Building Community through xSDK Software Policies

Piotr Luszczek, University of Tennessee

Ulrike Meier Yang, Lawrence Livermore National Laboratory

December 11, 2019

Who are we?

- Developers of **high-quality, robust, portable high-performance math libraries**
  - **Piotr Luszczek:**
    - Performance engineer and developer for multiple numerical libraries and benchmarks
    - Member of the xSDK4ECP project

  - **Ulrike Meier Yang**
    - More than 30 years of experience in numerical methods, parallel algorithms, scientific software development
    - PI of the xSDK4ECP project
    - software developer for parallel linear solvers library

Why are we leading this webinar?

- Tell the story behind the xSDK software policies
- Explain how the use of xSDK community policies **improves software quality, sustainability, and combined use of independent packages**, as needed for extreme-scale computational science and engineering
- Dive deeper into the actual policies; how they are defined
- Talk about their impact on software packages and application codes

Who are you?

- **Extreme-scale computational science community**
  - Developers of extreme-scale scientific applications
  - Developers of high-performance software packages and tools
  - Project leaders, stakeholders, program managers
  - Others

Learning objectives:

- Understand
  - Why are the software policies important
  - What does it take to develop them
  - What does it take to get code groups to accept and implement them
  - What are they about
  - Their impact

xSDK
ECP EXASCALE COMPUTING PROJECT

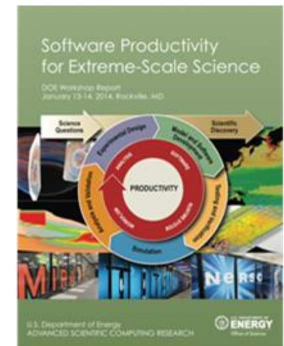
Webinar Outline:

- Introduction
- Motivation and History of xSDK and Community Policies
- Deep Dive: Community Software Policies
- Impact/Summary

xSDK history

- *Interoperable Design of Extreme-scale Application Software*
  - **First-of-a-kind project:** qualitatively new approach based on making productivity and sustainability the explicit and primary principles for guiding our decisions and efforts.
  - ASCR/BER partnership began in Sept 2014 (Program Managers: Bayer, Lesmes (BER), Ndousse-Fetter (ASCR))

- *Motivation:*
  - Enable ***increased scientific productivity,*** realizing the potential of extreme- scale computing, through ***a new interdisciplinary and agile approach to the scientific software ecosystem***.

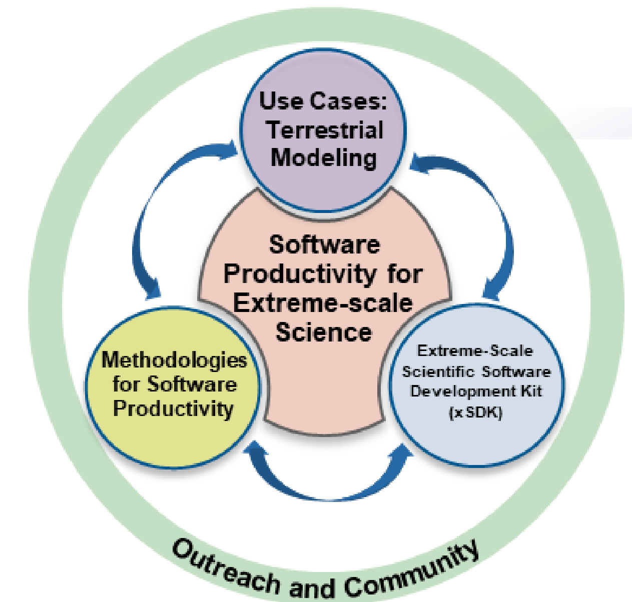


- *Objectives:*
  - Address confluence of trends in hardware and increasing demands for predictive multiscale, multiphysics simulations.
  - Respond to trend of continuous refactoring with efficient agile software engineering methodologies and improved software design

xSDK history

- ***Impact on Applications & Programs:***
  - Terrestrial ecosystem ***use cases tie IDEAS to modeling and simulation goals*** in two Science Focus Area (SFA) programs and both Next Generation Ecosystem Experiment (NGEE) programs in DOE Biologic and Environmental Research (BER).

- ***Approach***
  - **ASCR/BER partnership** ensures delivery of both crosscutting methodologies and metrics with impact on real application and programs.
  - **Interdisciplinary multi-lab team:** (ANL, LANL, LBNL, LLNL, ORNL, PNNL, SNL)
  - **ASCR Co-Leads:** Mike Heroux (SNL) and Lois Curfman McInnes (ANL), **BER Lead:** David Moulton (LANL)
  - ***Integration and synergistic advances in three communities*** deliver scientific productivity; outreach establishes a new holistic perspective for the broader scientific community.



xSDK EXASCALE COMPUTING PROJECT

Software libraries facilitate progress in computational science and engineering

- **Software library:** a high-quality, encapsulated, documented, tested, and multiuse software collection that provides functionality commonly needed by application developers
  - Organized for the purpose of being reused by independent (sub)programs
  - User needs to know only
    - Library interface (not internal details)
    - When and how to use library functionality appropriately

- **Key advantages** of software libraries
  - Contain complexity
  - Leverage library developer expertise
  - Reduce application coding effort
  - Encourage sharing of code, ease distribution of code

- **References:**
  - https://en.wikipedia.org/wiki/Library_(computing)
  - What are Interoperable Software Libraries? Introducing the xSDK

Why is reusable scientific software important?

