

pondělí 3. dubna 2017 od 18 hodin

záložní termín ve 20 hodin
(při kolizi s jinou výukou)

Ize získat až 30 bodů

6 otevřených otázek, každá za 6 bodů



Debugging, bugtracking a profiling

Michal Wiglasz

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2, 612 66 Brno - Královo Pole
iwiglasz@fit.vutbr.cz



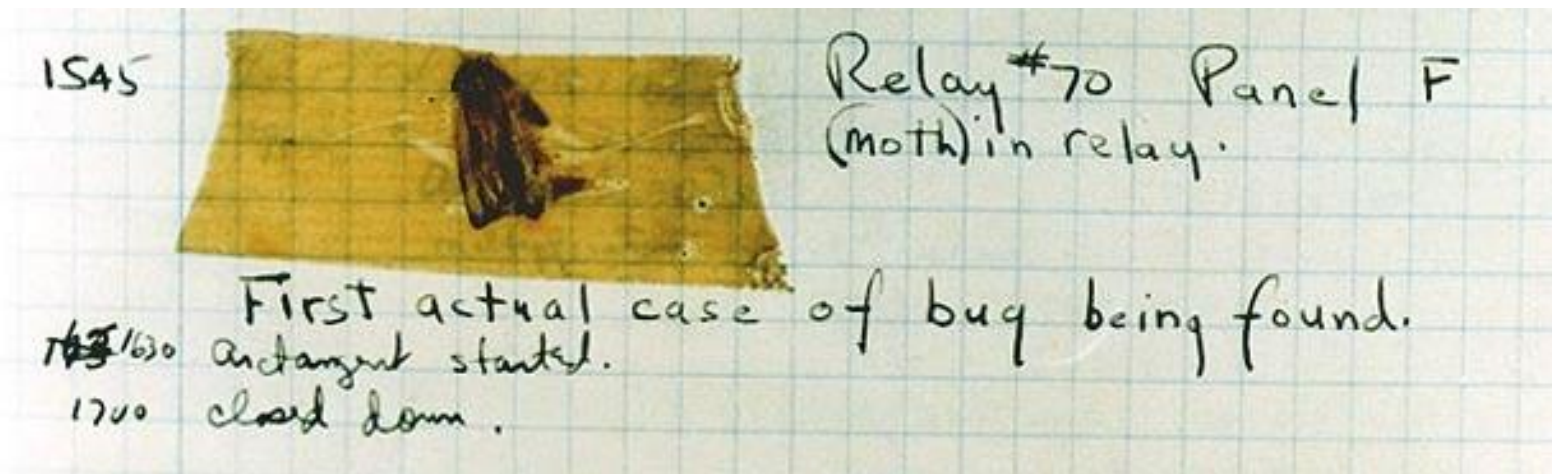
21. 3. 2017

Praktické aspekty vývoje software (IVS)

Původ slova **bug**?

Thomas Edison v dopise z roku 1878.

Historka o molu v relé počítače Mark II v roce 1947.



Mariner 1 (1962) – chyba při přepisu vzorce do kódu

Therac-25 (80. léta) – min. 5 mrtvých (údajně až 22)

- ozáření pacientů velkou dávkou radiace, pokud obsluha nastavovala přístroj příliš rychle

Patriot (1991) – 28 mrtvých, 98 zraněných

- nepřesnost systémových hodin
- po 100 hod. běhu 0,34 s → odchylka půl kilometru

USS Yorktown CG-48 (1997) – dělení nulou

Mars Climate Orbiter (1999) – zmatek v jednotkách

Y2K – a čeká nás problém roku **2038**

Boeing 787 Dreamliner – přetečení po 248 dnech

Chyby jsou nevyhnutelné.

- vytvořit bezchybný kód je téměř nemožné
- pro většinu aplikací není nutná absolutní bezchybnost
- ale jsou výjimky: zdravotnictví, vojenství, aerospace

Péče věnovaná testování by měla být úměrná náročnosti, důležitosti a nebezpečnosti zakázky.

Metodologie: extrémní programování (XP), programování řízené testy (TDD), lean development, Crystal, Adaptive Software Development, ...

Pád programu

- i laik pozná, že je něco špatně

Nekonečný cyklus či rekurze

- nemusí být jasné, jestli program něco dělá

Chyby ve výsledných hodnotách

- program zdánlivě funguje správně

Syntaktické chyby

- prohřešky proti gramatice jazyka
- program nelze přeložit
- u interpretovaných jazyků se odhalí až za běhu
- chytrý editor je označí už během psaní

Sémantické chyby

- program nedělá, co má
- obtížná autodetekce

Syntaktické:

- chybějící středník či závorka (v Pythonu odsazení)
- překlep v názvu proměnné, funkce, ...

