Architektura, návrh: struktury pro efektivní implementaci

KIV/ASWI 2014/2015

Obsah a cíl této části

- Co to je a jak vypadá architektura
  - statická struktura a dynamické aspekty celého systému
  - pohledy, vzory a dokumentace
- Dát návod jak tvořit podstatné části návrhu
  - tak, aby výsledky analýzy šly jednoznačně implementovat v konkrétním programovacím jazyce a provozní platformě
  - tak aby implementaci mohlo provádět více lidí, aby byla efektivní (co do výkonu i tvorby) a výsledek byl snadno udržovatelný

Klíčová rozhodnutí

- (proč – vize)
- (co – požadavky)

- Architektura
  - technologie
  - struktura
  - pravidla, konvence

- (jak – návrh)

Architektura

Architektura systému

- Klíčové aspekty organizace
  - jak je systém členěn, jak/proč dělá to co dělá
  - technologie
  - hrubé členění na subsystémy, závislosti a rozhraní mezi nimi
  - …
- Aspekty prolínající celou implementací (konvence)
  - jak systém navrhovat
  - vzory návrhu/implementace
  - mimo-funkční aspekty
  - …

> The fundamental organization of a system … its components, their relationships, … the environment, and the principles guiding its design and evolution.

Kontext, principy

- Kontext (daný)
  - okolí systému => vazby, důvody pro některá rozhodnutí
  - stakeholders => úhly pohledu => aspekty architektury
    - navíc oproti vizi a požadavkům
    - aspekty: logická struktura, procesní pohled, varianty nasazení, datová integrace, bezpečnost, provoz a podpora, provozní infrastruktura, rozhraní, …

- Principy architektury (volí si vývojáři)
  - efektivita – run-time, vývoj
  - jednotnost
  - srozumitelnost

Výběr technologie, omezení

- Programovací jazyk, databáze, knihovny
- Použití frameworků a hotových komponent
  - make vs buy decision

- Faktory ovlivňující/omezující výběr technologie
  - smluvní podmínky
  - standardy (i lokální)
    - viz disciplína Enterprise Architecture
  - kontext
  - marketing
  - technické znalosti

Zdůvodnění arch. rozhodnutí

- Zdůvodnění
  - proč je architektura právě taková
  - architektonicky důležitá funkčnost (use cases)
  - vztah k omezujícím podmínkám, historii

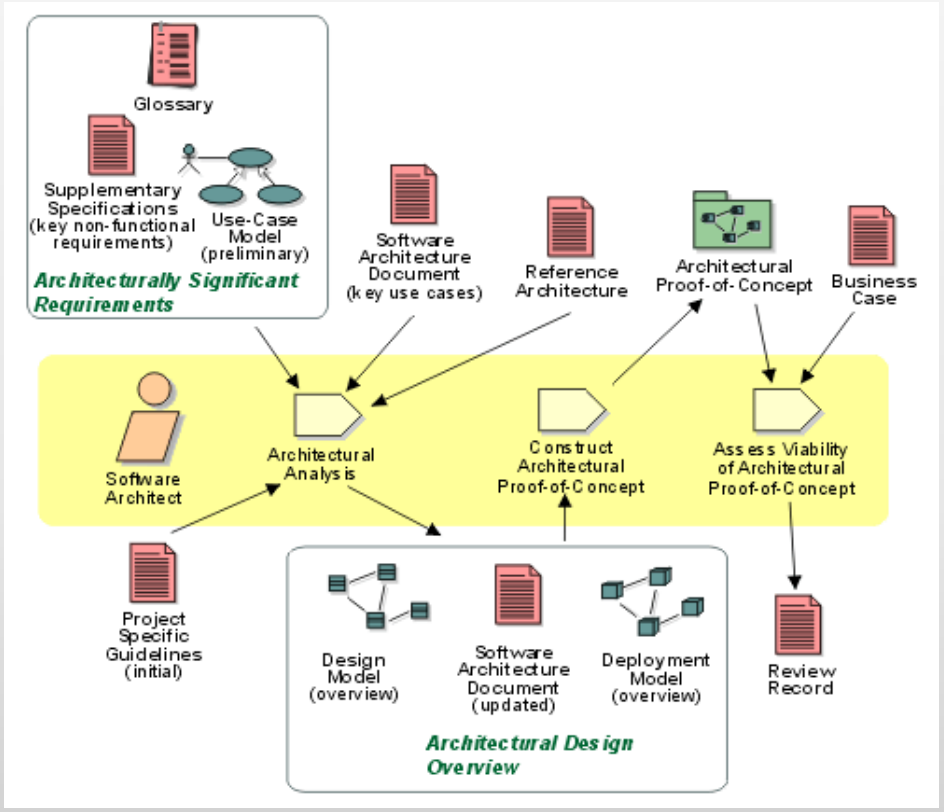
Příklad: Web KIV OpenCms

Validace architektury

- „Bude IOC, REL na tomto postavený splňovat LCO?“
  - executable architecture a její validace
  - architektonicky důležitá funkčnost (use cases)

- Mechanismy
  - návrh na základě klíčových use cases a mimofčních pož.
  - referenční architektura
  - proof of concept implementace
  - oponentura

Samostudium: UP Concept: Exectuable Architecture



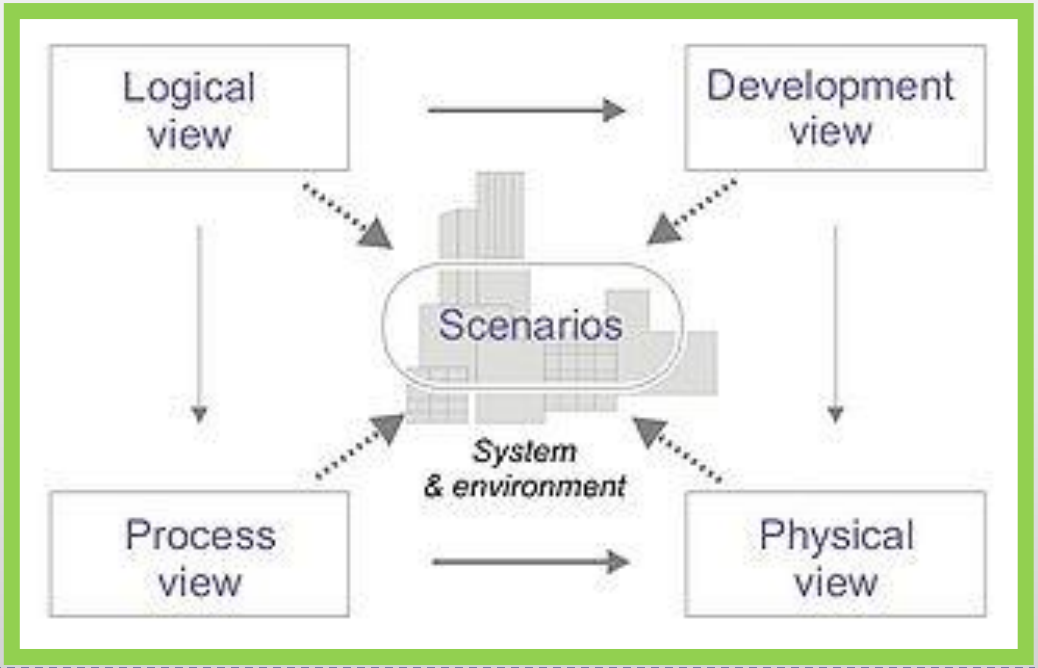
Architektonické struktury

Struktury = části architektury

- Co a proč je významné
- Např. RUP koncept „4+1 pohled“

P. Kruchten, “The 4+1 View Model of Architecture,” IEEE Software, vol. 12 (6), pp. 45-50, 1995.

- Obecný koncept:
  lokalita změn



logical view

Logické členění

- Motivace: monolitická aplikace nepřehledná

- Subsystém = skupina souvisejících prvků implementace tvořících funkční celek
  - funkčně soudržné (sada fčních požadavků)
  - často vázané na jednoho aktéra

horizontální vrstvy vs. vertikální domény (finance, personalistika, řízení, utility, …)

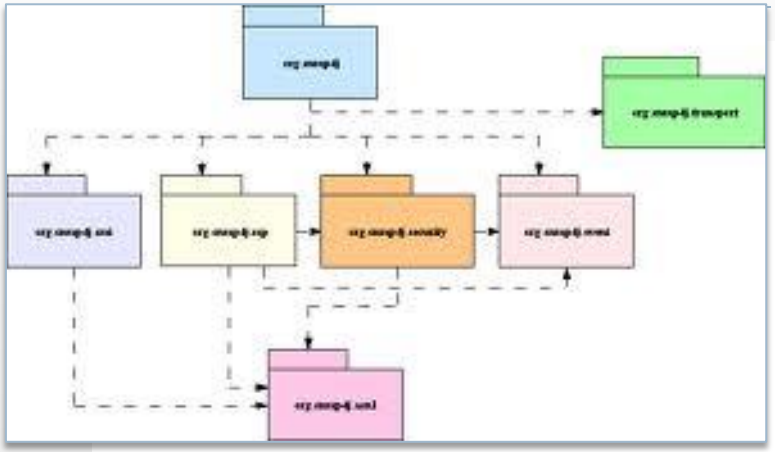
- Balík
  - agregace prvků implementace - jmenný prostor
  - snaha o přehlednost a srozumitelnost
  - oddělení veřejných a privátních prvků implementace

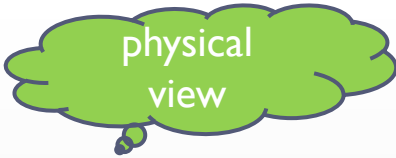
Logické členění – balíky

- Motivace: logický objektový model velký
- Členění pro získání
  - přehledu o systému
  - rozdělení implementace mezi členy týmu
- Co je balík
  - skupina souvisejících tříd, tvoří organizační celek
  - mapování do jazyka (vytváří jmenný prostor)
  - hierarchické vnořování
- Třídy balíku
  - funkčně příbuzné, v jedné vrstvě aplikace nebo kdekoli

nezávisle na tom, zda daný jazyk zná koncept „package“

UML: Balíky a jejich vazby





Fyzické členění

- Motivace: monolitická aplikace nepraktická
- Subsystém = skupina souvisejících prvků implementace tvořících funkční celek
  - funkčně soudržné (lokalita implementačních změn)
  - samostatná správa – nasazení, údržba, přístup

- Modul, komponenta, knihovna
  - celek vhodný pro samostatný vývoj, nasazení a údržbu
  - snaha o reuse = vícenásobnou použitelnost
  - => kvalitativní charakteristiky důležité
  - nutnost znát a spravovat závislosti

Jak najít balíky a moduly

D.Parnas: On the Criteria To Be Used in Decomposing Systems into Modules. Communications of the ACM, 15(12), 1972

- + kvalitativní charakteristiky dobrého návrhu
  - pravidla (information hiding, open-closed princip)
  - metriky (složitost, fan in/out)

Komunikace mezi moduly

- ROZHRANÍ
- ROZHRANÍ
- ROZHRANÍ

Různý výklad pojmu „komponenta“

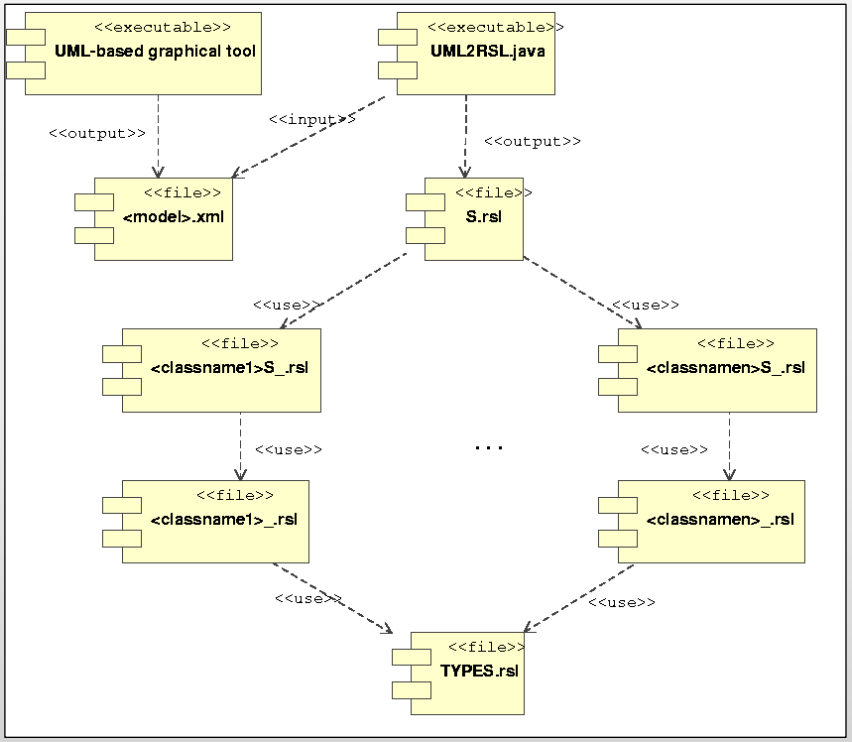
Komponentový přístup

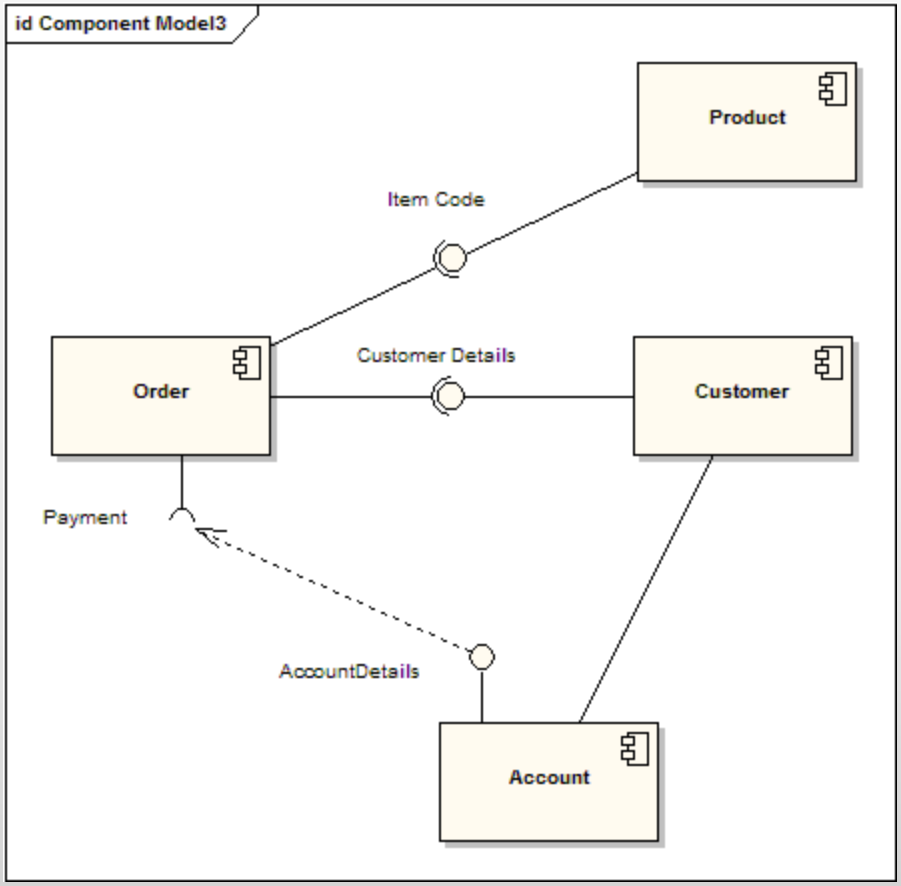
- Dotažení modularity, zapouzdření a rozhraní do konce
  - aplikace jako Lego skládačka (teoreticky)
- Komponenta
  - black-box – implementace „nedůležitá“ a „nedosažitelná“
  - rozhraní rozlišena na poskytovaná a vyžadovaná
    → vazby
  - explicitní specifikace rozhraní a vlastností
- Technologie
  - CORBA, EJB (částečně)
  - portlety, OSGi
  - Spring, PicoContainer

Inversion of Control (IoC) a Dependency Injection (DI)

UML: reprezentace komponent

- Rozdíl UML1 a UML2





Od logické struktury k fyzické

- Realizace logické struktury se provádí v konkrétních technických artefaktech
- Tyto je potřeba vytvořit
  - jednotlivci, týmy, projekty
  - technologické aspekty
  - vývojářská infrastruktura