User perspective:

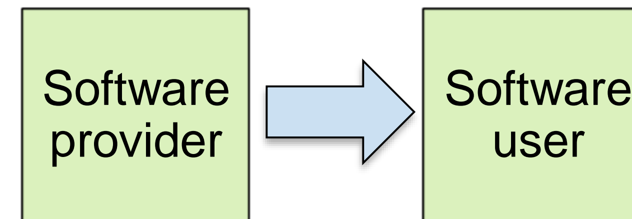
Focus on primary interests

- Reuse algorithms and data structures developed by experts
- Customize and extend to exploit application-specific knowledge
- Cope with complexity and changes over time

Provider perspective:

Share your capabilities

- Broader impact of your work
- Motivate new directions of research



- **More efficient, robust, reliable, sustainable software**
- **Improve developer productivity**
- **Better science**

xSDK EXASCALE COMPUTING PROJECT

Software libraries are not enough

- Well-designed libraries provide critical functionality … But alone are not sufficient to address all aspects of next-generation scientific simulation and analysis.
- Applications need to use software packages **in combination** on ever evolving architectures

> **"The way you get programmer productivity is by eliminating lines of code you have to write."**
>
> – Steve Jobs, Apple World Wide Developers Conference, Closing Keynote, 1997

xSDK
EXASCALE COMPUTING PROJECT

Need software <u>ecosystem</u> perspective

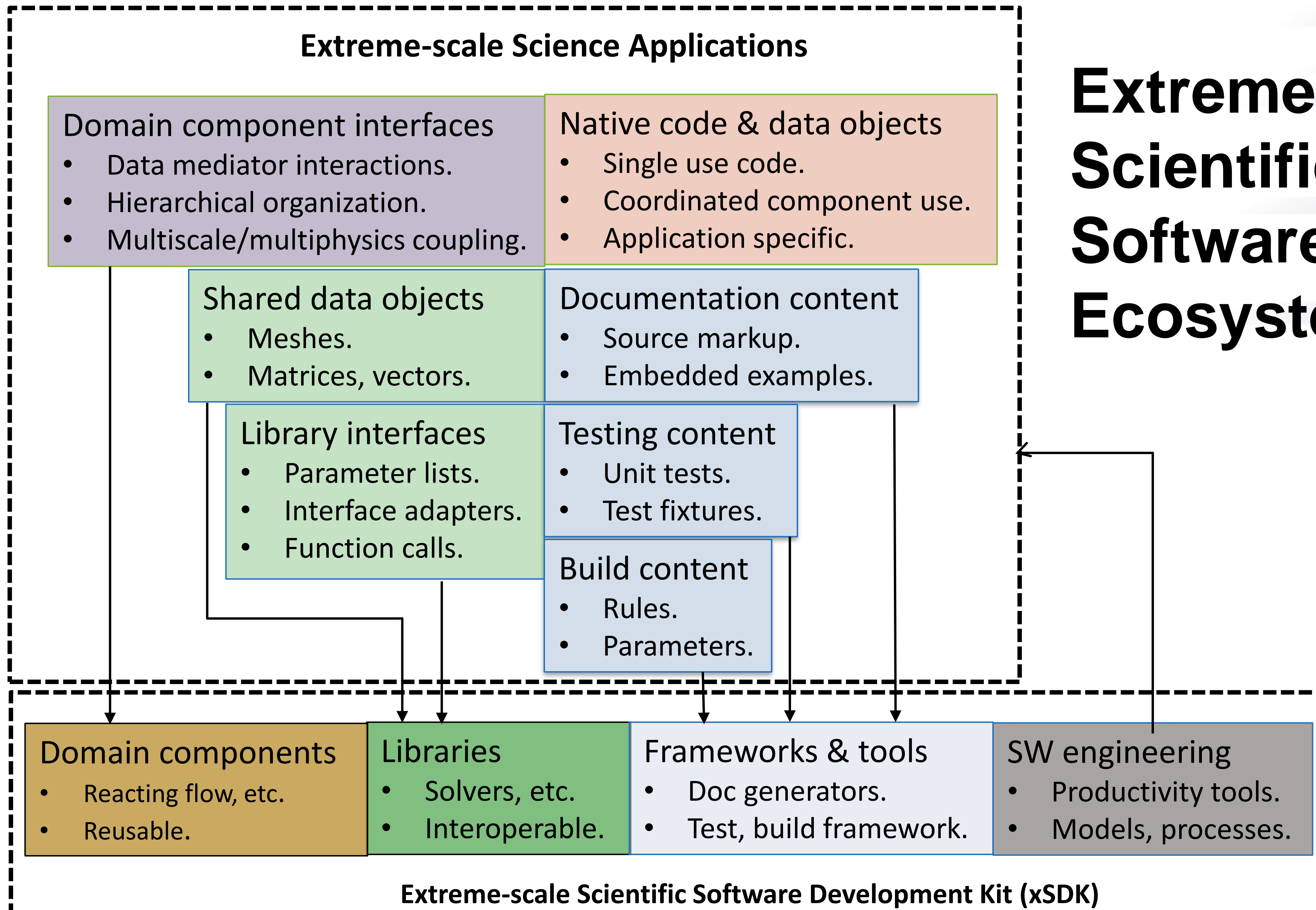
Ecosystem: A group of independent but interrelated elements comprising a unified whole

Ecosystems are challenging!

"We often think that when we have completed our study of one we know all about two, because 'two' is 'one and one.' We forget that we still have to make a study of 'and.' "

− Sir Arthur Stanley Eddington (1892−1944), British astrophysicist

Extreme-scale Scientific Software Ecosystem



Difficulties in combined use of independently developed software packages

Challenges:

- Obtaining, configuring, and installing multiple independent software packages is tedious and error prone.
  - Need consistency of compiler (+version, options), 3rd-party packages, etc.
- Namespace conflicts
- Incompatible versioning
- And even more challenges for deeper levels of interoperability

Levels of package interoperability:

- **Interoperability level 1**
  - Both packages can be used (side by side) in an application
- **Interoperability level 2**
  - The libraries can exchange data (or control data) with each other
- **Interoperability level 3**
  - Each library can call the other library to perform unique computations