Sémantické:

- chybný přístup do paměti
- dělení nulou
- chyba o jedničku
- přetečení
- nekonečný cyklus
- chyby v synchronizaci vláken

- Textové = překlepy
- Pády aplikace → debugger / valgrind
 - segmentation fault, buffer overflow...
- Úniky paměti (memory leak) → valgrind
- Problémy s výkonem → profiler
- Neočekávané chování
 - program dělá něco jiného, než by měl
 - chyba v programu nebo dokumentaci?
- Bezpečnostní problémy
 - např. neošetřené vstupy (SQL injection, CSRF, XSS, ...)

DEBUGGING

Postup:

1. Nalezení chyby (vývojář, tester, uživatel)
2. Reprodukce chyby
3. Zjednodušení problému na to důležité
4. Vytipování možných příčin
5. Zaměření se na pravděpodobné příčiny
6. Identifikace příčin
7. Oprava chyby
8. Testování (nejen opravy chyby)
9. Změna dokumentace

Umožňuje:

- krokování programu
- sledování hodnot proměnných (registrů CPU, RAM...)
- změna hodnot proměnných
- zastavení programu na určeném místě (breakpoint)
 - nepodmíněně
 - podmíněně
 - při změně hodnoty proměnné (watchpoint)

Při překladu je nutno přidat ladící informace

- jména proměnných, vazba instrukce ↔ řádek kódu...
- u gcc parametr **-g**

GDB – The GNU Project Debugger

- Konzolové rozhraní, standard pro UNIX-like systémy
- Ada, C, C++, Objective-C, Pascal...

DDD – Data Display Debugger

- grafická nadstavba nad konzolovými debuggery (GDB, DBX, WDB, Ladebug...)
- umí zobrazit grafy na základě dat programu

Prakticky každé IDE obsahuje debugger

- třeba i jako nadstavba nad GDB

- Textové rozhraní, velké množství funkcí
- HW i SW debugování
- Podporuje řadu procesorů a jazyků
- Vzdálené ladění (po síti, rs232) - embedded systémy
- Simulátor různých procesorů (bez periférií)
- Breakpointy, watchpointy, krokování (i zpět)
- Podporuje programy s vlákny
- Existují nadstavby (DDD, Eclipse)

<http://sourceware.org/gdb/current/onlinedocs/gdb/>

<https://www.root.cz/clanky/trasovani-a-ladeni-nativnich-aplikaci-v-linuxu-pouziti-gdb-a-jeho-nadstaveb/>

```
$ gcc ./segfault.c -o segfault -g
```

```
$ gdb ./segfault
```

```
GNU gdb (GDB) 7.12
```

```
...
```

```
Reading symbols from ./segfault...done.
```

```
(gdb)
```

```
$ gcc ./segfault.c -o segfault -g
```

```
$ gdb ./segfault
```

```
GNU gdb (GDB) 7.12
```

```
...
```

```
Reading symbols from ./segfault...done.
```

```
(gdb) run
```

```
Starting program: ~/segfault
```

```
Program received signal SIGSEGV, Segmentation fault.
```

```
0x00007ffff7aca301 in __strlen_sse2 () from /lib64/libc.so.6
```

```
(gdb)
```



```
$ gcc ./segfault.c -o segfault -g
$ gdb ./segfault

GNU gdb (GDB) 7.12
...
Reading symbols from ./segfault...done.
(gdb) run
Starting program: ~/segfault
Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7aca301 in __strlen_sse2 () from /lib64/libc.so.6
(gdb) backtrace
#0  0x00007ffff7aca301 in __strlen_sse2 () from
/lib64/libc.so.6
#1  0x00007ffff7ab199b in puts () from /lib64/libc.so.6
#2  0x0000000000400544 in main () at ./segfault.c:6
```

Základní příkazy GDB:

- **help [command]**
- **run** – spuštění programu
- **backtrace** – výpis zásobníku (call stack)
- **step** – postup o jeden krok (step into)
- **next** – krok, ale nevstoupí do cyklů a funkcí (step over)
- **break funkce** – breakpoint při zavolání funkce
- **break main.c:6** – breakpoint na konkrétním řádku
- **break main.c:6 if i>=10** – podmíněný breakpoint
- **watch myvar** – zastavení při změně hodnoty **myvar**
- **print myvar** – výpis hodnoty **myvar**
- **quit**

Funguje doplňování tabulátorem, šipky a zkratky (bt = backtrace)

- grafická nadstavba nad různými debuggery
- vývoj zamrzl (poslední verze v roce 2009)
- zobrazení datových struktur

DDD: /home/tester/xxx/factorial.c

```

File Edit View Program Commands Status Source Data Help
(): rsp
17 {
18     long n1 = 0;
19     long n2 = 10;
20     long n;
21     for (n=n1; n<n2; n++)
22     {
23         printf("1!d = %ld\n", n, factorial(n));
24     }
25
26     return 0;

```

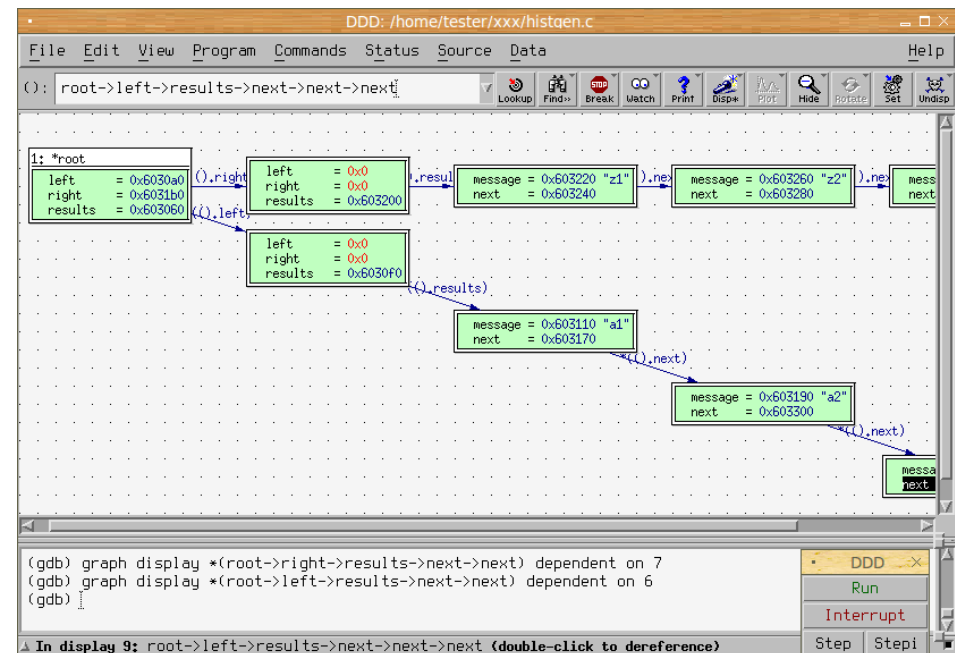
```

0x0000000040056e <+16>: movq  $0xa, -0x8(%rbp)
0x00000000400576 <+24>: mov  -0x10(%rbp), %rax
0x0000000040057a <+28>: mov  %rax, -0x18(%rbp)
0x0000000040057e <+32>: jmp  0x4005aa <main+76>
0x00000000400580 <+34>: mov  -0x10(%rbp), %rax
0x00000000400584 <+38>: mov  %rax, %rdi
0x00000000400587 <+41>: callq 0x40052d <factorial>
0x0000000040058c <+46>: mov  %rax, %rdx
0x0000000040058f <+49>: mov  -0x18(%rbp), %rax
0x00000000400593 <+53>: mov  %rax, %rsi
0x00000000400596 <+56>: mov  $0x400644, %edi
0x0000000040059b <+61>: mov  $0x0, %eax
0x000000004005a0 <+66>: callq 0x400410 <printf@plt>
0x000000004005a5 <+71>: addq  $0x1, -0x18(%rbp)
0x000000004005aa <+76>: mov  -0x18(%rbp), %rax
0x000000004005ae <+80>: cmp  -0x8(%rbp), %rax
0x000000004005b2 <+84>: jle  0x400580 <main+34>
0x000000004005b4 <+86>: mov  $0x0, %eax

```

(gdb) break 1d
Function "1d" not defined.
(gdb) break factorial.c:26
Breakpoint 2 at 0x4005b4: file factorial.c, line 26.
(gdb) |

Function "1d" not defined.



