Advanced Practical **Programming** for Scientists

Software Processes and Tools

Robert Gottwald, Matthias Miltenberger, Daniel Rehfeldt

Zuse Institute Berlin

June 9th, 2017

Software Processes and Tools

Agile Development and Continuous Integration

Tools for C/C++

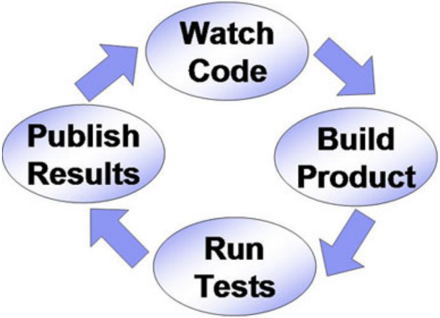
Time Measurement

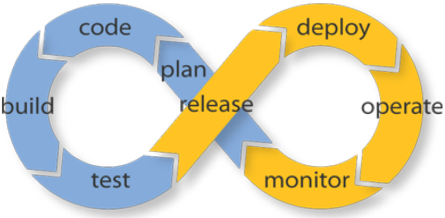
Agile Development

- avoid static/fixed development plans
- enable fast reaction to failures and problems
- esp. useful/applicable for small teams
- many agile methods
  - scrum
  - extreme programming
  - crystal clear
- basic concepts
  - pair/peer programming
  - code refactoring
  - continuous integration
  - release often
  - ...

Continuous Integration

- software engineering practice





Basic Concepts of CI

- automatic builds
- frequent (unit) tests
- dedicated build machine/server
- quick test results and analysis
- immediate feedback on broken builds (e.g. e-mail)
- requires version control (e.g. git)

Tools for CI

- Jenkins[1]
- features:
  - free, open source
  - written in Java
  - probably most popular (open source) CI tool right now
  - web interface
  - easy to deploy (`java jenkins.jar`)
  - easy to maintain (complete control using web interface)
  - large active community
  - many different plugins to extend the features
    (in fact, Jenkins is more of a framework that relies on its plugins)
  - different types of permission control (e.g. LDAP, or Jenkins' own one)

[1] `https://jenkins.io/`

Tools for CI

- hosted CI tool:
  Travis-CI[2]
- features:
  - build and test projects hosted on `GitHub.com`
  - free for open source projects
  - supports many programming languages
  - Berlin based company

[2] `https://travis-ci.org/`

Next Step

Continuous Deployment

- directly release build that passes all tests
- eliminates long release preparations
- users immediatly get new features
- but: not always applicable in practice
  (users may need to integrate new versions by hand)

Agile Development and Continuous Integration

Tools for C/C++

Time Measurement

valgrind[3]

- Integrates different tools for analyzing programs at runtime

Memcheck	For detecting memory-management problems
Callgrind	performance profiler
Cachegrind	cache profiler
Massif	heap profiler
Helgrind	thread debugger

[3] `http://valgrind.org/`

Static analysis

- run time checks like valgrind only find errors in the parts that are executed
- static code analysis can find errors at compile time
  - division by zero
  - integer overflows
  - uninitialized values
  - null pointer dereferences
- especially useful for hard to find bugs that appear only in some executions
- for C/C++:
  - clang static analyzer `https://clang-analyzer.llvm.org/`
  - cppcheck `http://cppcheck.sourceforge.net/`
  - coverity `http://www.coverity.com/`

Demonstration

Time for a demo!

Software Processes and Tools

Agile Development and Continuous Integration

Tools for C/C++

Time Measurement

Time Measurement

How to measure "execution time" of program that starts at time t_0 and ends at time t_1?

Time Measurement

How to measure "execution time" of program that starts at time t_0 and ends at time t_1?

- $t_1 - t_0$? (wall-clock time or wall time)
  - Problem: what about the time spent in I/O etc.?

Time Measurement

How to measure “execution time” of program that starts at time t_0 and ends at time t_1?

