Chair of Connected Mobility
TUM School of Computation, Information and Technology
Technical University of Munich

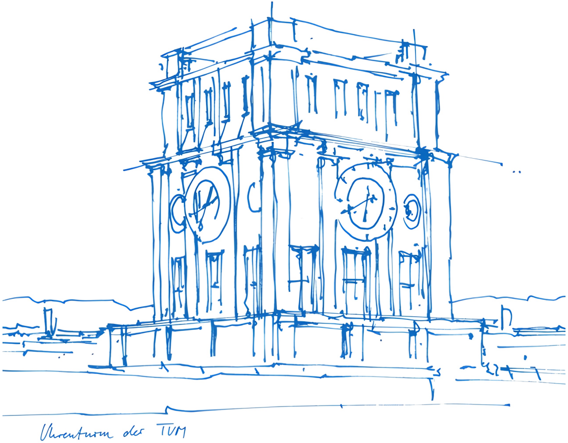
Open Source Lab

Utilities

Fabian Sauter, Christian Menges, Alexander Stephan

Chair of Connected Mobility
TUM School of Computation, Information and Technology
Technical University of Munich

Garching, November 23, 2022



Coverage

coverage 82%

Different Types:

- Statement Coverage (bad, don’t use it)
- Branch Coverage
- Condition Coverage
- Modified Condition / Decision Coverage (standard for high-risk avionics software: DO-178B, DO-178C)
- Path Coverage

```
x = 0
if (a == 0 || (b == 0 && c == 0)) {
       x = 1
}
```

Coverage

Different Types:

- Statement Coverage (bad, don’t use it)
- Branch Coverage
- Condition Coverage
- Modified Condition / Decision Coverage (standard for high-risk avionics software: DO-178B, DO-178C)
- Path Coverage

```
x = 0
if (a == 0 || (b == 0 && c == 0)) {
       x = 1
}
```

Coverage can help you to assess the quality of a test suite, but it should not be the reason why tests are written.

Ask, what is not covered instead of how much is not covered. (Example: If a project has 98% coverage, but the remaining 2% contain emergency shutdown or recovery routines, then this is not a good test suite)

Badge Source: https://camo.githubusercontent.com/1cde87ab5ba60df8d5283cfd2f24d8f5fea1f6c59561b48de1c7ac6a6747ab99/68747470733a2f2f6170702e636f646163792e636f6d2f70726f6a6563742f62616467652f436f7665726167652f38393836306165613566137346439393865633838346631613837356564306

Static Analyzers / Formatter

ALWAYS use it. There is no cheaper and easier way of finding bugs.

Tools depend on the language. A few examples:

C	gcc static-analyzer https://gcc.gnu.org/onlinedocs/gcc-12.1.0/gcc/Static-Analyzer-Options.html
C++	cppcheck https://github.com/danmar/cppcheck, clang-tidy https://clang.llvm.org/extra/clang-tidy/, clang-format https://clang.llvm.org/docs/ClangFormat.html (configuration generator: https://zed0.co.uk/clang-format-configurator/)
Go	go vet https://pkg.go.dev/cmd/vet, golangci-lint https://github.com/golangci/golangci-lint
Ruby	rubocop https://github.com/rubocop/rubocop, fasterer https://github.com/DamirSvrtan/fasterer, reek https://github.com/troessner/reek
XML-like (HTML, SVG, . . .)	W3 validator https://validator.w3.org/
Docker	Haskell Dockerfile Linter - hadolint https://github.com/hadolint/hadolint
Shell	ShellCheck https://github.com/koalaman/shellcheck
Tex	ChkTeX https://www.nongnu.org/chktex/
Javascript/Typescript	EsLint https://github.com/eslint/eslint
Java	PMD https://github.com/pmd/pmd
Rust	Miri https://github.com/rust-lang/miri, clippy https://github.com/rust-lang/rust-clippy, fmt https://github.com/rust-lang/rustfmt
Python	flake8 https://github.com/PyCQA/flake8