<https://www.gnu.org/software/ddd/manual/>

<https://mojefedora.cz/debugery-a-jejich-nadstavby-v-linuxu-2-cast/>

- Někdy to jinak nejde (např. embedded systémy)
- Výpisy přes printf
 - výpis „jsem zde“
 - výpis hodnoty proměnných
 - lze použít makra `__FILE__`, `__LINE__` aj.
- Logování
 - chybu lze najít i zpětně („když se nikdo nedíval“)
 - log lze odeslat technické podpoře
 - ale logování může ovlivnit chování programu
 - např. pravděpodobnost vzniku race condition
- Assert
 - ověření platnosti podmínky

- Hledá problémy v kódu bez jeho spuštění
- Může detekovat více chyb než překladač
- Také jako plugin do editoru či IDE

- Typová kontrola, neinicializovaná data
- Kontrola indexování polí,
- Přenositelnost konstrukcí
- Pravidla pro časté chyby
 - `if (confirmation = "yes") format_hdd();`
- Nedodržení stylu formátování
- Vlastní pravidla

Pro C: Lint, cppcheck, mygcc, codan...

Je třeba hledání chyb co nejvíce usnadnit:

- jasné názvy proměnných
- dodržování konvence pojmenování
- přehledné a konzistentní formátování
- dostatečně komentovaný kód
- seskupování kódu do skupin
- neduplikovat kód (Don't repeat yourself = DRY)
- nemít příliš hluboké zanoření (if, for, while)
- dodržovat best practices pro daný jazyk
 - Google Style Guide, PEP-8 pro Python, PSR-2 pro PHP...
- mít zapnuté varování překladače

-g

- vytváří ladící informace pro debugger

-ggdb

- rozšíření pro GDB

-Wall

- zapnutí všech varování

-Wextra

- zapnutí dalších varování

-pedantic

- striktně vyžaduje doržování normy (-std=xxx, -ansi)

<https://gcc.gnu.org/onlinedocs/gcc/Debugging-Options.html>

<https://gcc.gnu.org/onlinedocs/gcc/Warning-Options.html>

- Chyby mohou mít i exotické příčiny
 - chyba v překladači
- Chyba se nemusí projevit během ladění
 - race condition nevznikne, protože kód běží pomaleji
- Pozor na optimalizace
- Důležité je opravovat příčinu a ne následek

Nejlepší je chybám předcházet.

VALGRIND

- Valgrind = Posvátná brána do Valhally
 - Palác boha Ódina, který sem svolává padlé bojovníky, aby zde trénovali na poslední bitvu Ragnarök
- Původní název byl Heimdall
 - Strážce nordických bohů, který vidí stovky mil daleko ve dne i v noci, slyší růst trávu i vlnu na hřbetech ovcí
 - ... ale již existoval balíček s tímto názvem
- Nástroje pro ladění a profilování programů

- Pouze pro unixové systémy
- V podstatě virtuální stroj
 - analyzovaný program je oddělený od procesoru
 - podstatně pomalější běh programu (4-5krát)
- Umožňuje připojení GDB k běžícímu programu

- Memcheck – správa paměti, nejpoužívanější
- Callgrind – profilování
- Helgrind – detekce race conditions
- Cachegrind – profilování cache CPU

Existují i externě vyvíjené nástroje.

Memcheck detekuje:

- použití neinicializované paměti
- čtení/zápis do uvolněné paměti
- čtení/zápis mimo alokovaný blok
- čtení/zápis nad vrchol zásobníku
- úniky paměti (memory leaks)
- neshody v použití malloc / new vs. free /delete
- špatné použití POSIX knihovny pthreads
- překrytí ukazatelů v memcpy

Čtení z neplatné adresy:

```
Invalid read of size 4
  at 0x40F6BBCC: (within /usr/lib/libpng.so.2.1.0.9)
  by 0x40F6B804: (within /usr/lib/libpng.so.2.1.0.9)
  by 0x40B07FF4: read_png_image(QImageIO *)
                                     (kernel/qpngio.cpp:326)
  by 0x40AC751B: QImageIO::read() (kernel/qimage.cpp:3621)
Address 0xBFFFFFF0E0 is is 0 bytes after a block of size 40 alloc'd
```

Zároveň se snaží zjistit, kde se neplatná adresa vzala:

- již uvolněná paměť – hlásí, kde se volalo free
- adresa těsně za alokovaným blokem (chyba o jedničku?)

Použití neinicializované paměti:

```
Conditional jump or move depends on uninitialised value(s)  
  at 0x402DFA94: _IO_vfprintf (_itoa.h:49)  
  by 0x402E8476: _IO_printf (printf.c:36)  
  by 0x8048472: main (tests/manuel1.c:8)
```

Nehlásí kopírování neinicializovaných dat, ale až jejich použití, které může mít vliv na funkci programu

Úniky paměti:

LEAK SUMMARY:

definitely lost: 48 bytes in 3 blocks.

indirectly lost: 32 bytes in 2 blocks.

possibly lost: 96 bytes in 6 blocks.

still reachable: 64 bytes in 4 blocks.

suppressed: 0 bytes in 0 blocks.

S parametrem **--leak-check=full** je výpis podrobnější:

```
8 bytes in 1 blocks are definitely lost in loss record 1 of 14
  at 0x.....: malloc (vg_replace_malloc.c:...)
  by 0x.....: mk (leak-tree.c:11)
  by 0x.....: main (leak-tree.c:39)
```

```
88 (8 direct, 80 indirect) bytes in 1 blocks are definitely
lost in loss record 13 of 14
  at 0x.....: malloc (vg_replace_malloc.c:...)
  by 0x.....: mk (leak-tree.c:11)
  by 0x.....: main (leak-tree.c:25)
```


- Je dobré chyby opravovat shora dolů
 - Chyby níže ve výpisu mohou být jen důsledkem předchozích
- Některé chyby jsou v systémových knihovnách
 - Není třeba se jimi trápit
 - Valgrind je umí skrýt
- Memcheck není stoprocentní!
 - Například chyba o jedničku v poli alokovaném na zásobníku může přepsat jinou lokální proměnou
 - Ale z pohledu Valgrindu jde o korektní přístup do paměti – neví, že program pracuje s polem

BUG TRACKING

Také *Issue Tracking, Ticket System...*

- Aby mohl vývojář vyřešit problém, musí o něm vědět
- Více kódu → více chyb → potřeba lepší evidence
- Hlášení chyb a požadavků
- Přiřazování vývojářům či testerům
- Sledování životního cyklu chyb
- Sledování závislosti chyb
- Lze provázat s verzovacím systémem

Bugzilla, Trac, Redmine, Mantis, JIRA, Team Foundation Server, GitHub, GitLab...

Podle závažnosti:

- blokuující
- kritický
- významný
- normální
- méně důležitý
- triviální (drobnosti, chyby v překladu...)
- rozšíření (požadavek na vylepšení)

Požadavky jsou zpracovávány dle priority

- dle závažnosti, množství výskytů, ...

Hlášení chyby (požadavku) by mělo obsahovat maximum relevantních informací:

- jméno požadavku
- stručný popis
- verze SW
- verze operačního systému, knihoven, platforma...
- postup, jak chybu reprodukovat
- popis očekávaného chování
- popis skutečného chování
- kontaktní údaje

The screenshot shows a JIRA issue page for 'The printer is not working' (Issue ID: BLA-6). The interface includes a navigation bar with 'Dashboards', 'Projects', 'Issues', and 'Administration'. The issue title is 'The printer is not working' and it is categorized as a 'Bug' with a 'Major' priority. The status is 'Open' and 'Unresolved'. The issue was created on 06/Jan/12 at 5:25 PM and updated at 5:57 PM. The description states: 'The laser printer is not working. Please help me printing the attached file. Merged from BLA-7 - The printer is not working again. Please print the attache file as i am unable to use the printer again.' There are two attachments: 'Distributed_Version_Control_Systems_Why_and_How.pdf' (147 kB) and 'teaching.doc' (70 kB). The activity log shows two comments: one from 'admin' at 5:27 PM stating 'it was turned off but anyway i have printed it for you.' and one from 'user' at 5:34 PM stating 'Please help me as it is very urgent!!!!'. The page also features action buttons like 'Edit', 'Assign', 'Comment', 'More Actions', 'Start Progress', 'Resolve Issue', and 'Workflow', along with a 'Views' dropdown.

JIRA admin | Quick Search

Dashboards | Projects | Issues | Administration | + Create Issue

blabla / BLA-6

The printer is not working

Edit Assign Comment More Actions Start Progress Resolve Issue Workflow Views

Details

Type:	Bug	Status:	Open
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		

People

Assignee: admin
Reporter: user
Vote (0) Watch (0)

Dates

Due: 06/Jan/12
Created: Today 5:25 PM
Updated: Today 5:57 PM

Description

The laser printer is not working. Please help me printing the attached file.
Merged from BLA-7 - The printer is not working again
Please print the attache file as i am unable to use the printer again.

