profinit.
PROFESSIONALS IN IT

Architektura, design, konstrukce

Dnešní program

profinit.
PROFESSIONALS IN IT

- Úvod
  - Připomenutí – poslat téma zápočtové práce.
- Architektura, Design
  - Architektura webové aplikace
  - Design patterns
  - IoC, AOP …
- Konstrukce
  - Kuchařky, dočasná řešení …
- Když zbude čas
  - Integrace a integrační styly

Jak se pozná dobrá architektura

profinit.
PROFESSIONALS IN IT

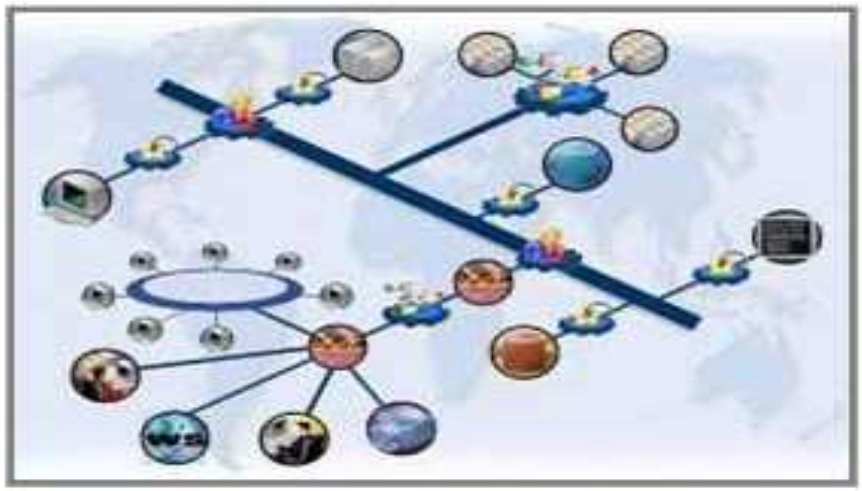
Dobrá architektura

profinit.
PROFESSIONALS IN IT

- Dokáže pojmout nové požadavky
- Systém je udržovatelný i pro letech v provozu
- Není zbytečně komplexní
  - Je pochopitelná
  - Je implementovatelná
  - Je zdokumentovaná :-)

Realita

- organizace mají typicky stovky aplikací
  - na každou věc je jedna „nejlepší“ aplikace
  - různé technologie a dodavatelé aplikací
- Conway’s law: “Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations.”



Architektura vs. design

profinit.
PROFESSIONALS IN IT

- Zjednodušeně:
  - Architektura – high-level
  - Design – detailní
- Fowler: „Architecture is about the important stuff. Whatever that is …“

