Softwarové architektury

definice, návrh, kvalitativní atributy, ladění

© Radek Ošlejšek
Fakulta informatiky MU
oslejsek@fi.muni.cz

Motivace – Problém

Proč softwarové architektury?

Jak zacelit mezeru mezi požadavky a kódem?

- Jak snížit riziko odchýlení kódu od požadavků?
- Jak snížit riziko nevhodně strukturovaného kódu?
- Jak snížit riziko obtížně pochopitelného a modifikovatelného kódu?

Motivace – Řešení

Role softwarové architektury

- Hrubá struktura systému
  - Vhodě definovaná, distribuovatelná
- Abstrakce na úrovni systému
  - Členění funkcionality do zodpovědností rozdělených mezi moduly/komponenty systému

Přehled přednášky

- Definice softwarové architektury
- Role softwarového architekta
- Návrh softwarové architektury
  - Specifikace požadavků
  - Návrh architektury
    - Vývojový proces
    - Techniky kvalitního návrhu
  - Ohodnocení SW architektury
    - Kvalitativní atributy SA
    - Taktiky ladění SA
- Stručný přehled navazujících témat

Definice Softwarové architektury

Architektura softwarového systému je sada **principiálních návrhových rozhodnutí** o systému. [Taylor et al. 2009]

Softwarová architektura nasazeného software je určena těmi aspekty, které jdou **nejobtížněji změnit**. [Klusener et al. 2005]

Softwarová architektura programu nebo výpočtového systému je struktura systému, kterou tvoří **softwarové elementy**, externě **viditelné vlastnosti těchto elementů**, a vzájemné **vztahy mezi nimi**. [Bass et al. 2003]

Architektura je základní struktura systému, vtělená do jeho **komponent**, jejich **vztahů vůči sobě** a **prostředí**, a sada principů zastřešujících jeho návrh a vývoj. [ANSI/IEEE Standard 1471/2000]

Co tvoří softwarovou architekturu?

Tři hlavní složky

1. **Moduly** = komponenty tvořící funkcionalitu systému (často běžící současně)
   - např. zapouzdřené části softwaru, balíky, procesy, vlákna, komponenty

2. **Konektory** = komunikační spoje a kanály (často s vlastní vnitřní logikou)
   - např. volání procedury či metody, kanály pro distribuci zpráv (v publish-subscribe stylu)

3. **Nasazení** = mapování modulů a konektorů na hardwarové (nebo softwarové) zdroje
   - např. fyzické zdroje a servery v případě hardwarových zdrojů (s parametry jako frekvence CPU, velikost HD, rychlost síťového linku)
   - Např. operační systém nebo aplikačního server v případě softwarových zdrojů

Výhody definované architektury systému

Vzájemná komunikace

- Sjednocení pohledu na systém mezi různými rolemi podílejícími se na vývoji (včetně různých rolí u zákazníka)

Systémová analýza

- Predikce kvalitativních atributů architektury

Znovupoužitelnost

- Stejná architektura (navržená za účelem splnění určitých nefunkčních kritérií) se může stát základem mnoha různých systémů
- Přenesení modulů z jiných systémů, nebo jejich návrh pro širší škálu systémů

Projektové plánování

- Odhad ceny, rozvržení milníků ve vývoji, závislostní analýza

Role softwarového architekta

Softwarový návrhář

- Musí být schopen rozpoznat, znovu využít nebo najít efektivní návrhová řešení a vhodně je aplikovat

Doménový expert

- Musí mít detailní znalost a pochopení aplikační domény systému, jejich hlavních vlastností a zvláštností

Technolog

- Musí vědět, jak bude jeho řešení technicky fungovat po reálném vyvinutí

Znalec SW/HW standardů

- Musí se výborně orientovat v relevantních standardech, přesně odhadnout a komunikovat jejich přínosy a dopady

Ekonom softwarového inženýrství

- Musí vhodně vyvážit celý projekt vývoje tak, aby co nejefektivněji realizoval zadané požadavky

Návrh softwarové architektury

Tři hlavní aktivity architektonického návrhu:

- **Specifikace požadavků**
  - Funkční a extra-funkční
- **Návrh architektury**
  - Základní návrhové otázky
  - Architektonické modely
  - Vývojový proces
  - Techniky kvalitního návrhu
- **Ohodnocení SW architektury**
  - Kvalitativní atributy SA
  - Metody hodnocení kvality
  - Taktiky ladění SA

Návrh softwarové architektury

Specifikace požadavků

- Funkční a extra-funkční

Návrh architektury

- Základní návrhové otázky
- Architektonické modely
- Vývojový proces
- Techniky kvalitního návrhu

Ohodnocení SW architektury

- Kvalitativní atributy SA
- Metody hodnocení kvality
- Taktiky ladění SA

Požadavky na SW architektury

Funkční požadavky = omezení kladené na funkcionalitu systému. Nemusí se ale nutně týkat jen funkcionality uvnitř komponent, ale také spolupráce mezi komponentami navzájem:

- K provedení `serviceA()` by nemělo být třeba více než 10 dalších komponent.
- Každá otevřená transakce (služba poskytovaná jinou komponentou) by měla být zavřena před tím než daná komponenta zahájí novou.
- Komponenta nesmí být zablokována při provádění žádné ze svých služeb (z důvodu čekání na výsledek volání jiné komponenty).

Nefunkční (extra-funkční) požadavky = omezení na způsob, jakým je systém implementován a realizuje svou funkcionalitu.

- Garantovaná doba odezvy v X% případů.
- Dostupnost X% v rámci každého měsíce.
- Kompatibilita softwaru/hardwaru.

