Robert Schumacher

Microsoft

Don't package your libraries, write packageable libraries!

Foreword

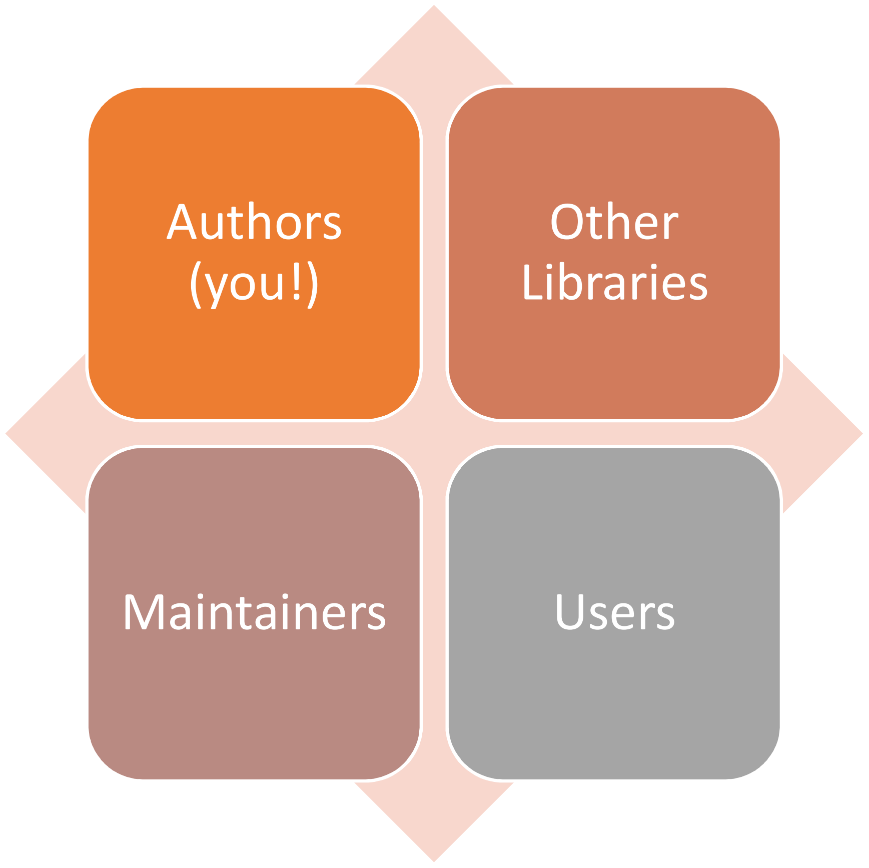
- This talk will be presented entirely from a packaging and maintainer viewpoint
- Every point presented has other tradeoffs that need to be evaluated for your project
- My objective is that you will be better engineers by being aware of the packaging implications of your design decisions

Foreword

- This talk will include real code from real projects
- This is not intended to shame – All of the projects mentioned are awesome and make all our lives easier.
- A handful of choices that make *packaging* harder can still be the right ones for the project to make.

- I sincerely thank all authors for their endless hours of effort to make these projects possible and hope they continue into the future!

Four Facets of the Packaging Ecosystem



Maintainers

- These are the volunteers which will package your library for you

- They are not contributors to your project
- They do not even use your project
- They do not know your dev workflows
- They do not know your code
- They are human and will make mistakes

Buildsystems

Use a popular buildsystem

- Today, that’s CMake, MSBuild, or Autoconf
- As a maintainer, I can handle your buildsystem being bad in the same ways that everyone else is bad.
    - I *can't* handle your buildsystem being bad in its own new and clever ways.

The reward for using a buildsystem before it's mainstream is to get packaged poorly or not at all.

Use a popular buildsystem (cont.)

- Boost: Boost.Build
- OpenSSL: 5.3k+ lines of Perl
  - https://github.com/openssl/openssl/blob/OpenSSL_1_0_2p/Configure
- FFmpeg: 7.4k+ lines of Bash
  - https://github.com/FFmpeg/FFmpeg/blob/efb65abedf40c0a5bc6eb76e6cf19b633a143444/configure
- Qt: Qmake

You discover that one of the above doesn’t build if there are spaces in the path.
Good Luck!