Architektura, Design

profinit.
PROFESSIONALS IN IT

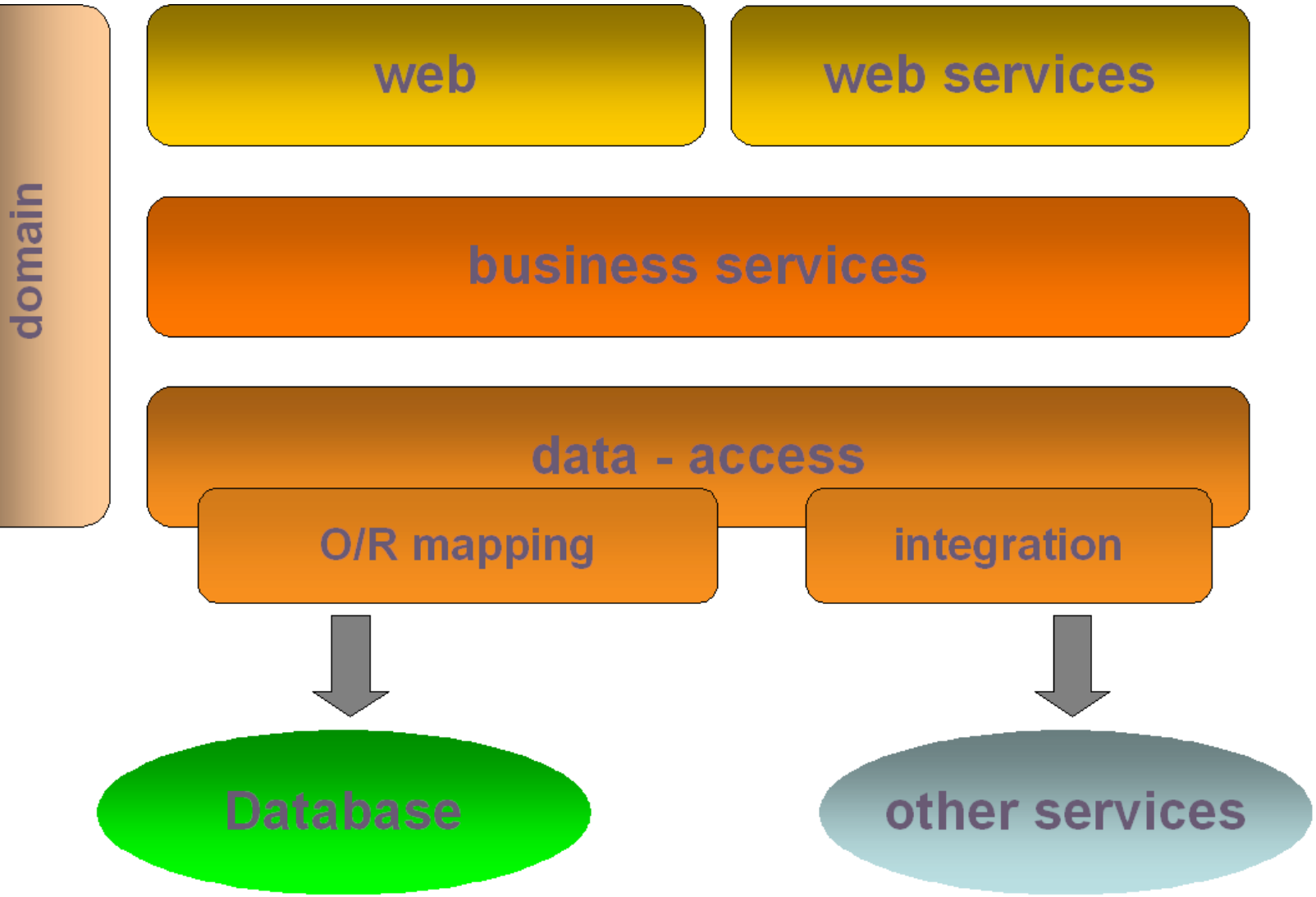
- Úroveň:
  - vývoj jednotlivé aplikace
    - objektově orientované programování
    - design patterns (architektonické) a jejich reálné použití na projektu
  - integrace aplikací

Dobrá rada

profinit.
PROFESSIONALS IN IT

- Neobjevujte kolo

Klasická architektura webové aplikace



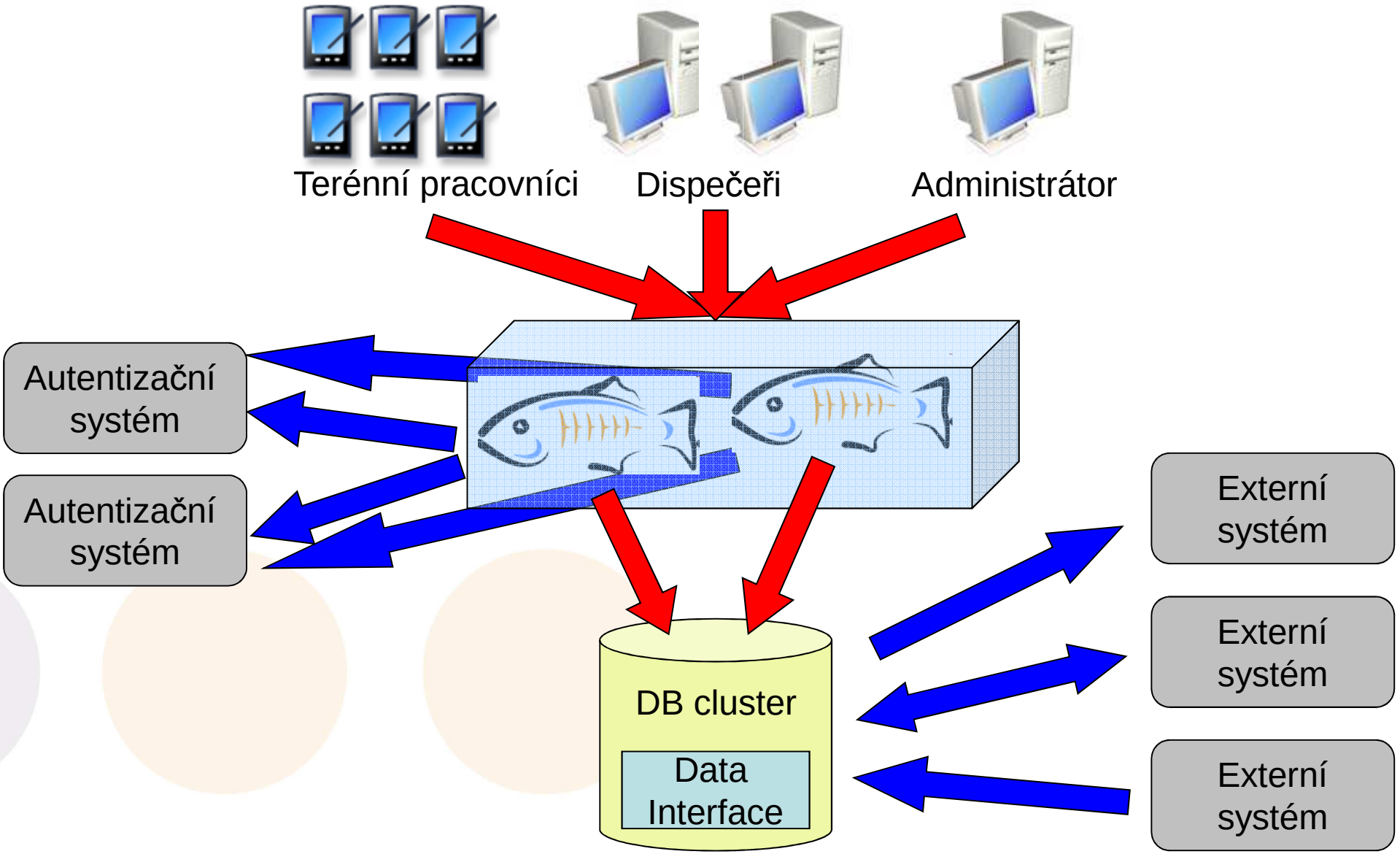
- Jednoduchá
- Flexibilní
  - Rozšířitelná
- Nevýhody:
  - Občas jen převolávání vrstev