Ref: What are Interoperable Software Libraries? Introducing the xSDK

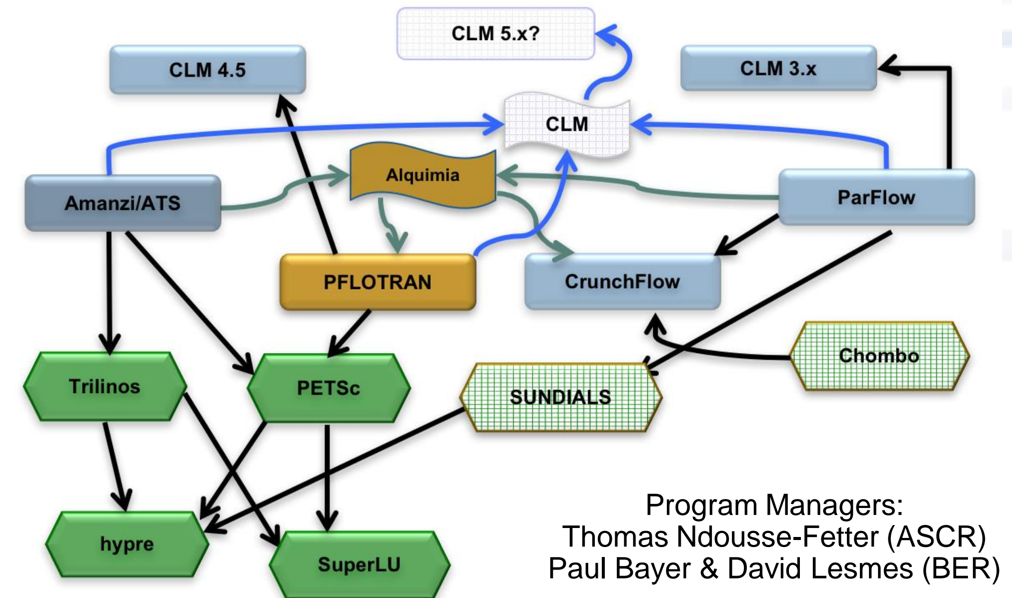
Motivation and history of xSDK

Next-generation scientific simulations require combined use of independent packages

- Installing multiple independent software packages is tedious and error prone
  - Need consistency of compiler (+version, options), 3rd-party packages, etc.
  - Namespace and version conflicts make simultaneous build/link of packages difficult
- Multilayer interoperability among packages requires careful design and sustainable coordination
- **Prior to xSDK effort, could not build required libraries into a single executable due to many incompatibilities**

xSDK history: Work began in ASCR/BER partnership, IDEAS project (Sept 2014)

Needed for BER multiscale, multiphysics integrated surface-subsurface hydrology models

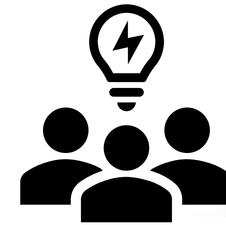


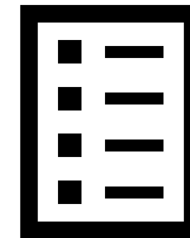
xSDK

EXASCALE COMPUTING PROJECT

How can you avoid the pitfalls mentioned?

- Ask application developers for feedback on most important issues
- Have brain storming sessions with experienced software developers to gather ideas
- Collect the input
- Formulate a set of rules that can help avoid these issues
- → xSDK standards
- What next?





How do you get buy-in from the software community?

- Communicate set of rules in various venues: meetings, papers, etc
- Use the right vocabulary!
  - Initially we suggested `*compliance to xSDK library standards*'
    - *Minimum compliance requirements*
    - *Recommended compliance*
  - New formulation:
    - `*Standards/ requirements*' replaced by `*community policies*'
    - `*Compliance'* replaced by `*compatibility*'
- Provide opportunity to give input

xSDK EXASCALE COMPUTING PROJECT

How do you make sure policies will stick around?

- Continue to ask for input from the community

We welcome feedback. What policies make sense for your software?

https://xsdk.info/policies

- Make updates as software practices are changing
  - Regular xSDK community policies releases
- Provide a process that allows change
  - Pull requests on github to change or add policies

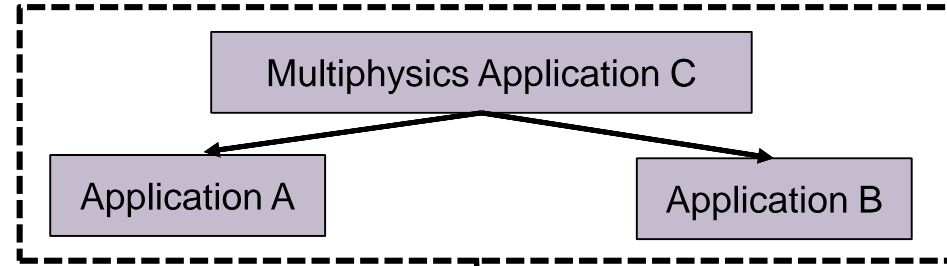
https://github.com/xsdk-project/xsdk-community-policies

xSDK EXASCALE COMPUTING PROJECT

xSDK History: Version 0.1.0: April 2016

https://xsdk.info

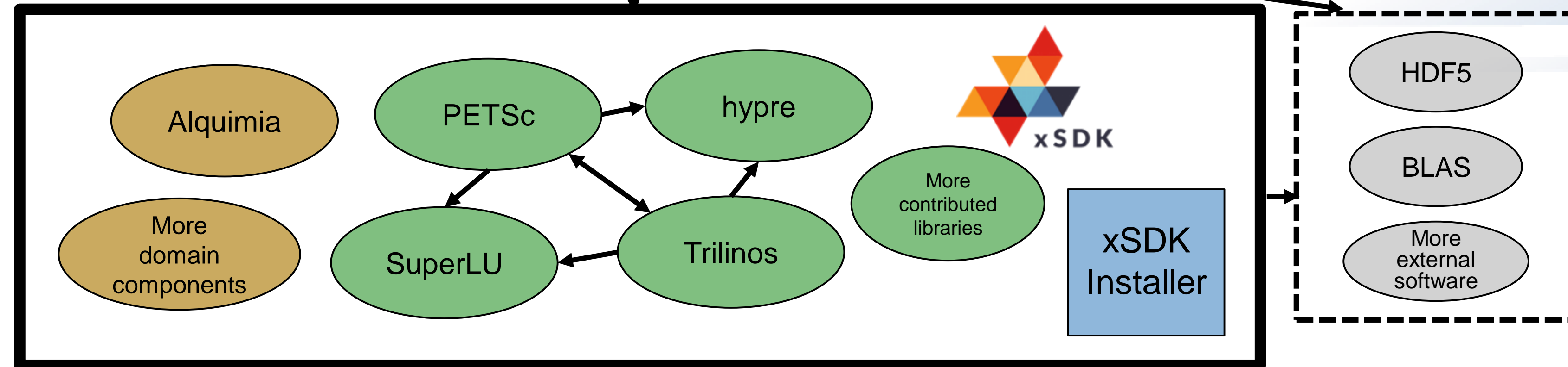
Notation: A → B:
A can use **B** to provide functionality on behalf of **A**

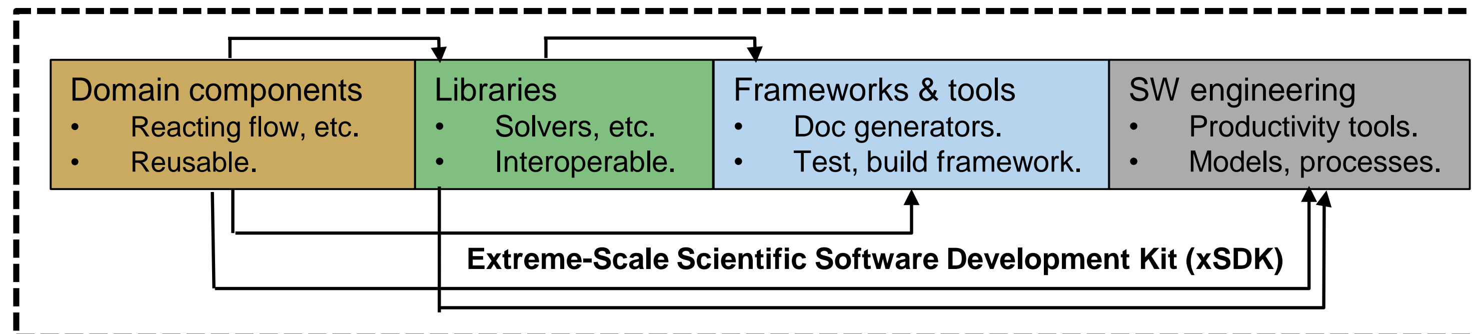


xSDK functionality, April 2016

Tested on key machines at ALCF, NERSC, OLCF, also Linux, Mac OS X

April 2016
- 4 math libraries
- 1 domain component
- PETSc-based xSDK installer
- **14 mandatory (5 rec.) xSDK community policies**

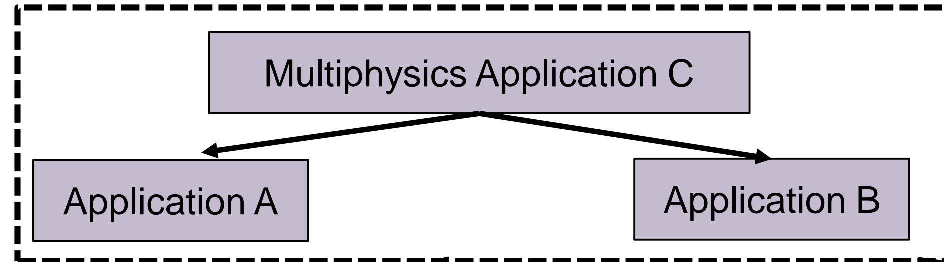




xSDK History: Version 0.5.0: November 2019

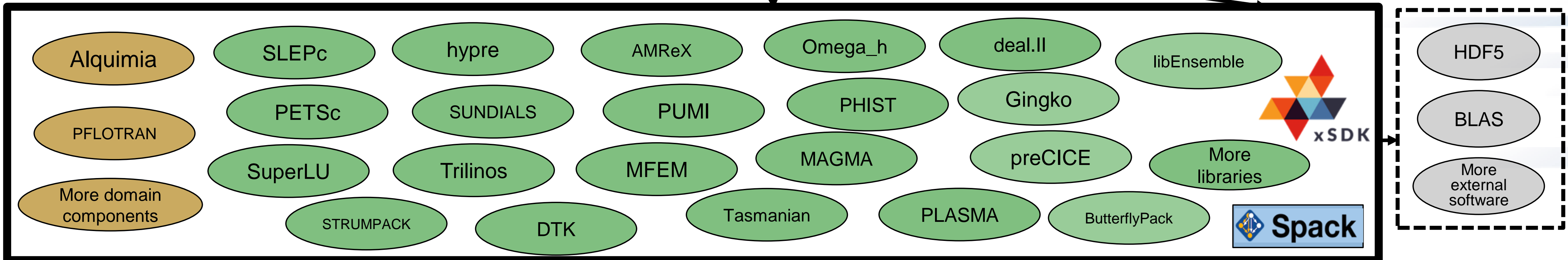
https://xsdk.info

Each xSDK member package uses or can be used with one or more xSDK packages, and the connecting interface is regularly tested for regressions.



