Sustainable Software Practices in Developing MATLAB

Pat Quillen
Engineering Manager, MATLAB Math and PDE

Collegeville Workshop on Sustainable Scientific Software, 24 July 2019

MathWorks as Producer and Consumer of Scientific Software

- MATLAB is a technical computing environment providing scientific software to our customers and also a platform for development of scientific software

- Many MathWorks products also sit on top of third-party scientific software libraries
  - BLAS, LAPACK, FFTW, UMFPACK, HSL_MA57, … and many more

I'll talk today about ways we've addressed sustainability challenges in developing and in using scientific software

Disclaimer: Following are my opinions and do not necessarily reflect those of MathWorks

Sustainability is our Business

- MathWorks around since 1984
  - MATLAB, even longer---late 1970s
  - ~5000 employees, ~55--60% are in Development
- Not everything is MATLAB…
  - MathWorks makes *a lot* of products---around 123 in total
  - Almost all of them require MATLAB
- Many customers have suites of MATLAB code/Simulink models
  - We need good software practices to build software serving internal and external customers alike

Company Decisions that have helped Sustainability

- Conversion to C
  - First MATLAB that shipped was PC-MATLAB implemented in C
- The Quality Initiative
  - Natural expression of Core Values, particularly Continuous Improvement
- Foundational Expression of QI: Fixed Time between releases
  - Since 2004, MATLAB (and all products) have been released roughly every six months
  - We have a well-defined release cycle
    - Active Dev is 8 iterations ~ 26 weeks with Freeze milestones
    - End-game ~ 3 months

The Quality Initiative

- Phased approach to improving our software
  - Focus on defect reduction, tracking, reporting
  - Consider sources of bugs in all phases of software lifecycle
  - *With emphasis on process improvement to eliminate defects and their sources*
- Targeted metrics
  - Bugs/Dev
  - Categorized bugs ← emphasis on the most severe bugs
  - Compiler warning reduction
- Began in earnest roughly ***20 years in***

