

# 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](mailto:iwiglasz@fit.vutbr.cz)



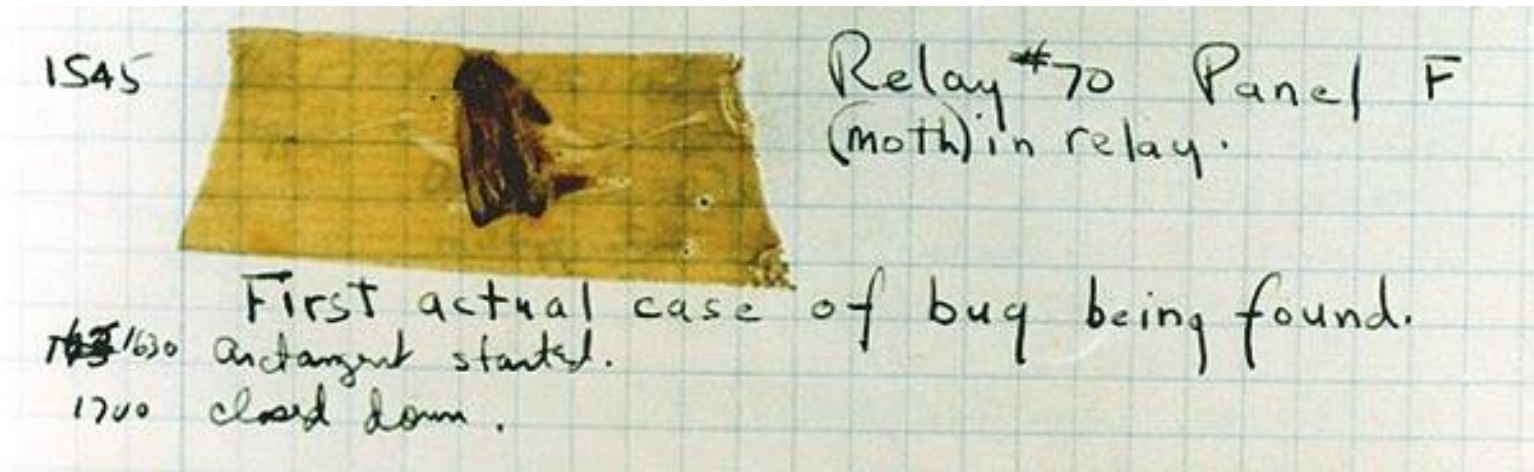
26. 3. 2019

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.



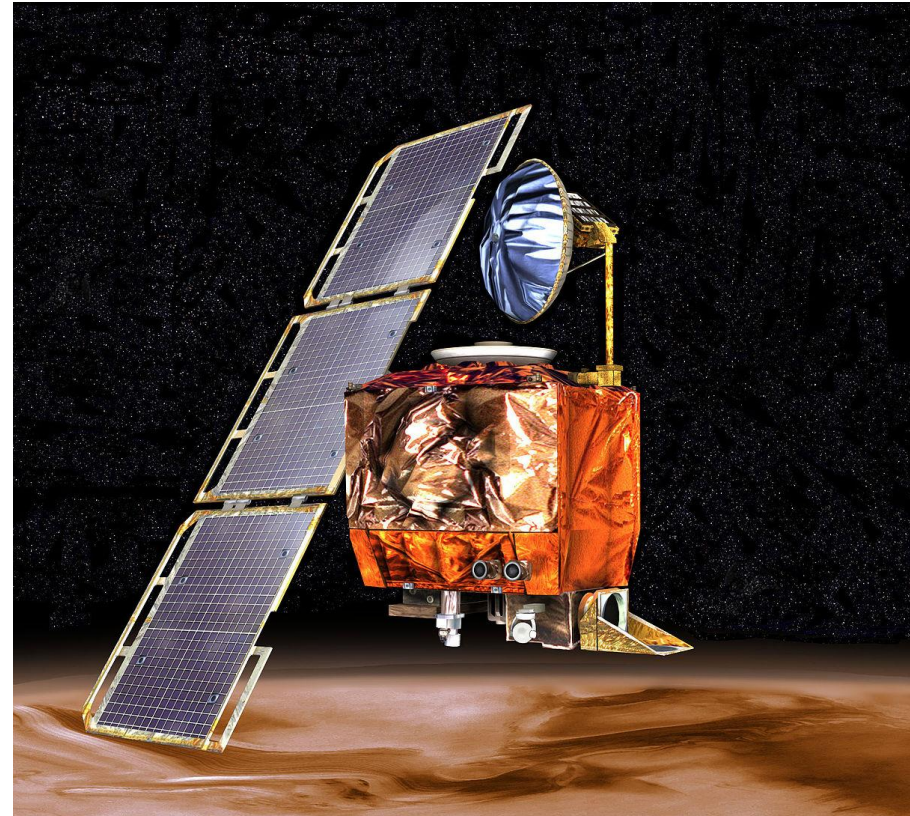
## Mariner 1 (1962)

- Americká sonda určená k průzkumu Venuše
- Chyba při přepisu vzorce naváděcího programu
- Z původního  $\overline{R_n}$  se během přepisu stalo  $\dot{R}_n$
- Čára znamená „vyhlazená data“  
(nevýznamné odchylky by neměly být brány v potaz)
- Ztráta 18,5 milionů dolarů  
(cca 150 milionů dnešních dolarů)



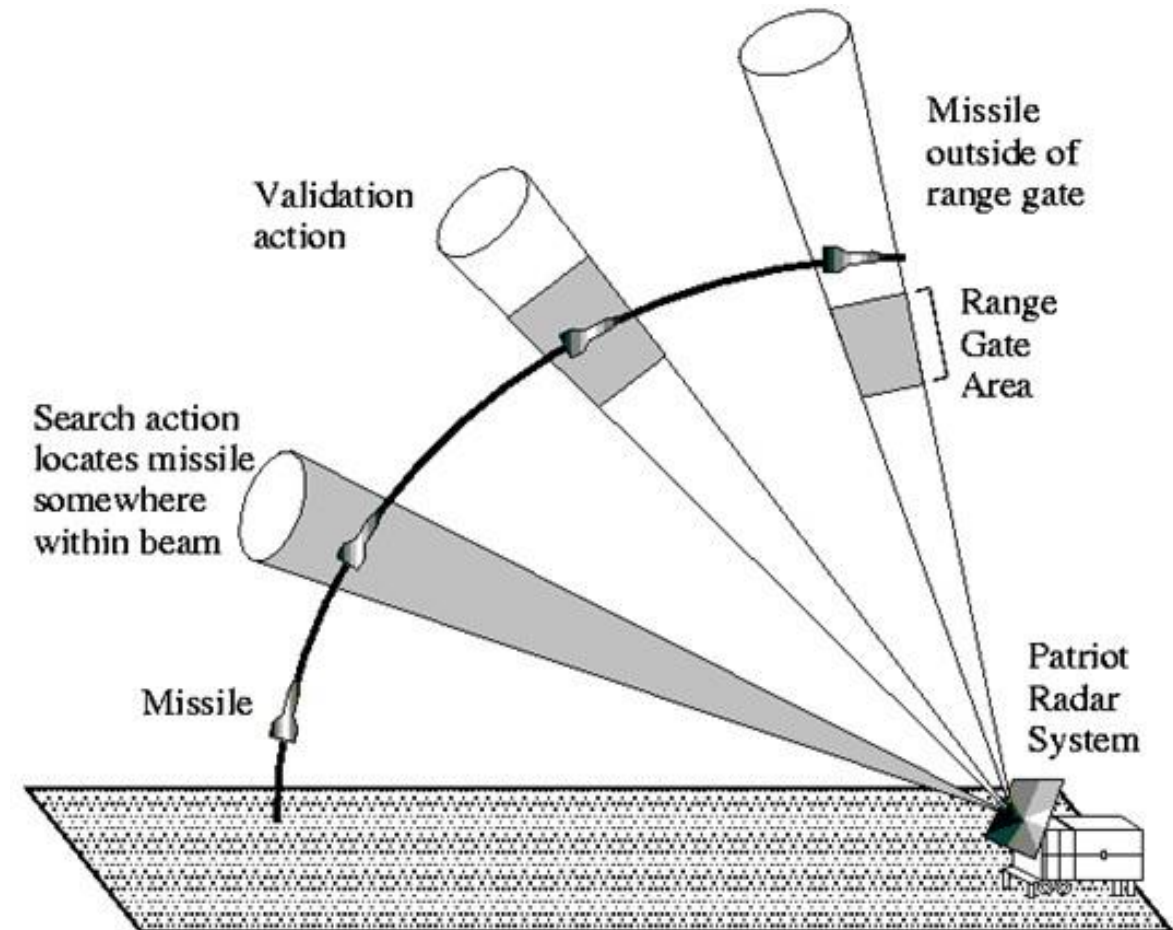
## Mars Climate Orbiter (1999)