Návrh softwarové architektury

Specifikace požadavků

- Funkční a extra-funkční

Návrh architektury

- Základní návrhové otázky
- Architektonické modely
- Vývojový proces
- Techniky kvalitního návrhu

Ohodnocení SW architektury

- Kvalitativní atributy SA
- Metody hodnocení kvality
- Taktiky ladění SA

Základní návrhové otázky

Příklad otázek, které si softwarový architekt musí zodpovědět:

- Existuje **předpis architektury**, který by měl být pro systém použit?
- Jaký přístup bude zvolen ke **strukturování systému**?
- Jak systém **dekomponovat** na subsystémy (moduly, komponenty)?
- Co může být znovu využito ze **starších projektů**?
- Co by mělo být znovu využito v **budoucích projektech**?
- Které komponenty mohou nebo musí být **zakoupeny**?
- Jaké **architektonické styly** bude vhodné použít?
- Jaké druhy **distribuce** jsou možné a vhodné?
- Jak komunikovat s **existujícím softwarem**?
- Jak přistupovat k **existujícím datům**?

Architektonické modely

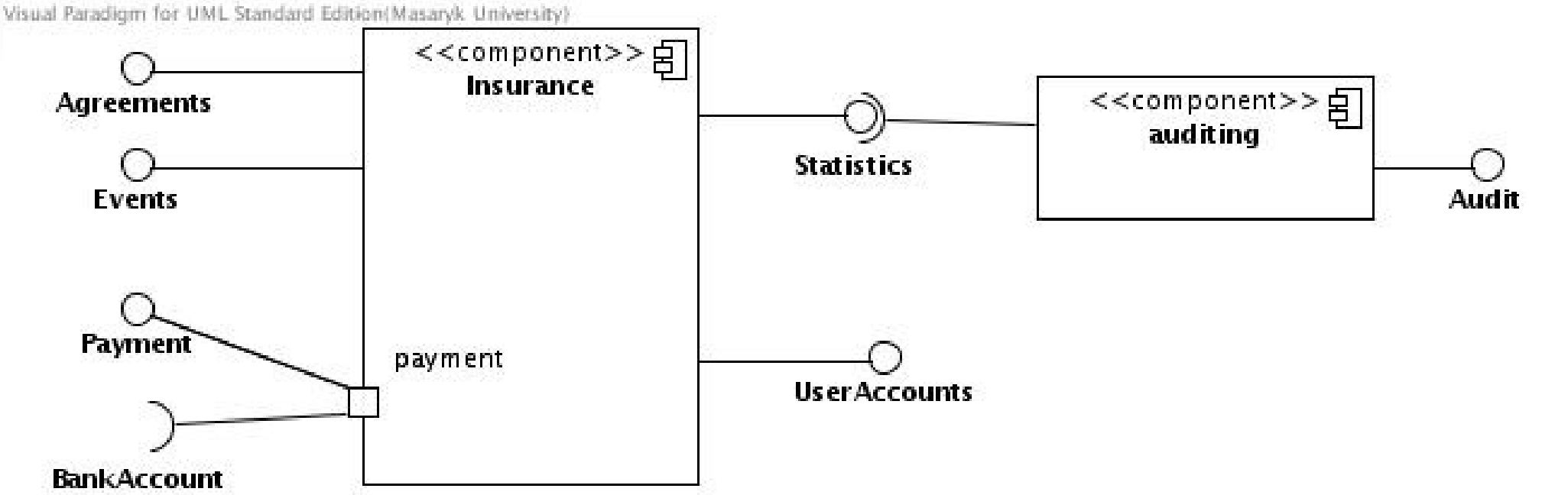
Definují

- Moduly – systémové komponenty
- Konektory – komunikační styly
- Nasazení – mapování na HW/SW zdroje