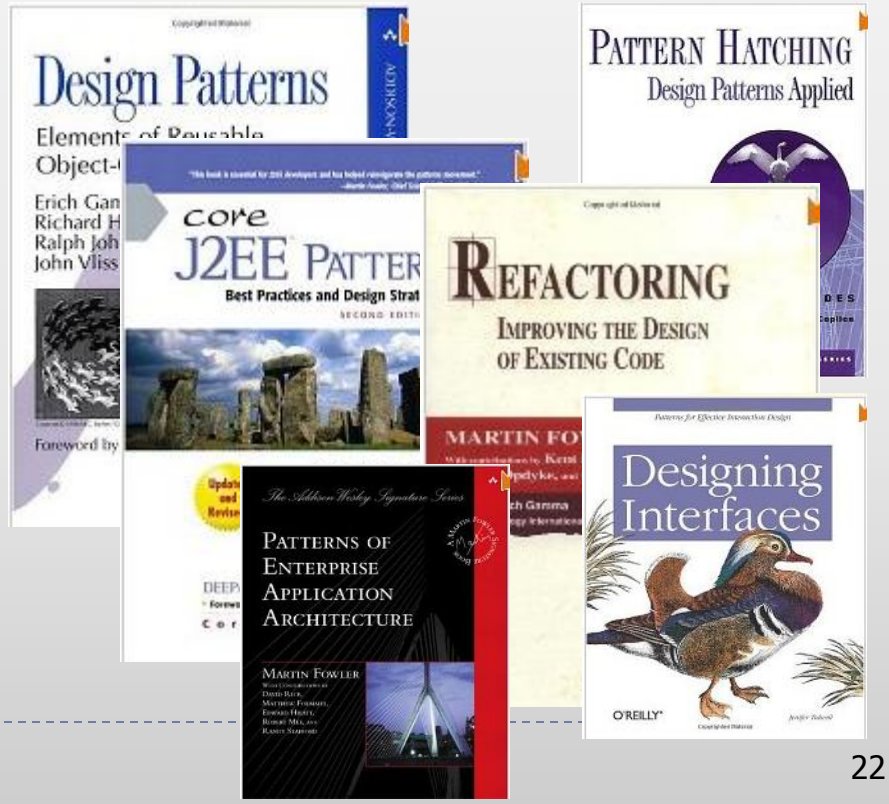
  => alokace prvků logické struktury do vývojových projektů

- Logický pohled: subsystémy, balíky => diagramy tříd
- Vývojový pohled: projekty, adresáře, sestavení
- Fyzický pohled: moduly, komponenty, knihovny

Konvence a politiky

- „Architectural policies“
  - obecná pravidla pro návrh v libovolné části aplikace
  - musí dodržovat všichni vývojáři

- Použité návrhové vzory
- Správa paměti
- Synchronizace, transakce
- Defenzivní programování
- Lokalizace (L10N, i18n)
- Dokumentace kódu
- …



Procesní pohled

- Struktura paralelizace v implementaci
  - procesy, vlákna; způsob synchronizace
  - komunikace – synchronní/asynchronní
  - propustnost, škálovatelnost, odolnost (deadlock, fault)
  - podpora (OS, jazyk, knihovny)
- Vazba na strukturu nasazení (distribuované systémy)

- Procesní model (workflow) systému
  - alokace aktivit do modulů implementace
  - synchronizace, předávání artefaktů

Dokumentace architektury

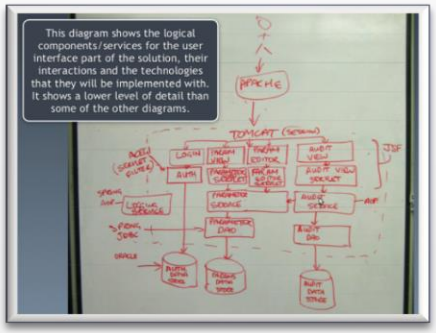
- Referenční architektura
  - dokument
  - kostra aplikace
- Dokument
  - RUP Artifact: Software Architecture Document
  - IEEE Std 1471, …
- Modely
  - UML: „implementation view“ (komponenty), „logical view“ (třídy, balíky), „process view“ (interakce, stavový model)
  - ad-hoc diagramy (Visio, tabule)

Samostudium: UP Checklist: Design

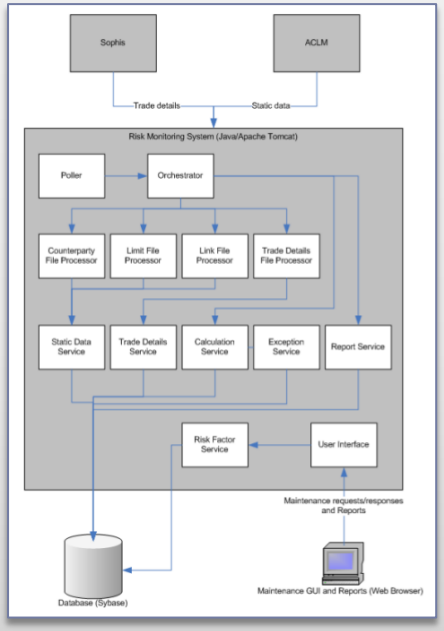
- www.codingthearchitecture.com → Software architecture document guidelines