- Software od Lockheed Martin předával výsledky v imperiálních jednotkách
- Software od NASA je očekával v jednotkách SI
- Výsledkem byla chybně vypočítaná trajektorie vstupu na oběžnou dráhu (po 286 dnech letu)
- Cena: 328,6 milionů dolarů (cca 507 milionů dnešních dolarů)



## Protiraketové střely Patriot (25. 2. 1991)

- 28 mrtvých, 98 zraněných
- systémové hodiny měřily čas v desetinách sekundy (jako celé číslo)
- čas se pak převáděl na desetinné číslo (tzn. vydělil se 10) na 24 bitech
- po 100 hodinách odchylka 0,34 s
- za tuto dobu irácká raketa Scud uletěla cca 800 metrů (a proto ji Patriot minul)



## USS Yorktown CG-48 (1997)

- Smart Ship, síť 27 počítačů
- Po zadání nuly na nesprávné místo v datech došlo k dělení nulou, přetečení bufferu a výpadku na 2,5 hodiny



## Therac-25 (80. léta)

- Minimálně 5 mrtvých (údajně až 22)
- Obsluha omylem zvolila použití vysokoenergetického paprsku, po opravě na nízkoenergetický se ale přesto spustil vysokoenergetický paprsek
- Chybějící code review
- Chybějící hardwarové pojistky (odstraněny při přechodu na čistě SW řízení)
- První testy proběhly až v nemocnici
- Systém poznal, že něco neseďí, ale chyby byly časté a šlo je ignorovat
- Race condition, pokud byl operátor moc rychlý
  - jedna proměnná pro více účelů



## Boeing 787 Dreamliner

- Po 248 dnech přejdou elektrické generátory do nouzového režimu (2015)
- Po 22 dnech se bez varování restartují všechny tři řídicí moduly (2016)





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, mapování instrukcí na místo ve zdrojovém 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

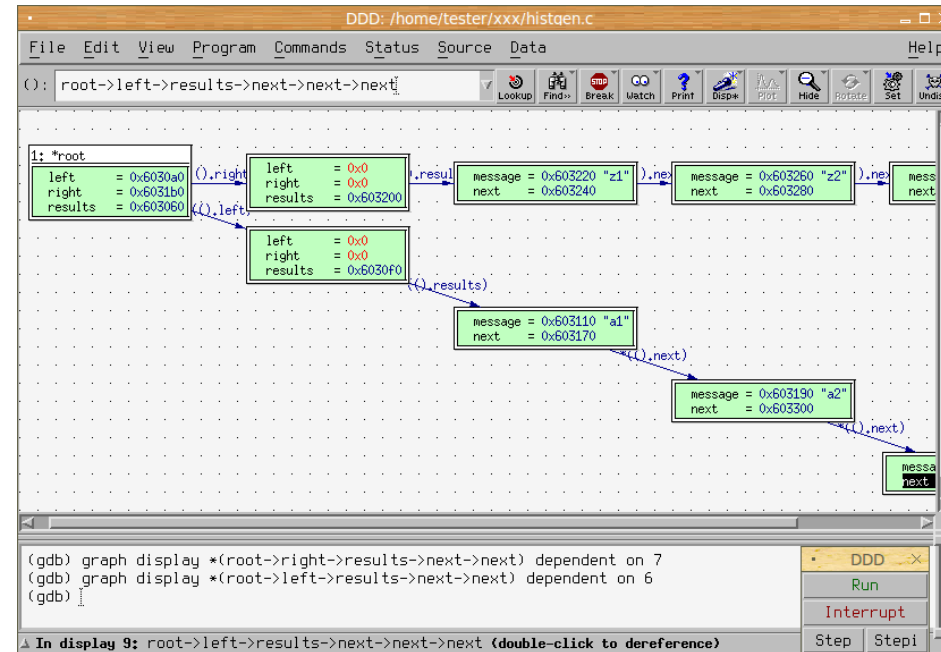
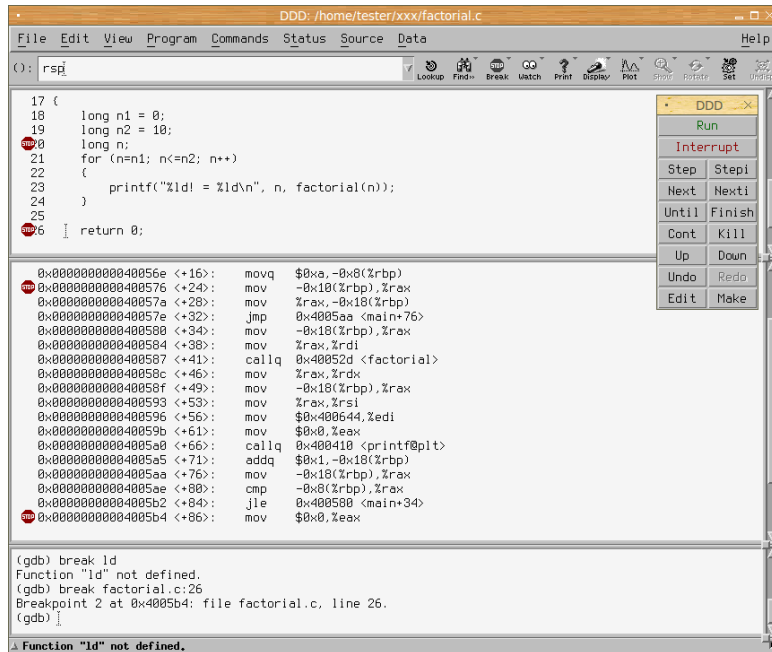
GNU gdb (GDB) 7.12
...
Reading symbols from ./segfault...done.
(gdb) run
Starting program: ~/segfault
Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7aca301 in __strlen_sse2 () from /lib64/libc.so.6
(gdb) backtrace
#0  0x00007ffff7aca301 in __strlen_sse2 () from /lib64/libc.so.6
#1 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