Je vůbec taková složitost potřeba?

profinit.
PROFESSIONALS IN IT

- Není
  - Ale rozhodnutí musí být činěno při znalosti důsledků
    - Např. obtížnější rozvoj.
- V naprosté většině případů se vyplatí „udělat to pořádně“
  - Člověk nikdy neví, co život přinese.
- U větších aplikací bývá architektura složitější.

Složitější architektura



Architektura a design

profinit.
PROFESSIONALS IN IT

- objektově orientované programování a design patterns
  - pokud chci rozumět DP, musím chápat OOP
    - polymorfismus
    - concrete inheritance
    - abstract inheritance
    - interface
    - abstract class
    - overriding
    - delegation
    - composition

Architektura a design

profinit.
PROFESSIONALS IN IT

- návrhové vzory od GoF* ve dvou větách
  - *Program to an interface, not an implementation.*
  - *Favor object composition over class inheritance.*

- *) GoF = Gang of Four – viz http://en.wikipedia.org/wiki/Design_Patterns_(book)

Návrhové vzory

profinit.
PROFESSIONALS IN IT

- kdy má smysl zabývat se návrhovými vzory
  - člověk zná, rozumí a aktivně používá základy OOP
  - člověk zná, rozumí a aktivně používá základní triky lowlewel OO Designu (na bázi polymorfismus, interface,overriding)
  - člověk chápe jaké problémy musí věcně při designu řešit (dekomponovat systém, poskládat systém, "příposlech" ...)
  - člověku nejsou cizí úvahy o předmětu dekompozice
  - člověk už něco navrh, naprogramoval, opravoval,modifikoval rozvíjel
  - a HLAVNĚ zpětně prostudoval, pochopil,diskutoval, redesignoval
  - člověk zjistí, že chápe *jaké* problémy, *proč* a *jak* GoF řeší;
  - člověk zjistí, že spoustu věcí udělal nějak podobně