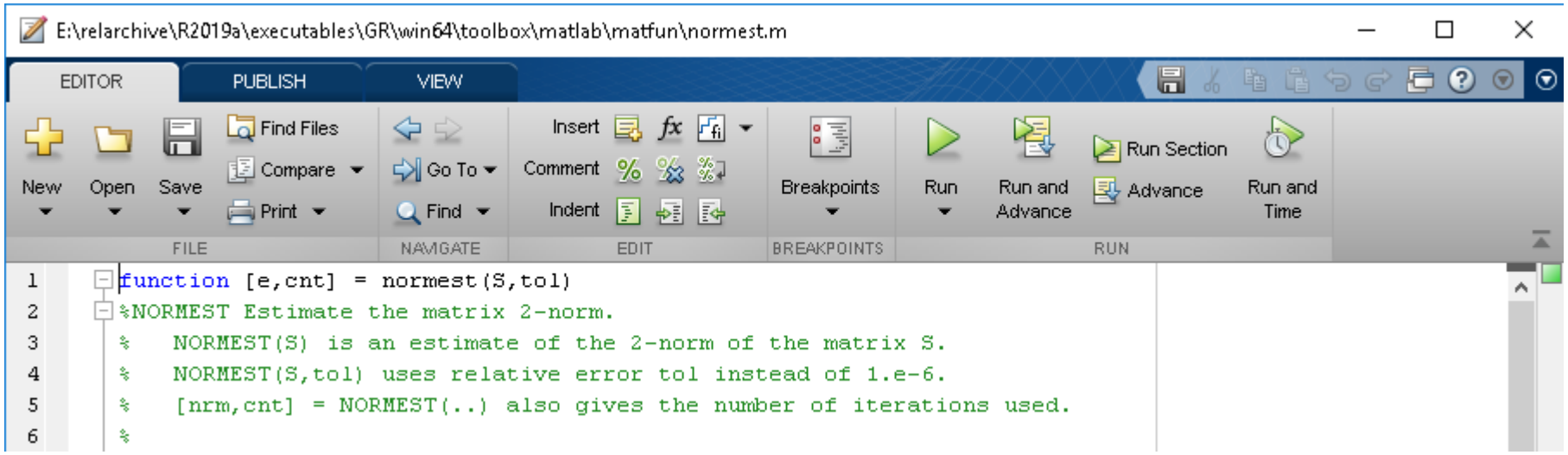
Testing and Integration

- Code organized into clusters in a system called BaT (Build and Test)
  - Clusters run tests charted to file changes for every submission; rejected if failures
  - Build and test code on all platforms (with some processor variants)
  - Continuous integration up the chain into main
- Testing occurs as part of build and as part of test
  - pkgtests and unittests built/executed with C++ modules
  - MATLAB Unit tests run during Test phase
  - Performance regression tests run 3 times weekly
- Company wide bashes are held
  - Defect reporting/enhancement requests rewarded!

Implementation strategies

- Define or adopt a coding standard and use tooling to enforce it
  - Minimally, compile at the highest warning level you can and at least consider lint
  - Lots of choices (C++ Core Guidelines, Google, C++ Coding Standards book)
  - Also, use standard libraries and stay off the bleeding edge

- MATLAB includes a Code Analyzer that automatically runs in the Editor



Light's green, code's clean

Refining Designs

- Bugs introduced at ***design time*** are the most expensive to eliminate

- Codified our software design process
  - Templates provided, revisited, and revised to make sure that they are still serving us
  - New hires are trained within the first 6—8 weeks
  - Local teams empowered to influence the process and templates
  - Codified how we do design review
  - Every customer visible MATLAB API goes through extensive design review

- Effect: Simpler designs with less surface area
  - True for both C++ and MATLAB APIs
  - Mileage varies across products

Managing Complexity and Adding Value with Layers of APIs

- Some APIs offered in MATLAB language (as .m)
- Which then use builtins from some shared library
- Which may depend on another shared library with almost all of the MathWorks-ish dependencies stripped out
  - Can be used by other products in a variety of ways
- Which then may depend on some 3rd party library

- At every level, expose only what is necessary

- Examples
  - `fft, decomposition, rand`

Speaking of third party libraries…

Questions we ask of Software we use in MATLAB

- Doc? Is it comprehensive and understandable?
- Test? Does any exist?
- Build? make, Cmake, autotools?
- Who do we run to with problems? Can we track resolution of our problems?
- Do you run on our supported platforms (Win, Linux, Mac)?
- How do you manage
  - Memory
  - Errors
  - Threads
  - Globals
  - And how much of that can we control and how?

Questions we ask of Software we use in MATLAB (continued)

- Under what circumstances are you reproducible?
- Numerically, what guarantees do you give?
- What about nonfinite/subnormal handling?
- How about recursion?
- What about messages you might want to dump to stdout? Can we suppress that easily?
- What are your dependencies and do we have conflicts?
- What about the license? Can we redistribute? Can we modify the code?

Before we go, a word about Backwards Compatibility

Keep the old code running---Backwards Compatibility

- Almost never throw anything away
  - Leverage "Discourage Use" process through doc and tooling
  - Keep around former code paths as undocumented package functions
    - Continue to test them, to some extent
  - Our big customers standardize on releases and don't move often (maybe every 3—6 years)

- Reproducibility
  - Guarantee run-to-run reproducibility under very strict circumstances
    - Same OS
    - Same MATLAB version
    - Same number of threads
    - Same inputs
  - Otherwise, it's a bug

Wrap Up

- Leverage build and test for continuous integration
- Track bugs, squash them, and report to your customers
- Adhere to standards, but stay away from the bleeding edge
- Spend time on design to design out bugs

- Observation: This wanted to be a talk about Technical Approaches but possibly turned out to be more cultural…

Thank you!