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

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

- nejvíc po ruce

The screenshot shows a code editor with a debugger. The code is in JavaScript and defines an Express.js application. A breakpoint is set at line 22, and the application is paused. The 'WATCH' panel shows the current state of variables, including 'sentiment' and 'level'. The 'CALL STACK' panel shows the current function call. The 'DEBUG CONSOLE' shows the output 'app listening on port: 3000'.

```
8 const path = require( path );
9
10 const app = express();
11 app.set('views', path.join(__dirname, 'client/views'));
12 app.set('view engine', 'pug');
13 app.use(bodyParser.json());
14 app.use(bodyParser.urlencoded({ extended: true }));
15 app.use(express.static(path.join(__dirname, 'client'), { maxAge: 31557600000 }));
16
17 app.get('/', async function (req, res) {
18   const data
19   let sentiment
20
21   for (let s
22     let newTweet = {
23       sentiment: s.sentiment,
24       level: util.getHappinessLevel(s.sentiment)
25     };
26   sentimentWithLevel.push(newTweet);
27 }
28
29 res.render('index', {
30   tweets: sentimentWithLevel,
31   counts: data.counts
32 });
```



## „printf debugging“

- výpis „jsem zde“, hodnoty proměnných, všeho zajímavého
- pomoci mohou i makra `__FILE__`, `__LINE__`, výpis stack trace...
- může se hodit takticky umístěný `if`, `assert`, `break`, `exit`, ...

```
..$time = $xml->location[0]->time[0];  
  
..$phase = $time->moonphase[0]['desc'];  
..dump($phase);  
..dump($time);  
..dump($time->moonphase[0]);  
..dump($time->moonphase[0]['desc']);  
..if (preg_match('#\(((^)+)\)#', $phase, $m)) {  
..    ..dump('matched');  
..    ..$phase = $m[1];  
..}  
..dump($phase);
```

## Logování

- chybu lze najít i zpětně (když se nikdo nedíval)
- log lze odeslat technické podpoře
  
- logování může ovlivnit chování programu
  - pravděpodobnost vzniku race condition
  - zpomalení
  - může snadno dojít místo na disku
  
- typicky:
  - uživateli se zobrazí stručná chybová hláška
  - vygeneruje se log a odešle se vývojáři (e-mailem, přes logovací službu)

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

- 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...

- 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
- dokonce i chyba v procesoru
  - <https://blog.cloudflare.com/however-improbable-the-story-of-a-processor-bug/>

## 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
- Sada nástrojů pro ladění a profilování programů



- Pouze pro unixové systémy
- V podstatě virtuální stroj
  - analyzovaný program je oddělený od procesoru
  - podstatně pomalejší 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:

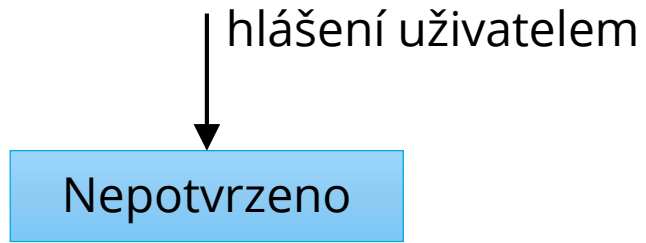
- blokuující
- kritický
- významný
- normální
- méně důležitý
- triviální (drobnosti, chyby v překladu...)
- rozšíření (požadavek na vylepšení)

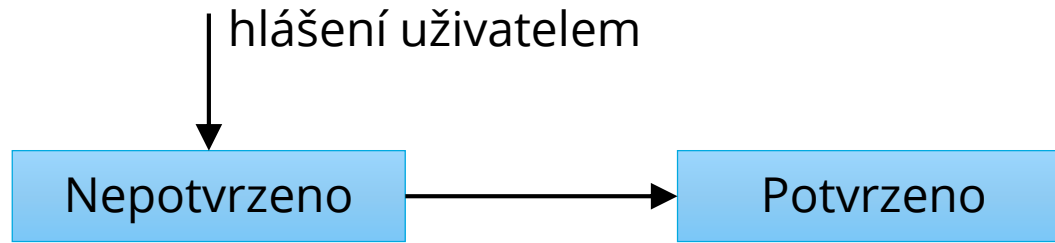
Požadavky jsou zpracovávány dle priority

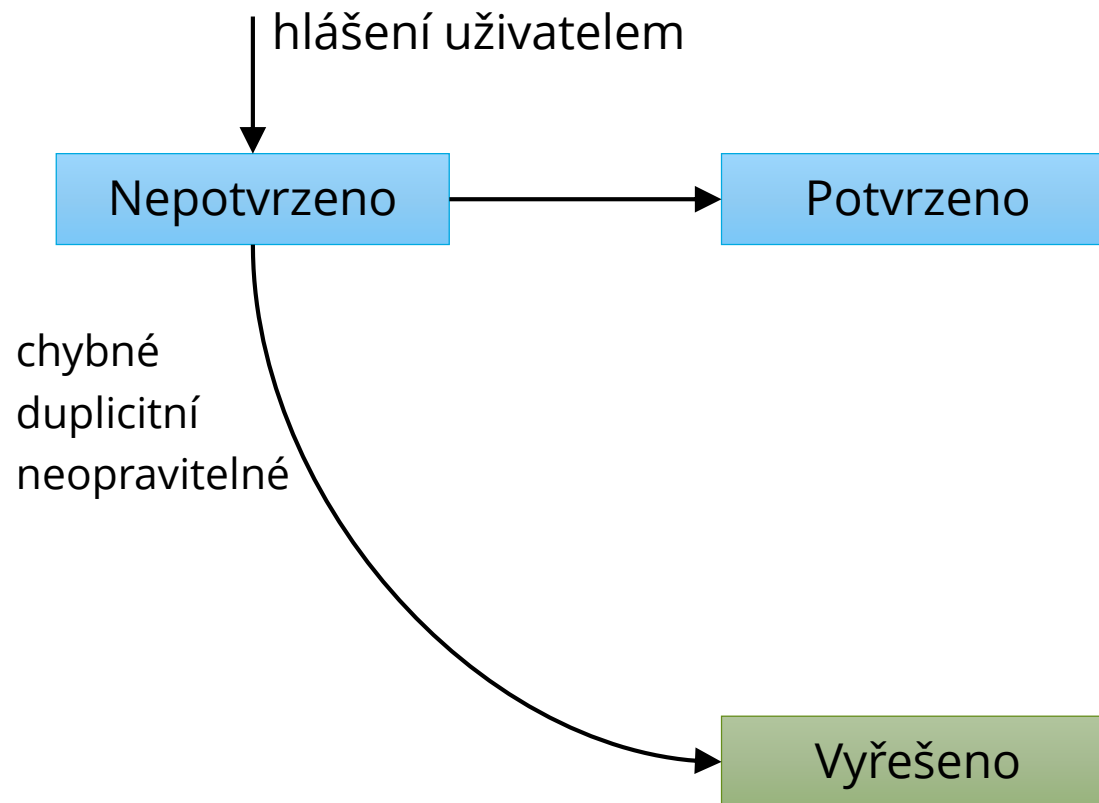
- dle závažnosti, množství výskytů, ...