Návrhové vzory – vztah lidí k DP

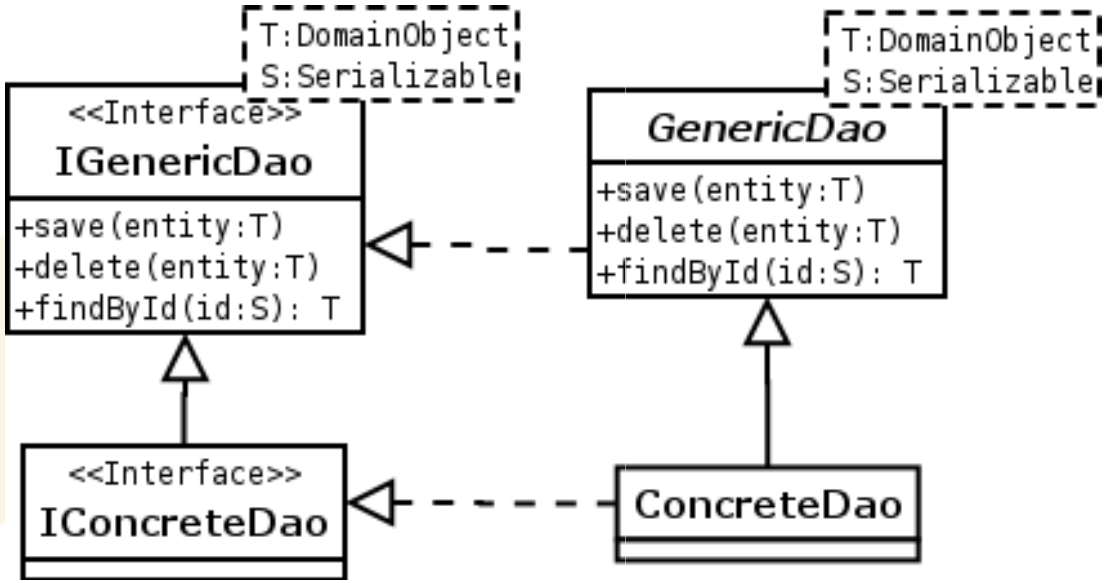
profinit.
PROFESSIONALS IN IT

- lidé, kteří znají základy OO a intuitivně od přírody navrhují elegantně, mají dar pro design;
  - DP padnou na úrodnou půdu, ale řeknou "no jasně"
- lidé, kteří po delší či kratší cestě na 1 až n pokusů tomu přijdou na chuť
- lidé, kteří přečtou všechny knihy, moc se jim to líbí, nikdy ale elegantně sami nenavrhnou nic co má víc než 5 programů

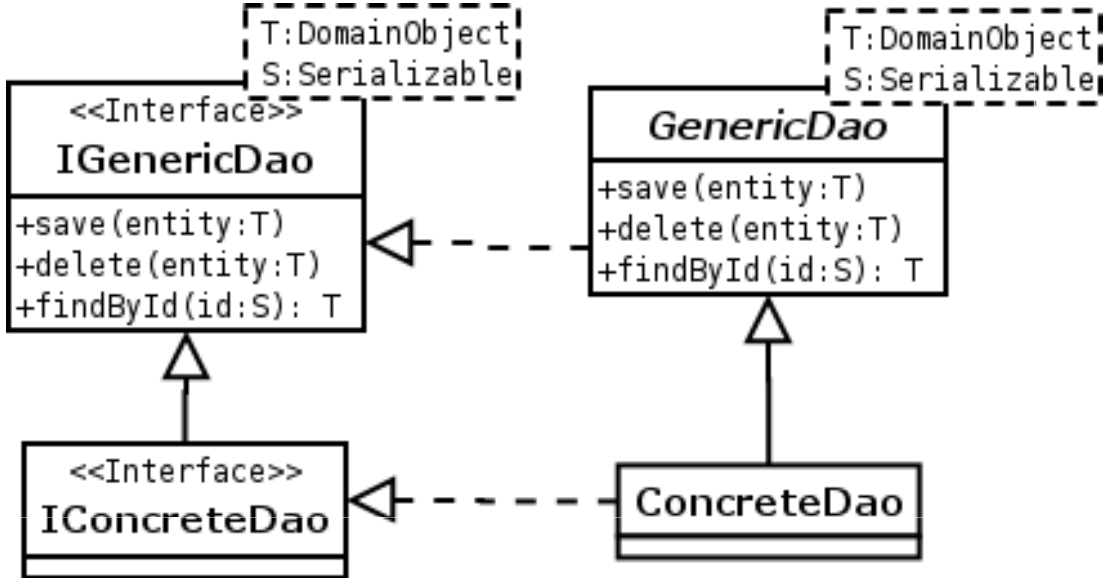
Příklad Design Patternu

profinit.
PROFESSIONALS IN IT

- Návrhový vzor GenericDAO
  - Oddělení kódu pracujícího s databází (ideálně prostřednictvím ORM) do samostatné vrstvy
  - Pro každý doménový objekt jeden interface a jeho implementace



GenericDAO



- IGenericDAO
  - Metody společné pro všechny DAO
  - Typicky obsahuje CRUD operace
- GenericDAO
  - Implementace základních metod
  - Ideální pokud máme k dispozici generické typy (Java 1.5)

AOP

- Jen další TLA? Buzzword?
  - Nikoliv. AOP je překvapivě užitečný koncept
- AOP = Aspect Oriented Programming
- Využití
  - Logování
  - Transakce
  - A mnoho dalšího
- Loose code coupling

Příklad (Spring)

```
<bean class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
        <property name="proxyTargetClass" value="false"/>
        <property name="beanNames" value="*Service,*Dao"/>
        <property name="interceptorNames" value="loggingAdvice"/>
</bean>

<bean id="loggingAdvice"
   class="org.springframework.aop.interceptor.CustomizableTraceInterceptor">
        <property name="useDynamicLogger" value="true"/>
        <property name="hideProxyClassNames" value="true"/>

        <property name="enterMessage" value="Vstupuji do metody '$[methodName]'
   tridy [$[targetClassShortName]]. Parametry: $[arguments]"/>

        <property name="exitMessage" value="Navrat z metody '$[methodName]' tridy
   [$[targetClassShortName]]. Navratova hodnota: $[returnValue]. Volani trvalo:
   $[invocationTime] ms"/>

        <property name="exceptionMessage" value="V metode '$[methodName]' tridy
   [$[targetClassShortName]] doslo k vyjimce $[exception]."/>
</bean>
```

IoC aka DI

profinit.
PROFESSIONALS IN IT

- Další TLA
- IoC = Inversion of Control
- DI = Dependency Injection

- Závislosti definujeme deklarativně
- Výhody
  - Flexibilita
  - Snadnost změn
- Spring, Google Juice, EJB 3, …

Nic se nemá přehánět

profinit.
PROFESSIONALS IN IT

```
@FilterDefs({
        @FilterDef(name="CPojisteni_policka", parameters = @ParamDef(name = "platnost", type = "date")),
        @FilterDef(name="CPojisteni_souhrnneLimity", parameters = @ParamDef(name = "platnost", type = "date")),
        @FilterDef(name="CPojisteni_okamzikyAAkce", parameters = @ParamDef(name = "platnost", type = "date")),
        @FilterDef(name="CPojisteni_algoritmy", parameters = @ParamDef(name = "platnost", type = "date"))
})
@Cache(usage = CacheConcurrencyStrategy.READ_ONLY)
@Entity
@Table(name = "c_pojisteni")
public class CPojisteni extends Konfigurace {

    @OneToMany(fetch = FetchType.LAZY,
                        mappedBy = "pojisteni")
    @Fetch(value = FetchMode.SELECT)
    @Sort(type = SortType.NATURAL)
    @Filter(name = "CPojisteni_policka",
              condition = TemporalniVztah.FILTER_PLATNOSTI)
    @Cache(usage = CacheConcurrencyStrategy.READ_ONLY)
    private SortedSet<CPojisteniPolicko> policka;

…
}
```

profinit.
PROFESSIONALS IN IT

Konstrukce

Konstrukce - dnešní obsah

profinit.
PROFESSIONALS IN IT

- Úvod
- statická analýza kódu
- konvence pro psaní kódu
- kuchařky
- dočasná řešení

Příklad – najdi chybu

profinit.
PROFESSIONALS IN IT

```
Connection con;
try {
 con = getConnection();
 Statement stmt = con.createStatement();

 ResultSet rs = stmd.executeQuery("SELECT jmeno, prijmeni "
                                  + "FROM osoby");
 while (rs.next()) {
   Osoba osoba = new Osoba();
   osoba.setJmeno( rs.getString("jmeno") );
   osoba.setPrijmeni(rs.getString(" prijmeni ") );
   osoby.add(osoba);
 }
} catch (Exception e) {
   …
} finally {
   con.close();
}
```

Překvapivá pointa :-)

```
Connection con;
try {
 con = getConnection();
 Statement stmt = con.createStatement();

 ResultSet rs = stmd.executeQuery("SELECT jmeno, prijmeni "
                                  + "FROM osoby");
 while (rs.next()) {
   Osoba osoba = new Osoba();
   osoba.setJmeno( rs.getString("jmeno") );
   osoba.setPrijmeni(rs.getString(" prijmeni ") );
   osoby.add(osoba);
 }
} catch (Exception e) {
   …
} finally {
   con.close();
}
```

- Chybou je boilerplate kód

Programovat přece všichni umíme

profinit.
PROFESSIONALS IN IT

- Ale důležitá je kvalita
  - Modularizace
  - Dodržování konvencí
  - Komentování …
- Jak se zlepšovat
  - Čtení cizího kódu
  - Code revisions
  - Čtení chytrých knížek
  - Open source contribution

Konstrukce

profinit.
PROFESSIONALS IN IT

- konvence pro psaní kódu
  - pro Java
    - Sun Code Convention CodeConventions.pdf
    - Naše sepsané po X revizích kódu
      - Java Profinit_JavaPrgTechniques.doc
      - DBS Profinit_DbsPrgTechniques.doc

Konstrukce

- kuchařky
  - v designu jsme vymysleli, že to budeme dělat takhle
    - víme, proč to tak děláme
    - víme, že to má nevýhody
    - ale budeme to dělat takhle a ne jinak
      - **udržovatelnost**
    - tedy na to napíšeme kuchařku, podle které to udělá kde kdo
      - příklad kuchařky pro VC Balíčků IPBPBAL_KucharkaProV&C.doc
- nedělat „dočasná“ řešení
  - pak při dodávce koukáte, kam je co nastavené a nechápete

Dočasná řešení (z praxe ;-))

profinit. PROFESSIONALS IN IT

```
private String encPass = „12345678“
  //TODO pouzit silnejsi heslo!!!
```

- Nebo:

```
try {
…
} catch (Exception e) {
    e.printStackTrace() // TODO
  osetrit lepe!
}
```

Statická analýza kódu

profinit.
PROFESSIONALS IN IT

- statická analýza kódu
  - najde vám hodně chyb zcela zadarmo
  - puštěno po létech vývoje na aplikaci Balíčky našlo 955 chyb
  - příklad
    - Comparison of String parameter using == or !=
    - Class defines equals() and uses Object.hashCode()
    - Invocation of toString on an array
    - Method may fail to close database resource
    - Method ignores return value
      - String dateString = getHeaderField(name); dateString.trim();
  - Nástroje – PMD, FindBugs

FindBugs Report

Metrics

118666 lines of code analysed, in 3148 classes, in 371 packages.

Metric	Total	Density*
High Priority Warnings	150	1.26
Medium Priority Warnings	810	6.83
Total Warnings	**960**	**8.09**

(Defects per Thousand lines of non-commenting source statements)*

Summary

Warning Type	Number
Bad practice Warnings	308
Correctness Warnings	87
Malicious code vulnerability Warnings	203
Multithreaded correctness Warnings	20
Performance Warnings	119
Dodgy Warnings	223
Total	**960**

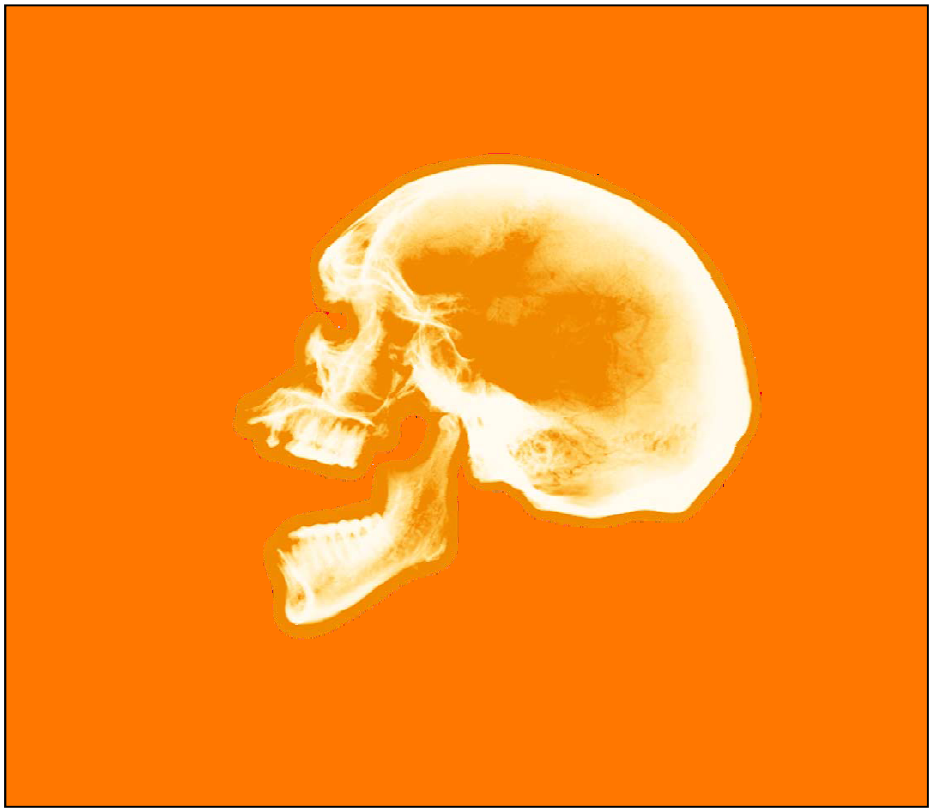
Cvičení

- Navrhněte objektový model této učebny.