Header-only is not a Packaging Panacea

- "What's hard about copying a few headers around?"
  - How will you generate config.cmake/.pc files?
  - How will I run your tests?
- You lose the ability to encapsulate
- You are forcing "static linkage" for consumers
- It's extremely easy to end up with circular dependencies between header-only libraries
- In the wild:
  - Boost v1.68 has *multiple* circular dependencies between header-only components
    - https://pdimov.github.io/boostdep-report/
    - Shout out to pdimov for his work on analyzing the boost internal dependencies!

Use your buildsystem’s standard constructs

- CMake:
  - `BUILD_SHARED_LIBS`
  - `find_package()`
  - `find_dependency()`
  - Use targets instead of macros!

- In the wild:
  - Libpng v1.6.35 uses `PNG_SHARED` and `PNG_STATIC`
    - https://github.com/glennrp/libpng/blob/v1.6.35/CMakeLists.txt#L70
  - Expat v2.2.6 uses `BUILD_shared`
    - https://github.com/libexpat/libexpat/blob/R_2_2_6/expat/CMakeLists.txt#L34

Do not use header-checks

- This is highly susceptible to contamination
  - Example:
    - You see `openssl.h` and assume OpenSSL 1.1 is available
    - In fact, it was provided by libressl
    - You try to use some openssl-specific thing and fail to build
    - I didn’t even want you to build with SSL anyway! ☹
- Partial support becomes poisonous
  - Nobody can provide a partial `unistd.h` or `sys/mmap.h` because your build will suddenly believe it’s on Linux instead of Windows and fail

Do not use header-checks (cont.)

- Libxml2 v2.9.4 checks for `dirent.h`, `finite()`, `fpclass()`, `fp_class()`, `isnand()` then does nothing with them
  - https://github.com/GNOME/libxml2/blob/bdec2183f34b37ee89ae1d330c6ad2bb4d76605f/configure.ac#L561
  - Fixed in latest master 👏

- Exiv2 2018-09-18 checks for `mmap()` and, if found, uses it on Windows and fails
  - https://github.com/Exiv2/exiv2/blob/f06b69fa56051204ef3b8f1379ed2a0346672d22/src/basicio.cpp#L436

Do not “detect” different settings

- Give the builder options to require that a setting is ON or OFF
- Respect those settings – don’t undermine them
  - CMake note: every use of `find_package()` without `REQUIRED` is suspect

- In the wild:
  - Cpprestsdk v2.10.6 uses an old, embedded copy of websocketspp if `find_package()` quietly fails
    - https://github.com/Microsoft/cpprestsdk/blob/v2.10.6/Release/cmake/cpprest_find_websocketpp.cmake#L6-L13

Remember: The maintainer will make mistakes! Don’t leave spikes in pits ☺

Don’t depend on having a host compiler

- Most common offender: building code generators during the build
- If you must depend on the host, have separate build stages that can be packaged independently and imported

- In the wild:
  - See previous custom buildsystems – several are built during their parent builds
  - Protobuf v3.6.1 does this perfectly with `--with-protoc=` ✨
    - https://github.com/protocolbuffers/protobuf/blob/v3.6.1/src/README.md

Don’t hardcode a single package manager

- This includes the `ThirdParty/` “package manager” ☺
- All package managers need to intercept your dependencies to handle ODR, updates, security, consistency, and more
- Hardcoding any particular package manager makes this *much* more difficult

Do not use runtime plugins

- They always imply custom build and deployment logic *in all consuming projects* that is extremely difficult and fragile

- In the wild:
  - Qt5 (when dynamic linking) has many plugin folders which need to be handled with extreme special casing (see: `windeployqt`)
    - https://github.com/qt/qttools/blob/5.11/src/windeployqt/main.cpp#L799-L828

Don’t muck with compiler flags

- CMake: `CMAKE_CXX_FLAGS`
- Add an escape hatch if you must

- In the wild:
  - Caffe2 v0.8.1 forces `/MT` instead of `/MD` when building static libs on MSVC
    - https://github.com/caffe2/caffe2/blob/v0.8.1/CMakeLists.txt#L96-L105
    - Fixed in master! 👌

Dependencies

Don’t have them

Hide dependencies

Don’t include `Windows.h` in your public headers

Too much macro configuration that is impossible to encapsulate (WIN32_LEAN_AND_MEAN, WINSOCK, etc)

In the wild:

OpenSSL v1.0.x pulled in `windows.h` inside `rand.h`

- https://github.com/openssl/openssl/blob/OpenSSL_1_0_2p/crypto/rand/rand.h
- Fixed in v1.1 ☃

Do not have optional dependencies

- Provide optional features as separate libraries with separate builds

- The user doesn’t always know what they want; rebuilding everything every time they change their mind is awful

Do not have optional dependencies (cont)

OpenCV v3.4.1 optionally depends on pretty much every graphics format library written before 2013

https://github.com/Microsoft/vcpkg/blob/7881abfc29c916330e868118b29606cb32c51b16/ports/opencv/CONTROL#L1-L78

FFMpeg says "Hold my beer"

https://github.com/FFmpeg/FFmpeg/blob/release/4.0/configure#L203

Adopt newer dependencies quickly

- If you use Boost, you should be on 1.68 right now

- If your dependency releases once a year
  - You take 6mo to adopt it
    - Someone else takes 3mo to adopt you
      - Someone else 1.5mo to adopt that
        - Someone else …
- Users can use N-1 just barely before N is released.

We must move faster.

ABI, API & Support

Don’t replace APIs when using C++17 vs 14

- For all major compilers and STLs, newer standards are additive and compatible
- We do this to make it easy for users to upgrade their code incrementally
- This means that you need one binary that can serve *both* versions

Note: only applies if you support both C++14 and C++17.

Don’t replace APIs when using C++17 vs 14

- Most commonly, this is broken by optional polyfills.

- Example:
  - Abseil provides a custom `absl::string_view` or typedefs `std::string_view` based on the c++ standards level
    - https://github.com/abseil/abseil-cpp/blob/e01d95528ea2137a4a27a88d1f57c6cb260aafed/absl/strings/string_view.h#L33-L41
    - https://github.com/abseil/abseil-cpp/blob/2a62fbdedf64673f7c858bc6487bd15bcd2ca180/absl/base/config.h#L354-L365

Don’t worry about ABI; worry about API

- Avoid breaking source compatibility and provide migration guidance if you do
  - Ideally in a way that can be applied by someone who isn’t familiar with either codebase

- If you must maintain ABI compatibility
  - Your public binary interface should be C
  - You should use caller-allocated memory
  - You should not expose any dependencies in your interface

Have a clear, single supported source version

- If you must support multiple versions, make them simultaneously consumable
  - Separate headers, separate symbols

- Multiple supported versions make it harder for others to depend on you
  - If X, Y, and Z are “supported” for customers to be using today, which should I test with? Will every downstream library make that same conclusion?

Have a clear, single supported source version

- Qt has multiple supported versions in flight
  - And has the macros to (theoretically) avoid ODR violations!
  - https://github.com/qt/qtbase/blob/c5307203f5c0b0e588cc93e70764c090dd4c2ce0/src/corelib/global/qglobal.h#L164-L188
- OpenSSL is simultaneously supporting v1.1.x and v1.0.x
  - Changed the library names but not the header names

Use a standard license

- When you use a standard license, it is infinitely easier for everyone downstream to know exactly what to do

- Make sure that license is readily accessible in a dedicated file

- In the wild:
  - Sqlite3 is in the "public domain" (Surprisingly, not well defined)
    - No dedicated license file!
    - https://sqlite.org/copyright.html

Code

Break source, not behavior

- v1: `void throws_on_error()`
- v2: `std::error_code throws_on_error() noexcept`
- Users: 😢
- Nobody tracks test code coverage *of their dependencies*

Namespace *all the things*

- Symbols with `namespace`
- Headers with `foo/`

- In the wild:
  - Libjpeg-turbo v1.5.3 exports `/jconfig.h`
  - Nlohmann-json v3.0.1 exported `/json.h`
    - https://github.com/nlohmann/json/blob/v3.0.1/README.md
    - Fixed in v3.1.0 🎂

Don’t be hostile to the compiler

- \`#if _MSC_VER > 1900\` + \`#error\`
- \`#if !$COMPILER\` + \`#error\`
- \`#if _MSC_VER > 1900\` + \`#message SPAM\`

- In the wild:
  - PMDK v1.4.2 hardcoded a failure for \`_MSC_VER\` > 1911
    - https://github.com/pmem/pmdk/blob/1.4.2/src/common/util.h#L287-L291
    - Fixed in master! (at least, lifted to >=2000) 🚀

Patterns

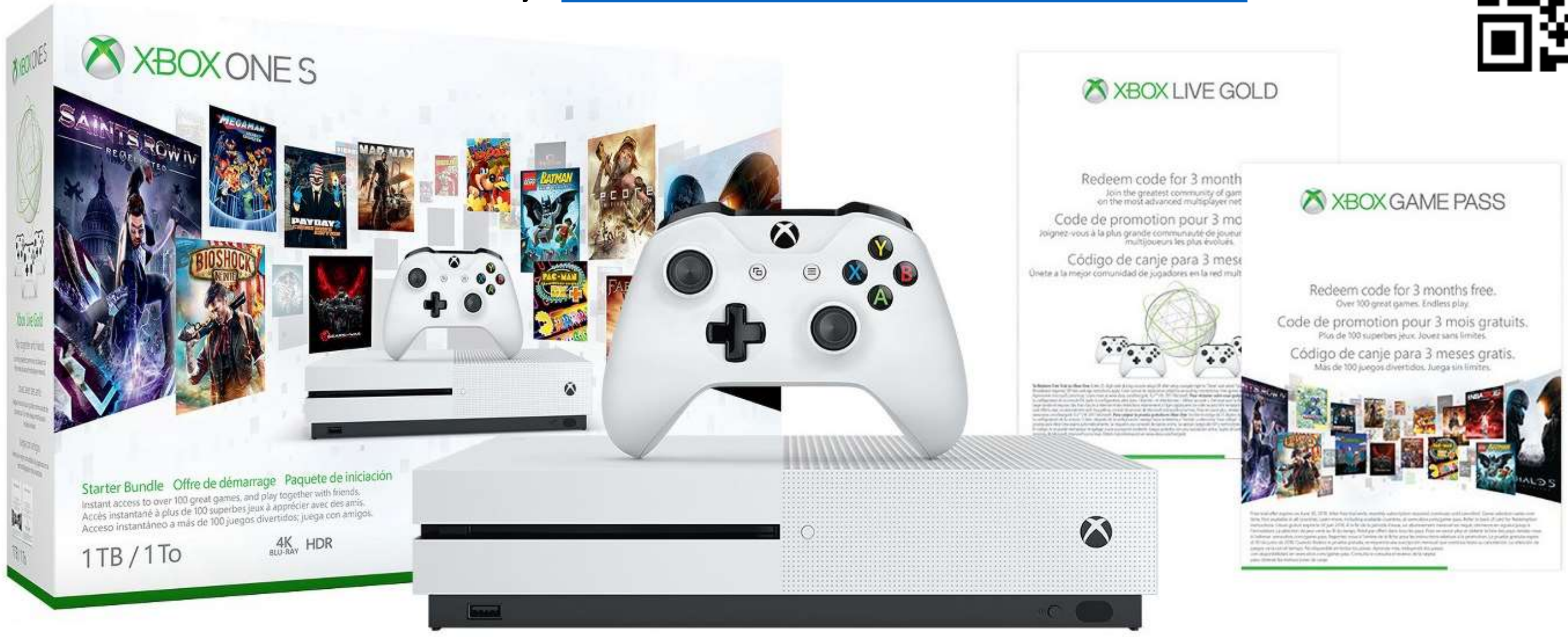
Patterns

- Maintainers will make mistakes
- Packaging is more than your library + your user
- Many effects are hard to see locally

Shameless advertisement: vcpkg!

- All examples in this talk were taken from vcpkg’s 750+ catalog of libraries for Mac, Windows, and Linux
- Our community deals with these problems so users don’t need to ☺
- Help us out by sending a PR!
- https://github.com/Microsoft/vcpkg

Take our survey https://aka.ms/cppcon


XBOX ONE S
Starter Bundle Offre de démarrage Paquete de iniciación
1TB / 1To
4K BLU-RAY HDR
XBOX LIVE GOLD
XBOX GAME PASS
Redeem code for 3 months free.
Code de promotion pour 3 mois gratuits.
Código de canje para 3 meses gratis.