Příklad

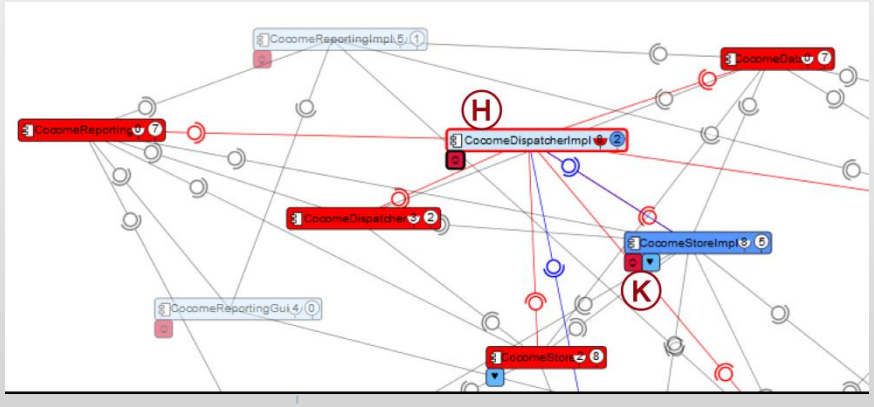
- Web KIV architektura modulů
- Tabule



- Visio



nástroje – architecture exploration

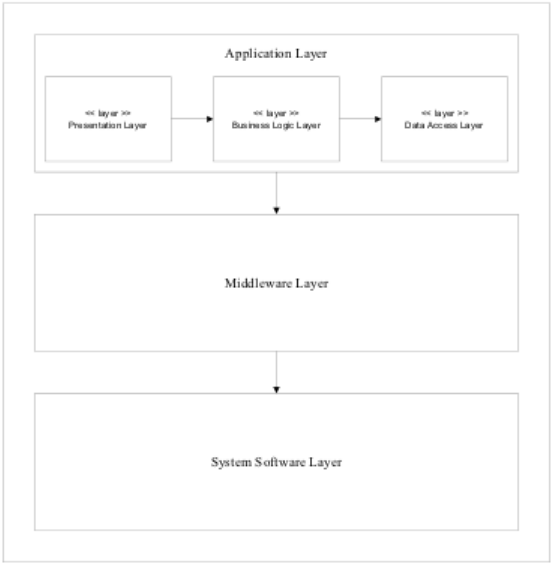


http://www.codingthearchitecture.com/presentations/yowsydney2014-software-architecture-sketches

Protipříklady

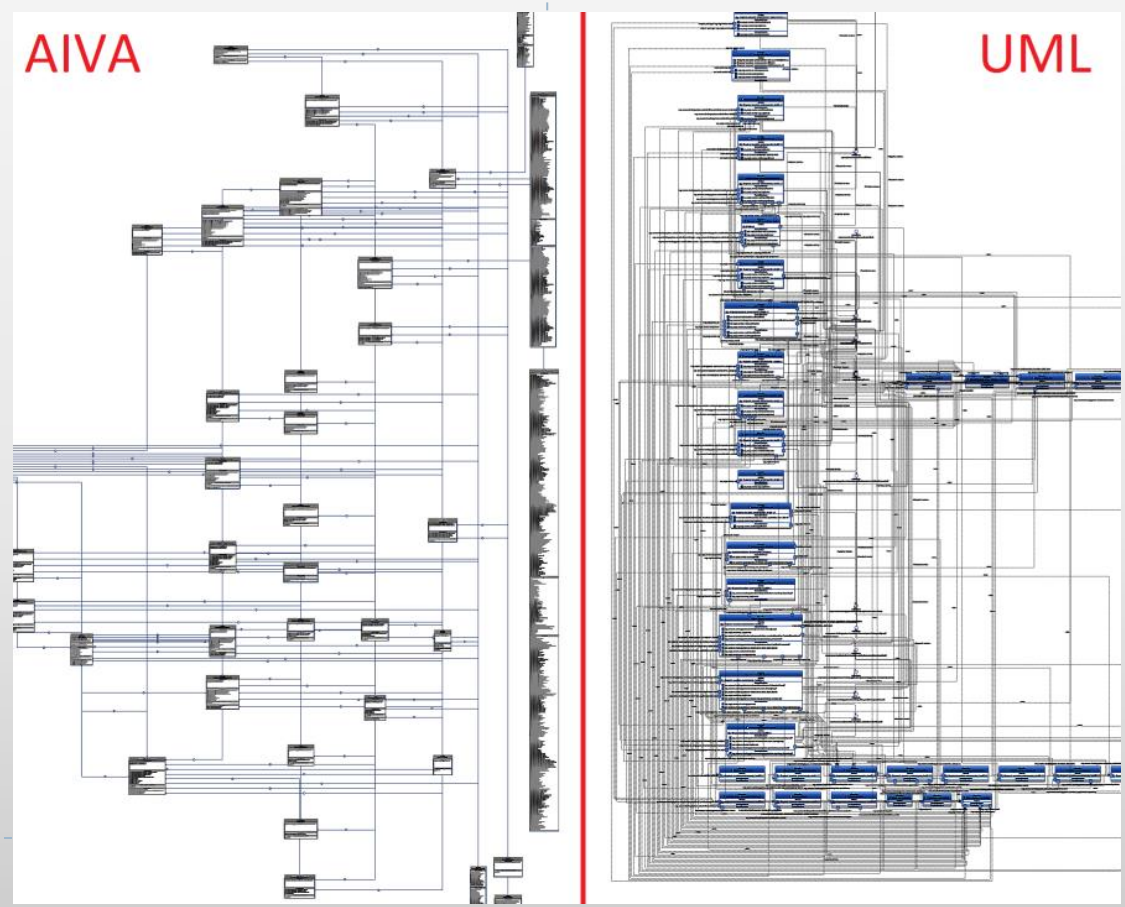
5.4.3 WUT Software Layers

The WUT problem domain and solution space layers are graphically depicted in Figure 3. Note the directional dependencies between the layers within the problem domain as well as between the application layer and the Middleware and System Software layers.



zbytečné

nečitelné



4+1 v souvislostech

Vazby mezi pohledy

- Dáno souvislostmi mezi prvky systému
  - vazbami artefaktů a logických celků
  - vazbami artefaktů mezi sebou
  - aktivitou či pasivitou artefaktů za běhu
  - nasazením artefaktů na provozní infrastrukturu
- Dáno postupem tvorby
  - priorita funkčností a omezení rizik ovlivňují postup vytváření
  - závislosti artefaktů definují postup skládání
  - připravenost infrastruktury ovlivňuje nasazení

Kdy začít členit architekturu

- Členění ovlivňuje plánování a postup prací
- Standardní projekty
  - možno odhadnout nebo stanovit předem
- Menší systémy
  - rozdělení na základě analýzy
- Velké projekty (analýza trvá dlouho)
  - nahrubo předem → rozfázování prací včetně analýzy
  - vodítko: aktéři, znalost struktury fyzického systému, možnosti reuse, vývojové kapacity
  - přesně až během návrhu architektury

Samostudium: UP Task: Refine the Architecture

Architektura z ptačí perspektivy (Design in the Large)

High-level architecture

- Základní hrubé členění celého systému
  - funkční subsystémy, vrstvy
  - vzájemné vazby
  - vazby na okolní sw
  - vazby na hw
  - důsledky na EFP

- Typické koncepty
  - vrstvy
  - filtry
  - služby
  - sdílená data

Architektonické styly

- Zavedené zvyky a standardy

"An architectural pattern expresses a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them."