Architektonické modely – moduly

Moduly = systémové komponenty

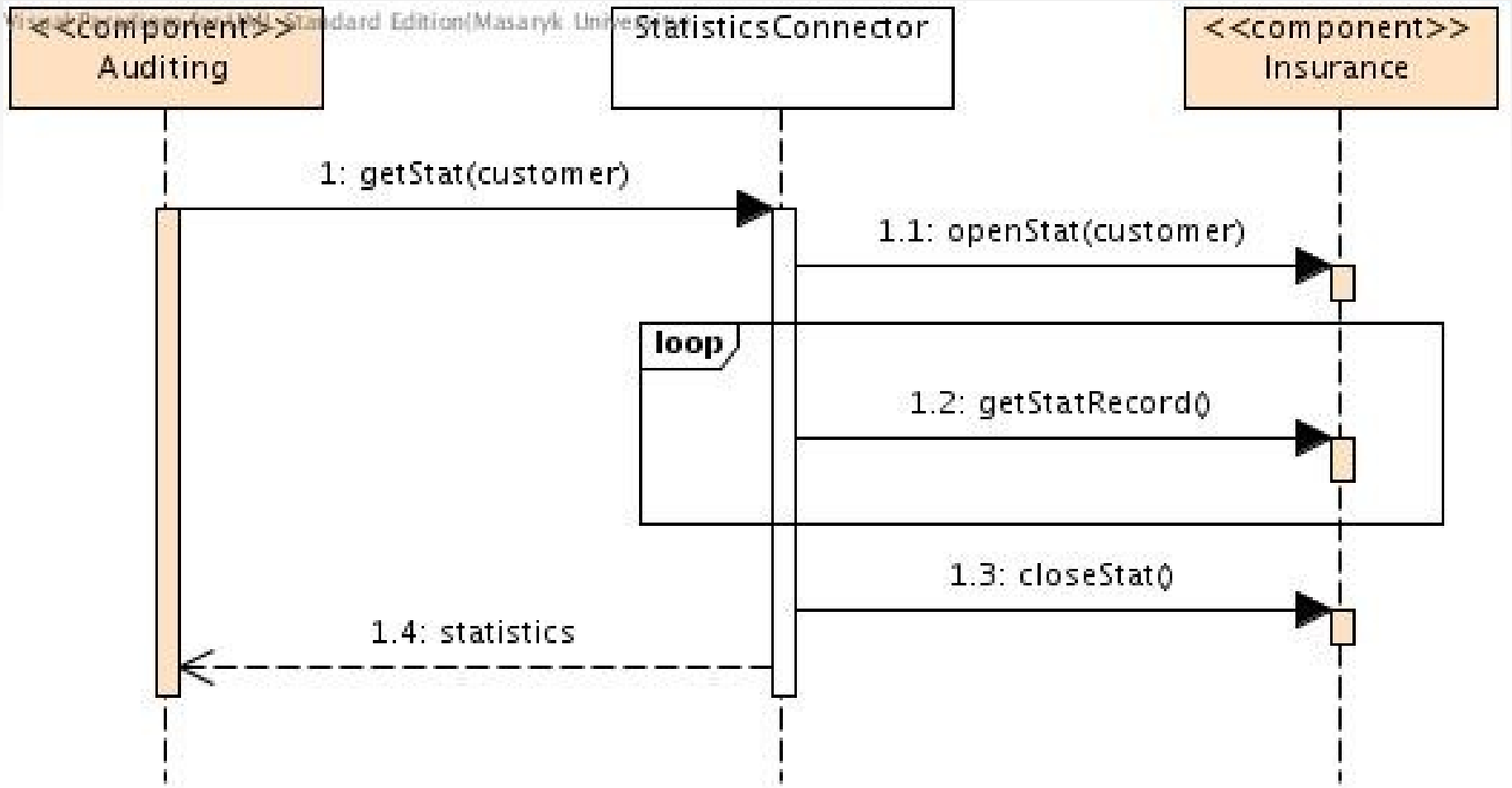
- **Model statické struktury** – definuje vnitřní strukturu komponent systému, které mohou být složené z dílčích podkomponent (nazýváme *složené*) nebo obsahovat přímo kód (ty nazýváme *primitivní*)
- **UML** – diagram komponent (pro složené komponenty), diagram tříd (pro primitivní komponenty)



Architektonické modely – konektory

Konektory = komunikační styly

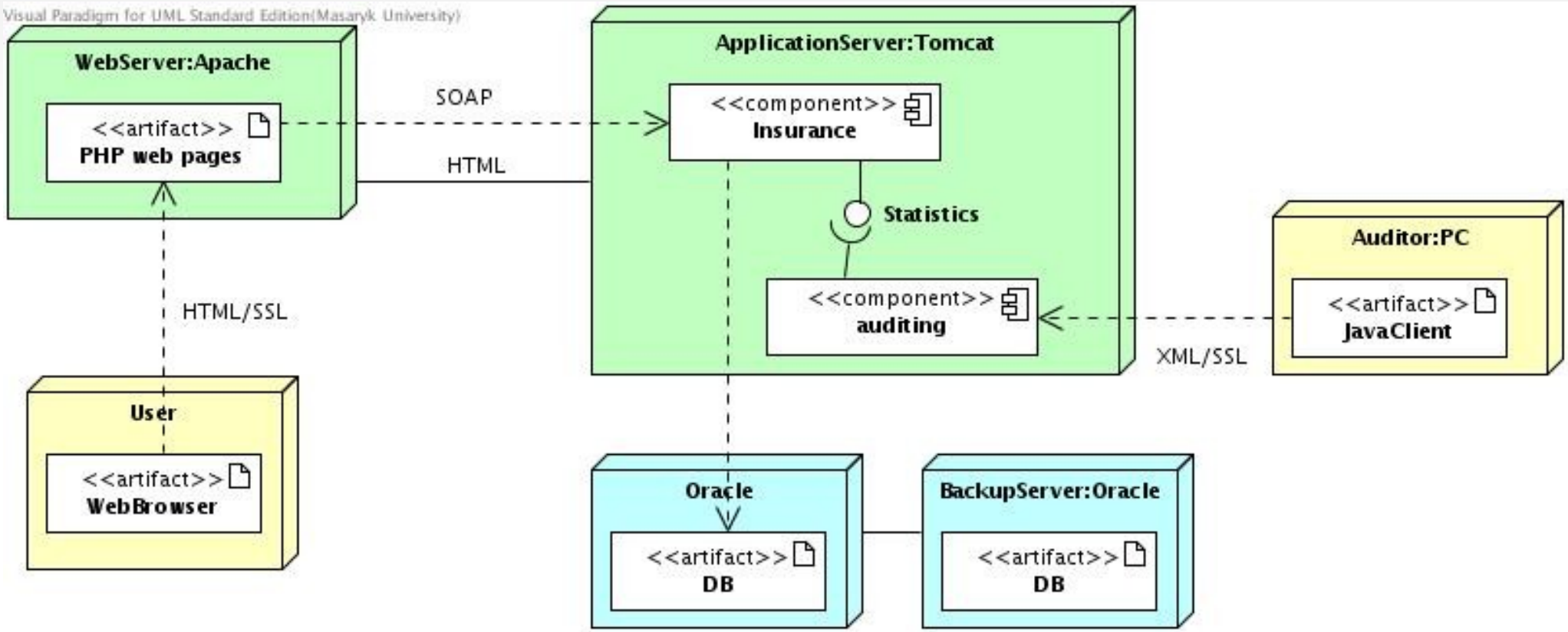
- **Procesní (dynamický) model** – definuje vzájemnou interakci mezi komponentami systému
- **UML** – sekvenční diagramy, komunikační diagramy, diagramy aktivit



