Building Portable Software: Finding a Middle Ground

William Gropp
wgropp.cs.illinois.edu

What Do I Mean By a “Middle Ground”?

- There are well-established software engineering practices that have been developed over the years for “scientific software”, e.g., including:
  - Library API design
  - Coding standards
  - Testing standards and coverage analysis
  - Documentation and training
  - Build systems
  - Delivery

Experimental

Site	Build Name	Update	Configure		Build		Test			Build Time
		Files	Error	Warn	Error	Warn	Not Run	Fail	Pass	
serrano-login6	maint-full-intel-cmake	0	0	3	0	50 +50	0	0	88	10 hours ago
serrano-login6	MJA_testUpdates-full-gnu-cmake	0	0	3	0	0	0	17	60	10 hours ago
serrano-login6	maint-full-gnu-cmake	0	0	3	0	0	0	0	88	10 hours ago
golubh2	maint-full-coverage-gnu-cmake-serial	0	0	2	0	0	0	0	88	10 hours ago
golubh2	maint-full-coverage-gnu-cmake	0	0	2	0	0	0	0	88	11 hours ago

Coverage

Site	Build Name	Percentage	LOC Tested	LOC Untested	Date
cab688	maint-full-coverage-gnu-cmake	28.48%	16166	40591	7 hours ago
cab688	maint-full-coverage-gnu-cmake	28.46%	16102	40481	11 hours ago
golubh2	maint-full-coverage-gnu-cmake	28.68%	17149	42654	11 hours ago
golubh2	maint-full-coverage-gnu-cmake-serial	28.24%	16900	42951	10 hours ago

- But very much aimed at “batch” oriented process of building applications
- There is an entire community that is building apps and tools using different methods and technologies
- Can we take advantage of the best of both worlds?

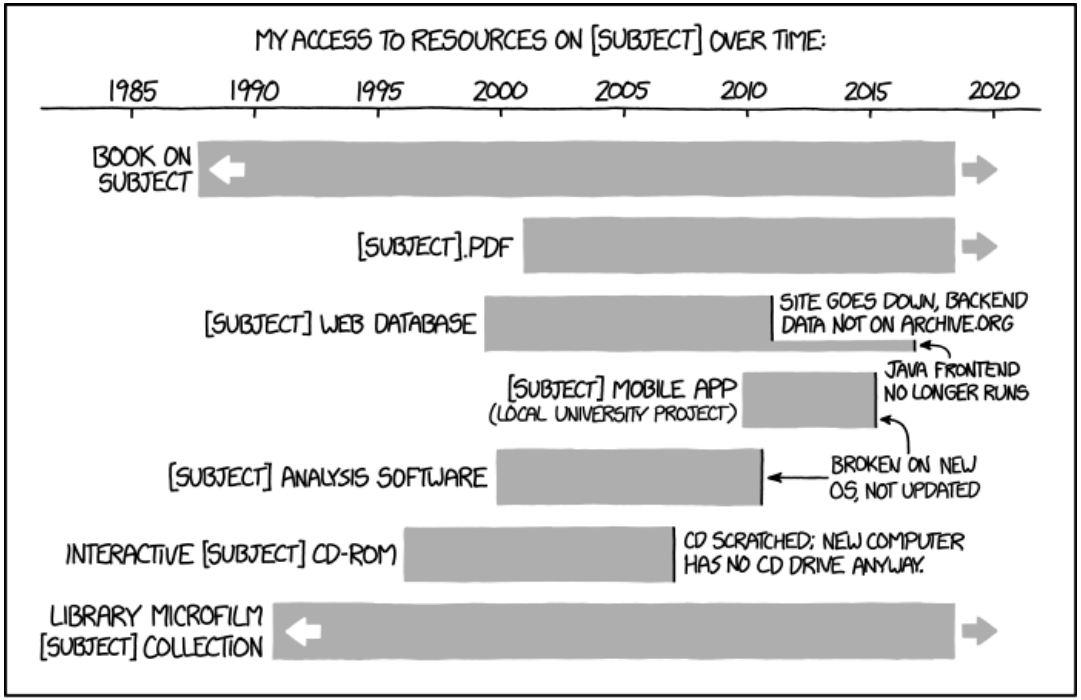
20th Century Approach

- Define a library API
  - Either by algorithm (early libraries) or function (PETSc, later frameworks)
- Careful adherence to language standards, use languages designed to provide performance (Fortran, C, C++)
- Build system
  - Imake, autoconf, cmake, …
  - If you think language wars are bad, try build systems
- User installs, uses binaries
  - Lucky user has admins install library (correctly if user is really lucky)
- Very successful and *still* the core of many applications
  - Same code likely to still run 10, 20, or even 30 years later
- But many challenges
  - Best for expert users

21st Century Approach

- (I won't do this justice)
- Define a rich framework that extends some existing system (which is already itself rich)
- Exploit flexible implementation strategies, including
  - Interpreted or JIT compilation; advanced analysis and compilation
  - Leverage other frameworks ("stand on the shoulders of giants")
  - Use package installers or containers or similar technology to simplify installation and use
  - Maintain an escape to call those old-fashioned libraries
- Exploit rapid evolution in languages and tools
  - But some things may not run next week. This is a feature

Portability in Time vs. Greatest Capability



Its not enough to have a great idea and implementation

- Powerful tools with zillion capabilities but poor or incomplete documentation can add as many problems as they solve. Well-meaning tutorials aren't enough
- A downside of a rich environment
  - What parameters/methods/interactions are available/relevant?
  - How do I make "this" change?
  - Should I even make this change (am I trying to impose the wrong model on the tool)?

Two Communities (?)

- Old hands (me)
  - Numerical libraries
  - Decades of backward compatibility
  - (Mostly) Batch, command-line tools
- New blood (many of you)
  - All manner of tools
  - Rapid change, follow innovations in other tools, systems
  - Interactive (at least as an option)
  - Graphical (or other productivity-oriented) interface

Can We Take the Best from Both Communities?

- Use new approaches to deliver software to a broader user community
- Leverage lessons on writing portable, *persistent*, high performance software
- More effective ways to compose and integrate tools into application workflows
- Documentation, training, and design
  - Tension between completeness and ease of use
- Best ways to get feedback from user community
  - You often get only one chance with new users

Two Criteria for Collaboration

- Bill's rules:
  1. Desperation. We're all excited about new things that we can do. But we can only find time for the ones that we're desperate to complete.
  2. Commitment to Outreach. Build it and they will come rarely works (just often enough to mislead). Particularly in HPC, strong outreach efforts are necessary, including
     - Documentation
     - Examples
     - Accepting and responding to bug reports
     - Meeting with users *on their own turf* and showing them your solution