Diskuse

profinit.
PROFESSIONALS IN IT

- Komentáře
- Otázky
- Připomínky
- Upřesnění
- Poznámky
- …



Optional slajdy – když zbyde čas

Proč integrace?

profinit.
PROFESSIONALS IN IT

- máme naše „nejlepší“ aplikace
  - jak přidáme novou funkcionalitu
  - kterou aplikaci upravíme?
- vztah uživatelů k hranicím systému
  - uživatelům je jedno, který systém danou akci řeší
  - uživatel očekává, že danou logickou akci bude dělat na jednom místě
- sdílení dat

integrační výzvy:-)

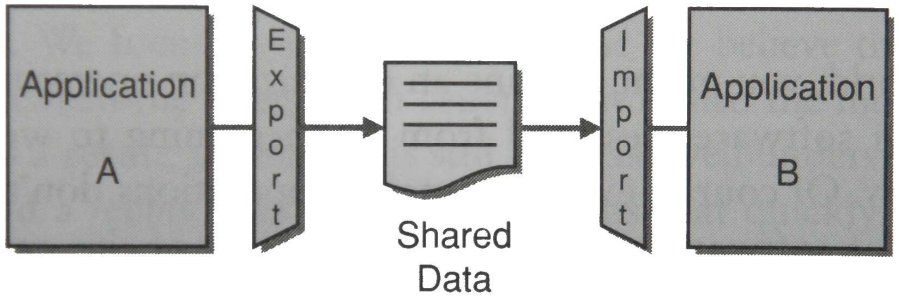
profinit.
PROFESSIONALS IN IT

- integrované řešení má velký rozsah a dopad
  - výpadek stojí hodně peněz a ovlivňuje hodně lidí
- staré systémy
- nedostatek standardů
  - XML a WebServices nestačí na všechno (ale hodně pomůžou:-)
  - rozdílné platformy, na kterých řešení funguje
    - taková drobnost - little a big endian v ukládání čísel

Integrační styly

- File transfer
- Shared database
- Remote procedure invocation
- Messaging

Integrace - přenos souborů



- Soubory jsou univerzální.
- Formát?
  - Musí být univerzální.
- Není třeba znát implementační detaily – soubor je „rozhraním“ aplikace.
- Problémy
  - Unikátnost jmen souborů
  - Mazání starých souborů – kdo to udělá a jak pozná, že soubory již nejsou potřeba?
  - Vícenásobný přístup – zamykání?
- Kdy (jak často) je vytvářet a jak často je „konzumovat“?
  - Pokud updaty nejsou časté – systémy mohou být „out of synchronization“

Integrace – sdílená databáze

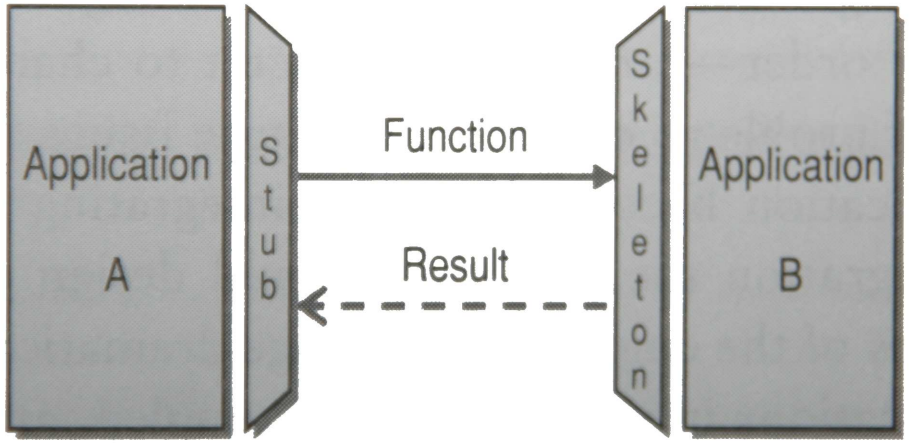
profinit.
PROFESSIONALS IN IT

- Integrované aplikace používají stejnou databázi
- Transakce!
  - Vícenásobný přístup – ze všech integrovaných aplikací.
- SQL je standardizované a velmi rozšířené
- Problém – dobře navrhnout sdílenou databázi.
  - Schéma, které splňuje potřeby vícero aplikací je velmi těžké navrhnout.
  - Když už ho navrhneme, je těžké ho udržovat.
  - Většinou má není optimální z hlediska výkonu (hodně joinů).
- Hrozí deadlocky.
- Používání sdílené databáze zpomaluje kritické aplikace => tlak na separaci schémat.
- Distribuované aplikace mají pomalý přístup ke sdílené databázi.

