A Roadmap for Sustainable Ecosystems of CSE Software

Roscoe A. Bartlett, Ph.D.

bartletrra@ornl.gov

http://web.ornl.gov/~8vt/

Oak Ridge National Laboratory

Trilinos Software Engineering and Integration Lead
CASL VERA Software Engineering Lead

Computational Science and Engineering Software Sustainability and Productivity Challenges (CSESSP) Workshop

Rockville, MD

October 15 - 16, 2015
Overview of CSE Software Ecosystem Challenges

Overview of CSE Software Ecosystems:
- Sophisticated cutting-edge algorithms implemented by PhD experts from different fields
- Packages independently implemented, maintained, and released by different organizations and institutions
- Many packages constantly developed over many decades and changes to programming models, computer architectures, etc.
- Many APPs (i.e. customers) need access to the latest versions of some packages (e.g. driving research).

Motivating/example ecosystems:
- Trilinos: 68 native pkgs, 90 upstream TPLs (third party libraries), many critical downstream pkgs/apps
- CASL VERA: 18 repositories integrated with almost CI, 10 upstream TPLs => TPLs #1 portability issue!
- SNL SIERRA: Uses 30+ upstream pkgs/TPLs (including Trilinos, PETSc, etc.)
- IDEAS xSDK: Trilinos, PETSc, SuperLU, HYPRE (and several upstream TPLs) and BER app codes

Challenges to Sustainable Ecosystems of CSE Software:
1. **Lifecycle and software quality of individual packages**: Is a package by itself ready to be used by customers and participate in an ecosystem?
2. **Sustainability of software packages**: Is a package sustainable over long lifecycle?
3. **Maintaining compatibility of packages in the ecosystem**: Can the compatibility of interdependent packages be maintained over decades and satisfy customer needs?
4. **Building a compatible set of packages for a given application from source**: Can a compatible set of interdependent packages be effectively deployed to customers?
Roadmap for Sustainable CSE Software Ecosystems

1. **Lifecycle and software quality of individual packages:** Is a package by itself ready to be used by customers and participate in an ecosystem?
   - Lean/Agile lifecycle for CSE software:
     - Exploratory (EX) => Research Stable (RS) => Production Growth (PG) => Production Maintenance (PM)
     - Existing software grandfathered in using *Legacy Software Change Algorithm*

2. **Sustainability of software packages:** Is a package sustainable over long lifecycle?
   - Self-Sustaining Software: open-source license, strong automated tests, clean design/code, minimal controlled internal and external dependencies (stopping at standards)

3. **Maintaining compatibility of packages in the ecosystem:** Can the compatibility of interdependent packages be maintained over decades and satisfy customer needs?
   - Continuous Integration (CI) => e.g. Trilinos packages, Google online apps (5K+ developers)
   - Almost Continuous Integration (ACI) => e.g. INL MOOSE, CASL VERA, SIERRA/Trilinos, ...
   - Punctuated Releases => Semantic Versioning Standard X.Y.Z, sets of backward compatible releases (i.e. fixed X, increment Y), buildable against multiple versions of upstream packages

4. **Building a compatible set of packages for a given application from source:** Can a compatible set of interdependent packages be effectively deployed to customers?
   - Build & Install wrappers around heterogeneous build systems (CMake, autotools, raw makefiles, etc.) => e.g. CMake ExternalProject, Spack, PETSc --download-xxx, CASL VERA TPLs
   - Uniform build system for all packages: => e.g. SNL SIERRA (replaced native build process with new bjam files for 30+ TPLs), TriBITS/CMake (Trilinos, CASL VERA (Trilinos, SCALE/Exnihilo, COBRA-TF, MPACT, ...)), Google online apps (2K+ projects)
Example: Maintaining Compatibility and Deploying Packages Over Many Released Versions

Assumptions:

- Start out all compatible packages, version 1.0
- New releases on same cadence (e.g. every quarter/year, etc.)
- Upgrade to most current allowed version of upstream packages
- No coordination/staging between package developers or releases
- Package ‘A’ breaks backward compatibility with each release, all other packages maintain backward compatibility

Release Set 1: A1, B1, C1, D1, E1, F1
Release Set 2: => All release against A1!
  • A2
  • B2: A1
  • C2: B1(A1), A1
  • D2: B1(A1)
  • E2: C1(B1(A1), A1), A1
Release Set 3: => Can’t all use A2!
  • A3
  • B3: A2
  • C3: B2(A1), A2 => A1
  • D3: B2(A1)
  • E3: C2(B1(A1), A1), A2 => A1

Release Set 4: => Most stuck with A1 or A2!
  • A4
  • B4: A3
  • C4: B3(A2), A3 => A2
  • D4: B3(A2)
  • E4: C3(B2(A1), A1), A3 => A1
Release Set 5: => Five versions of A in use!
  • A5
  • B5: A4
  • C5: B4(A3), A4 => A3
  • D5: B4(A3)
  • E5: C4(B3(A2), A2), A4 => A2

- Developers for Package A have to support current and 4 prior releases!
- Some downstream customers stuck with very old versions of some packages!