Doporučené postupy v programování

Úvod, vývoj software

Lubomír Bulej

KDSS MFF UK

Otázka na úvod

Kam zařadit vývoj software?

Věda?

Řemeslo?

Umění?

Poznámky:

- Věda, zvlášť v našem oboru, to je algebra, logika, diskrétní matematika, ..., teorie informace, atd. Věda, to je čistý svět jasných (jak pro koho) pravd, nevyvratitelných důkazů a (v našem případě) turingových strojů.
- Cožpak věda nestačí? Zdá se že ne - jedna věc je něco vymyslet, druhá věc je něco realizovat. Matematická formule se nedá spustit ani nenajde článek na webu, stejně jako v projektu na barák se nedá bydlet. Takže vývoj software je také proces, jehož výsledkem je něco hmatatelného.
- Cožpak věda a řemeslo nestačí? Co je to umění? Zajímavá je definice umění jako "dokumentace tisíce zajímavých rozhodnutí".

  dokumentace
  : ... dá se to sdílet s ostatními ...

  tisíce
  : ... mělo by v tom být vidět úsilí ...

  zajímavých rozhodnutí
  : ... sada rozhodnutí, které stojí za to zaznamenat ...

- "The creation of art is not the fulfillment of a need but the creation of a need. The world never needed Beethoven's Fifth Symphony until he created it. Now we could not live without it." ~ Louis Kahn.
- Umění poskytuje prostor pro kreativitu, intuici, estetiku, nápady přicházející znenadání a odnikud, nadšení a zápal. Existuje snaha některé aspekty formalizovat a přesunout je do

kategorie "řemeslo". Na některé typy projektů to funguje, ale eliminovat esenci umění z programování asi nejde.

- Rands in Repose: Signs of Art
- David Majda: Malá noční úvaha nad programováním

Vývoj implikuje proces

Vývoj software je jako...

- psaní (knihy, článku, dopisu)?
- pěstování plodin?
- pěstování ústřic (kvůli perlám)?
- stavba (katedrály, rodinného domu, psí boudy)?
- ...?

Poznámky:

- Psaní knihy je většinou individuální záležitost, kniha když se dopíše tak je hotová, při psaní knihy je důraz kladen na originalitu.
- Při pěstování plodin se zasadí semena, vyrostou rostlinky, o ty se člověk stará až nakonec přinesou zasloužené plody.
- Pěstování ústřic zachycuje růst a vývoj softwaru jako proces nabalování tenkých vrstev na nějaké malé jádro.
- Představa "stavby" software se zdá užitečnější než představa jeho "psaní" nebo "pěstování". Je kompatibilní s tím, co naznačuje metafora "ústřičné farmy", ale navíc poměrně intuitivně implikuje potřebu různých fází plánování, příprav, vlastní stavby, konečných úprav, inspekcí v průběhu a na závěr stavby, a koneckonců i údržby.
- Další podobnosti: projekt, schválení projektu, příprava stavby, základy, hrubá stavba, zastřešení, vnitřní rozvody, omítky, malování, vnitřní vybavení, zahrada. Ale také např. náročnost a cena změn prováděných během stavby, jako třeba posunutí zdi.
- Stavební metafora navíc škáluje s velikostí projektů. Jinak se připravuje stavba rodinného domu, stadionu pro 50000 lidí, katedrály, nebo psí boudy. S rostoucí složitostí vyniká potřeba vyšší míry organizace a roste závažnost důsledků špatného plánování. A konečně, u stavby je prostor i pro umění.

Metafora jako cesta...

... k pochopení procesu vývoje software

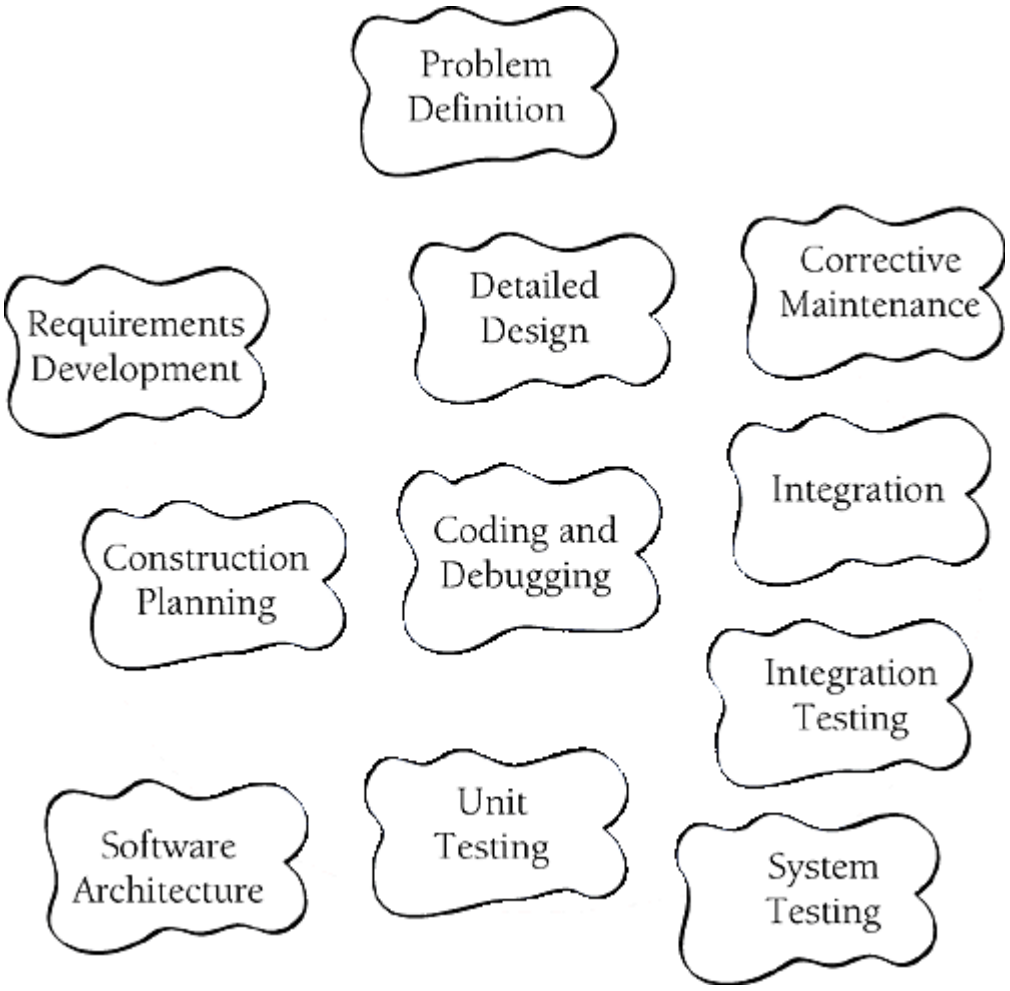
- heuristika vs. algoritmus
- obecný návod jak přistupovat k problémům
- některé metafory jsou lepší než jiné

Poznámky:

- Síla analogie. Přirovnáním něčeho složitého a těžko pochopitelného k něčemu co je dobře známé a pochopitelné je možné získat určitý vhled, který v důsledku umožňuje pochopit složité. Dá se tomu říkat modelování.
- Pozor! Metafora je pouze heuristika a jako taková nedává přesný návod jak najít odpověď, ale jak ji hledat. Kdybychom znali přesný postup, tak by byl vývoj software o mnoho jednodušší, ale tak daleko zatím nejsme a asi nikdy nebudeme. Největší problém při vývoji softwaru je konceptualizace problému a mnoho chyb při programování jsou ve skutečnosti konceptuální chyby.

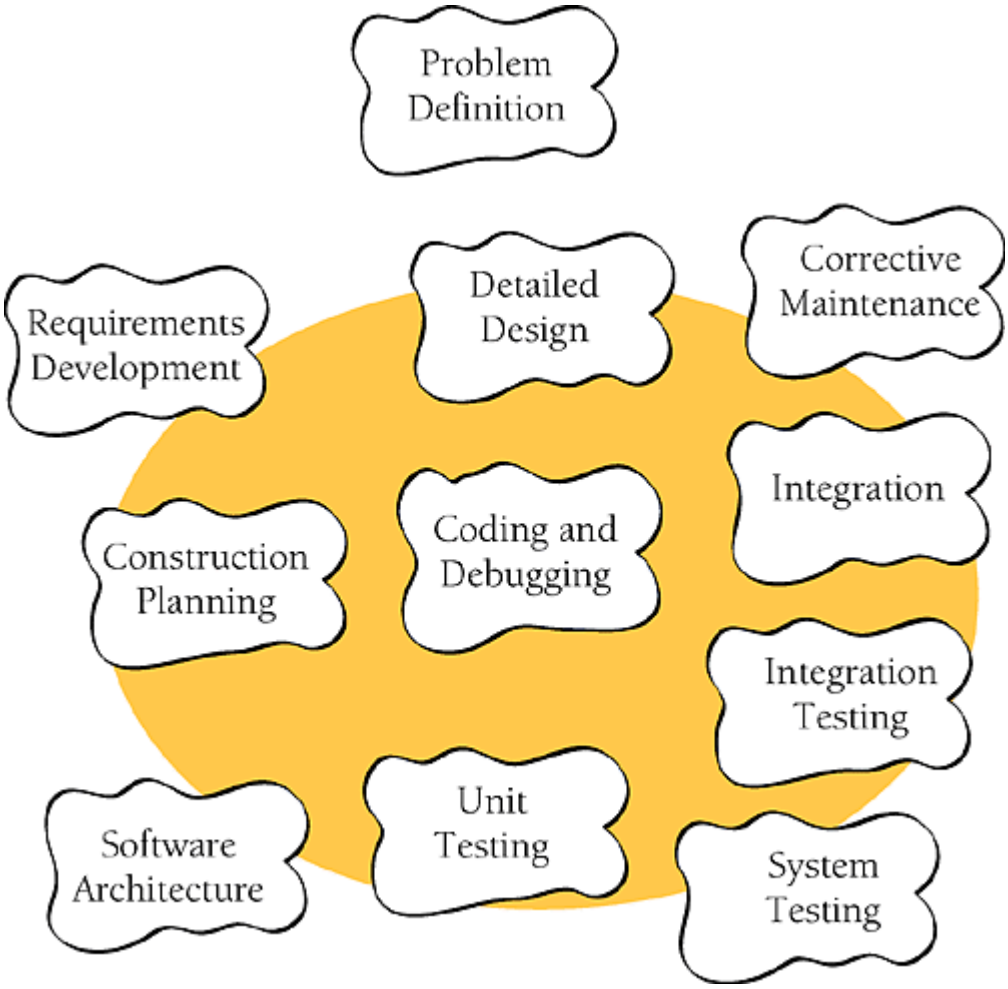
O čem bude tento předmět?

O vývoji software...



