

Praktické aspekty vývoje software

Jaroslav Dytrych

Fakulta informačních technologií Vysokého učení technického v Brně
Božetěchova 1/2. 612 66 Brno - Královo Pole
dytrych@fit.vutbr.cz



2. února 2017

- Úvod do předmětu
- Praktická pravidla pro psaní udržitelného kódu
- RefaktORIZACE
- Efektivní využití IDE (Integrated Development Environment)

- V rámci předmětů bakalářského programu budete zpracovávat řadu samostatných i týmových projektů.
- V praxi se můžete stát členy malého týmu či pracovat zcela samostatně (např. tvorba webů)
- Lze vystačit i bez znalostí IVS (v r. 2011 první běh předmětu)
- Ale nástroje a postupy prezentované v IVS Vám mohou pomoci zefektivnit týmovou spolupráci, usnadnit Vám jednotlivé kroky při vývoji SW (od analýzy požadavků až po doručení zákazníkovi) a podpořit vytváření systémů udržitelných po delší době, nikoliv jen v měsíci, kdy vznikly.

- Jinou možností je, že se v rámci nového zaměstnání setkáte s vágně pojatým procesem vývoje software, volnými či neexistujícími normami a změtí nekompatibilních nástrojů.
- Předmět IVS by vás měl naučit, které nástroje a postupy se vyplatí zavést a které nikoliv.
- Je také možné, že se v praxi setkáte s dobře definovaným procesem vývoje, normami a sdílenou sadou nástrojů. V takovém prostředí musíte být schopní si rychle osvojit využití nástroje a danou koncepci vývoje.
- V IVS byste měli získat zkušenosti s typickými postupy, což by mělo pomoci ke snazšímu pochopení specifik nových nástrojů

- Jak se naučit programovat v konkrétním programovacím jazyce / používat vývojové nástroje specifické pro konkrétní jazyk (viz např. předmět ISJ).
- Teoretické základy, na nichž stojí různé nástroje (např. pro odhalování chyb) – na to jsou jiné specializované předměty.
- Vyčerpávající přehled dostupných nástrojů, s nimiž se lze v praxi setkat.
- Vyčerpávající popis možností určitého nástroje – ten lze dohledat v dokumentaci.

- Řízení projektu (MS Project, Trello, Jira, ...)
- Testování (GoogleTest, JUnit, GCOV, ...)
- Správa verzí (GIT, Mercurial, Bazaar, ...)
- Generování dokumentace (Doxygen, JUnit, phpDocumentor, ...)
- Sestavování částí (Make, CMake, QMake, ...)
- Odhalování chyb (debugger)
- Evidování chyb a námětů (Redmine, Trac, Bugzilla, ...)
- Profiling (GProf, cProfile, JProfiler, ...)
- Integrace (Hudson)
- Nasazení (Windows Installer, Debian Package Manager, ...)
- ...

- 1 Úvod, praktická pravidla pro psaní udržitelného kódu, refaktorizace a efektivní využití IDE
- 2 Testování software, TDD (Test-Driven Development) a jeho použití při vývoji v týmu
- 3 Týmová spolupráce, komunikace, sdílení dat v týmu, základy návrhu a plánování projektu
- 4 Systémy pro distribuovanou správu verzí, GIT
- 5 Typy dokumentace, generování programové dokumentace z kódu, identifikace existujících komponent a využívání knihoven dostupných na různých platformách
- 6 Sestavení programů, Make, Cmake a Qmake
- 7 Uživatelská rozhraní
- 8 Debugging, bugtracking a QA
- 9 Demo cvičení dle Vašich preferencí a půlsemestrální test
- 10 Nasazení programů
- 11 Urychlování výpočtů, možnosti paralelizace a profiling
- 12 Programovací jazyky a paradigmaty, SWIG a práce se starším kódem
- 13 Přednáška pozvaných expertů z firemní praxe

- Pavel Dedík

<https://www.linkedin.com/in/paveldedik>

- CTO at Mergado <http://www.mergado.cz/>
- Přejde na 4. přednášku 28. 2. 2017 a bude mít krátkou prezentaci o využití GITu v praxi. Společně s přednášejícím Michalem Wiglaszem následně předvede demo na využití GITu.

- Poslední přednáška 2. 5. 2017

- Téma bude upřesněno později
- Pravděpodobně Honza Javorek

<https://www.linkedin.com/in/honzajavorek>

<http://honzajavorek.cz/>

- Software Engineer & Python Activist
- Maskot české Python komunity
- Pracuje pro Apiary (Oracle) <https://apiary.io/>

- 1. projekt: Definice testů (18 bodů)
- 2. projekt Projekt zaměřený na spolupráci v týmu (52 bodů)
 - Veřejné obhajoby v 1. týdnu zkouškového období
 - Z obhajob je pořizován záznam a veřejně zpřístupněn
 - V závěru dostanete detailní zpětnou vazbu (ne jen čísla)
- Půlsemestrální písemný test (30 bodů)
- Úkoly k procvičení (bez bodů, dle možností zpětná vazba)

- Server ivs.fit.vutbr.cz je určen pro řešení projektů.
- Připojit se lze přes ssh z IP adres v doménách .cz a .sk (vyžaduje funkční reverzní DNS – ověřit lze např. na <https://remote.12dt.com/>)
- Login je stejný jako na ostatní servery FIT, heslo Vám bude zasláno e-mailem nejpozději začátkem 2. týdne semestru.
- Chovejte se na něm prosím slušně a neomezujte své kolegy jeho přetěžováním.
- Pokud se nemůžete připojit, zašlete mi svoji IP adresu e-mailem na dytrych@fit.vutbr.cz (posílejte prosím veřejnou a nikoliv lokální IP). Pokud se Vám adresa často mění, pošlete název a rozsah svého poskytovatele připojení (můžete hledat např. na <http://ipindex.dihe.de/>).

Praktická pravidla pro psaní udržitelného kódu

- Unix vznikl v roce 1969 a od té doby je stále v provozu. Počítače se systémem Unix mají za sebou již více než stovky tisíc provozních hodin.
- Programátoři v systému Unix mají za sebou desítky let zkušeností. To, co museli v historii řešit, my dnes pokládáme za samozřejmé.
- Porozumění základním konceptům je užitečné, abychom "znovu neobjevovali kolo" a byli schopni vytvářet SW, který bude rovněž moci přežít desítky let.

- Navrhujte programy odspodu nahoru a snažte se, aby byly co nejjednodušší.
- Vytvářejte jednoduché nástroje, které vzájemnou spoluprací dokáží plnit složité úkoly (program by měl dělat jednu věc a to opravdu dobře).
- Počítejte s tím, že výstup každého programu může být využit jako vstup jiného programu (i u GUI – např. kopírovat do schránky).
- Předpokládejte určitou inteligenci uživatele (vyhněte se zbytečným dotazům, dialogům apod.).
- Při funkcionálním programování pište malé funkce pro řešení specifických problémů.
- Při objektově orientovaném programování by objekt měl zastřešovat právě jednu entitu a to úplně. Obdobně by jedna metoda měla provádět jednu činnost.
- Navrhujte SW tak, aby mohl být rychle prototypován a zkoušen (ideálně v řádu týdnů).
- ...

- Pravidlo srozumitelnosti (srozumitelnost je lepší než chytrost)

Správně

```
int Sum = 0;
for (int i = 1; i < N; ++i) {
    Sum = Sum ^ i;
}
```

Špatně

```
int Sum = (N & (N % 2 ? 0 : ~0) |
((N & 2)>>1) ^ (N & 1));
```

- Pravidlo kompozice (navrhujte programy tak, aby je bylo možné připojit k dalším programům)

Správně

```
int c;
while ((c = getc(stdin)) != EOF) {
    putchar(toupper(c));
}
```

Špatně

```
int c;
FILE *f;
f = fopen("input.txt", "r");
while ((c = getc(file)) != EOF) {
    putchar(toupper(c));
}
fclose(f);
```

- Pravidlo transparentnosti (navrhujte programy tak, aby bylo co nejsnazší jejich ladění a kontrola)

Správně

```
int maxCount = 50;
int sum = 0;
for (int i = 1; i < maxCount && buf[i] > 0; i++) {
    sum = sum + buf[i];
}
```

Špatně

```
int sum = 0;
for (int i = 1; i < 50 && buf[i] > 0; i++)
    sum = sum + buf[i];
```