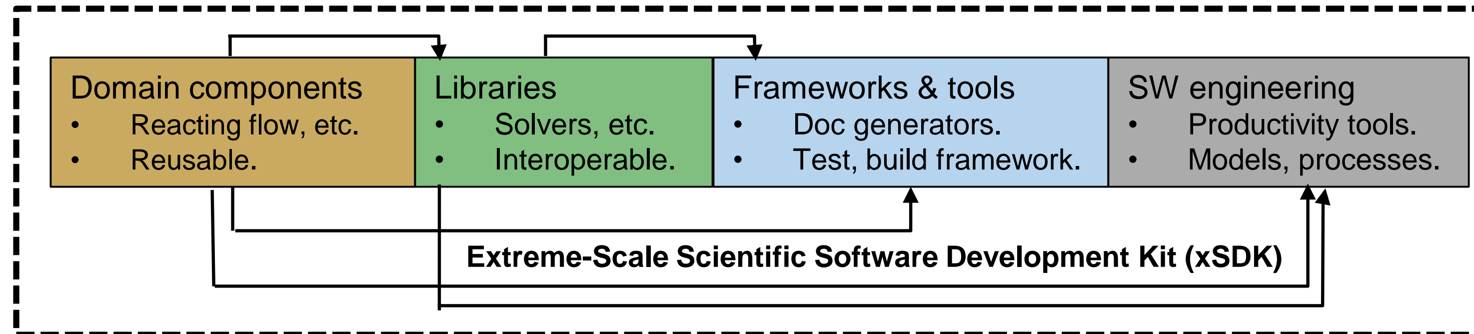
xSDK functionality, Nov 2019

Tested on key machines at ALCF, NERSC, OLCF, also Linux, Mac OS X



November 2019

- 21 math libraries
- 2 domain components
- **16 mandatory (7 rec) xSDK community policies**
- Spack xSDK installer



Impact: Improved code quality, usability, access, sustainability

Foundation for work on performance portability, deeper levels of package interoperability

xSDK Dependency Graph

xSDK Member
Dependency

xsdk
py-libensemble dealii mfem alquimia
py-petsc4py precice oce slepc gsl sundials pflotran
intel-tbb phist petsc omega-h
eigen muparser nanoflann strumpack trilinos
boost butterflypack hypre matio netcdf-c
py-numpy py-mpi4py plasma suite-sparse arpack-ng netlib-scalapack superlu-dist tasmanian hdf5
py-setuptools amrex openblas parmetis pumi p4est
python ginkgo glm metis openmpi
gettext expat libffi cmake hwloc
bzip2 libbsd libxml2 sqlite openssl adol-c numactl libpciaccess
diffutils tar xz zlib automake libtool util-macros
libiconv autoconf
perl m4
gdbm libsigsegv
readline
ncurses
pkgconf

Questions?

Deep Dive: Community Software Policies

M1: Support Autoconf or CMake Options

- The configuration options are meant make xSDK packages compatible during installation
- The options are described in the xSDK Community Installation Policies
  - `https://github.com/xsdk-project/installation_policies/README.md`

- The use of Autoconf or CMake is optional
  - For example, PETSc uses Python for configuration
  - Makefile-only builds are not allowed
    - PLASMA introduced CMake-based configuration and build for compatibility with xSDK
    - SuperLU transitioned from manually editing "make.inc" to CMake/Ctest for increased productivity and robustness
      - Management of dependencies became easier (turn ParMetis on/off, inclusion of machine-dependent files)
      - Platform-specific versions were introduced (MT = multi-threaded, DIST = distributed, GPU)
  - SUNDIALS had to re-align the configuration options to be compatible with xSDK
    - Some duplication was introduced so that the redundant options would keep compatibility for existing users

xSDK Installation Policies

1. --dis/enable-xsdk-defaults
   USE_XSDK_DEFAULTS=[YES,NO]
2. --prefix=path
   CMAKE_INSTALL_PREFIX=path
3. CC=cc CXX=cxx FC=fc CPP FFLAGS FCFLAGS CFLAGS CXXFLAGS CPPFLAGS LDFLAGS
   CMAKE_C_COMPILER=cc
4. --dis/enable-debug
   CMAKE_BUILD_TYPE=[Debug,Release]
5. --dis/enable-shared
   BUILD_SHARED_LIBS=[YES,NO]
6. --dis/enable-<language>
   XSDK_ENABLE_<language>=[YES,NO]
7. --with-precision=[single,double,quad]
8. --with-index-size=[32,64]
9. --with-blas-lib=-lblas --with-lapack-lib=...
10. --with-<package> --with-<package>-lib=. –with-<package>-include=.
11. In xSDK mode, do not rely on env. variables such as LD_LIBRARY_PATH
12. Compile, install, “smoke” test with “make”, “[sudo] make install”, “make test_install”
13. After install, provide machine-readable provenance

xSDK EXASCALE COMPUTING PROJECT

M2: Provide Comprehensive Test Suite

- The test suite cannot require commercial software
- Significant portion of the test suite should:
  - Complete on a workstation within a few hours.
  - Be suitable for running in batch-only environment.
- There are differences in test suite coverage between xSDK packages
  - For deal.II, there are many interfaces to other libraries that are verified by the test suite
  - hypre introduced a new test suite to check for errors on any platform
  - MFEM tests for versioning of dependent libraries to verify interoperability
  - PETSc/TAO introduced new features and parallelism to their test suite for more robust and scalable testing
  - SuperLU introduced a comprehensive regression suite of unit tests
    - Compatibility with Travis CI allowed to use an integration pipeline that includes every git commit
  - Trilinos now has a more regular testing that includes its interfaces to hypre and PETSc
    - Documentation received a refresh as well

M3: No Direct Use of MPI_COMM_WORLD

- Only user-provided communicator should be used for communication
- It's OK to provide a sequential version (no MPI required)
  - But not with a dummy (single rank) implementation because there would be a clash of symbols if multiple packages do it
  - Compatibility with other packages that use MPI is not required if sequential version is installed
- Some packages in xSDK were designed with communicators in mind while others had to adjust to accommodate policy M3

M4: Be Portable

- Support common platforms
  – xSDK is regularly tested on macOS and variants of Linux
  – Machines at major DOE sites are also used for testing
- Support common compiler toolchains
  – LLVM Clang, GNU C/C++/gfortran
- Support vendor toolchains
  – Sometimes necessary for integration with low-level hardware features
- Some xSDK packages had to adjust their building process to accommodate the portability
  – PHIST did away with some platform/toolchain specific settings (OpenMP or -march) to decouple from a particular compiler

M5: Provide Reliable Contact Information

- Web form or email are required
  - Joining mailing lists is not enough because they lack focus
    - Users care for their own bugs only, not all the other bugs that usually are sent to the bug reporting mailing lists by the entire users' community

xSDK ECP EXASCALE COMPUTING PROJECT

M6: Respect System Resources and Settings

- For example: the handling mode for floating-point exception
  - It’s OK to change the error handling mode, but only if allowed by the user through, for example, an API call
- If changing system settings: save the original state and resort it upon exit
- If saving state is not possible, provide the user ability to prohibit state change

xSDK EXASCALE COMPUTING PROJECT

M7: Use Permissive Open-Source License

- Permissive licenses include MIT and 3-clause BSD.
- Strong copyleft licenses (e.g. GPL) are not compatible.
- Weak copyleft licenses (e.g. LGPL or GPLv2 with runtime exception) are accepted, (but inclusion of such packages could be optional in the future)
- Restricting commercial use is not allowed (industry doesn’t like weak copyleft.)
- Some of the xSDK packages, such as Trilinos, include custom versions of external packages that might be licensed under a different license than the main source code.

M8: Make Version Information Available at Runtime

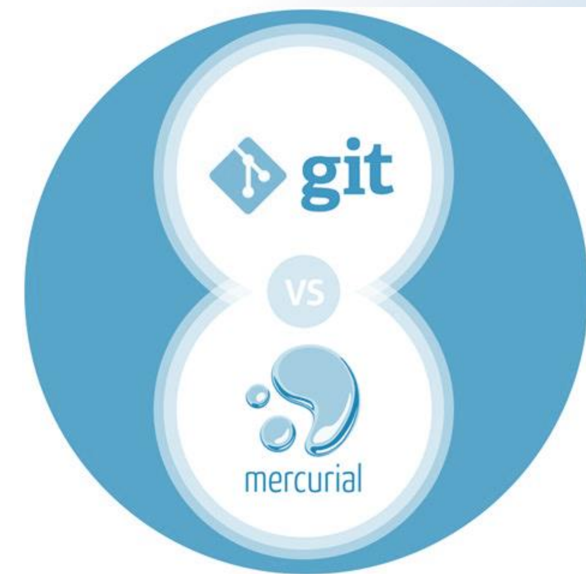
- For regular releases, full version information should be available through an API call
- For in-development builds, the version can be identified by a commit ID
- The options used to configure and build the package should also be provided
- Note:
  - It is hard to track versions of all dependent packages and it is not required which may prevent rebuilding the package in the same state
    - See Spack concretization for how to track the entire dependence tree of packages

M9: Use a Namespace for Externally Visible Artifacts

- A package-specific namespace has to be provided for:
  - Macros, symbols, library names, and header files
- For example, a matrix class from different xSDK packages could clash so they should be in a namespace:
  - `hypre_matrix`, `magma::matrix`, `petsc_matrix`, `plasma_matrix`, `sundials_matrix`, `trilinos::matrix`
- Don't forget to prefix configuration options:
  - For example `HAS_LONG_LONG` should be `PLASMA_HAS_LONG_LONG` if it gets included in the installed PLASMA headers
- Introducing a namespace for necessary for some xSDK packages
  - Simple macros were unprefixed in deal.II
  - Many functions needed `hypre_` prefix
  - PLASMA had a library called Core Blas that needed `PLASMA_` prefix
  - When SUNDIALS introduced compatible API for time integrators and nonlinear solvers, MFEM interfaces to SUNDIALS had to be updated
  - New namespaces in SuperLU allowed simultaneous use of the three library versions (serial, multithreaded, and distributed) both internally and externally

M10: Use Version Control for Development

- The repo does not have to be public but must be accessible by the xSDK team
- We recommend support for pull requests
  - xSDK uses them heavily for development and collaboration
  - Other developers will likely use them as well
- Majority of xSDK packages use Git, some use Mercurial
  - SuperLU transitioned from SVN to Git
    - This enabled distributed development, bug fixing, and external contributions:
      - Windows and static/shared libraries' builds contributed by external developers
  - MAGMA and PLASMA will be transitioning from Bitbucket to Git
    - Due the recent sunsetting notice of Mercurial VCS by Atlassian's Bitbucket



xSDK EXASCALE COMPUTING PROJECT

M11: No “Hardwired” print() or other I/O

- Any print() or I/O must have an option to be disabled by the user
  - This must be done in programmatic way
    - Environment variables cannot be changed at a reliable order in simulations involving complex dependencies
- Allow the output to be redirected by the user
  - Default file descriptors such as `stdout` or `stderr` might not be reliable at scale
- If the package prints by default then the output should be limited to a single process, e.g., rank 0.
- Print-statement debugging is common and considered indispensable by some developers but removing them is necessary for compatibility
  - In hypre, new reporting level was introduced to ensure that there is no I/O by default
  - In PUMI, over 700 calls to *`printf()` or iostream had to be dealt with (wrappers, redirection)