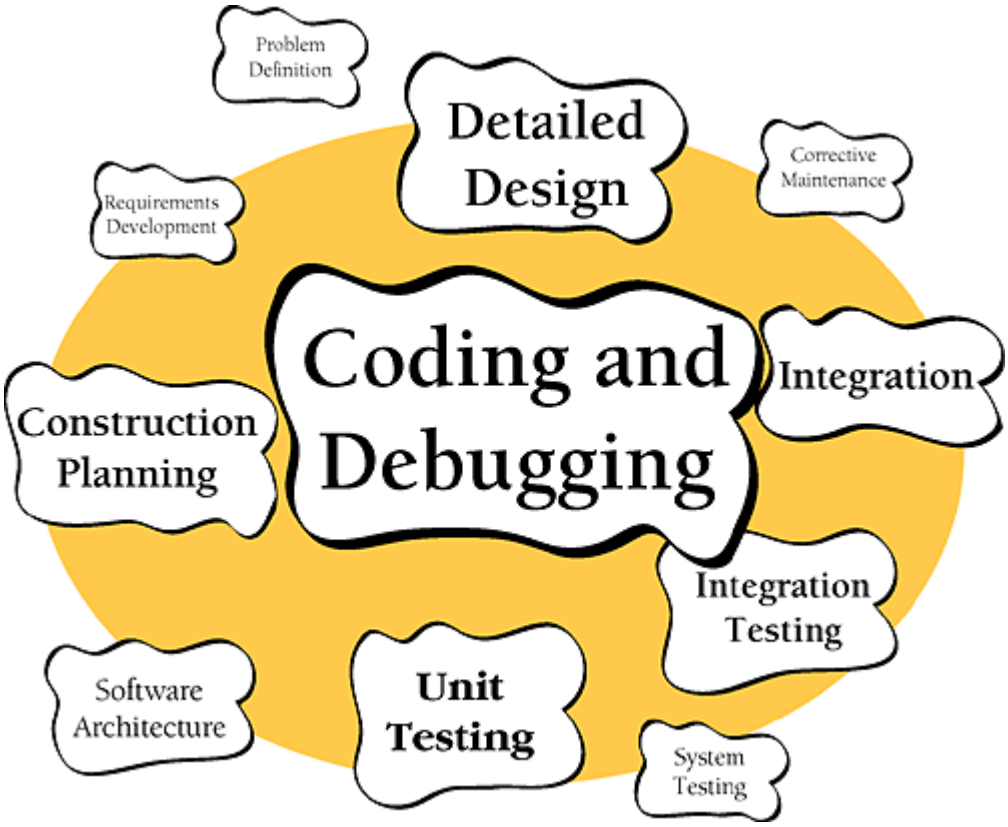
O čem bude tento předmět?

... a zejména o aktivitách, které se týkají programování ...



O čem bude tento předmět?

... tedy o psaní kódu a souvisejících činnostech.



Proč právě programování?

Co je na psaní kódu zvláštního?

Kód = nejdůležitější část software!

Kód = nejdůležitější část software!

V čem spočívá důležitost kódu?

- Design se dá odbýt nebo dělat průběžně
- Dokumentace nemusí existovat
- Testování můžeme nechat na uživatelích

Ale kód je to, co ...

- nakonec běží – nedá se vynechat
- přesně odráží stav projektu
- zabere 30 – 80 % času projektu

Jak to souvisí s vámi?

Programování ve škole

- složité problémy v jednoduchém kontextu

- důraz většinou kladen na vědecký aspekt práce
- řemeslný aspekt programování (většinou) opomíjen

Programování ve firmě

- jednoduché problémy ve velmi složitém kontextu
- složité problémy ve velmi složitém kontextu
- důraz kladen na funkčnost a včasné dodání

Důsledkem jsou

- problémy při realizaci složitějších projektů
  - náročnější semestrální práce, diplomové práce, SW projekt
- potenciální ekonomické ztráty

Poznámky:

- Umělecký aspekt si musí najít každý sám.
- Problémy např. na OS – semestrálky přes 5KLOC, neznámé prostředí, velký rozsah abstrakcí.
- Jedním z důsledků je také vznik tohoto (a jiných) předmětů.
- Za výstižný postřeh ohledně rozdílu mezi programováním ve škole a v průmyslu vděčím T. Pochovi.

Jak má tento předmět situaci zlepšit?

Náplň předmětu

- Ukázat praktické programátorské techniky, které vedou k psaní přehlednějšího, kvalitnějšího a lépe udržovatelného kódu.
- Vysvětlit proč je používat a na příkladech ilustrovat jejich správné použití.

