Conserving Legacy Code

From handwritten Makefile to modern build systems and activatable archivation

deRSE25 27.02.2025

by J. Ph. Thiele, J. Fuhrmann

TeSCA was highly innovative

- In terms of mathematics, physics and industrial application:
  - New PDE theory and numerical mathematics for semiconductor devices
  - Leading tool in ‚the East' and even worldwide (only realized 1992, after reunification)
  - Still relevant today with semiconductors being ubiquitous and even needed in quantum devices
- In terms of usability and technology (in the ‚feature complete' version of 1996)
  - Efficient Fortran code capable of 3D simulations
  - Dedicated scripting language for simulation control
  - Onboard graphics based on X11 → Steering of output through scripting
  - Validated against physical experiments → High trust in results by user community

Development Timeline

- 1983: beginning of project
- 1995/96: Last major changes with only smaller revisions and bugfixes afterwards
- 2005: Collection of examples (script files etc.) from users
- 2014: Retirement of last main developer, continued interest → collaborative effort:
  - Relocation of source code to mercurial for shared version control (now GitLab)
  - Introduction of CMake and CTest with small integration tests
  - New revision of the user manual
- 2018: Release of GCC 8 → Incompatibilities lead to floating point errors
- 2023/24: Renewed interest due to new developments in Julia WIAS-PDELib (Poster by P. Jaap)
  - Smaller revisions to CMake
  - Using Docker to build an executable that runs on modern systems

The challenging parts of legacy software (technical)

- Version control wasn’t a thing → folders, tar files and mails where used
- Compilation was initially done with ‘*gfortran *.F*’, i.e. not even a Makefile
- No coding standards like:
  - Naming conventions
  - Frequent useful comments
  - Consistent formatting
- Unit testing wasn’t a thing, the first xUnit framework SUnit was developed in 89
- No software design

The challenging parts of legacy software (people)

- Institutional knowledge is gone
  - No one understands the full code base
  - No one knows what challenges were faced during development
  - Sometimes hardly anyone speaks the used programming language
- There is almost no (time) budget, but it would be needed for
  - Understanding the code base
  - Understanding all of the numerical methodology
  - Writing (unit) tests to safeguard further development
  - Refactoring/Redesigning the code

Why CMake?

- Check for third party libraries and subpackages like X11, Xau, etc.
- Better handling of linking libraries and targets
- Modularity: CMakeLists can include one another (e.g. in subfolders of src)
- Out of source builds
- Common options can be included, e.g. for specific machines
- CTest allows for easier setup of a test suite

Docker/Podman containers

- Allows running of different (linux) operating systems in containers
- Easy to test if things run in a `barebones' setting without your local installation
- `document' all the steps to get a software running on a new system
- One basis for GitHub actions / GitLab CI → similar to write
- Multi stage files are possible → Easy way to remove temporary build files
- In our case: Use Ubuntu 20.04 which still has gcc-7 in its repositories
  - ! Word of caution: as long as they are still available

Docker: Our steps with TeSCA

- Get TeSCA to build and tests to run in container → exec runs in Docker, but no graphics output
- Figure out how to connect X window with docker to run larger examples as a test → exec with graphics
- Try building a statically linked executable
  - Dynamic: External libraries are `somewhere' on the machine and can be shared by multiple programs
    → user has to install external libraries like X11 and the gfortran7 library
  - Static: **All** code needed for the execution is inside the executable
    → user can directly run the executable (on Linux)
  - → Fully static did not work because of X11: depends on **too many libraries,** very large exec, graphics
- TeSCA without X11? Failure: Too strongly coupled and plotting would be too different for users
- Final alternative: statically link GCC and dynamically link everything else
  - User has to install X11-devel (from package manager, so easy to do)
  - → Portable executable runs on different linux machines (and WSL) after install of X11

Dockerfile (Setup of container)

-

```
#Base image is Ubuntu 20.04
FROM ubuntu:20.04 as build

USER root

#Install prerequisite software for building TeSCA
RUN apt-get update &&\
    apt-get install -y gcc-7 gfortran-7 make cmake libx11-dev numdiff &&\
    ln -s /usr/bin/gfortran-7 /usr/bin/gfortran

RUN mkdir /tmp/tesca-src

ADD . /tmp/tesca-src
```

Dockerfile (Building and executable container)

-

```
RUN cmake -S/tmp/tesca-src -B /tmp/tesca-build\
    -C/tmp/tesca-src/cmake/linux-gfortran64-opt.cmake\
    -DCMAKE_MAKE_PROGRAM=make &&\
    cmake --build /tmp/tesca-build &&\
    cmake --build /tmp/tesca-build --target test

FROM ubuntu:20.04
COPY --from=build /lib/x86_64-linux-gnu /lib/x86_64-linux-gnu
COPY --from=build /tmp/tesca-build/tesca-dynamic /bin/tesca
COPY --from=build /tmp/tesca-src/examples /root/tesca-examples

WORKDIR /root
```

Dockerfile (Building and only executable)

-

```
RUN cmake -S/tmp/tesca-src -B /tmp/tesca-build\
    -C/tmp/tesca-src/cmake/linux-gfortran64-opt.cmake\
    -DCMAKE_MAKE_PROGRAM=make -DTESCA_STATIC_GCC=ON&&\
    cmake --build /tmp/tesca-build --parallel &&\
    cmake --build /tmp/tesca-build --target test

FROM scratch
COPY --from=build /tmp/tesca-build/tesca-dynamic /tesca
```

Conclusions

- A distributed version control system is great
- CMake makes adding options much easier (e.g. TESCA_STATIC_GCC) and is better to maintain
- Switching to CMake is much easier when a main developer is still around!
  - Keeping track of software and developers is important
- Docker is a good tool, when aware of the caveats like long term availability
  - Do my installation steps work on different machines?
  - In this case: no need to bui;d GCC 7 by hand
- The flags `*-static-libgcc -static-libgfortran*' are great when you require an `ancient' version of GCC
- Users were able to use the executable and get expected results
- Will be a good basis for `model model comparisons' and gaining trust in the newer Julia developments