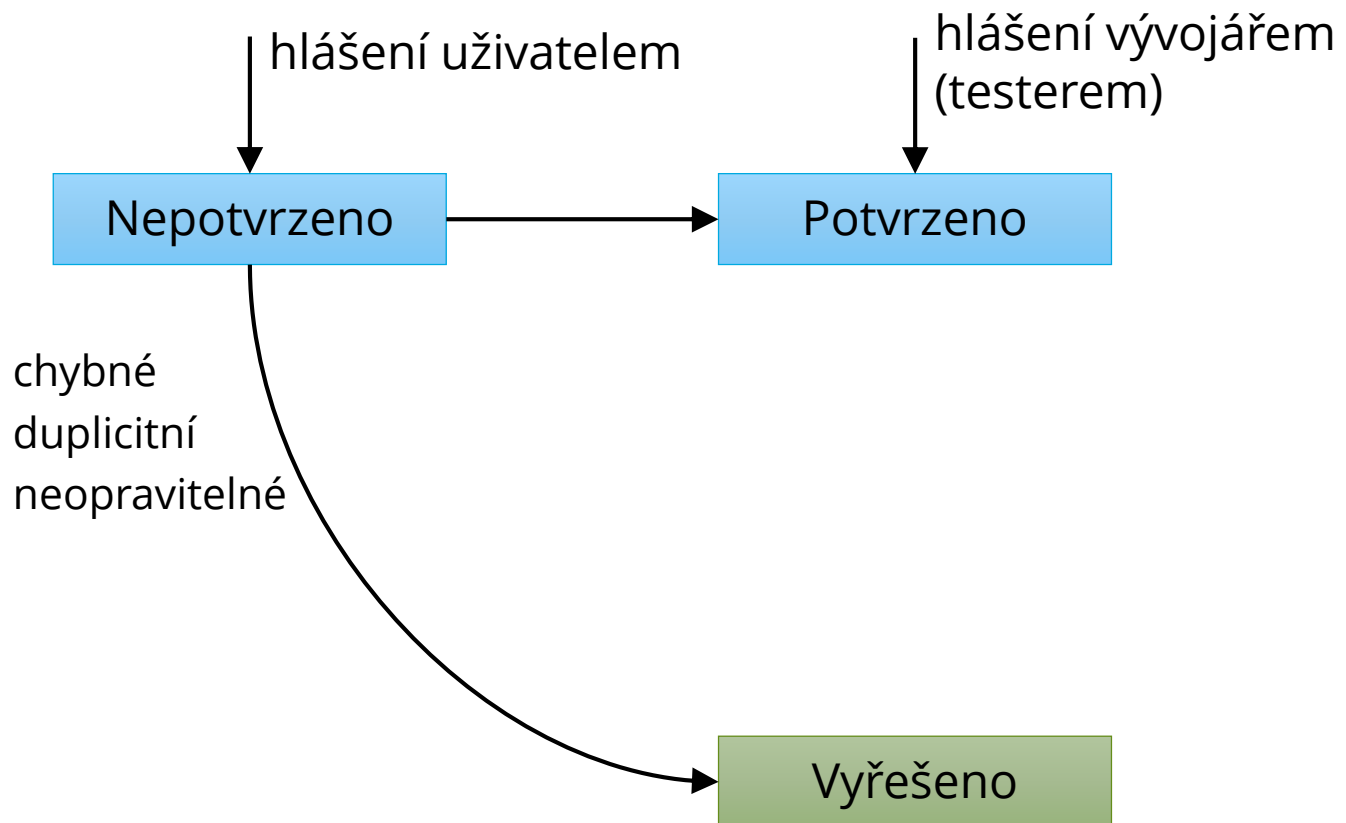
Hlášení chyby (požadavku) by mělo obsahovat maximum relevantních informací:

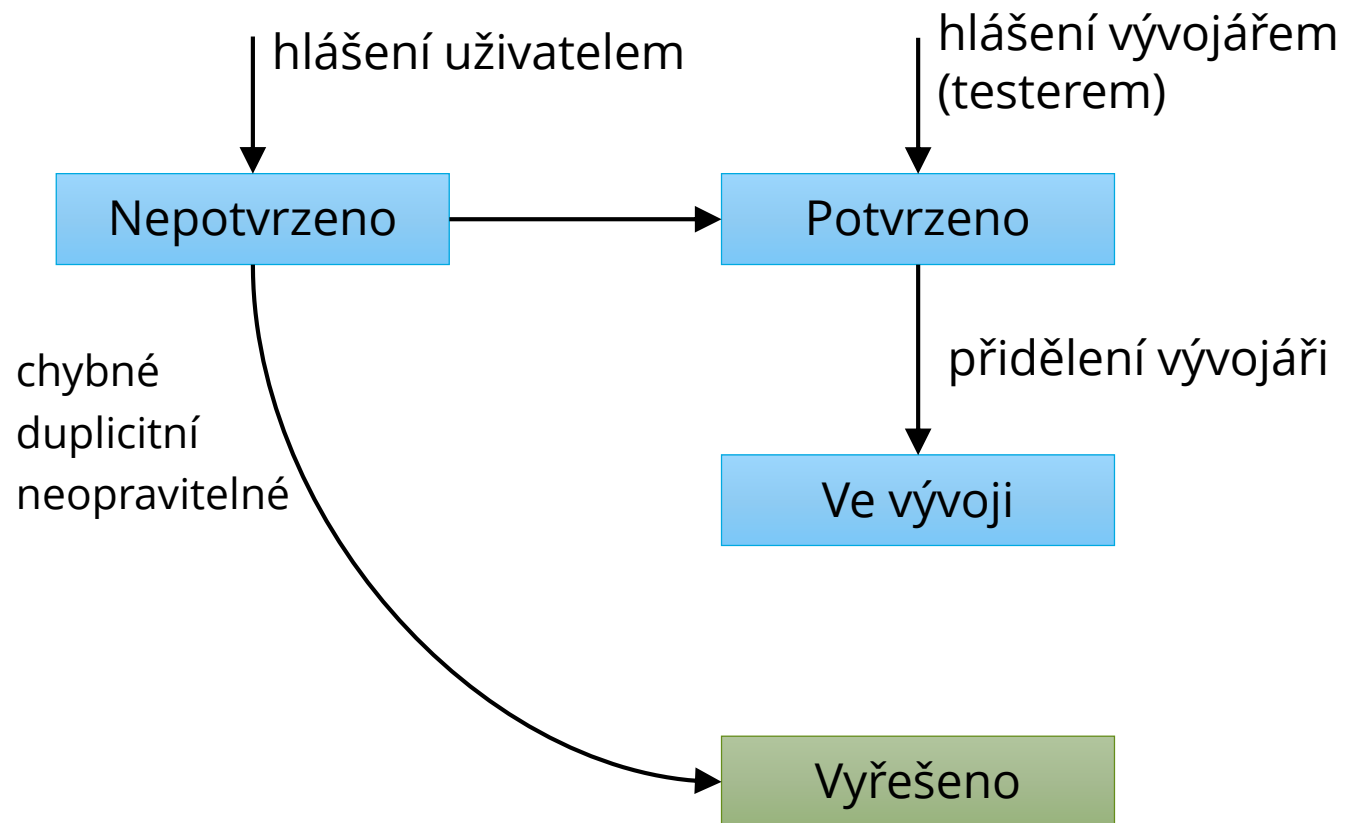
- jméno požadavku
- stručný popis
- verze SW
- verze operačního systému, knihoven, platforma...
- postup, jak chybu reprodukovat
- popis očekávaného chování
- popis skutečného chování
- kontaktní údaje