Attachments

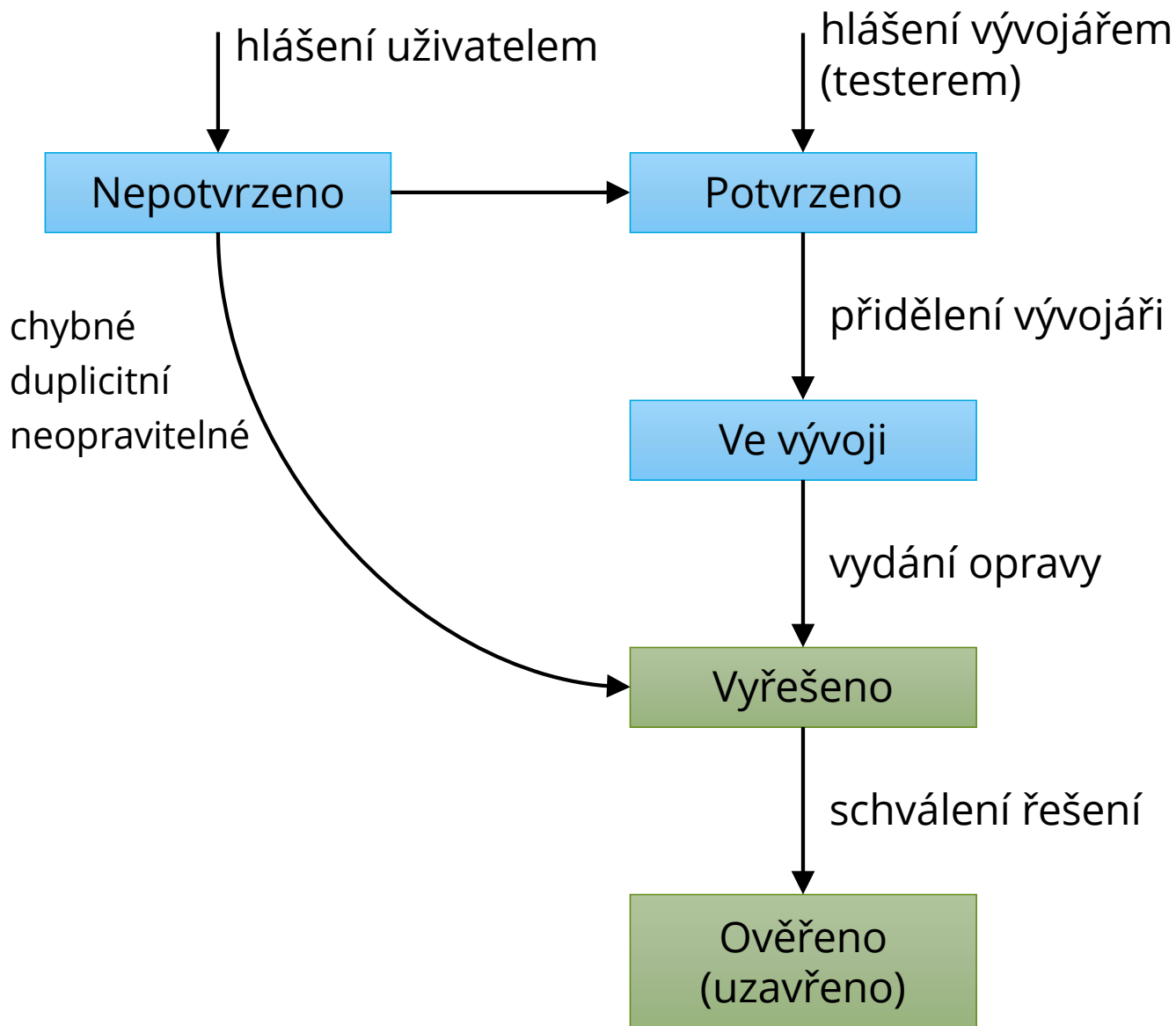
Distributed_Version_Control_Systems_Why_and_How.pdf	147 kB	06/Jan/12 5:32 PM
teaching.doc	70 kB	06/Jan/12 5:25 PM