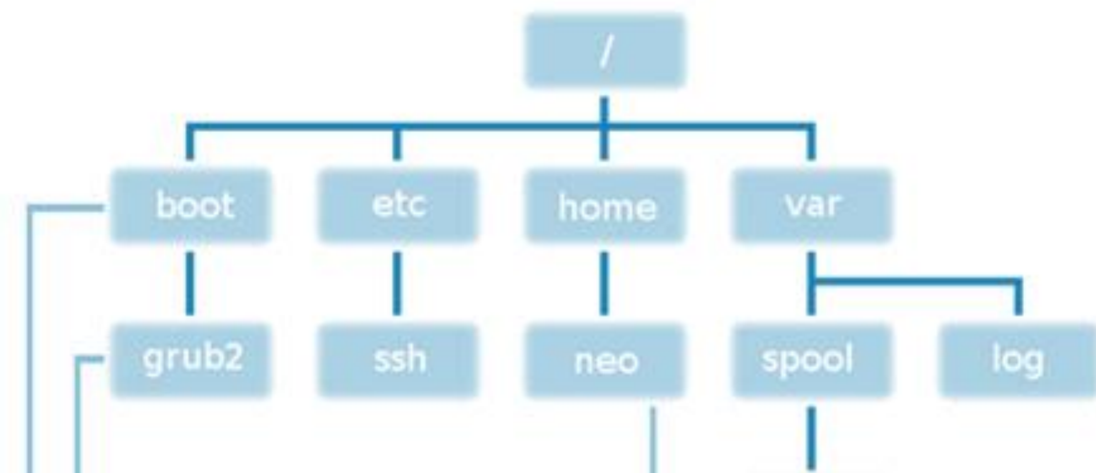
xSDK EXASCALE COMPUTING PROJECT

M12: Allow Use of External Dependencies

- If your packages includes pieces of other software then you must allow to link against the system-provided copy
  - The included software is the one that is externally developed
- There are several ways to enable the use of the external software
  - Disable the included version during the build
  - Confine the included version to a new namespace
- There are many examples of including external software
  - Subset of BLAS and LAPACK is often included if only their limited functionality is required
  - Trilinos included early versions of `boost::any` and SparseSuite but they were kept with changed file names and a new namespace
- Some xSDK packages have hardwired external dependencies
  - For example, MAGMA requires hardware accelerator API such as CUDA, HIP, or SYCL

M13: Use <prefix> for Installation

- The package must follow the configuration process that uses `--prefix` option to specify the destination for where the headers and libraries should be installed
  - Headers go to `<prefix>/include`
  - Libraries go to `<prefix>/lib` or `<prefix>/lib64` depending on the system
- The version numbers should **not** be embedded in filenames of headers or libraries
  - Exception: sonames and the links:
    - `<prefix>/lib/lib<package>.so`
      `<prefix>/lib/lib<package>.so.X`
      `<prefix>/lib/lib<package>.so.X.Y.Z`



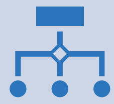
xSDK EXASCALE COMPUTING PROJECT

M14: 64-bit Pointers by Default

- Packages should build by default with 64-bit pointers
- 32-bit pointers are optional

xSDK EXASCALE COMPUTING PROJECT

M15: Compatibility with the Policies is Sustainable



The package cannot have one branch in the repo for non-compatible versions and another branch for the compatible ones

Maintaining the compatibility should be part of the standard release process for all the packages

In other words: occasional fixes are insufficient

Trilinos added continual maintenance of documentation, interfaces to external packages, and regular testing

xSDK
ECP EXASCALE COMPUTING PROJECT

M16: Production-Quality Installation Process

- The package should be compatible with the installation tool of xSDK
  - Since xSDK 0.2.0-alpha (released in April 2017), xSDK installs with Spack

xSDK ECP EXASCALE COMPUTING PROJECT

R1: Make Public Source Code Repository

- Public repositories encourage collaborative development
- Pull requests further facilitate user feedback through improvements and bug fixes

R2: Use Debugging Tools for Running the Test Suite

- One example is Valgrind that could check for memory corruption
  - The price is a slowdown in execution and may not be suitable for a frequent use

xSDK EXASCALE COMPUTING PROJECT

R3: Make Error Reporting Consistent and Configurable

Packages should establish a policy for error conditions

- Returning error codes
- Propagating exceptions

It should be possible to change this behavior through an API call

- For production use: return error codes
- For debugging: abort upon error

It is prohibited to call `abort()`, `exit()`, or `MPI_Abort()` unconditionally

No unconditional printing during error handling

The errors should be documents as

- Recoverable
- Causing resource loss
- Leaving the process in undefined behavior

The calling code will decide what to do with each class of errors

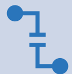
R4: Release System Resources ASAP

System resources should be as soon as they are not needed, including

- Closing open files
- Freeing heap memory
- Freeing MPI communicators, data types

This will ensure no gradual exhaustion of resources for long or large-scale runs

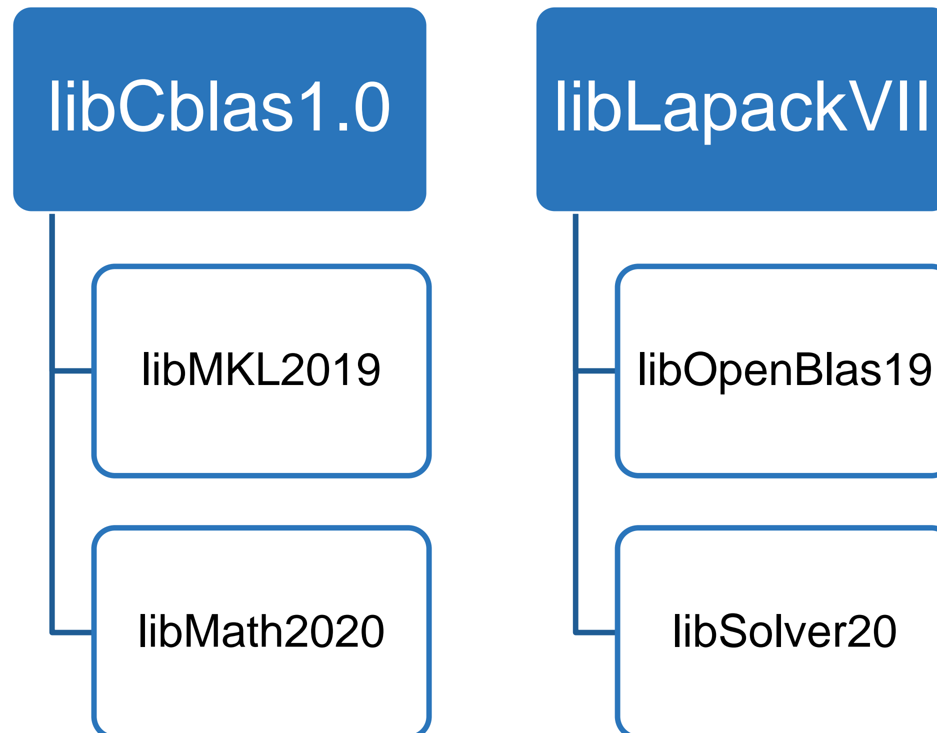
Any resource that has to be released by the user must be clearly documented