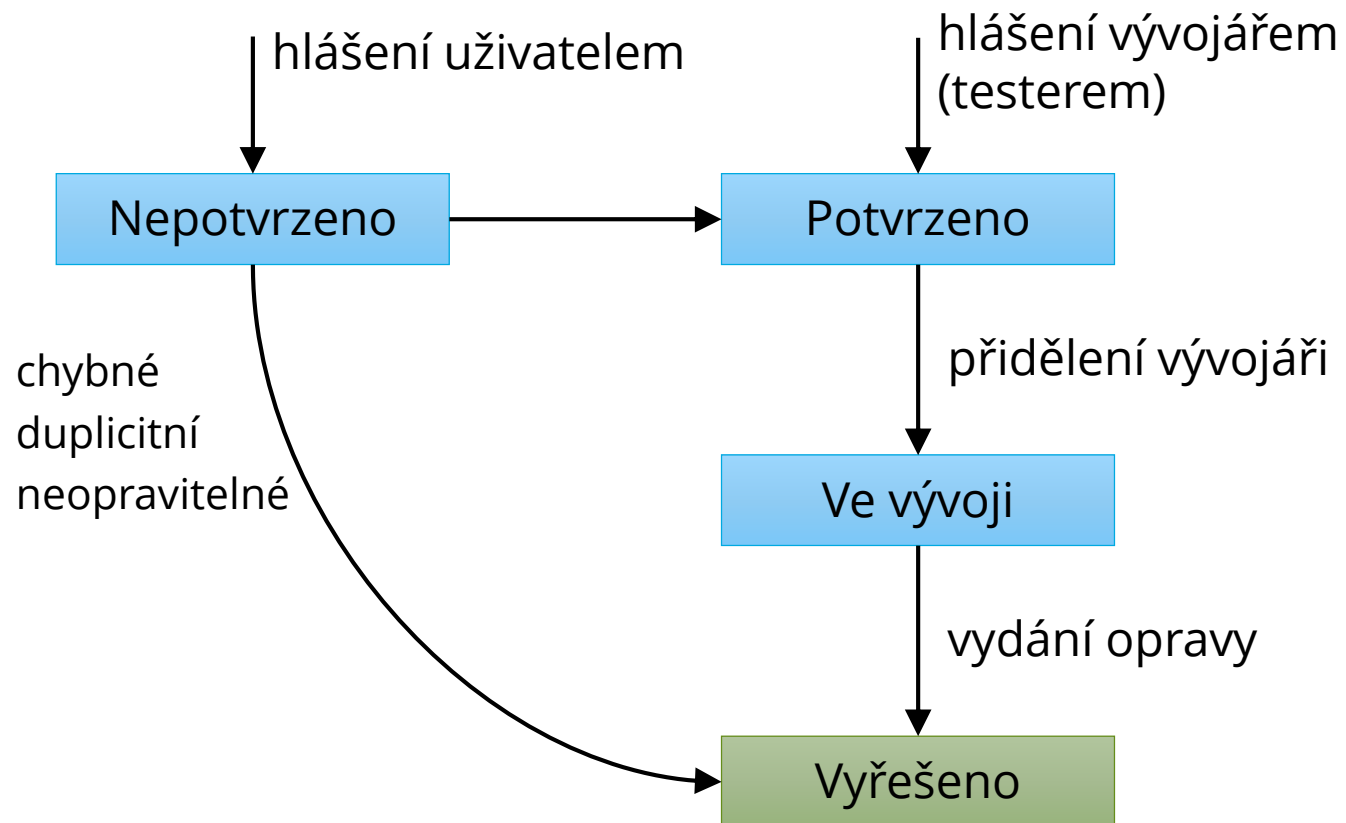


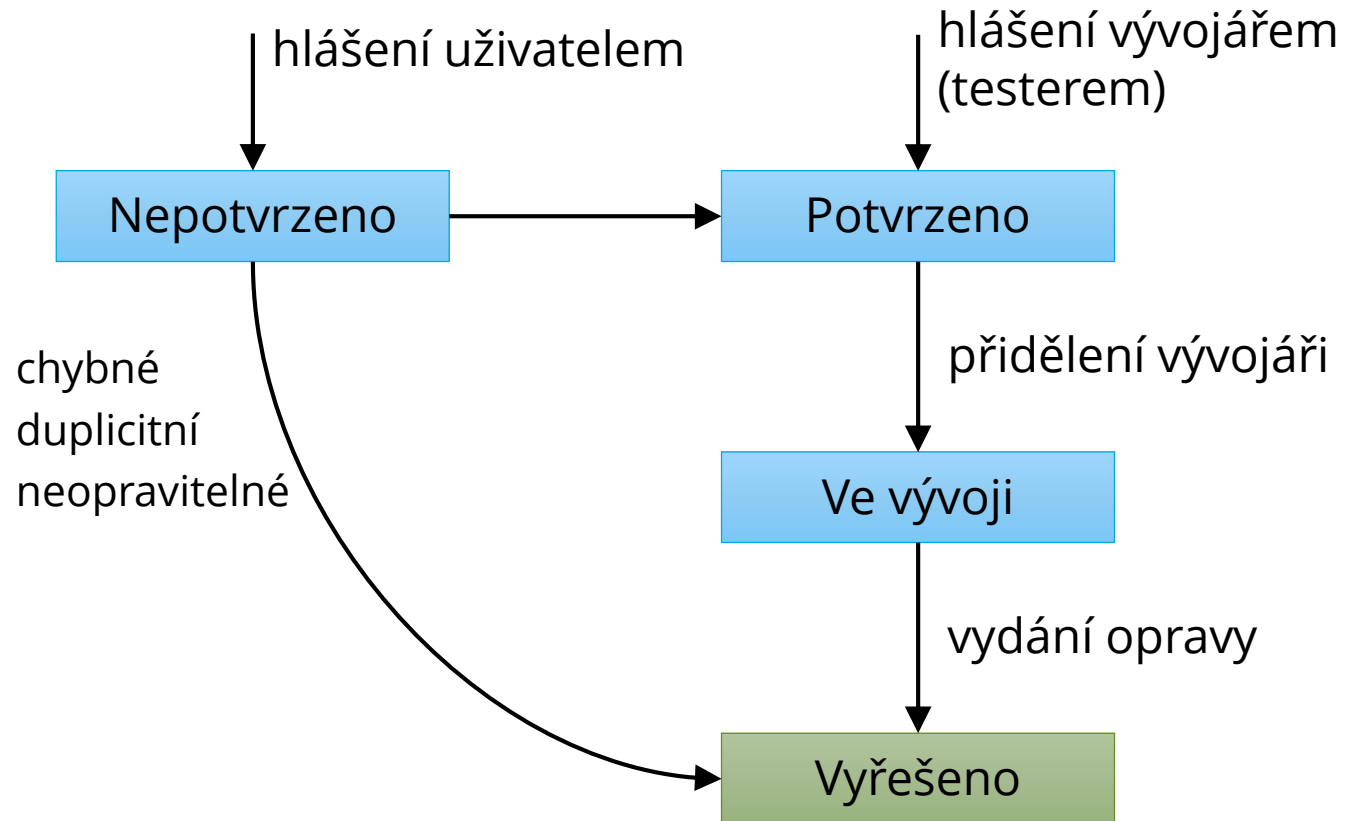




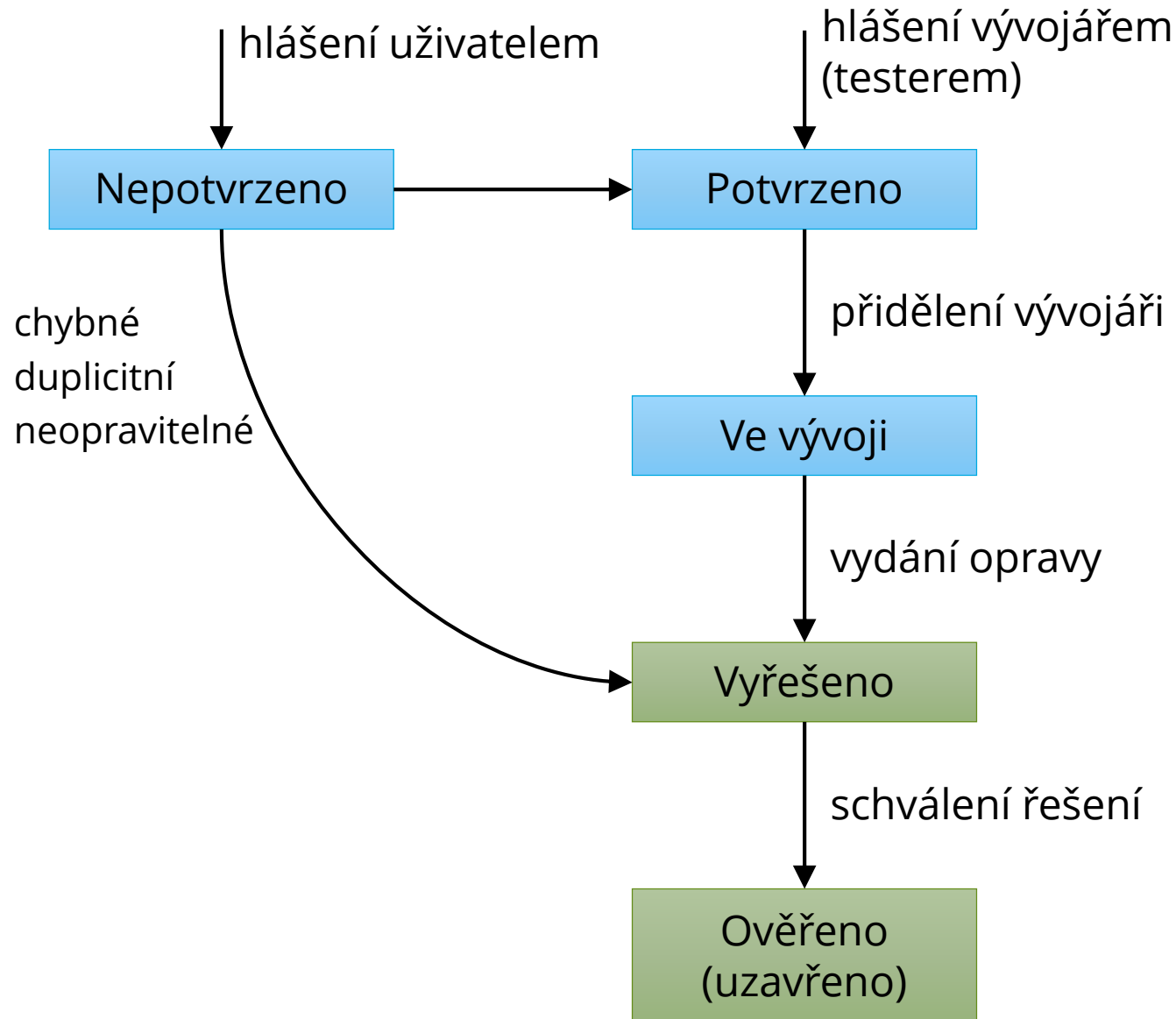