Architektonické modely – nasazení

Nasazení = mapování na HW/SW zdroje

- **Model nasazení** – definuje strukturu zdrojů systému (včetně jejich charakteristik) a mapování modulů a komunikačních vazeb na tyto zdroje
- **UML** – diagram nasazení



Vývojový proces

Hlavní úlohy vývojového procesu:

1. Identifikuj systémové **komponenty**
2. Identifikuj **rozhraní** komponent
3. Navrhni **konektory** mezi komponentami (rozhraními)
4. Identifikuj části systému, které by měly být **alokovány** na stejný uzel
5. Ohodnoť **kvalitu** architektury

Architektonické návrhové praktiky

Úlohy 1.-3. mohou být realizovány v různém pořadí v závislosti na zvolené metodice

Top-down přístup – postupné zjemňování architektury

- Navrhni systém jako jednu komponentu s definovanými rozhraními
- Rozlož komponentu na podkomponenty první úrovně a vazby mezi nimi
- Rozkládej podkomponenty na nižší úrovně až po primitivní komponenty
- Implementuj, dohledej nebo nakup komponenty odpovídající navrženým primitivním komponentám

Bottom-up přístup – sestavování architektury z předem daných komponent

- Vyber z knihoven kandidáty na komponenty, které by mohly být součástí systémů
- Zkoušej komponenty propojovat dohromady do složených komponent, využívej jen ta jejich rozhraní, která jsou pro požadovanou funkcionalitu nezbytná
- Postupně dojdi až po komponentu nejvyšší úrovně (= systém)
- Odstraň nezapojené komponenty (které nakonec nebyly použity)

Techniky kvalitního návrhu

Architektonické vzory

- Praxí ověřené praktiky návrhu architektur
- Znovupoužitelné pro různé systémy

Metriky kvalitního návrhu

- Kvantitativní ohodnocení kvality návrhu podle objektivních kritérií (množství vazeb mezi moduly, velikost modulů, apod.)

Korektnost z podstaty konstrukce (correctness by construction)

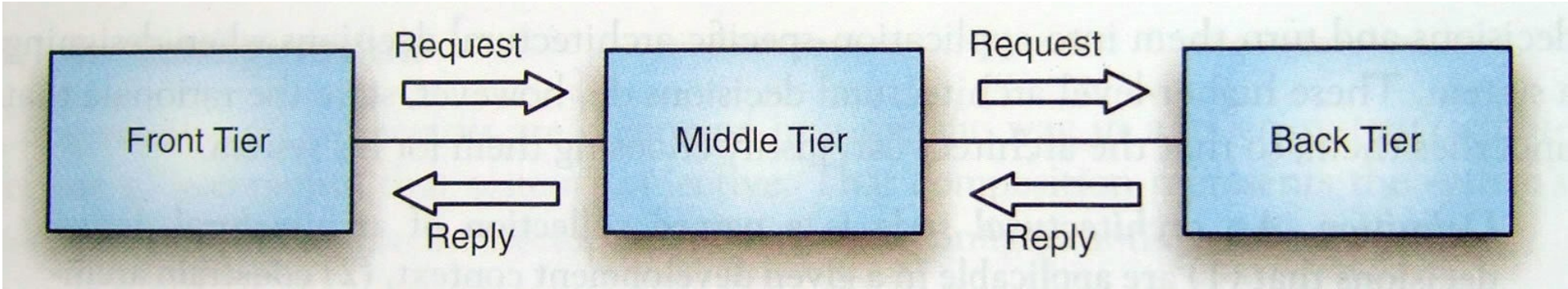
- Techniky návrhu v identifikovatelných krocích, kdy korektnost všech jednotlivých kroků garantuje kvalitní výsledek

Architektonické vzory

Architektonický vzor je pojmenovanou kolekcí architektonických návrhových rozhodnutí, která jsou aplikovatelná na znovu se objevující návrhové problémy a parametrizovatelná na různé kontexty softwarového vývojového procesu, ve kterých se dané problémy objevují. [Taylor et al.]

Jednoduchý příklad:

Třívrstvá architektura

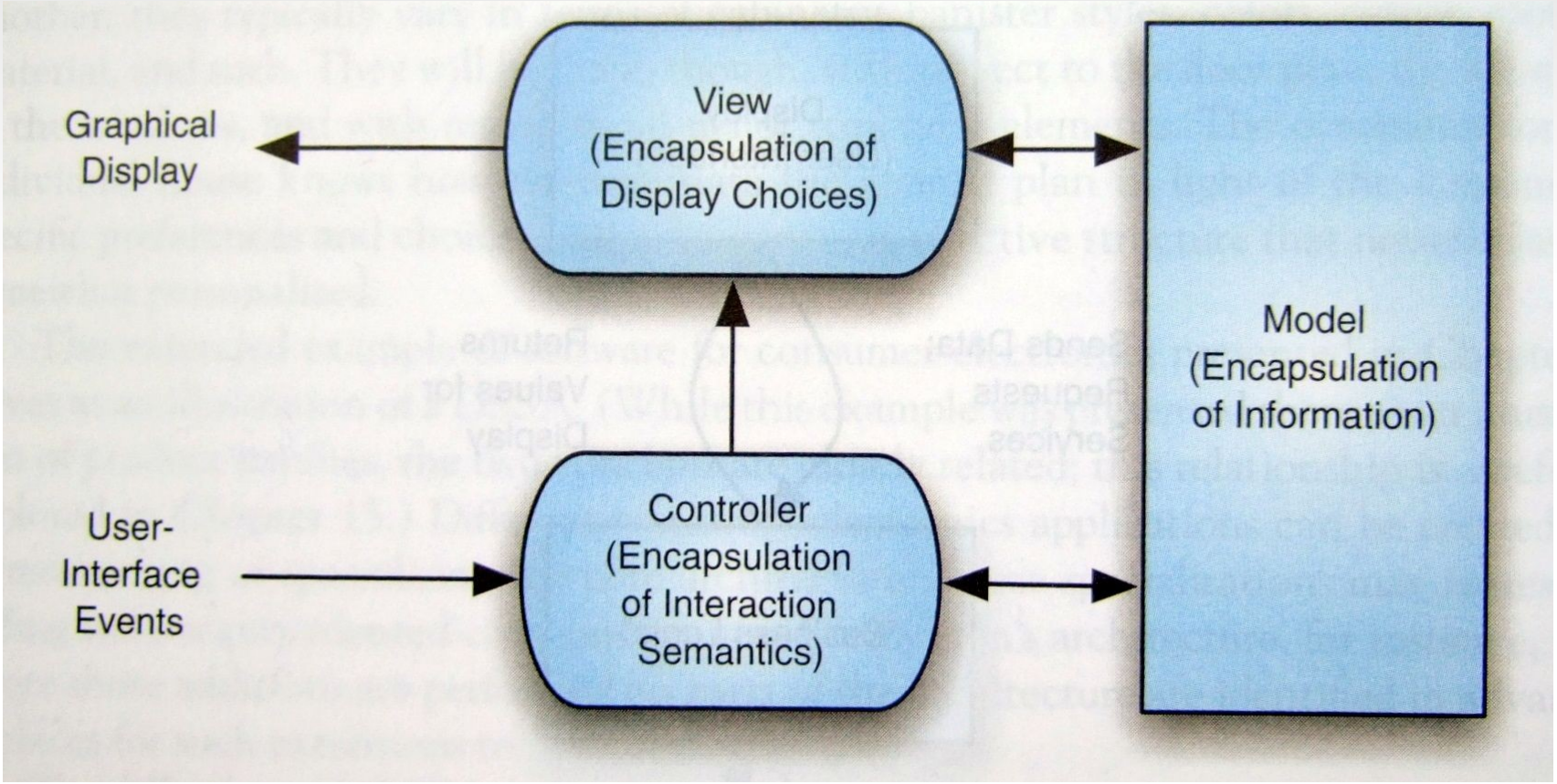


[Taylor et al. 2009]

Další příklady architektonických vzorů

Příklad vzoru řešícího lokální problém:

Model-View-Controller (grafického rozhraní)