Cíle předmětu

- Schopnost navrhovat rozumné abstrakce v poslední (detailní) fázi návrhu software.
- Schopnost rozpoznat nekvalitní kód a přeměnit jej v kvalitní.
- Vzbudit vnitřní potřebu psát kvalitní kód, ať už proto, že je to správný způsob jak programovat, nebo proto, že se to vždy nějakým způsobem vrátí.

Tématická osnova

Úvod

- proč psát kvalitní kód, návrh software,
- inherentní a zavlečená složitost.

Návrh tříd

- návrh rozhraní (API), dědičnost vs. kompozice,
- coupling a decoupling, modularizace a vrstvy abstrakce.

Návrh metod

- pseudokód, lokalita vs. duplicita kódu,
- ošetření chyb, defenzivní programování.

Tématická osnova

Dokumentace

- zdrojový kód, komentáře,
- popis rozhraní, high-level dokumentace.

Psaní kódu

- proměnné, konstanty, názvové konvence, datové typy,
- příkazy, řídící struktury, kód řízený daty.

Zlepšování kódu

- testování, refaktoring, ladění.

Pro koho je předmět určen?

Ideálně

- 3. ročník Bc. studia
- 1. ročník NMgr. studia

Méně ideálně

- 1. ročník Bc. studia
- 2. a vyšší ročník NMgr. studia

Poznámky:

- Ideální cílovou skupinou jsou studenti, kteří už mají nějaké programování za sebou, ale zatím např. neprogramovali v týmu nebo nemuseli udržovat nějaký software funkční.
- Méně ideální je předmět pro studenty z nižších ročníků, kteří si sice předmět mohou zapsat, ale motivace může být těžko pochopitelná, protože si ještě neprošli svým programátorským peklem.
- Méně ideální je předmět rovněž pro studenty z vyšších ročníků, kterým (v závislosti na zkušenostech) mohou věci zde předkládané přijít samozřejmé (i když pravděpodobně ne všechny).

Organizace předmětu

Formát 2/2, KZ

- důraz na práci v semestru
- cvičení dle harmonogramu na webu (cca 12 za semestr)

Hodnocení

- >= 87% bodů ... 1
- >= 73% bodů ... 2
- >= 60% bodů ... 3
- jinak 4

Organizace předmětu

Cvičení – Lubomír Bulej (Yours Truly)

- zadání domácích úkolů, prezentace týmů
- v případě potřeby doplňkový materiál

Web předmětu

http://d3s.mff.cuni.cz/teaching/nprg043

- obecné informace, slajdy z přednášek
- harmonogram cvičení, zadání úkolů a termíny

GitLab issue tracker

https://gitlab.mff.cuni.cz/teaching/nprg043/2024-summer/forum

- oznámení, dotazy, tvorba teamů

Programovací jazyky

Přednášky: primárně Java

- jednoduchý, v praxi rozšířený jazyk s C-like syntaxí
- (většinou) umožňuje zabývat se podstatou problému

Nicméně...

- znalost jiných jazyků rozšiřuje obzory
- použití správného nástroje na daný problém
- dynamické jazyky: přímočaré řešení/velmi rychlé prototypování

Úkoly: C++, C#, Java (Scala)

- týmové rozhodnutí, neškodí vyzkoušet jiný jazyk

Literatura

McConnel, S. "Code Complete", 2nd Edition, Microsoft Press, 2004

- vyčerpávající, dobrá argumentace, nutno více průchodů
- McConnell, S. "Dokonalý kód – Umění programování a techniky tvorby software", Computer Press, 2006

Bloch, J. "Effective Java", 2nd Edition, Addison-Wesley, 2008

- nejen o Javě – věnuje se i zásadám návrhu
- Bloch, J. "Java Efektivně. 57 rad softwarového experta"

Freeman, E., Robson, E., Sierra, K., and Bates, B. "Head First Design Patterns", 2nd edition, O'Reilly, 2014

- návrhové principy a jejich aplikace v návrhových vzorech

+ odkazy roztroušené po slajdech

Dotazy?

Motivační příklad...

Jak vypadá špatný kód?