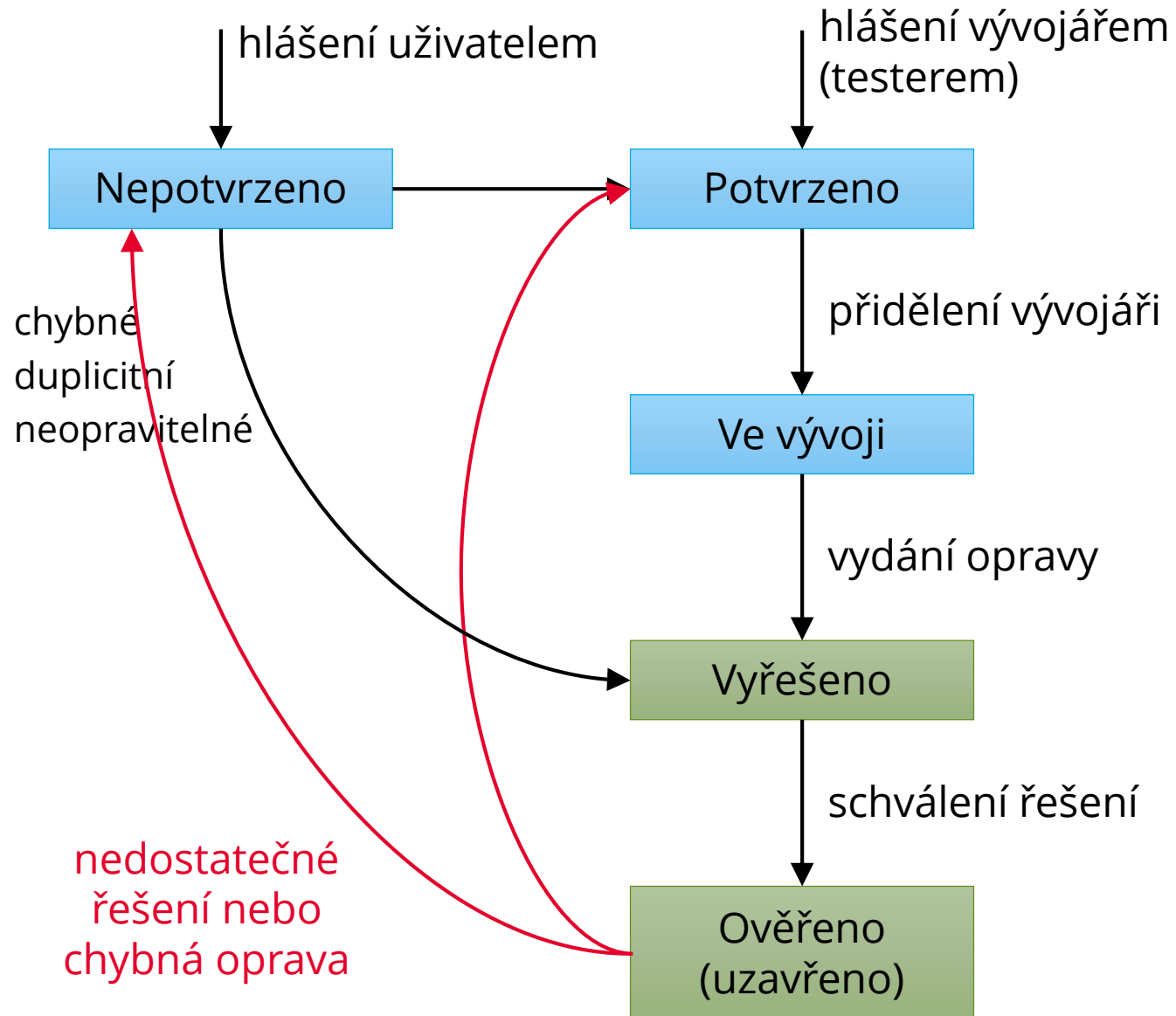




- Možná řešení:
- chybné
  - opraveno
  - duplicitní
  - neopravitelné (wontfix)
  - rozpracováno
  - později