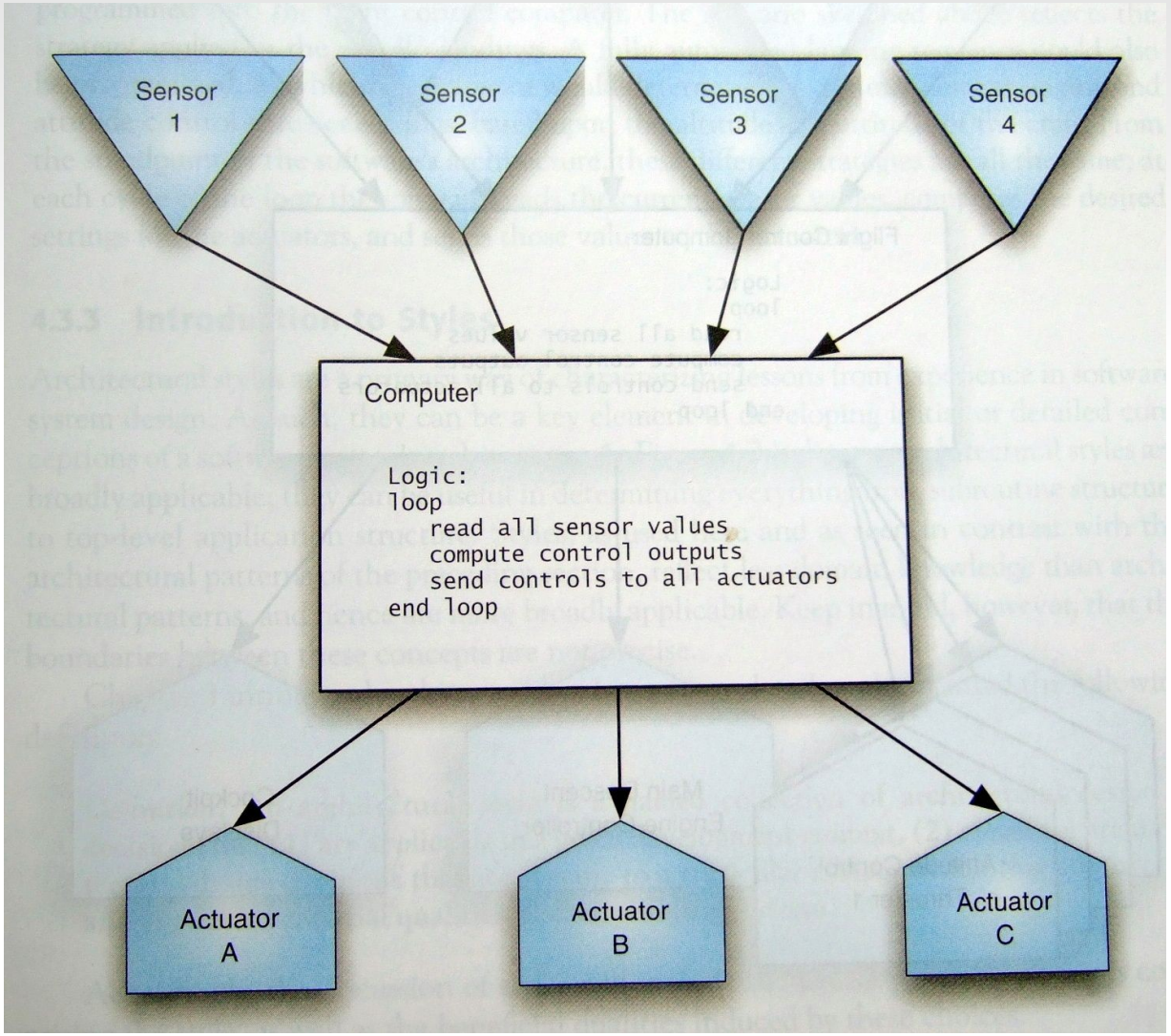


[Taylor et al. 2009]

Další příklady architektonických vzorů

Příklad vzoru pro specifickou doménu:

Sense-Compute-Control (vestavěné systémy)



[Taylor et al. 2009]

Architektonické styly

Často zaměňovány s architektonickými vzory!

> **Architektonický styl** je pojmenovanou kolekcí architektonických návrhových rozhodnutí, která (1) jsou aplikovatelná v daném kontextu vývojového procesu, (2) omezují architektonická návrhová rozhodnutí specifická konkrétnímu systému v rámci daného kontextu a (3) zajišťují určité kvality v každém výsledném systému. [Taylor et al.]

Jednoduché příklady:

Z pohledu programovacího jazyka

- Hlavní program a procedury/funkce
- Objektově orientovaný styl

Z pohledu komunikace

- Volání procedur/metod/služeb
- Zasílání zpráv
- Publish-subscribe notifikace

Postavení architektonických vzorů

Návrhové vzory

- Obecná řešení problémů při návrhu kódu

Architektonické styly

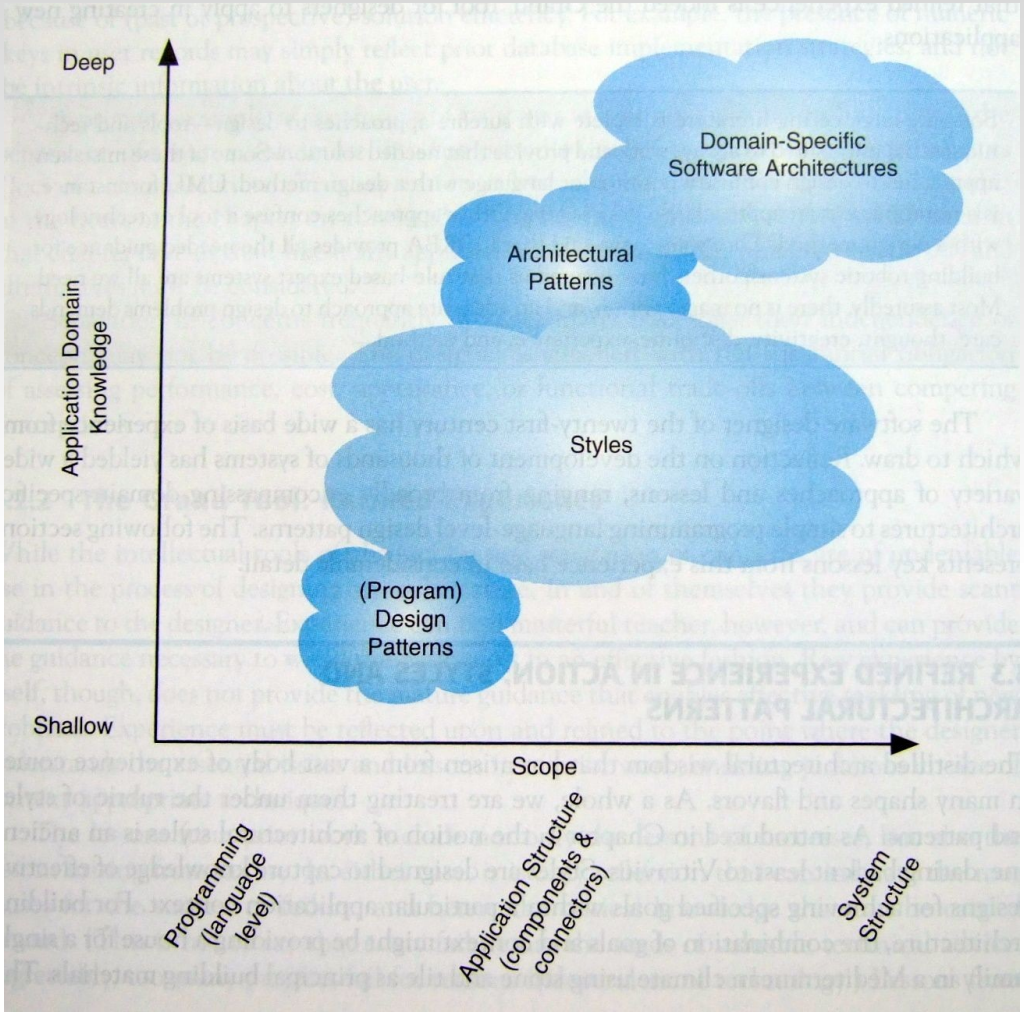
- Průřezové architektonické principy s vlivem na kód

Architektonické vzory

- Obecná řešení problémů při návrhu architektury

Doménově specifické softwarové architektury

- Předpisy kompletních struktur aplikací dle zvolené domény



[Taylor et al. 2009]

Návrh softwarové architektury

Specifikace požadavků

- Funkční a extra-funkční

Návrh architektury

- Základní návrhové otázky
- Architektonické modely
- Vývojový proces
- Techniky kvalitního návrhu

Ohodnocení SW architektury

- Kvalitativní atributy SA
- Metody hodnocení kvality
- Taktiky ladění SA

Kvalitativní atributy SA

Týkají se zejména nefunkčních (extra-funkčních) požadavků na systém:

> **Nefunkční (extra-funkční) požadavek** softwarového systému je omezení na způsob, jakým je systém implementován a realizuje svou funkcionalitu. [Taylor et al.]
>
> **Extra-funkční atribut** je potom kvalitativní aspekt, kterého se může extra-funkční požadavek týkat.

Příklady základních extra-funkčních kvalitativních atributů:

- **Výkonnost (performance)** – propustnost, doba odezvy, efektivita využití zdrojů
- **Spolehlivost (reliability)** – bezchybný provoz, dostupnost, robustnost, zotavitelnost
- **Bezpečnost (security)** – důvěrnost, integrita, dostupnost
- **Škálovatelnost (scalability)** – zátěž (daná požadavky), souběžná připojení, velikost dat
- **Udržovatelnost (maintainability)** – modifikovatelnost, přizpůsobivost

Metody hodnocení kvality

Monitorování a testování = ověření kvality existujícího systému

- Levná a často používaná metoda
- Použitelné až po implementaci systému
- Výsledky nutno brát s rezervou, závisí na počtu testovacích běhů

Predikce z modelu = předpověď kvality vytvářeného systému

- Nutno mít k dispozici (zjednodušený) model systému (vytvářený a upřesňovaný v průběhu návrhu)
- Použitelné v průběhu celého procesu návrhu architektury
- Přesnost výsledků závisí na detailnosti modelu

Formální verifikace = ověření hypotéz o modelu systému

- Nejdražší a nejpřesnější metoda
- Ověřováno na modelu vytvořeném speciálně za účelem verifikace (částečně možno generovat automaticky z kódu nebo návrhových modelů)
- Pro zaručení přesnosti nutno investovat nemalé úsilí vyladění modelu (a úrovně jeho detailu)

Taktiky ladění SA

Laděním softwarových architektur rozumíme úpravu architektury za účelem optimalizace zvoleného kvalitativního atributu.

Výhody:

- Praxí vyzkoušené techniky zvyšující kvalitu dle daného atributu

Pozor:

- Optimalizace není garantovaná, často jde jen o drobné vylepšení
- Nebezpečí zhoršení kvality z pohledu jiných atributů

Je třeba aplikovat s rozumem a rozumět vzájemným souvislostem mezi jednotlivými atributy a vlivy taktik na všechny z nich.

Taktiky – Výkonnost (Performance)

Výkonnost reflektuje schopnost softwarového systému naplnit požadavky na rychlou dobu odezvy a vysokou propustnost systému při minimalizaci použití výpočtových zdrojů.

Minimalizuj počet adaptorů a wraperů úpravou rozhraní

- **Jak:** Vyčištění rozhraní a úprava signatur služeb na rozhraních
- **Efekt:** Snížení prostředníků, skrz které musí volání projít, což jeho zpracování zpomaluje

Zjednoduš komunikaci skrz rozhraní

- **Jak:** Nabídni více rozhraní ke stejné funkcionalitě
- **Proč:** Každé z rozhraní může být určeno pro jiný runtime kontext (např. klienty přistupující z jiné platformy, využívající jiné datové formáty), který může obsluhovat efektivněji

Taktiky – Výkonnost (Performance)

Odděl data od výpočtu

- **Proč:** Reprezentace dat tak může být snáze optimalizována bez zásahu do výpočtových komponent; a výpočtové algoritmy optimalizovány bez zásahu do dat.

Přehodnoť použití broadcast konektorů

- **Proč:** Hromadné rozesílání zpráv (broadcast) zvyšuje spolehlivost doručení zprávy, ale zatěžuje výkon systému. Proto by nemělo být použito zbytečně.

Nahraď synchronní komunikaci asynchronní kdekoli je to možné

- **Proč:** Při synchronní komunikaci nejpomalejší komponenta brzdí všechny ostatní, které jsou zapojeny do řetězce volání, jehož součástí je.

Alokuj často komunikující komponenty blízko sebe

- **Proč:** Minimalizuje se zdlouhavá komunikace mezi uzly přes síť

Taktiky – Spolehlivost (Reliability)

Spolehlivost softwarového systému je pravděpodobnost, že systém bude provádět očekávanou funkcionalitu dle návrhových omezení bez chyb a výpadků po daný časový úsek.