You can win an Xbox One S - Starter Bundle

9/27 10:30 – 12:00 // *Breckenridge Hall*
Thoughts on a More Powerful and Simpler C++ (5 of N), ***Herb Sutter***

Other sessions

~~Monday, September 24th~~

- ~~14:00 – 15:00~~
  ~~How to Write Well-Behaved Value Wrappers~~
  - ~~by Simon Brand~~
- ~~15:15 – 16:15~~
  ~~How C++ Debuggers Work~~
  - ~~by Simon Brand~~

~~Tuesday, September 25th~~

- ~~14:00 – 15:00~~
  ~~What Could Possibly Go Wrong?: A Tale of Expectations and Exceptions~~
  - ~~by Simon Brand and Phil Nash~~
- ~~15:15 – 15:45~~
  ~~Overloading: The Bane of All Higher-Order Functions~~
  - ~~by Simon Brand~~

~~Wednesday, September 26th~~

- ~~12:30 – 13:30~~
  ~~C++ Community Building Birds of a Feather~~
  - ~~with Stephan T. Lavavej and others~~
- ~~14:00 – 15:00~~
  ~~Latest and Greatest in the Visual Studio Family for C++ Developers 2018~~
  - ~~by Marian Luparu and Steve Carroll~~
- ~~15:15 – 15:45~~
  ~~Don't Package Your Libraries, Write Packagable Libraries!~~
  - ~~by Robert Schumacher~~

Wednesday, September 26th

- ~~15:15 – 15:45~~
  ~~What's new in Visual Studio Code for C++ Development~~
  - ~~by Rong Lu~~
- 15:50 – 16:20
  Value Semantics: Fast, Safe, and Correct by Default
  - by Nicole Mazzuca
- 16:45 – 17:45
  Memory Latency Troubles You? Nano-coroutines to the Rescue! (Using Coroutines TS, of Course)
  - by Gor Nishanov
- 18:45 – 20:00
  Cross-Platform C++ Development is Challenging – let Tools Help!
  - by Marc Goodner and Will Buik

Thursday, September 27th

- 9:00 – 10:00
  Inside Visual C++'s Parallel Algorithms
  - by Billy O'Neal
- 15:15 – 15:45
  ConcurrencyCheck – Static Analyzer for Concurrency Issues in Modern C++
  - by Anna Gringauze
- 16:45 – 17:45
  Class Template Argument Deduction for Everyone
  - by Stephan T. Lavavej

Don't assume your user's link model

- Don't use `LoadLibrary`/`dlopen`
- Don't use `DllMain`
- Glib on Windows uses `DllMain()` which prevents static linking :'(
    - https://github.com/GNOME/glib/blob/2.58.1/glib/glib-init.c#L273-L294
- Don’t use mutable globals

Default to portability

- If you support SSE2 and AVX, default to SSE2

- This is because portability is *extremely* hard to test
  - The maintainer will not see your flag because everything seems to run fine
  - The user who's hunting for every perf gain possible *will* find your flag

- In the wild:
  - Rocksdb v5.14 defaults to `-march=native` / `/arch:AVX2`
    - https://github.com/facebook/rocksdb/blob/5.14.fb/CMakeLists.txt#L194-L210

Don't make me build things I don’t need

- Samples, docs, alternate flavors, tools
- A crowd favorite is to combine this with dep autodetection
  - "If I can find doxygen, then generate my docs by default"
  - This translates to “The build runs 200% slower on that machine that appears identical”
- In the wild:
  - Fast-RTPS v1.6.0 builds and installs examples with no easy OFF
    - https://github.com/eProsima/Fast-RTPS/blob/v1.6.0/CMakeLists.txt

Don't build with `-Werror`/`/WX` by default

- Your user will use a newer compiler. Your code will break. Your user will be sad. 😢
- Note: Definitely turn it on for your CI and dev work!

- In the wild:
  - Folly 2018.09.24 throws `/Werror` unconditionally on unix
    - https://github.com/facebook/folly/blob/v2018.09.24.00/CMake/FollyCompilerUnix.cmake#L19

Golden rule(s) of source layout:

- Lay out your sources like you don't have a buildsystem
- Lay out your sources like you have 10 buildsystems

Ship often

- It is the package manager’s job to help the user manage change
- Shipping more gives users more choice: half now, half later, instead of everything later.
- Minimizing delta between “last stable” and “master” enables:
  - Backporting changes
  - Useful bug reports
  - Fewer fixed-in-master-but-unavailable bugs

Have clear public headers