"Routine from hell"

```
void HandleStuff( CORP_DATA & inputRec, int crntQtr,
   EMP_DATA empRec, double & estimRevenue, double ytdRevenue,
   int screenX, int screenY, COLOR_TYPE & newColor,
   COLOR_TYPE & prevColor, StatusType & status, int expenseType )
{

int i;
for ( i = 0; i < 100; i++ ) {
   inputRec.revenue[i] = 0;
   inputRec.expense[i] = corpExpense[ crntQtr ][ i ];
   }
UpdateCorpDatabase( empRec );
estimRevenue = ytdRevenue * 4.0 / (double) crntQtr;
newColor = prevColor;
status = SUCCESS;

if ( expenseType == 1 ) {
     for ( i = 0; i < 12; i++ )
           profit[i] = revenue[i] - expense.type1[i];
     }
else if ( expenseType == 2 ) {
          profit[i] = revenue[i] - expense.type2[i];
          }

else if ( expenseType == 3 )
```

```
profit[i] = revenue[i] - expense.type3[i];
}
```

"Routine from hell"

Jak na vás příklad působí?

- nutkání konat násilné činy (na autorovi)…
- kód nic moc, ale to je život, s tím se nedá nic dělat…
- normálka, nechápu proč je kolem toho takový povyk …

Jak špatný ten kód byl?

- dalo by se na něm najít alespoň 15 vad
- od formátování, přes názvy proměných a číselné konstanty, množství a uspořádání parametrů, až po fakt, že dělá řadu věcí, které spolu nesouvisí

Proč vlastně psát kvalitní kód?

Obecně se to vyplatí

- Kód je jednou napsán, mnohokrát čten a upravován
  - "ztráta" času při psaní se vrátí nejpozději když nastanou problémy
  - problémy dříve nebo později nastanou

Technologický dluh

- Hack je půjčka (času), jednou ji musíme vrátit (i s úroky)
  - je lepší žít bez dluhů, pokud to jde
  - příliš velké zadlužení může vést k "bankrotu"

Poznámky:

- Kvalitní reimplementace bude trvat déle, protože problém a jeho řešení nebude z kódu dobře patrné, detaily jsme zapomněli, celé je to potřeba znovu otestovat…
- Pokud to jde, je samozřejmě lepší žít bez dluhů, ale někdy to nejde, např. když se blíží release. Poté je však potřeba začít dluh co nejrychleji splácet, aby "úroky" nenarostly do neúnosné výše.
- Se spoustou dluhů můžeme nakonec "zbankrotovat", to je když se z kódu stane neudržovatelný žmolek.
- Steve McConnell: 10× Software Development – Technical Debt
- Steve McConnell: 10× Software Development – Technical Debt Decision Making
- Development Short-Cuts Are Not Free: Understanding Technology Debt

Proč také psát kvalitní kód?

Individuální motivace

- snaha odvést co nejlepší práci, z principu

- vědomí, že s tím, co napíšu, budu muset žít

Tlak okolí

- vedení (pokud rozumí programování)
- kolegové (pokud jsou motivováni sami)
- může být rovněž vyvíjen opačným směrem

Poznámky:

- Zkušenost: u jednorázových prací je na místě důsledná kontrola.

Ale proč hlavně psát kvalitní kód?

Zjednodušuje návrh

- kvalitní kód umožňuje zvládat složitost
- návrh sám o sobě je dosti složitý problém

Patří návrh do programování?

- návrh někdy dělá přímo ten, kdo píše kód
- návrh může být pseudokód třídy
- návrh může být pár diagramů ilustrujících vztahy mezi třídami
- detailní návrh tříd a funkcí jde často za hranici návrhu architektury

V čem je samotný návrh tak složitý?

Při návrhu software je nutno čelit mnoha výzvám

- návrh je zapeklitý problém (wicked)
  - k pochopení problému je nutné ho nejprve vyřešit
  - dobré řešení se od špatného příliš neliší
- návrh je "špinavý" proces
- návrh je o kompromisech a prioritách
- návrh vytváří, ale hlavně omezuje možnosti
- návrh je nedeterministický
- návrh je výsledkem procesu
  - dobrý návrh se neobjeví jen tak, zničeho nic

Co je při vývoji software nejdůležitější?

Zvládnutí složitosti!

Inherentní vs. zavlečená složitost

Inherentní složitost

- vlastnosti, které nějaká věc musí mít, aby byla tou věcí
- vychází z podstaty daného problému, nelze odstranit

Zavlečená složitost

- vlastnosti, které nějaká věc zrovna má, ale nedefinují ji
- určena konkrétním řešením problému, ne jeho podstatou
- dlouhodobě se snažíme o její eliminaci
  - reprezentace data, přepojování pointerů, cykly,
  - explicitní alokace/dealokace paměti, reference counting,
  - problémy s paralelizací, odlišnosti mezi OS,
  - ...

Proč je zvládnutí složitosti tak důležité?

Velký rozsah úrovně abstrakcí

- "od bitů po megabajty"

Omezené lidské možnosti

- mozek dokáže najednou "udržet" pouze malý počet konceptů
- je jednodušší "uchopit" více jednoduchých konceptů než jeden složitý

Jak udržet složitost pod kontrolou?

Vyhnout se neefektivnímu návrhu v důsledku

- složitého řešení jednoduchého problému
- jednoduchého, ale nesprávného řešení složitého problému
- složitého, ale nevhodného řešení složitého problému

Počítat s tím, že inherentní složitosti se nelze vyhnout

- Minimalizovat množství inherentní složitosti, kterou je nutné se zabývat v libovolném okamžiku
- Globálně omezovat šíření složitosti zavlečené
  - použitím expresivnějšího jazyka, pokročilejší technologie a frameworků, odsunutím složitosti do knihoven

Poznámky:

- Použití expresivnějšího jazyka: Ruby, Python, ...
- Použití pokročilejší technologie: smart pointers, garbage collection
- Použití knihoven/frameworků: generování unikátního ID, ...

Příklad: omezení zavlečené složitosti

Použití indexových proměnných…

```
float frubbish = 0.0;

for (int i = 0; i < foo.length; i++) {
  for (int j = 0; j < bar.length; j++) {
    for (int k = 0; k < zap.length; k++) {
      frubbish += frubbishDelta (foo[i], bar[j], zap[k]);
    }
  }
}
```

… nahrazeno iterací s podporou v jazyce

```
float frubbish = 0.0;

for (float eachFoo : foo) {
  for (float eachBar : bar) {
    for (float eachZap : zap) {
      frubbish += frubbishDelta (eachFoo, eachBar, eachZap);
    }
  }
}
```

Příklad: omezení zavlečené složitosti

Transformace dat v cyklu…

```
List <FieldInsnNode> result = new ArrayList <> ();
for (AbstractInsnNode insn : Insns.asList (insns)) {
    if (!AsmHelper.isStaticFieldAccess (insn)) {
        continue;
    }

    FieldInsnNode fieldInsn = (FieldInsnNode) insn;
    String fieldName = ThreadLocalVar.fqFieldName (insn.owner, insn.name);
    if (!  tlvIds.contains (fieldName)) {
        result.add (fieldInsn);
    }
}
```

Příklad: omezení zavlečené složitosti

Transformace dat pomocí objektových streamů…

```
List <FieldInsnNode> fieldInsns = Insns.asList (insns)
    .stream ().unordered ()
    .filter (AsmHelper::isStaticFieldAccess)
    .map (insn -> (FieldInsnNode) insn)
    .filter (insn -> {
        String fieldName = ThreadLocalVar.fqFieldNameFor (insn.owner, insn.name);
        return tlvIds.contains (fieldName);
    })
    .collect (Collectors.toList ());
```

Vlastnosti dobrého návrhu software

- Minimální možná složitost
- Snadná údržba
- Minimální propojenost částí (minimal coupling)
- Rozšiřitelnost
- Znovupoužitelnost
- Portabilita
- Žádné zbytečnosti

- Stratifikace
- Použití standardních technik/nástrojů

Poznámky:
Jak už to tak bývá, svět není černobílý a dobré vlastnosti jdou často jedna proti druhé. V celé řadě oblastí je nutné dělat ústupky, to je v pořádku, jen je nutné ty ústupky dělat vědomě.