Valgrind is useful for finding memory leaks (see R2)

R5: Export Ordered List of Library Dependencies

- When a package includes a list of its library dependencies then every user package will know to include them during linking



xSDK EXASCALE COMPUTING PROJECT

R6: Document Versions of Suitable Dependencies

- This includes the software outside of xSDK
- The documentation should be machine readable for automatic verification
- This facilitates periodic releases of xSDK

xSDK EXASCALE COMPUTING PROJECT

R7: Provide Common Information Files in the Repo

- `README` or `README.md`
  - Brief description, installation information, web page link
- `LICENSE` or `LICENSE.md`
  - Full text of package licensing terms or a link to it
- `SUPPORT` or `SUPPORT.md`
  - Contact information to get help (see M5)
- `CHANGELOG` or `CHANGELOG.md`
  - Important changes over time or a link to the list of changes (no need for every commit message)
- See online for more information
  - `https://github.com/kmindi/special-files-in-repository-root/blob/master/README.md`

Adding, Changing, Retiring Community Policies

- xSDK policies are updated regularly (but don't confuse them with software releases)
- To maintain a community, its members have to agree on the set of policies or any changes over time
- xSDK team members seek input from the larger community of users and arrive at consensus (or majority) how to take the feedback into account
- Recommended policies migrate to become mandatory ones
- New policies arise due to changes in hardware and software

Seek community input

Discuss feedback

Consensus vote

Compatibility with xSDK community policies

To help developers of packages who are considering compatibility with xSDK community policies, we provide:

- Template with instructions to record compatibility progress
- Examples of compatibility status for xSDK packages
  - Explain approaches used by other packages to achieve compatibility with xSDK policies
- Available at https://github.com/xsdk-project/xsdk-policy-compatibility

xSDK Community Policy Compatibility for PETSc

This document summarizes the efforts of current and future xSDK member packages to achieve compatibility with the xSDK community policies. Below only short descriptions of each policy are provided. The full description is available here and should be considered when filling out this form.

Please, provide information on your compabilty status for each mandatory policy, and if possible also for recommended policies. If you are not compatible, state what is lacking and what are your plans on how to achieve compliance. For current xSDK member packages: If you were not compliant at some point, please describe the steps you undertook to fulfill the policy.This information will be helpful for future xSDK member packages.

Website: https://www.mcs.anl.gov/petsc

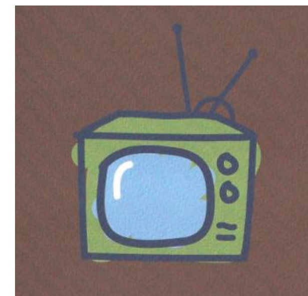
Mandatory Policies

Policy	Support	Notes
M1. Support xSDK community GNU Autoconf or CMake options.	Full	PETSc uses the GNU Autoconf options. The implementation is done with python code.
M2. Provide a comprehensive test suite for correctness of installation verification.	Full	PETSc has over 1000 test examples and a test harness that can execute the examples in parallel. It also collects information on the failures and can display them graphically, e.g., see ftp://ftp.mcs.anl.gov/pub/petsc/nightlylogs/archive/2017/09/19/master.html
M3. Employ userprovided MPI communicator (no MPI_COMM_WORLD).	Full	All PETSc objects take a MPI communicator in the constructor, allowing the user complete control over where each object exists and performs its computations.
M4. Give best effort at portability		

Impact of xSDK Software Policies

- Improved code quality, and usability of individual libraries (or application codes)
- Addresses challenges in interoperability and sustainability of software developed by diverse groups at different institutions
- Enables common build of libraries
- Foundation for work on deeper levels of interoperability and performance portability
- Base for new sets of software policies
- Engages community

It takes all kinds.



Think outside the box.

xSDK community policies

now also on github: https://github.com/xsdk-project/xsdk-community-policies

We welcome feedback. What policies make sense for your software?

https://xsdk.info/policies

xSDK compatible package:
Must satisfy mandatory xSDK policies:

M1. Support xSDK community GNU Autoconf or CMake options.
M2. Provide a comprehensive test suite.
M3. Employ user-provided MPI communicator.
M4. Give best effort at portability to key architectures.
M5. Provide a documented, reliable way to contact the development team.
M6. Respect system resources and settings made by other previously called packages.
M7. Come with an open source license.
M8. Provide a runtime API to return the current version number of the software.
M9. Use a limited and well-defined symbol, macro, library, and include file name space.
M10. Provide an accessible repository (not necessarily publicly available).
M11. Have no hardwired print or IO statements.
M12. Allow installing, building, and linking against an outside copy of external software.
M13. Install headers and libraries under <prefix>/include/ and <prefix>/lib/.
M14. Be buildable using 64 bit pointers. 32 bit is optional.
M15. All xSDK compatibility changes should be sustainable.
M16. The package must support production-quality installation compatible with the xSDK install tool and xSDK metapackage.

Also **recommended policies**, which currently are encouraged but not required:

R1. Have a public repository.
R2. Possible to run test suite under valgrind in order to test for memory corruption issues.
R3. Adopt and document consistent system for error conditions/exceptions.
R4. Free all system resources it has acquired as soon as they are no longer needed.
R5. Provide a mechanism to export ordered list of library dependencies.
R6. Provide versions of dependencies.
R7. Have README, SUPPORT, LICENSE, and CHANGELOG file in top directory.

xSDK member package: Must be an xSDK-compatible package, *and* it uses or can be used by another package in the xSDK, and the connecting interface is regularly tested for regressions.

xSDK EXASCALE COMPUTING PROJECT

Some useful links

- http://xsdk.info/
- http://xsdk.info/policies/
- https://github.com/xsdk-project/xsdk-community-policies
- http://ideas-productivity.org/resources/howtos/

Acknowledgments

- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

xSDK EXASCALE COMPUTING PROJECT

Questions?