Activity

All | Comments | Work Log | History | Activity | Source | Reviews

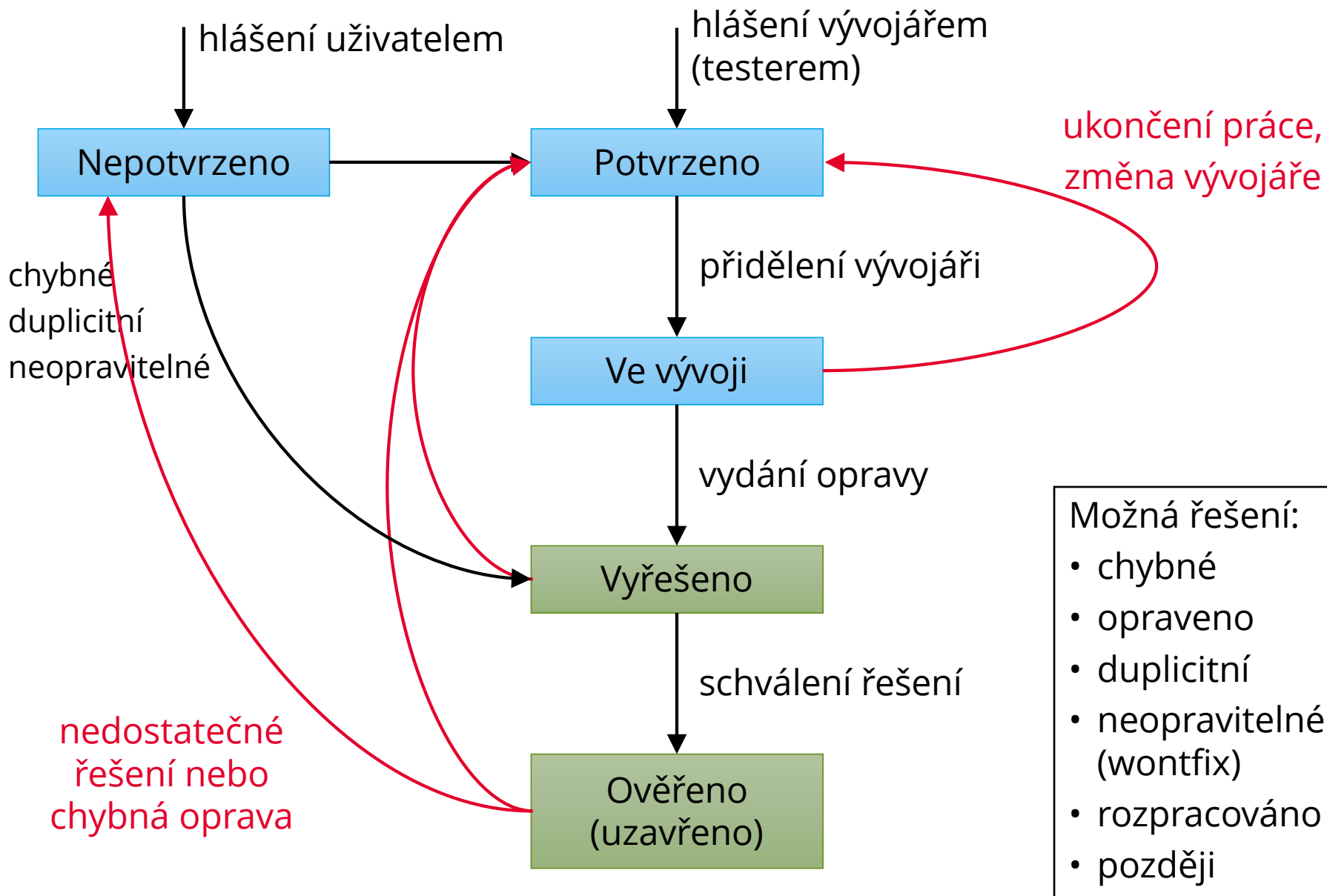
admin added a comment - 06/Jan/12 5:27 PM
it was turned off but anyway i have printed it for you.

user added a comment - 06/Jan/12 5:34 PM
Please help me as it is very urgent!!!!



Možná řešení:

- chybné
- opraveno
- duplicitní
- neopravitelné (wontfix)
- rozpracováno
- později



PROFILING

- Program funguje správně, ale pomalu
- Profiler pomáhá identifikovat místa v programu, kde se spotřebuje nejvíce strojového času
 - 80 % strojového času se stráví nad 20 % kódu

Postup:

1. Měření

- získání údajů o běžícím programu
- počet vyvolání funkcí, počet iterací cyklů, čas strávený v jednotlivých částech kódu, čekání na I/O...

2. Analýza

- statistické vyhodnocení naměřených dat
- vizualizace pomocí tabulek a grafů

3. Optimalizace

Sampling (statistický přístup)

- periodické přerušení, ve kterém se zaznamená aktuální poloha v programu
- nepřesné, ale nezpomaluje tolik běh programu
- AMD CodeAnalyst, Apple Shark, Intel Vtune

Instrumentace programu

- do programu se vloží volání speciální funkce
- více ovlivňuje rychlost programu
- některé chyby se nemusí projevit, jiné se naopak začnou projevovat

Flat profile

- čas strávený v jednotlivých funkcích
- počet volání

Graf volání (Call graph)

- pro každou funkci: odkud byla volána, jaké funkce volala
- jak dlouho každé volání funkce trvalo

Anotovaný kód (Annotated source)

