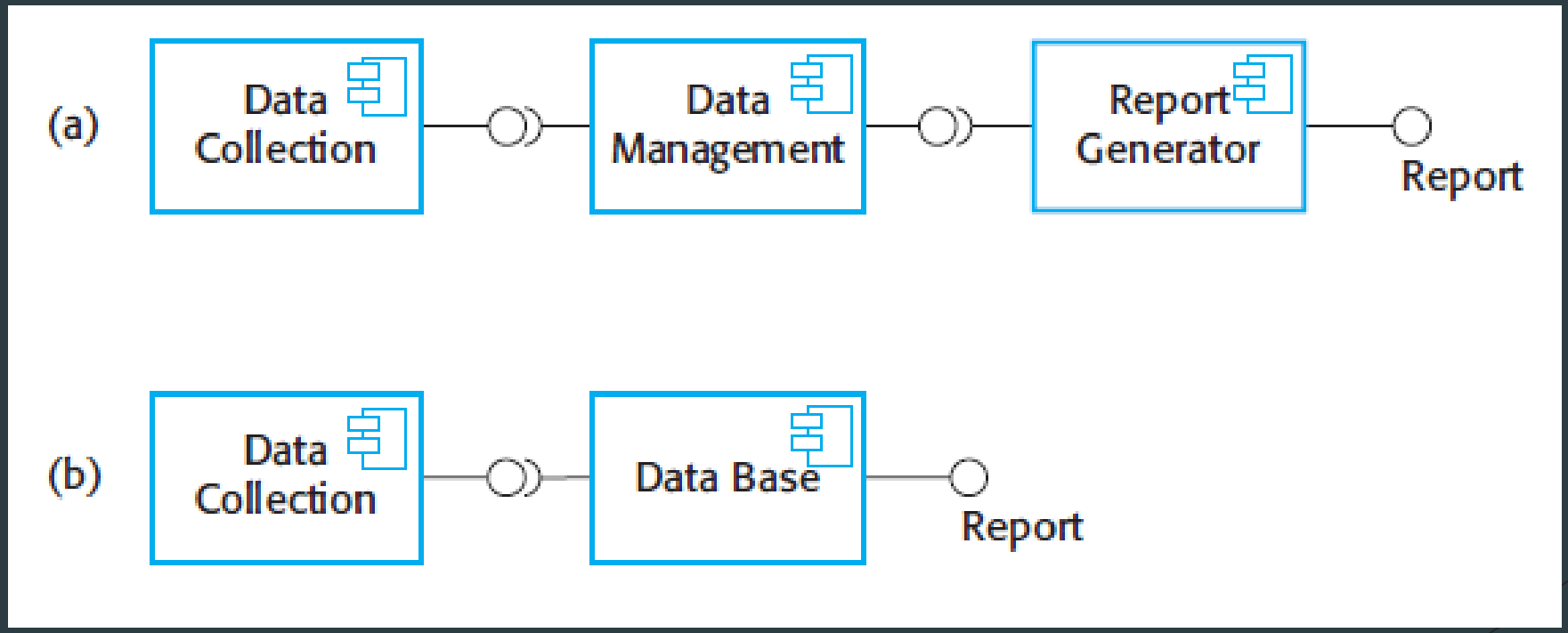
c) Kompozicija dodavanjem
   - Koristi se da se napravi nova komponenta od dve postojeće
   - Interfejs pruženih i traženih usluga je kombinacija tih interfejsa komponenti A i B
   - Same komponente A i B su nezavisne i tako se i pozivaju kroz novi interfejs

Problemi pri integraciji

- Potreba za neistorodnim komponentama – vodi do nekompatibilnih interfejsa
  - Različitost parametara (imena, broja ili tipova)
  - Različitost operacija (nepoklapajući nazivi u “requires” i “provides” interfejsima)
  - Nekompletnost pruženih operacija (“provides” je samo podskup “requires”)
  - Rešenje je pisanje adaptera (gledati na adapter kao na proxy)
- Konflikti pri samoj integraciji
  - Često se sistem može sastaviti na više načina
  - Takođe često nijedan nije idealan, pa se vrši odabir kompozicije na osnovu funkcionalnih zahteva
  - Kompromisi kao svakodnevni deo razvoja (brza isporuka sistema, dobre performanse, dobra skalabilnost i lako proširenje često ne idu zajedno)

Jednostavan primer

- Date su dve realizacije sistema sa istom funkcionalnošću:



Nastavak…

a) Ovo rešenje pruža:
   - Zdravo odvojene odgovornosti
   - Bolju prilagodljivost (baze podataka je jako teško menjati!)
   - Obe komponente se mogu lako zameniti boljim implementacijama
b) Ovo rešenje pruža:
   - Brzinu pristupa i dobru organizovanost podataka
   - Veća sigurnost sistema, sigurnosni mehanizmi baze podataka štite podatke
   - Potencijalno brža isporuka sistema zbog manjeg broja komponenti

Koje rešenje je bolje?

Hvala na pažnji!!!