Five (+1) worlds

Při vývoji je nutné zohlednit typ vyvíjeného software

Základní typy software

1. Krabicový
2. Interní
3. Embedded
4. Hry
5. Na jedno použití

6. Software jako služba

Poznámky:
- Five Worlds
- Výčet typů nejspíš není přesný ani úplný

Krabicový software

Prostředí, kde běží, není pod kontrolou

- Nutno hodně testovat
- Nelze spoléhat na předinstalované komponenty (knihovny)
- Nutnost rychlého běhu (nelze upgradovat HW)

- Konkurence: důraz na snadné používání, vzhled
- Manažerský pohled: scalable, na vývoj lze vynaložit poměrně velké množství prostředků

Interní software

Prostředí, kde běží, je pod kontrolou

- Stačí testovat na několika málo konfiguracích
- Lze spoléhat na předinstalované komponenty (knihovny)
- Lze optimalizovat upgradem HW

- Není tlak na snadné používání, vzhled (uživatelé typicky nemají na výběr)
- Důraz na rychlost vývoje (cyklus i v řádu týdnů a dnů)

- Manažerský pohled: non-scalable, na vývoj nelze přidělit tolik prostředků

Embedded software

Omezené zdroje a možnost aktualizace

- Důležitá stabilita a bezchybnost
- V některých případech efektivita kódu důležitější než jeho čistota

Poznámky:
Opravdu je update tak obtížný? Software solutions to hardware problems

Hry

Maximální využití zdrojů, omezená životnost

- Efektivita kódu často důležitější než jeho čistota
- Typicky jen jedna verze – důležitá stabilita a bezchybnost

Software na jedno použití

Skripty, konverze mezi datovými formáty, apod.

- Čistota kódu nehraje téměř žádnou roli
- Efektivita nehraje téměř žádnou roli
- Pozor, aby se z takového software nestal produkční kód!

Software jako služba

Aplikace (typicky webové) hostované na serverech

- Možná nejpoužívanější platforma současnosti a téměř určitě budoucnosti
  - **Než začnete psát jakoukoliv aplikaci, položte si otázku, zda nebude lepší ji napsat jako aplikaci webovou**
- Samozřejmě se nejedná o platformu univerzální:
  - Odezva GUI (hry, práce s grafikou)
  - Velká množství dat (grafické programy)
  - Bezpečnost (např. účetní program)
- Svoboda volby OS, platformy, jazyka

Poznámky:

- Starost o servery (24/7)
- Optimalizace typicky motivována:
  - Snížením ceny za HW (servery)
  - Snížením ceny za bandwidth
  - Zrychlením odezvy

Software jako služba

Proč psát software jako službu?

- Extrémně snadný update všech uživatelů najednou
  - Ze zvyšování kvality mají užitek všichni uživatelé
  - Žádná starost o zpětnou kompatibilitu, staré formáty dat, apod.
  - Jednoznačná preference inkrementálního vývoje
  - Možnost okamžitě opravit chyby, které se navíc snadno reprodukují

    "Often I could fix the code and release a fix right away. And when I say right away, I mean while the user was still on the phone." – Paul Graham

- Snadné monitorování chování uživatelů

Software jako služba

- U větších aplikací: "City of code"
  - Spousta programů, co spolu spolupracují
  - Typicky v různých jazycích
  - Vlastní/cizí

"As well as buildings you need roads, street signs, utilities, police and fire departments, and plans for both growth and various kinds of disasters." –Paul Graham

Poznámky:

- Paul Graham: The Other Road Ahead
- Steve Yegge: It's Not Software