Nulová varianta

Vrstvená architektura

- Delegování na podřízené
- Komunikace se sousedy

- Vrstvy např.
  - prezentace / řízení / doména / business služby / technická infrastruktura / knihovní třídy / systémové

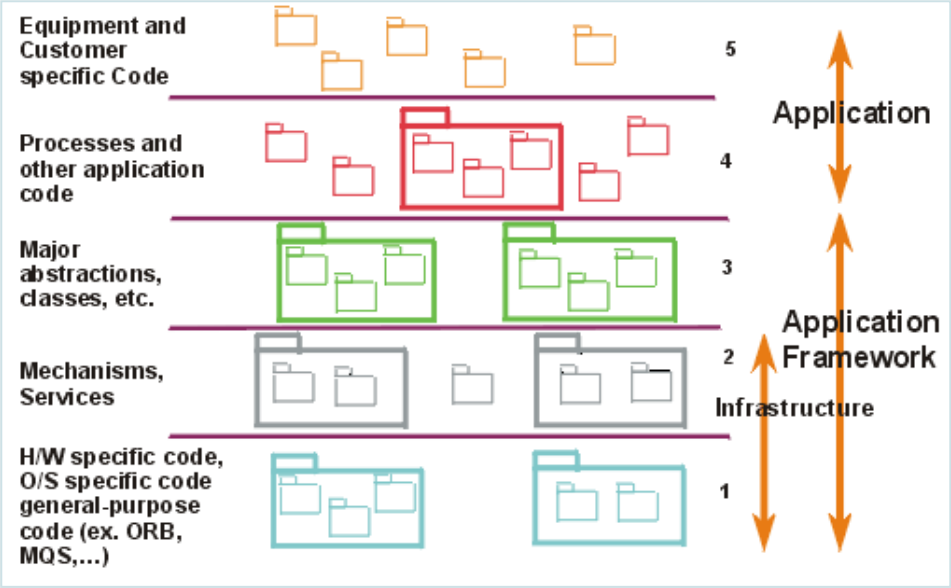


Image from IBM RUP 2005 Concept: Architecture

- Klient-server
  - tlustý klient
- 3-vrstvé a vícevrstvé
  - oddělení prezentace, business logiky a datové části
  - dnes standard

Další architektonické styly

- SOA
  - varianta: Enterprise Service Bus
- Pipes and filters („kolona“)
- Blackboard
- Broker
- Map-Reduce
  - e.g. http://architects.dzone.com/articles/how-hadoop-mapreduce-works

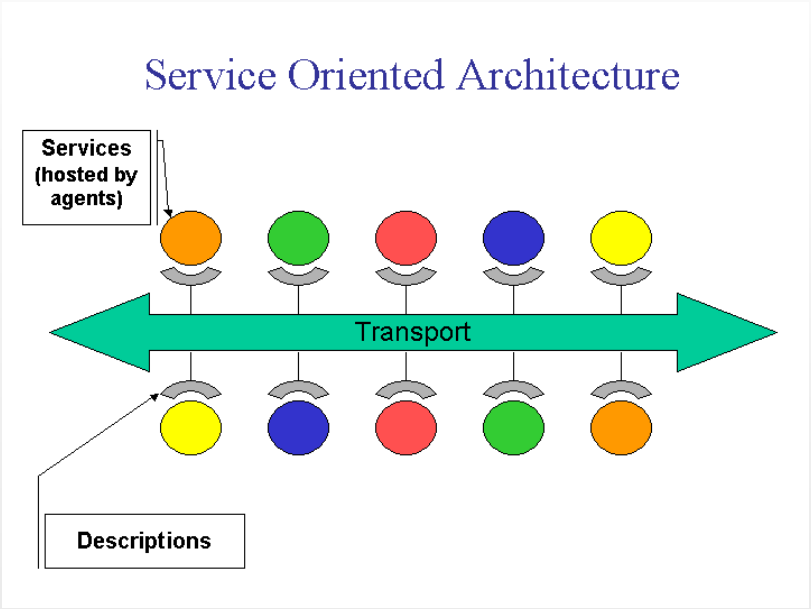


Image from http://lists.w3.org/Archives/Public/www-ws-arch/2003Feb/0030.html

Třívrstvý model

MVC struktura aplikace

- 3vrstvá architektura „v malém“
  - oddělení vrstev, které se nejčastěji mění
  - typicky uživatelské rozhraní

- Základ: návrhový vzor Model-View-Controller
- Realizace pomocí tříd/objektů
  - hraniční – rozhraní
  - řídící – mechanismy
  - datové – perzistence údajů
  - + knihovní, systémové

Hraniční třídy

- Obsahují funkčnost přímo závislou na okolí
  - dialog s uživateli
  - komunikace s externími systémy
    - aktéři přistupují k systému pouze přes hraniční objekty
- Úlohy hraničních objektů
  - prezentace / čtení informací
  - zapouzdření externích prvků
    - předávání dat od / k interním objektům
  - zpracování informací, jejich uchovávání
- Podle toho základní atributy a metody

Hledání hraničních tříd

- Objekty zřejmé z popisu rozhraní v PU
  - specifikace požadavků, znalost systému
  - odraz v prototypu (možno přímo převzít)
- Alespoň jeden H objekt pro každého aktéra
  - nemusí platit pro abstraktní aktéry
- Rozbor případů použití
  - vyznačit místa s rozhraním na systém
  - H objekt obsahuje primárně funkčnost závislou na podobě rozhraní (ne zpracování)
- Obvykle kombinace => konsolidace výsledků

Datové (entitní) třídy

- Zapouzdřují informace uchovávané delší dobu
  - typicky přežívají mezi případy použití
- Atributy
  - uchovávaná informace, jednoduché i strukturované
    - samostatně zpracovávaná data → separátní objekty
- Metody
  - funkčnost svázaná s přístupem k informacím
  - životní cyklus objektu, přístup k datům, zpracování dat, složitější přímo související analýzy apod.