Pečlivě zkontroluj externí závislosti komponent

- **Proč:** Je třeba zajistit, aby chybné chování jedné komponenty mělo minimální vliv na bezchybnost ostatních

Umožni u vybraných komponent promítnutí jejich stavu ven, a definuj invarianty stavu

- **Proč:** Pokud není možné u některé komponenty získat dostatečnou garanci jejich spolehlivosti, může přístup k aktuálnímu stavu komponenty umožnit jejím klientům vyhodnotit její aktuální "zdravotní stav" za běhu systému; invarianty pak slouží jako kritéria, dle kterých lze "zdravotní stav" kdykoli posoudit

Nasaď vhodné chybové reportovací mechanismy

- **Proč:** Když komponenta selže, měla by mít možnost informovat o příčinách chyby zbytek systému (např. pomocí výjimek)

Taktiky – Spolehlivost (Reliability)

Integruj kontrolu spolehlivosti komponent do konektorů

- **Proč:** Snížení pravděpodobnosti propagace chyb za hranici komponenty

Vyhni se existenci kritických míst (single points of failure)

- **Jak:** Replikace komponent, rozložení komponent na různé komponenty dle zodpovědností, posílení kontrolních schopností konektorů kolem takových komponent
- **Proč:** Jde o existenci míst, jejichž selhání s vysokou pravděpodobností ochromí celý systém

Integruj do systému automatické zálohování kritické funkčnosti a dat, a mechanismy zotavení

- **Proč:** Snížení rizika narušení spolehlivosti v případě ztráty dat nebo narušení funkčnosti

Integruj do systému monitorování jeho aktuálního "zdravotního stavu"

- **Proč:** Možnost rychlé reakce na vzniklé problémy

Taktiky – Škálovatelnost (Scalability)

Škálovatelnost je schopnost softwarového systému adaptovat se na nové požadavky ohledně velikosti a rozsahu systému.

Zajisti každé komponentě dostatečnou integritu, jasně definovaný účel a srozumitelná rozhraní

- **Proč:** Přidávání nových komponent nebo replikace stávajících tak bude mít minimální vliv na ostatní části systému

Distribuuj zdroje dat

- **Proč:** Předcházení častému úzkému místu při růstu velikosti systému (mnoho komponent přistupujících k jednomu zdroji dat)

Identifikuj data, která bude vhodné replikovat

- **Proč:** Opět možnost obsluhy více klientů přistupujících k datům současně

Taktiky – Škálovatelnost (Scalability)

Zajisti každému konektoru dostatečnou integritu a jasně definovaný účel

- **Proč:** Důvod analogický komponentám

Zvaž nahrazení přímých závislostí nepřímými

- **Proč:** Přímé závislosti (skrz asociace, tj. synchronní volání) nejsou ideální při růstu systému, protože vyžadují multiplikaci takových vazeb; vhodnější je volnější propojení skrz nepřímé závislosti (skrz posílání zpráv, např. broadcastingem)

Eliminuj úzká místa systému (komponenty a konektory používané více klienty současně)

- **Proč:** Zamezení brzdění systému v případě nárůstu počtu takových klientů

Nasaď na vhodných místech paralelní zpracování

- **Proč:** Urychlení složitých výpočtů vyžadovaných rostoucím počtem klientů

Taktiky – Udržovatelnost (Maintainability)

Udržovatelnost je úroveň obtížnosti změny softwarového systému v reakci na nové požadavky, změny prostředí nebo ladění chyb.

Odděl různé zodpovědnosti do různých komponent/sjednoť stejné zodpovědnosti do stejných komponent

- **Proč:** Snazší lokalizace míst, která si žádají úpravu

Vyčisti komponenty od operací souvisejících s interakcí, ne funkcionalitou

- **Proč:** Logika interakce by měla být soustředěna do konektorů, kde je lépe dohledatelná

Udržuj komponenty malé a kompaktní

- **Proč:** Snáz tak můžeš upravit malou část funkcionality systému výměnou vybrané komponenty

Izoluj data od výpočtu

- **Proč:** Zvýší se pravděpodobnost, ze při změně dat nebo výpočtu nebude třeba zasahovat do druhého z nich

Taktiky – Udržovatelnost (Maintainability)

Odděl různé komunikační principy do různých konektorů

- **Proč:** Snazší lokalizace míst, která si žádají úpravu; včetně komponent, které mohou být touto úpravou ovlivněny

Vyčisti konektory od operací souvisejících s funkcionalitou, ne interakcí

- **Proč:** Funkcionalita by měla být soustředěna do komponent, kde je lépe dohledatelná

Eliminuj nepotřebné závislosti

- **Proč:** Vyšší počet závislostí snižuje srozumitelnost systému

Hierarchizuj architekturu

- **Proč:** Hlubší vymezení zodpovědností komponent, možnost pohledu na systém na různých úrovních abstrakce

Stručný přehled navazujících témat

- **Komponentové softwarové inženýrství**
  - Vývoj systémů z COTS (Components of The Shelf)
- **Servisně-orientované architektury (SOA)**
  - Vývoj systémů integrací autonomních služeb
- **Aspektově orientované architektury**
  - Integrace průřezových koncernů (crosscutting concerns)
- **Softwarové produktové řady**
  - Rodiny produktů s jednotným jádrem a variačními body
- **Dynamické a adaptivní architektury**
  - Architektury schopné adaptovat se na run-time změny prostředí
- **Model-driven architektury (MDA)**
  - M2M transformace a zjemňování modelů

Shrnutí

- Definice softwarové architektury
- Role softwarového architekta
- Návrh softwarové architektury
  - Specifikace požadavků
    - Funkční a extra-funkční
  - Návrh architektury
    - Základní návrhové otázky
    - Architektonické modely
    - Vývojový proces
    - Techniky kvalitního návrhu
  - Ohodnocení SW architektury
    - Kvalitativní atributy SA
    - Metody hodnocení kvality
    - Taktiky ladění SA
- Stručný přehled navazujících témat