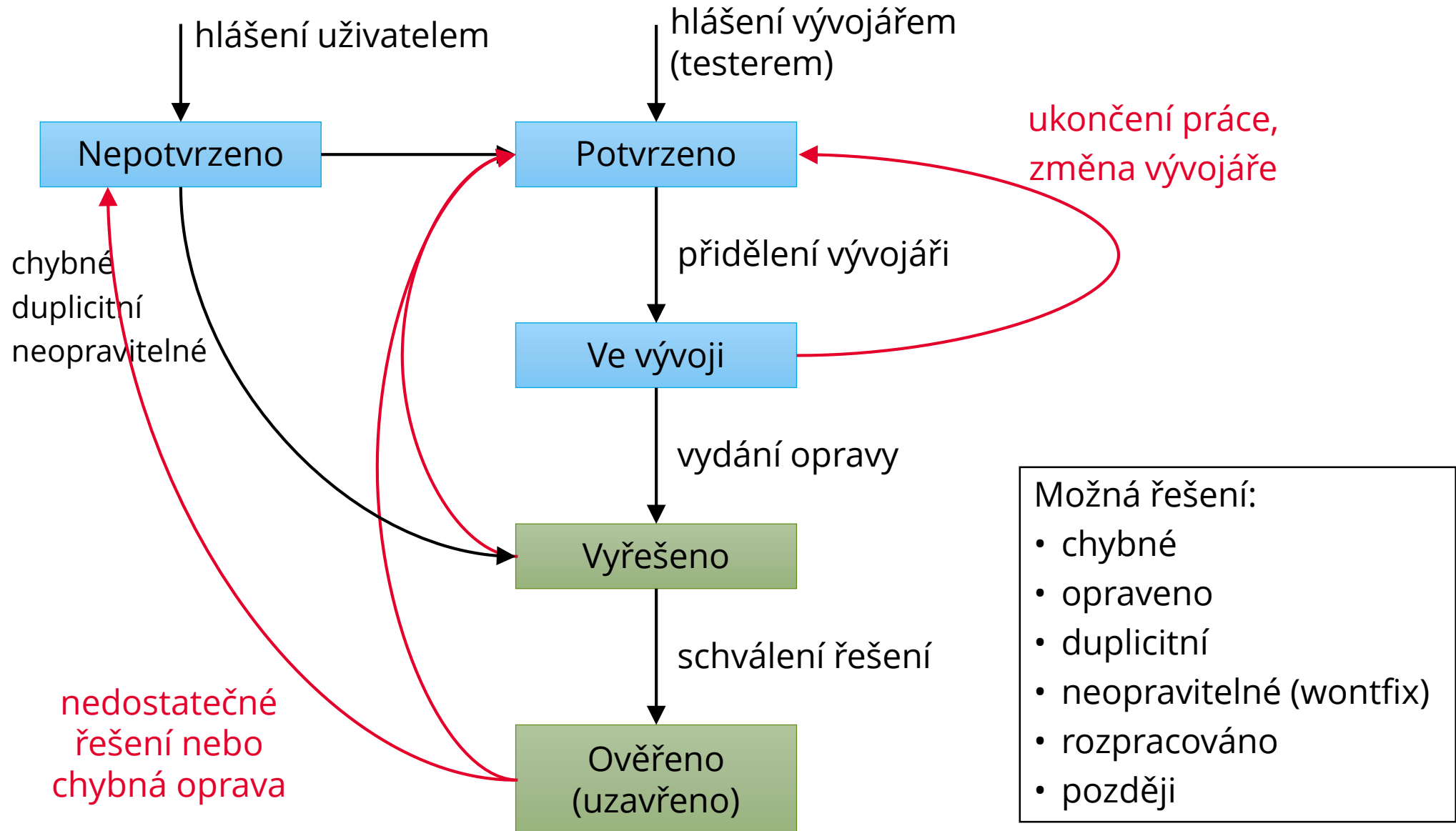


- Možná řešení:
- chybné
  - opraveno
  - duplicitní
  - neopravitelné (wontfix)
  - rozpracováno
  - později



Možná řešení:

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



The screenshot shows a JIRA issue page for 'blabla / BLA-6' with the title 'The printer is not working'. The issue is a 'Bug' with 'Major' priority and 'Open' status. It was reported by 'user' and assigned to 'admin'. 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 interface includes navigation tabs (Edit, Assign, Comment, etc.), a 'Details' section with metadata, a 'Description' section, an 'Attachments' table, and an 'Activity' section with comment filters.

Attachment Name	Size	Created
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

User	Time	Comment
admin	06/Jan/12 5:27 PM	it was turned off but anyway i have printed it for you.
user	06/Jan/12 5:34 PM	Please help me as it is very urgent!!!!

## Jak to bude s kachničkou? #3

New issue

Open Redak37 opened this issue 14 days ago · 1 comment



Redak37 commented 14 days ago



Jak to bude tento semestr, je nějaký plán, od kdy bude zase normálně fungovat bar a tak podobně?



2



Toaster192 commented 2 days ago

Member



Potřebujeme zlepšit evidenci pohybu věcí na baru

proto pracujeme na novem systemu který potřebujeme prvne dodelat

viz. :

*Isi frontend vývojář? Otravuje tě, že Kachna je otevřeva v provizorním režimu a chceš pomoct?*

Hledáme pomoc s implementací webové aplikace pro informační systém postavený na REST API (zdokumentovaném pomocí OpenAPI 3). Pivní odměna jistá (případně dle domluvy v rámci možnosti). V případě zájmu nás neváhej kontaktovat ([xbudis02@stud.fit.vutbr.cz](mailto:xbudis02@stud.fit.vutbr.cz) nebo prostřednictvím Discordu SU).

Jde na poměry SU o velký projekt, který byl dlouho odkládán, a proto jsme se do toho rozhodli "hodit vidle" aby se konečně realizoval.

michkot added **question** **SU / Kachna** labels 20 hours ago



Write Preview



Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Styling with Markdown is supported

Comment

Assignees

No one assigned

Labels

SU / Kachna

question

Projects

None yet

Milestone

No milestone

Notifications

Subscribe

You're not receiving notifications from this thread.

3 participants



# PROFILING



- Program funguje správně, ale pomalu
- Malé urychlení jedné malé smyčky může způsobit velké urychlení celého programu.
- Pravidlo 80/20:  
80 % strojového času se stráví nad 20 % kódu
- Profiler pomáhá identifikovat místa v programu, kde se spotřebuje nejvíce strojového času