Hledání datových objektů

- Zřejmé z výsledků analýzy
  - často doménové objekty
  - data vstupující do systému
- Rozbor případů použití
  - vyznačit data používaná při zpracování
  - vyhledat přímo svázané operace
  - zapouzdřit do objektů
- Úpravy
  - separovat společné vlastnosti => dědičnost
- Realizace v SQL/RDBMS až později

Řídící třídy

- Řízení aplikace
  - stavové přechody, mechanismy (business rules)
  - funkčnost, která se může změnit nezávisle na datech
  - některé nelze nebo není dobré svázat s hraničními / datovými třídami
    - komplexní kontroly nebo zpracování
    - koordinace více objektů

=> jejich vyčlenění do specielních tříd

- Vlastnosti řídících tříd
  - obvykle jen pro daný případ použití
  - obvykle poměrně malé, těžiště v několika metodách

Hledání řídících objektů

- Typicky v rozborech případů použití
  - transakční operace
  - izolování hraničních a datových objektů
  - zajištění komunikace mezi objekty
- Technika
  - pro začátek jeden případ použití – jeden řídící objekt
  - hledat typické chování, přiřadit řídícímu objektu
  - více různých => vyrobit potřebné další řídící objekty
  - typické chování nenalezeno => řídící objekt zbytečný

Systémové a knihovní třídy

- Realizace nízkoúrovňových úloh
  - komunikace
  - soubory
  - zobrazování
  - …
- Cíl: reuse (využití již hotového)
  - znalost knihoven
  - namapování hraničních, případně datových tříd

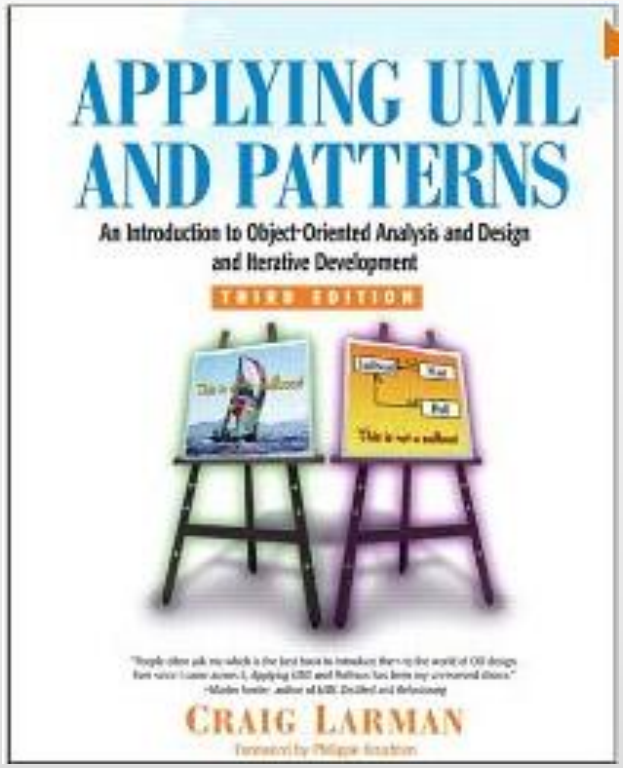
Úvaha o strategiích alokace řízení

- Hraniční x datové x řídící objekty
  - každý může obsahovat řídící mechanismy
- Cíl: minimalizace dopadu, lokalita změn
  - uživatelské / systémové rozhraní
  - reprezentace a předávání informace
  - malá změna požadavků => lokální změna implementace
- Druhy aplikací podle typu řízení
  - výpočetní a řídící systémy, dialogové, kombinované
  - vyvážené

Detailní návrh („design in the small“)

Zodpovědnosti a metody tříd

- Cíl: interní funkčnost jednoznačně odvozená od požadované vnější
  - trasovatelnost požadavků (údržba!)
- Rozbor případů použití
  - akce → zodpovědnosti tříd
  - zodpovědnost → metoda
- Kdy
  - jakmile potřebuji (zodpovědnosti)
  - jakmile je dost informací (metody)
    - může potřebovat 0..3 iterace
  - obvykle Z:M = 1:N



Samostudium: RUP Activity: Use-Case Analysis

Mechanismy spolupráce tříd

- Mechanismy = více tříd na realizaci jedné funkčnosti
  - ideální stav: zobecnitelné
- Využití návrhových vzorů
  - Listener, Strategy, …
  - návrh postupu předávání zpráv (zodpovědnosti)
  - soulad s architektonickými principy a pravidly
- Mechanismy rámují jednotlivé zodpovědnosti
  - rozbor scénáře případu užití
  - správně identifikovat spouštěcí událost

Vztah PU – třídy/metody

Realizace případu užití

- Scénář jednoho PU
  - popisuje konkrétní interakci instancí => mechanismy
  - generuje metody jejich tříd
  - model „Use Case Realization“
- Soubor scénářů
  - vytváří celkový souhrn chování třídy
  - při vhodném výběru její úlohu => rozhraní

- Výsledek: tvorba/doplňování objektového modelu
  - třídy
  - metody, parametry
  - algoritmy