- Ke každému řádku kódu je přidán počet vykonání

- Kombinuje statistický přístup s instrumentací
- Čas strávený v jednotlivých funkcích pomocí periodického sledování
 - čas běhu programu by měl být výrazně vyšší než perioda vzorkování (0,01 s)
- Počet volání funkcí pomocí instrumentace
- Výstup: flat profile, graf volání

```
$ gcc -Wall -Wextra -O2 -g -pg ./program.c -o program
$ ./program
$ ls
program gmon.out
$ gprof program gmon.out > output-file.txt
```

- Soubor gmon.out vznikne v pracovním adresáři
- Každý modul je třeba přeložit pro profilování (-pg)
 - v opačném případě nebude na výstupu gprof

Pro zvýšení přesnosti lze zkombinovat více běhů:

```
gcc -Wall -Wextra -O2 -g -pg ./program.c -o program
```

```
# první běh
```

```
$ ./program
```

```
$ mv gmon.out gmon.sum
```

```
# opakuj n-krát
```

```
$ ./program
```

```
$ gprof -s program gmon.out gmon.sum
```

```
# souhrnné výsledky
```

```
$ gprof program gmon.sum > output-file.txt
```


Each sample counts as 0.01 seconds.

%	cumulative	self		self	total	
time	seconds	seconds	calls	s/call	s/call	name
33.86	15.52	15.52	1	15.52	15.52	func2
33.82	31.02	15.50	1	15.50	15.50	new_func1
33.29	46.27	15.26	1	15.26	30.75	func1
0.07	46.30	0.03				main

Časy které nejsou mnohem větší než vzorkovací perioda nejsou příliš věrohodné

granularity: each sample hit covers 2 byte(s) for 0.02% of 46.30 seconds

index % time self children called name

```
[1]  100.0  0.03  46.27          main [1]
      15.26 15.50    1/1    func1 [2]
      15.52 0.00    1/1    func2 [3]
-----
      15.26 15.50    1/1    main [1]
[2]  66.4   15.26 15.50    1    func1 [2]
      15.50 0.00    1/1    new_func1 [4]
-----
      15.52 0.00    1/1    main [1]
[3]  33.5   15.52 0.00    1    func2 [3]
-----
      15.50 0.00    1/1    func1 [2]
[4]  33.5   15.50 0.00    1    new_func1 [4]
-----
```

- Součást valgrindu
- Výrazně pomalejší běh programu, ale není třeba speciální překladač
- Analýza např. pomocí KCachegrind

```
$ valgrind --tool=callgrind ./program
```

```
$ ls
```

```
program program.out.pid
```

The screenshot shows the KCachegrind interface. On the left, a table lists the call stack with columns for 'Incl.', 'Self', 'Caller', 'Funkce', and 'Umístění'. The function 'isSudokuSolved' is highlighted in green. On the right, a call graph visualizes the execution flow. The root node is 'doSudoku' (20 025), which calls 'isSudokuSolved' (20 025). This function then branches into 'isBlockSolved' (9 153), 'isRowSolved' (5 310), and 'isColSolved' (5 310). Each of these further calls 'bToR' (1 782), 'bToC' (1 620), and 'isBadNumber' (3 159) respectively. The interface also includes a 'Flat Profile' window and various tool options at the top.

- Profiler napoví, kde se vyplatí optimalizovat
- Je zbytečné optimalizovat funkci, které se zavolá jednou a neběží dlouho
- Výstup profileru může být zatížen statistickou chybou
- Vhodné pro rozsáhlejší programy
- Může pomoci odhalit chyby
 - více/méně volání funkce, než se očekává
- Spuštění s profilerem ovlivňuje běh programu
 - rychlost, heisenbugs
- Optimalizace kódu může zhoršit čitelnost
 - Mnohdy je lepší optimalizovat algoritmus než implementaci

QUALITY ASSURANCE

- Proces zajišťování kvality
- Týká se všech fází vývoje, pro procesy i pro výstup

Kontrola kvality procesů

- ISO 9001, CMM, SPICE
- certifikovat lze i proces výroby nekvalitních produktů

Kontrola kvality výstupu (software)

- code review
- testování
- QA inženýr kontroluje práci vývojářů

Hodnocení vyspělosti procesů v organizaci

1 - Počáteční (Initial)

- Procesy jsou realizovány adhoc

2 - Opakované (Repeatable)

- Dodržuje se určitá kázeň nezbytná pro provádění základních opakovaných procesů

3 - Definovaná (Defined)

- Procesy organizace jsou zdokumentovány

4 - Řízená (Managed)

- Procesy jsou řízeny a provádí se měření jejich výkonnosti pomocí KPI

5 - Optimalizovaná (optimized)

- Procesy jsou trvale zlepšovány, existuje inovační cyklus na procesech a řízení

iwiglasz@fit.vutbr.cz