Integrace – vzdálené procedury

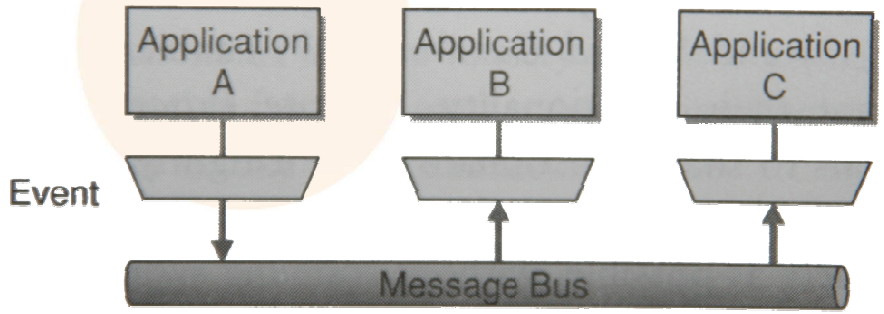
profinit. PROFESSIONALS IN IT

- Jedna aplikace volá procedury jiné aplikace
- Aplikace se samy starají o integritu svých dat.
  - Selecty a updaty jsou realizovány pomocí volání funkcí
  - Změna interních datových struktur nijak neovlivní ostatní aplikace.
- Mnoho technologií – CORBA, COM, .NET Remoting, Java RMI, RPC, Web Services
  - Web Services obvykle nemají problémy s firewally (díky použití http protocolu).
- Problém – odlišný výkon lokálního a vzdáleného volání.
- Problém – vzdálené volání může selhat (a aplikace na toto nemusí být připravena).
- Tightly coupled applications



Integrace - messaging

- Podobá se přenosu souborů
  - Mnoho malých datových paketů – mohou být vytvářeny a přenášeny rychle a jednoduše.
  - Je potřeba detekovat ztrátu packetu.
  - Stejně jako v předchozím případě je schéma skryto před ostatními aplikacemi => snadnější údržba.
  - Přenos dat probíhá asynnchronně
    - Odesílatel není blokován (i v případě, že je třeba získat potvrzení o doručení).
    - Odeslání zprávy nevyžaduje, aby příjemce byl on-line.



Integrace - příklad

profinit.
PROFESSIONALS IN IT

- Pojišťovna - integrace přes sdílenou databázi
- Integrované aplikace
  - předpisy (inkasní, zajistné, provizní)
  - účtování
  - inkaso/exkaso
  - pojistné události
  - zajištění
  - složenky, přeplatky/nedoplatky/vratky
  - **Pojištění Podnikatelských Rizik**
  - **ČKP**
  - **Balíčky - Notebook, internet, intranet**

Integrace – příklad (2)

profinit.
PROFESSIONALS IN IT

- Snaha o
  - Jednoduchý vývoj
  - Integrace dat co nejjednodušší
- Integrace mezi interními systémy
  - **DATOVÁ INTEGRACE přes jednu DB**
  - Pattern tabulek a procedur rozhraní
  - Ve většině případu integrace point-to-point

Integrace – příklad (3)

profinit.
PROFESSIONALS IN IT

- Výhody
  - Zpočátku jednoduché
  - Vše v jedné db
    - snadno dostupná data
    - Snadnější údržba db
- Nevýhody
  - Přísný diktát na nové systémy (u ČKP by bylo výhodnější jiné řešení)
  - Nemožnost zapojení externího systému bez úprav
  - Pattern tabulek a procedur rozhraní
  - Integrace point-to-point

Integrace – příklad (4)

profinit.
PROFESSIONALS IN IT

- Řešení pomocí integrační platformy by bylo v některých případech výrazně jednodušší
  - programování přímo podporovaných konceptů
  - dohled
  - ...
- Velká omezení pro vývoj dalších systémů
- Přechod na jinou integrační strategii složitý
- Prvotní rozhodnutí o integrační strategii důležité