A more complete list can be found here:
https://analysis-tools.dev/tools

Dynamic analysis

- asan (address sanitizer)
- ubsan (undefined behavior sanitizer)
- lsan (leak sanitizer)
- tsan (thread sanitizer)
- DMon (current research, OSDI 21)
- ...

```
./leak.out
=================================================================
==81400==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 40 byte(s) in 1 object(s) allocated from:
#0 0x7fb07a0f1db0 in __interceptor_malloc ../../../../src/libsanitizer/lsan/lsan_interceptors.cpp:54
#1 0x401147 in main (/open-source-lab/leak/leak.out+0x401147)
#2 0x7fb079f44d09 in __libc_start_main ../csu/libc-start.c:308

SUMMARY: LeakSanitizer: 40 byte(s) leaked in 1 allocation(s).
```

Listing 1 Example: Leak detected by leak sanitizer

Compiler

A compiler not only transforms your code into another form, it also is crucial for software quality, performance and security.

Best practices:

- enable warnings (e.g. `-Wall`)
- enable security mechanisms (e.g. Stack protection, Relocation Read-Only (RELRO))
- test with optimization enabled

The performance of generated code can vary between compilers. Comparing compilers can be helpful (e.g. for C++: `gcc`, `clang` and `icc` (not open-source :( ))

Warning: Sometimes bugs are caused by the compiler!

Fuzzing

Traditional testing requires an oracle. But it is always possible to test for crashes ⇒ Fuzzing

Fuzzing generates (semi-)random input and tries to crash the application

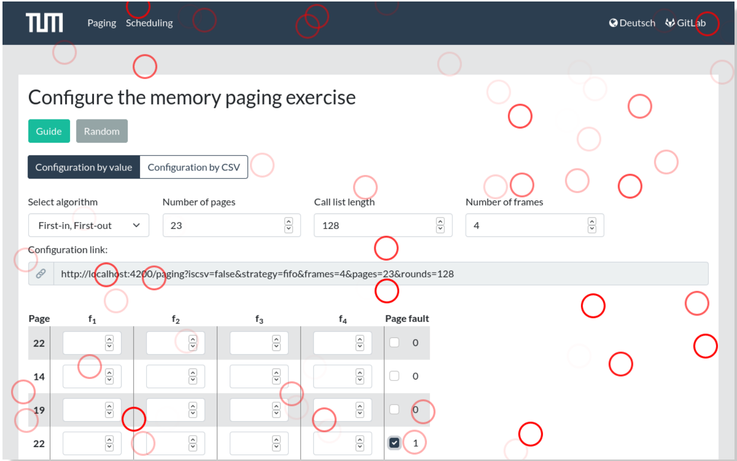
OpenSSL Heartbleed (CVE-2014-0160) bug could have been found by using fuzzing

Google's OSSFuzz initiative for open source projects https://github.com/google/oss-fuzz

Popular fuzzer: AFL fuzzer https://github.com/google/AFL

Some languages, like Go (v1.18+), provide native support for fuzzing.

UI Fuzzing - Monkey Testing



Configure the memory paging exercise

Select algorithm: First-in, First-out

Number of pages: 23

Call list length: 128

Number of frames: 4

Configuration link:

http://localhost:4200/paging?iscsv=false&strategy=fifo&frames=4&pages=23&rounds=128

Page	f_1	f_2	f_3	f_4	Page fault
22					☐ 0
14					☐ 0
19					☐ 0
22					☑ 1

https://github.com/marmelab/gremlins.js/

Formal verification

Unfortunately, still mostly used for academic purposes, but rises in popularity as algorithms become more complex. Very powerful.

ESBMC model checker https://github.com/esbmc/esbmc

Isabelle/HOL Verifies Standard ML programs https://isabelle.in.tum.de/

ivy Proofs protocols, such as leader election, raft, etc. https://github.com/kenmcmil/ivy

If you come up with a completely new, fundamental algorithm, you should definitely consider this.