- $t_1 - t_0$? (wall-clock time or wall time)
  - Problem: what about the time spent in I/O etc.?
- Alternative: measure only the time spent in CPU? (CPU time)
  - Problem: is this time actually being spent in the program?

Time Measurement

How to measure "execution time" of program that starts at time t_0 and ends at time t_1?

- $t_1 - t_0$? (wall-clock time or wall time)
  - Problem: what about the time spent in I/O etc.?
- Alternative: measure only the time spent in CPU? (CPU time)
  - Problem: is this time actually being spent in the program?
- Alternative: measure time CPU spent executing code in user space (user time)

CPU Time

- CPU time: amount of time CPU was used for processing instructions of program or of operating system. Can therefore be divided in
  - user time
  - system time (time spent in kernel space)

CPU Time

- CPU time: amount of time CPU was used for processing instructions of program or of operating system. Can therefore be divided in
  - user time
  - system time (time spent in kernel space)
- CPU time is measured in clock ticks (computers' notion of passing of time) or seconds
- CPU time $\leqslant$ wall time.

CPU Time

- CPU time: amount of time CPU was used for processing instructions of program or of operating system. Can therefore be divided in
  - user time
  - system time (time spent in kernel space)
- CPU time is measured in clock ticks (computers' notion of passing of time) or seconds
- CPU time $\leqslant$ wall time.
- CPU usage refers to CPU time as a percentage of CPU's capacity

CPU Time in Unix

top or htop command to get information including user time and details of all processes.

```
Terminal                    ×          Terminal                    ×          Terminal
top - 15:58:03 up 7 days, 24 min,  4 users,  load average: 0.11, 0.12, 0.09
Tasks: 268 total,   1 running, 267 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.8 us,  0.3 sy,  0.0 ni, 98.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 32853460 total, 19747376 free,  2848244 used, 10257840 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 28968140 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
14653 bzfrehfe  20   0 3076624 1.022g 193540 S   4.0  3.3  64:57.84 Web Content
21155 adm_timo  20   0 4376940 217388  17948 S   0.7  0.7   0:29.97 java
  809 mongodb   20   0  388320  52136  18092 S   0.3  0.2  23:55.25 mongod
14600 bzfrehfe  20   0 2557064 426296 151308 S   0.3  1.3  40:39.68 firefox
    1 root      20   0  185660   6252   3944 S   0.0  0.0   0:03.20 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.13 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:02.00 ksoftirqd/0
    7 root      20   0       0      0      0 S   0.0  0.0   3:01.94 rcu_sched
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
    9 root      rt   0       0      0      0 S   0.0  0.0   0:01.58 migration/0
   10 root      rt   0       0      0      0 S   0.0  0.0   0:01.47 watchdog/0
   11 root      rt   0       0      0      0 S   0.0  0.0   0:01.46 watchdog/1
   12 root      rt   0       0      0      0 S   0.0  0.0   0:01.65 migration/1
   13 root      20   0       0      0      0 S   0.0  0.0   0:01.41 ksoftirqd/1
```

Execution Time of Your Program (Unix)

/usr/bin/time -v to measure your program. For exercise 1:

```
[08.06.17, 16:17] bzfrehfe@opt59 ~/projects/lectures/appfs17/appfs/exercise1 (master)
 <91> /usr/bin/time -v ex1x ex1.dat
Valid values Loc1 50004466 with geo mean: 36.7817
Valid values Loc2 49994581 with geo mean: 36.7826
File: ex1.dat with 100001235 lines
        Command being timed: "ex1x ex1.dat"
        User time (seconds): 12.14
        System time (seconds): 0.67
        Percent of CPU this job got: 67%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:18.98
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 792852
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 10915
        Voluntary context switches: 7994
        Involuntary context switches: 64665
        Swaps: 0
        File system inputs: 4060288
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
[08.06.17, 16:18] bzfrehfe@opt59 ~/projects/lectures/appfs17/appfs/exercise1 (master)
 <92>
```

Exercise

Exercise 6: Extend your program for exercise 5 such that it measures (and prints) both it's own wall-clock and user time.

Questions about exercise 5?