- Pravidlo robustnosti (robustnost je dítětem transparentnosti a jednoduchosti)

Správně

```
char *line = NULL;
size_t len = 0;
printf("Enter a value :");
read = getline(&line, &len, stdin);
...
free(buff)
```

Špatně

```
char str[100];
printf("Enter a value :");
gets(str);
```

- Pravidlo reprezentace (zahalte znalosti do dat tak, aby logika programu mohla byt hloupá a robustní)

Správně

```
char* messages[7];
strings[0] = "you typed zero";
strings[1] = "n is a perfect square";
strings[2] = "n is an even number";
strings[3] = "n is a prime number";
strings[4] = "n is a perfect square";
...
printf("%s\n", strings[n]);
```

Špatně

```
if (n == 0) printf("you typed zero");
else if (n == 1 || n == 4 || n == 9)
    printf("n is a perfect square");
else if (n == 2 || n == 6 || n == 8)
    printf("n is an even number");
else if (n == 3 || n == 5 || n == 7)
    printf("n is a prime number");
```

- Pravidlo nejmenšího překvapení (při návrhu rozhraní se snažte, aby dělalo vždy co nejméně překvapující věci)

Správně

```
int multiply(int a, int b) {  
    return a * b;  
}
```

Špatně

```
int multiply(int a, int b) {  
    return a + b;  
}
```

- Pravidlo hospodárnosti (programátorův čas je drahý, šetřete svůj čas na úkor času počítače)

Program v C

```
int wordCount(char *line) {  
    int count = 0;  
    while(isspace(*line)) line++;  
    int len = strlen(line);  
    bool onWord = true;  
    for(int i = 0; i < len; i++) {  
        if(isspace(line[i])) {  
            if(onWord) {  
                count++;  
            }  
            onWord = false;  
        }  
    }  
    // ... ještě 10 řádků  
}
```

Program v pythonu

```
def wordCount(line):  
    words = line.strip().split()  
    return len(words)
```

- Pravidlo optimalizace (nejprve vytvořte prototyp, zprovozněte a pak se teprve pusťte do jeho optimalizace)

Správně

```
float f[100], sum;
...
sum = 0;
for(i=0; i<100; i++) {
    sum += f[i]; }
```

Špatně

```
float f[100], sum0, sum1, sum2, sum3, sum;
...
sum0 = sum1 = sum2 = sum3 = 0;
for(i=0; i<100; i+=4) {
    sum0 += f[i];
    sum1 += f[i+1];
    sum2 += f[i+2];
    sum3 += f[i+3];
}
sum = (sum0 + sum1) + (sum2 + sum3);
```

- Pravidlo rozmanitosti
 - Nedůvěřujte všem tvrzením typu „existuje jediný správný způsob“.
- Pravidlo rozšiřitelnosti
 - Navrhujte pro budoucnost, protože ta nastane dříve, než si myslíte.
- ...

- Máte-li dobrý důvod, klidně kterékoliv pravidlo porušte, ale:
 - nejprve je třeba pochopit, proč dané pravidlo máme a jaké bude mít jeho nedodržení důsledky,
 - udržujte kód konzistentní a přehledný,
 - vždy myslíte na ty, co po Vás budou pokračovat v práci.
- Příkladem může být optimalizační “finta”, která upřednostní chytrost a rychlost před srozumitelností – pak je ale nutné dobře komentovat, případně uvést neoptimální kód jako komentář.

- Formátování musí být takové, aby byl kód co nejpřehlednější.
 - Kód bez komentářů je pro ostatní často nečitelný (např. regulární výrazy jsou tzv. "write-only").
 - Kódování znaků a konce řádků musí být jednotné.
 - Odsazování musí být důsledné.
 - Odsazování o hodně znaků je nepřehledné (je nutné skrolovat vertikálně i horizontálně). Časté jsou 2 mezery nebo 1 tabulátor.
 - I volné řádky mají svůj význam (za funkcí, mezi bloky, ...).
 - ...
- Je vhodné využívat obvyklé vzory a postupy, které jsou často podpořené v IDE.
- Formátování musí být v celém programu jednotné.
- Při vývoji v týmu je na začátku potřeba stanovit pravidla.
- S dodržováním jednotného nastavení v rámci týmů a projektů Vám může pomoci EditorConfig
<http://editorconfig.org/>

- Konfigurační soubor pro přizpůsobení nastavení editoru (.editorconfig).
- Některá IDE a editory jej přímo podporují, pro ostatní je potřeba zásuvný modul (Code::Blocks, NetBeans, Eclipse, ...).
- Nejdůležitější jsou kódování, konce řádků a odsazování.

Ukázka

```
root = true
[*]
charset = utf-8
end_of_line = lf
insert_final_newline = true
[Makefile]
indent_style = tab
[*.{c,cpp,h}]
indent_style = space
indent_size = 2
```

Použitelnost programů

- Pravidlo mlčenlivosti (nemá-li program co překvapivého říci, neměl by říkat nic)

Správně

```
$ ls empty_dir  
$
```

Špatně

```
$ ls empty_dir  
$ Vítejte v programu ls. Vámi požadovaný  
adresář je prázdný, chcete vypsát obsah  
jiného adresáře? [Y|N]: N  
$
```

- Pravidlo opravy (pokud se program musí zhroutit, ať to udělá hlasitě a co nejdříve)

Správně

```
$ wget www.google.com
Překládám www.google.com (www.google.com)...
nezdařilo se: Neznámé jméno nebo služba.
wget: adresu počítače www.google.com nelze přeložit
$
```

Špatně

```
$ wget www.google.com
Překládám www.google.com (www.google.com)...
nezdařilo se: Neznámé jméno nebo služba.
Zkouším znovu...
Překládám www.google.com (www.google.com)...
nezdařilo se: Neznámé jméno nebo služba.
Zkouším znovu...
...
```

- Nezavádějte zbytečně mnoho vyžadovaných pozičních parametrů.
- Nenabízejte konfigurační přepínače proto, co může být spolehlivým způsobem zjistitelné automaticky (např. autodetekce formátu vstupního souboru).
- Uživatelé by neměli mít přístup k optimalizačním přepínačům.
- Nesnažte se přidat přepínačem to, čehož by se dalo dosáhnout zapouzdřením programu do skriptu nebo jeho začleněním do zřetězení (např. přepínač pro stránkování výstupu).
- Zamyslete se nad použitím:

```
cp [OPTION]... SOURCE... DIRECTORY
```

```
cp [OPTION]... -t DIRECTORY SOURCE...
```

```
find /ivs/test/ -type f | head -5 | xargs cp -t /ivs/t2
```

- 1 předpona projektu
- 2 spojovník nebo podtržítka
- 3 číslo verze
- 4 tečka
- 5 src nebo bin (nepovinné)
- 6 tečka, spojovník nebo podtržítka
- 7 binární typ a možnosti (nepovinné)
- 8 přípona archivu nebo kompresního formátu

lcov-1.13-1.noarch.rpm

lcov-1.13-1.src.rpm

git_2.7.4.orig.tar.xz

git_2.7.4-0ubuntu1_i386.deb

mercurial-4.0.2.win-amd64-py2.7.msi

- Raymond, E.S.: *The art of unix programming*, ISBN-0131429019, Pearson Education, 2003
- Balzarotti, D.: *Software Development The UNIX Philosophy*, Eurecom – Sophia Antipolis, France
- Cantin, C.: *Basic Introduction to UNIX/linux*, 2010
- Lande, J.: *Unix Tutorial*, 2010

RefaktORIZACE

- Každá část programu musí mít jednu jednoznačnou a spolehlivou podobu.
- Opakování vede k nekonzistencím – kdykoliv spatříte duplicitní kód, je to známka nebezpečí.
- RefaktORIZACE je proces zabývající se vnitřním preuspořádáním kódu, aniž by se změnila jeho funkčnost. Zahrnuje především:
 - Odstraňování duplicit
 - Přejmenování částí kódu (pro sjednocení, zpřehlednění apod.)
 - Zpřehlednění (např. rozdělením na menší části)

- Zlepšit interní kvalitu aplikace
- Lepší škálovatelnost
- Vylepšení čitelnosti a udržitelnosti kódu
- Zjednodušení struktury kódu

- Zjednodušení podmínek cyklů a podmíněných výrazů
- Úpravy řetězců (kódování, překlad)
- Přesunutí opakujícího se kódu do nové funkce.
- Zkracování funkcí vytvářením nových
 - Kód dlouhé funkce přesuneme do více krátkých, které z dané funkce voláme.
 - Ve funkci na nejvyšší úrovni je dobře vidět posloupnost operací.
 - Funkce na nižší úrovni dělají právě jednu činnost – dojde-li k chybě, je jasnější, co se zrovna provádělo.
- Vhodná přejmenování funkcí a proměnných
- Vytknutí společných vlastností do třídy, ze které se bude dědit (tedy např. třídy Malíř a Scochař nebudou mít duplicitní vlastnosti a metody, ale vše společné bude vytknuto do třídy Umělec, ze které budou dědit).
- ...

- Nikdy neměňte funkcionalitu i strukturu současně.
- Refaktorizujte průběžně a často.
- Neponechávejte původní kód (nekopírujte ale přesouvejte).
- Po refaktorizaci vždy otestujte, zda nedošlo ke změně chování programu.
 - Před refaktorizací je vhodné napsat a spustit testy.

- <https://refactoring.com/>
- <https://sourcemaking.com/refactoring>
- <http://www.integralist.co.uk/posts/refactoring-techniques.html>

Efektivní využití IDE

- Uživatelská přívětivost
 - Zvýrazňování syntaxe
 - Doplnění kódu
 - Refactoring
 - Rychlejší navigace po projektu
- Funkčnost
 - Verzování kódu
 - Debugging
 - Automatická kompilace a linkování
 - Spuštění programu

- Automatické
 - Autoindentace
 - Automatická tvorba závorek
 - Zvýraznění odpovídajících si závorek
 - Doplnování jmen proměnných a metod objektů
 - Kontrola názvů proměnných
- Z nabídky
 - Vyhledání chyby
 - Pokročilé vyhledávání
 - Přepínání mezi definicí a deklarací
 - Přeložit & Spustit
 - ...

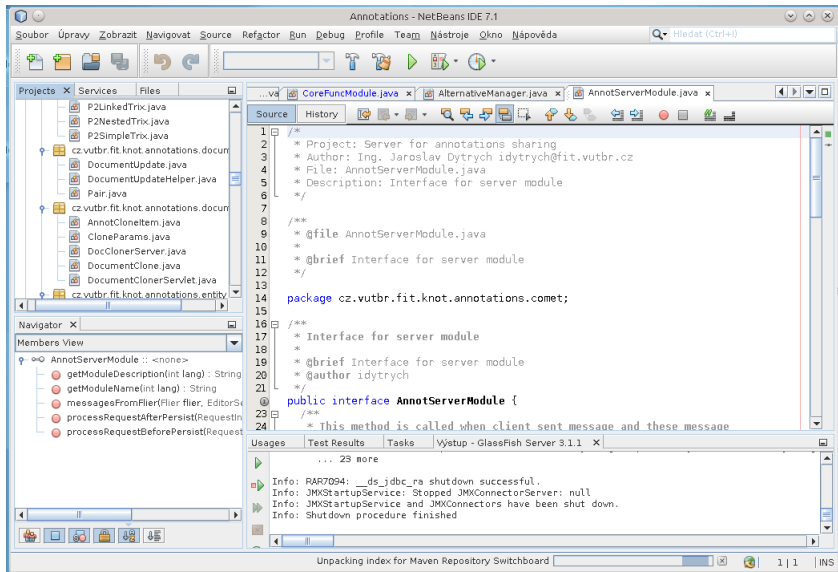
- Linux
 - Anjuta
 - KDevelop
- Windows
 - Visual Studio
- Mac
 - Xcode
- Multiplatformní
 - QtCreator
 - Code::Blocks
 - Bluefish
 - Eclipse
 - Netbeans

Grafická část

- Správce projektů
- GUI Designer
- Editor zdrojového kódu
- Zobrazení výstupu
- Vestavěná nápověda

Funkce

- Debugger
- Kompilátor
- Linker
- Interpret



- Přídavná funkčnost
 - Podpora dalších programovacích jazyků
 - Vývoj mobilních aplikací
 - Nástroje pro testování
 - Knihovny
 - Správa verzí
 - ...
- Obvykle se instalují ve správci doplňků

Děkuji za pozornost!