Logging / Tracing

- Important for debugging and auditing
- Store important events with timestamp
- Use consistent format
- Consider performance and storage overhead
- Libraries can help

```
[ debug ] [thread 172236] PubMed request done.
[ info ] [thread 172236] PubMed: Requested 50 out of 56 abstracts.
[ info ] [thread 172236] PubMed: Parsing abstracts.
[warning ] [thread 172236] PubMed: No Abstract node found for: 30460988.
[warning ] [thread 172236] PubMed: No Abstract node found for: 27389065.
[warning ] [thread 172236] PubMed: No Abstract node found for: 25075404.
[ info ] [thread 172236] PubMed: Parsing abstracts done.
[ debug ] [thread 172236] PubMed request done.
[ info ] [thread 172236] PubMed: Requested 100 out of 56 abstracts.
[ info ] [thread 172236] PubMed: Parsing abstracts.
[warning ] [thread 172236] PubMed: No Abstract node found for: 9306861.
[ info ] [thread 172236] PubMed: Parsing abstracts done.
[ info ] [thread 172236] PubMed: Requesting 56 full text links...
```

Listing 2 Log example

Meaningful Crashes

Not helpful:

```
Segmentation fault (core dumped)
```

Better:

```
Stack trace (most recent call last):
#12 ...

Segmentation fault (core dumped)
```

Example:

- https://github.com/bombela/backward-cpp

Performance matters

Many people don't care about performance too much until it gets a problem. DON'T be one of them.

Slow programs are annoying, expensive and are contributing to climate change.

Always ask yourself, whether the algorithm or data-structure you are using is appropriate.

Example:
We once had a program doing a lot of computations on intervals. At the beginning, these were organized in an array. Later the implementation was changed to an interval tree which improved performance by an order of magnitude.

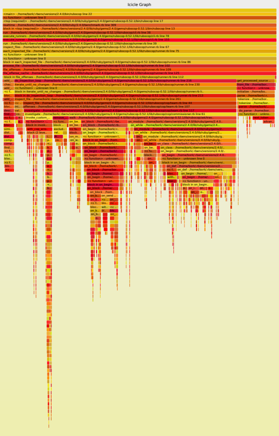
Performance measurements / Profiling

Easiest way: `time` or `hyperfine` (https://github.com/sharkdp/hyperfine)

Use profilers, often they are integrated in IDEs, making usage extremely simple.

In complex scenarios, use `perf` or `vTune` (not open source, but still really nice), to find the performance bottlenecks.

Make sure you compiled your program with enabled optimizations!



Example output of profiler rbspy[1]

[1] https://github.com/rbspy/rbspy.github.io/blob/main/src/static/images/rubocop-flamegraph.svg

Package/Dependency management

Use package managers:

C++	conan, CPM
Ruby	RubyGems
Javascript/Typescript	npm
Java	maven, gradle
Rust	cargo
Fortran	fpm
Python	pip

Some languages provide native support for package management, e.g. Go.

Keep your dependencies up to date (Use tools, such as dependabot (covered later))

Use in conjunction with security scanners to detect vulnerable versions and license risks.

What should you use?

Always pick a combination of the described techniques and use the more advanced ones where applicable.

What should you use?

Always pick a combination of the described techniques and use the more advanced ones where applicable.

Want to learn more about testing?

Visit Alexander Pretschner’s course on Advanced Topics of Software Testing:
https://campus.tum.de/tumonline/ee/ui/ca2/app/desktop/#/slc.tm.cp/student/courses/950602762

What should you use?

Always pick a combination of the described techniques and use the more advanced ones where applicable.

Want to learn more about testing?

Visit Alexander Pretschner's course on Advanced Topics of Software Testing:
https://campus.tum.de/tumonline/ee/ui/ca2/app/desktop/#/slc.tm.cp/student/courses/950602762

Concluding Remarks

Don't believe that important and famous projects are always following these rules. Many don't. Always have a look at the build scripts to see what is going on in a project. Adding some of these techniques can reveal bugs, you would have never deemed possible in a well known project.

Next step: Automation!