Doplňující info: RUP Guidelines: Use-Case Realization

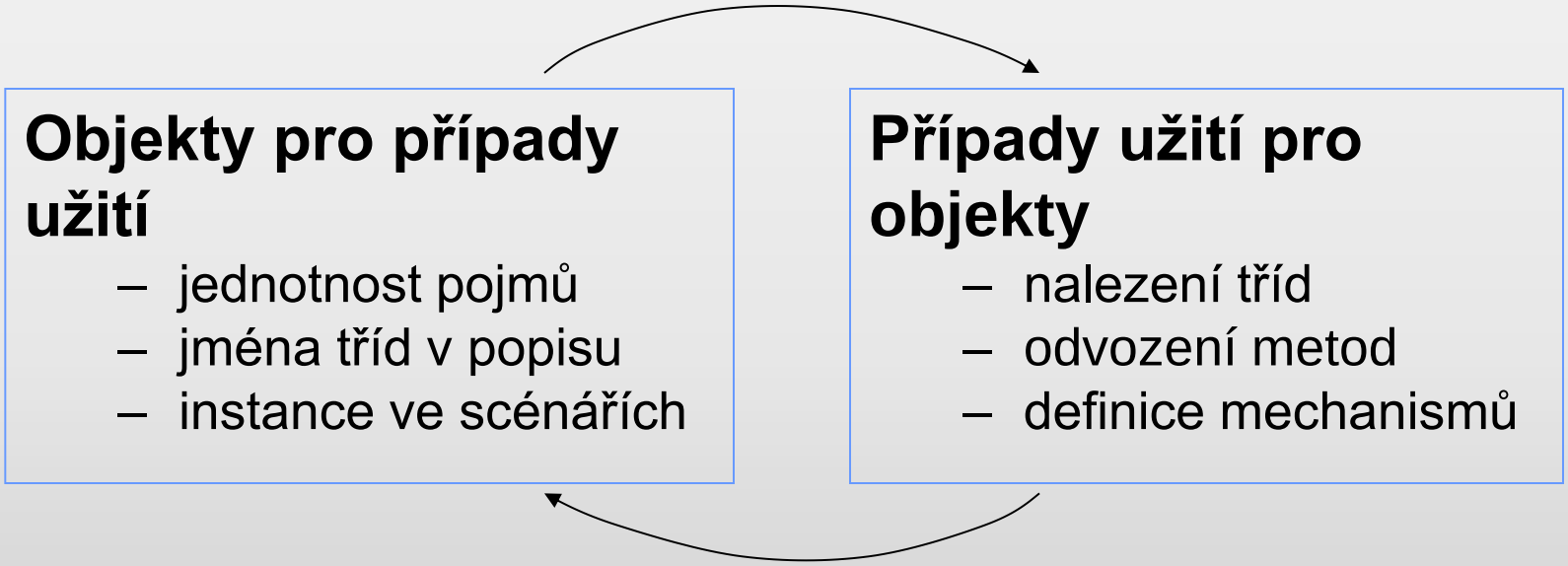
Stavové modely objektů

- Přechody mezi stavy
  - na základě volání metod
  - podmíněné částmi interního stavu

- Objekty řízené podněty
  - nezávislost na stavu: volání metod v libovolném pořadí
  - časté pro datové a servisní objekty
- Objekty řízené stavem
  - stav určuje možná volání: pořadí metod tvoří protokol
  - přechody mezi stavy chráněné podmínkami
  - časté pro řídící objekty, akční členy

Souvislost analýzy struktury a dynamických aspektů

- Iterativnost = základní vlastnost objektové analýzy a návrhu



Nutnost konsolidovat model

- Z analýzy scénářů
  - pohled na jednu roli třídy => některé metody třídy
  - ze souboru scénářů všechny metody třídy potřebné pro realizaci jednotlivých případů použití
- Typicky práce více lidí a/nebo rozložená v čase

- Výsledek: nekonzistence rozhraní
  - názvy metod, syntaxe volání
  - duplikáty, podobnosti, rozpory v sémantice
  - příčiny: různé scénáře, různí autoři

Doplňkové informace: RUP Checkpoints: Design Class

Úpravy rozhraní a vnitřku tříd

- Konsolidace syntaxe a sémantiky tříd
  - komplementarita metod (Create – Destroy)
  - znalost domény a implementace
    => další atributy a metody
  - snaha o minimalitu, reuse
  - viditelnost vlastností (public, private)

- Následné zpětné úpravy scénářů

Úpravy celkového chování třídy

- Role třídy (zodpovědnosti, mechanismy)

  => sady metod, rozhraní
  - diagram tříd, komponent, kompozitů
- Postup implementačního mechanismu

  => pořadí volání metod, protokol
  - obvykle stavové modely

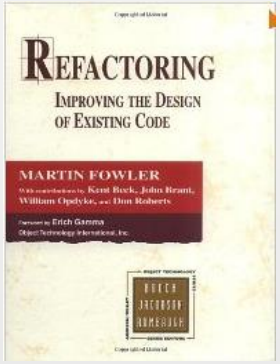
- Dohromady kontrakt třídy
  - důležitá součást specifikace návrhu

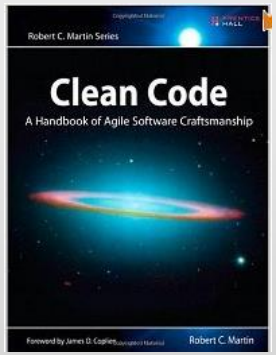
Úpravy vztahů a modelu

- Úpravy objektového modelu
  - vyhledání opakujících se prvků => rodičovské třídy
  - vyjasnění vztahů (asociace => agregace, c-s, …)
  - nové řídící objekty, nové vztahy mezi třídami

- Refactoring

  = změna implementace bez změny funkčnosti
  - mění design (v extrémním případě architekturu) „za pochodu“
  - podmíněno používáním automatizovaných testů





Samostudium: RUP/XP Guidelines: Test-first Design and Refactoring a části Concept: Agile Practices and RUP

Rekapitulace

Architektura a návrh: rekapitulace

- Známe (skoro) všechny detaily implementace
  - rozpracovaná funkčnost validuje architekturu
  - vazba na konkrétní technologie
  - vyřešený způsob komunikace na okolní prostředí
  - struktura aplikace a rozhraní mezi částmi
  - používané konvence a návrhové vzory

- Co dál
  - naprogramovat co zbývá
  - integrační testy