## 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 (1/100 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 > gprof-report.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 > gprof-report.txt
```

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
31.75	9.90	9.90	1	9.90	9.90	new_func1
31.62	19.76	9.86	1	9.86	9.86	func2
22.17	26.68	6.91	1	6.91	16.82	func1
0.23	26.75	0.07				main

Časy které nejsou o moc větší než  
vzorkovací perioda nejsou příliš věrohodné

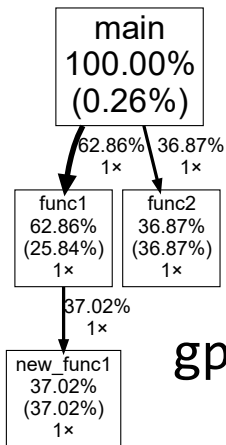


granularity: each sample hit covers 2 byte(s) for 0.04% of 26.75 seconds

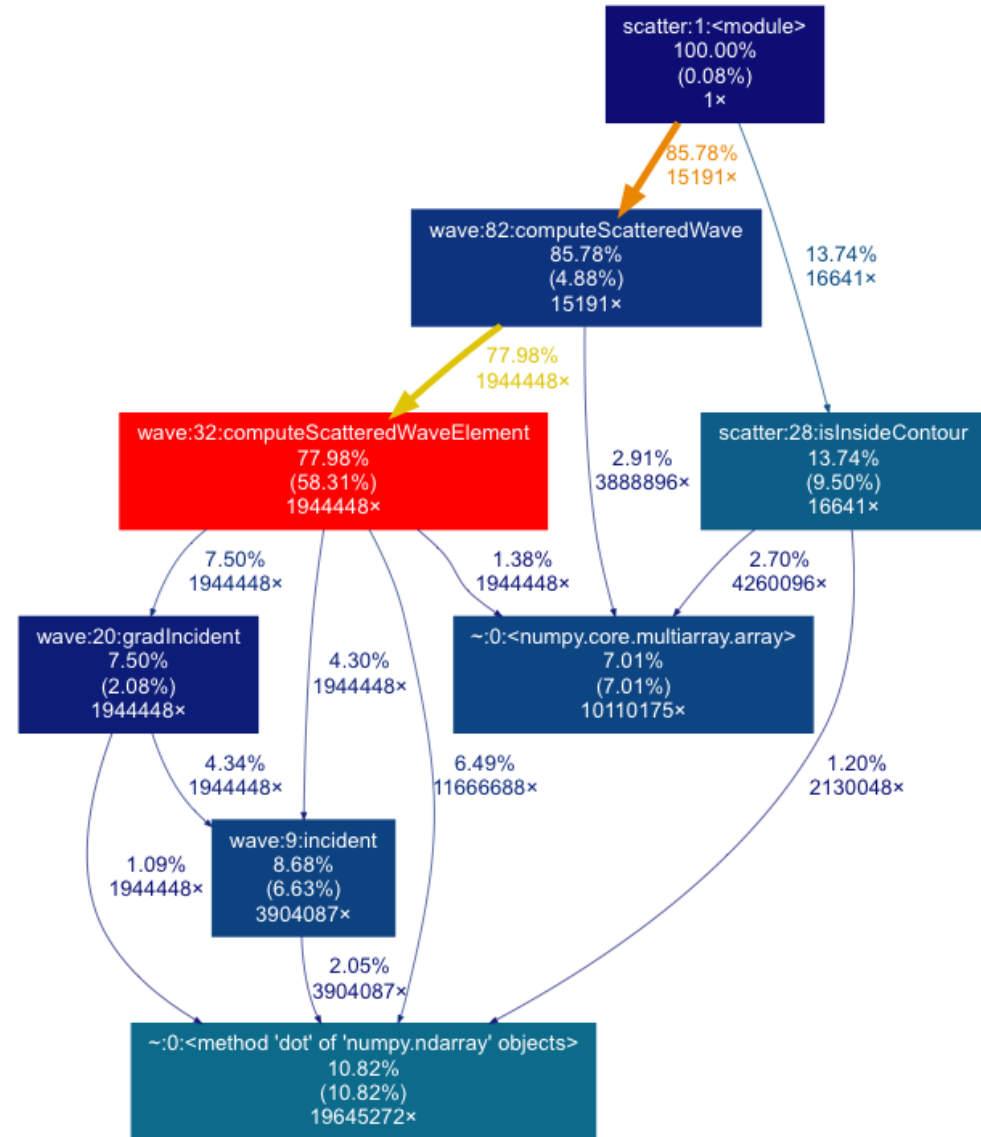
index	% time	self	children	called	name
					<spontaneous>
[1]	100.0	0.07	26.68		main [1]
		6.91	9.90	1/1	func1 [2]
		9.86	0.00	1/1	func2 [4]
-----					
		6.91	9.90	1/1	main [1]
[2]	62.9	6.91	9.90	1	func1 [2]
		9.90	0.00	1/1	new_func1 [3]
-----					
		9.90	0.00	1/1	func1 [2]
[3]	37.0	9.90	0.00	1	new_func1 [3]
-----					
		9.86	0.00	1/1	main [1]
[4]	36.9	9.86	0.00	1	func2 [4]
-----					

- <https://github.com/jrfonseca/gprof2dot>
- Převodník do formátu pro GraphViz
- Umí i jiné vstupní formáty než gprof

```
$ gprof2dot gprof-report.txt > gprof-report.dot
$ dot -Tpng -ogprof-report.png gprof-report.dot
$ dot -Tsvg -ogprof-report.svg gprof-report.dot
```



`gprof2dot -c print`





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

```
$ valgrind --tool=callgrind ./program
$ ls
program program.out.pid

$ ./gprof2dot.py -f callgrind program.out.pid > vis.dot
```

Soubor Pohled Přejít Nastavení Nápověda

Otevřít Reload Force Dump Nahoru Zpět Vpřed % Show Relative Costs Percentage Relative to Parent Skip Cycle Detection Instruction Fetch

### Flat Profile

Hledat:  ELF Object

Self	ELF Object
123 941	libc-2.8.90.so
98 329	ld-2.8.90.so
22 912	proj3

Incl.	Self	Caller	Funkce	Umístění
149 099	11	1	0x00000000004008C0	proj3
146 898	31	1	main	proj3: proj3.c
145 609	36	1	doOperation	proj3: proj3.c
122 368	37	1	readMatrix	proj3: proj3matrix.c
58 009	2 000	1	_readMatrix	proj3: proj3matrix.c
21 516	21	1	doSudoku	proj3: proj3.c
20 025	252	1	isSudokuSolved	proj3: proj3sudoku.c
9 153	4 698	9	isBlockSolved	proj3: proj3sudoku.c
5 310	4 257	9	isRowSolved	proj3: proj3sudoku.c
5 310	4 257	9	isColSolved	proj3: proj3sudoku.c
3 159	3 159	243	isBadNumber	proj3: proj3sudoku.c
2 683	236	1	allocMatrix	proj3: proj3matrix.c
1 782	1 782	81	bToR	proj3: proj3sudoku.c
1 689	144	1	freeMatrix	proj3: proj3matrix.c
1 620	1 620	81	bToC	proj3: proj3sudoku.c
1 470	11	1	writeValidation	proj3: proj3.c
1 258	284	1	doParams	proj3: proj3.c
54	25	1	_libc_csu_init	proj3
29	6	1	0x00000000004007A8	proj3: crt1.S, crtn.S
22	4	1	0x0000000000404988	proj3: crt1.S, crtn.S
18	18	1	0x0000000000400910	proj3
11	11	1	0x0000000000404950	proj3
6	6	1	0x0000000000400980	proj3
6	6	1	0x00000000004008EC	proj3: crt1.S

### isSudokuSolved

Typy Callers All Callers Source Code Callee Map

isBlockSolved	9 153	isRowSolved	5 310	isColSolved	5 310
bToR	1 782	bToC	1 620	isBadNumber	1 053
				isBadNumber	1 053

```

graph TD
    doSudoku[doSudoku  
20 025] --> isSudokuSolved[isSudokuSolved  
20 025]
    isSudokuSolved --> isBlockSolved[isBlockSolved  
9 153]
    isSudokuSolved --> isRowSolved[isRowSolved  
5 310]
    isSudokuSolved --> isColSolved[isColSolved  
5 310]
    isBlockSolved --> bToR[bToR  
1 782]
    isBlockSolved --> bToC[bToC  
1 620]
    isBlockSolved --> isBadNumber1[isBadNumber  
3 159]
    isRowSolved --> isBadNumber2[isBadNumber  
1 053]
    isColSolved --> isBadNumber3[isBadNumber  
1 053]
    
```

Caller Map Části Call Graph Callees All Callees Assembly Code

callgrind.out.12582 [1] - Total Instruction Fetch Cost: 245 182

- 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 vspělosti 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](mailto:iwiglasz@fit.vutbr.cz)