redhat.

Making Akraino Successful

Frank Zdarsky
Senior Principal Software Engineer
Office of the CTO, Red Hat

2018-08-23

Key Success Drivers

- Clarity of mission (the “WHY”) -- aspirational but not open-ended
- Clarity of deliverables (the “WHAT”) -- each with clear business value for its users
- Goal-oriented approach (the “HOW”) -- focussed on deliverables, not process

Classifying Open Source Community Projects

... by main type of deliverable:

- development project
  - focus on developing a component / system ➔ code, documentation
  - ex.: OpenStack, Kubernetes, ONAP, ODL, EdgeX
- **integration project**
  - focus on *enabling* integration ➔ e2e test & tools, RFEs or patches for e2e functional gaps, code for auxiliary components enabling e2e use case
  - typically not *prescribing* integration ➔ code as basis for project distributions
  - ex.: OPNFV
- specification project
  - focus on interface/protocol specification ➔ spec, reference implementation
  - ex.: OCI

Classifying Open Source Community Projects

... by industry focus:

- industry-specific (ex.: OPNFV, ONAP)
- **industry-agnostic** (ex.: OpenStack, Kubernetes, ODL)

... by {internal, external} project coupling:

- **loosely coupled** (ex.: Kubernetes + eco-system projects under CNCF)
- tightly coupled (ex.: ONAP, most of OpenStack)

... by community health

- **diverse and development-centric** user and developer community
- imbalanced and/or discussion-centric user and developer community

Open Source Project “Anti-Patterns”

- Confusing *project* with *product*.
  - agility vs stability goal conflict
  - features vs {backporting, multi-release upgrades, ...} dev resource prioritization
  - premature security hardening, performance tuning, ...
- Forcing coordinated releases on time-based release cycles without need.
  - Binds 100%+ dev resources for months prior to release for small marketing gain.
  - Often surfaces problem of tight sub-project / component coupling.
- Establishing “Conformance Testing Program” in non-specification projects.
- Reinventing instead of improving / working with eco-system.
- Carrying patches against upstreams / not enforcing strict “upstream-first” best-practice.
- Developing & maintaining integration code outside of projects to be integrated.

Input to TSC Governance Discussions

- Overall guiding principles:
  - Community needs “ownership” of its deliverable, freedom to define its evolution.
  - Community needs to be inclusive (➜ contributions not limited to membership) and diverse (➜ quotas per company on privileged roles).
  - Privileged roles should be time-limited and require renewal by default.
- TSC Member role:
  - define overall technical architecture & direction, resolve technical dispute, set quality standards & best-pratices, prioritize work to keep the project focussed
  - elected from and by ATCs
- TSC Co-Chair role:
  - representative, moderator, facilitator, steward… not leader!
  - elected from and by TSC Members

Input to TSC Governance Discussions (cont.)

- Committer role:
  - define technical direction of a sub-project, ability to commit
  - promoted from the ATCs by the existing Committers of that sub-project based on the ATC's proof of of expertise, experience, and contributions.
- Active Technical Contributor (ATC) role:
  - promoted from Contributor after a significant, measurable number of technical contributions (code, reviews, docs) over a defined measurement period
- (Sub-)Project Lifecycle:
  - high bar for new projects (narrow scope, clear motivation + problem statement + objectives + KPIs, contributor diversity), regular progress/maturity review

Input to Blueprint Definition Discussions

- Why Blueprints? Means to drive convergence, facilitate building/operating edge stacks.
  - for users: tested deployment config as basis for customization and procurement
  - for vendors: opportunity to target development on fewer deployment configs
- How many? What level of detail? 1 BP per pod? per use case? per (use case, vendor)?
  - need to separate outcomes (POD, HW+SW stack, enabled workloads; want *few*) from implementations (CPU Arch A vs B, OpenStack C vs D, …; will have *many*).
- ⇒ BPs should focus on the outcome (WHAT), avoid prescribing implementation (HOW)!
  - enables technical innovation/evolution, multiple solutions per BP
- How do “seed code” and BPs relate to each other?
  - Airship is advanced e2e impl., but w/ Airship-specific design choices & data model
- ⇒ BPs need to generalize, start from architecture-level requirements; seed code is one implementation (reference implementation?) of it.

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos