Release Engineering Strategies: How LLVM and GCC Navigate Development and Maintenance

David Edelsohn, %company%, GCC Steering Committee
Tom Stellard, Red Hat, LLVM Release Manager

Outline

Introduction to GCC

GCC Development and Release Process

Introduction to LLVM

LLVM Development and Release Process

Comparative Analysis

Insights

Conclusion

Q&A at Roundtable

GCC History and Evolution

Compiler for the GNU Project

Advantages over Portable C Compiler (PCC)

Cygnus Solutions targeted embedded processor SDKs

Widely adopted by BSDs and by Linux

Linux system compiler for major Linux distributions

First C Compiler Release March 1987

C++ added January 1990

EGCS 1.0 December 1997

GCC 2.95 July 1999

GCC 2.96 RH

GCC 3 (major features) March 2001

New development process adopted July 2001

GCC 4 (SSA) April 2005

GCC 14.1 March 2024

GCC Development Model

Structured contribution process

- Steering Committee
- Release Managers, Global Reviewers, Port Maintainers, Feature Maintainers
- GCC: 6605 commits by 389 developers in 2023

Focus on stability and backward compatibility; GCC Trunk should be stable

Predictable development process and release cycle (yearly)

- Open Development: 7 months
- Bug fixes and port-specific changes: 2 months
- Regression fixes:3 months

Extensive testsuite and Linux distribution rebuild sniff test; limited CI system

GCC Release Model

Predictable maintenance releases for 3 years

- Unified maintenance in GCC repository
- All bugfix backports must be upstream and in all newer release branches
- Update releases at 3 months, 6 months, 1 year, and 2 years anniversaries
- Linux distros continued shared maintenance in GCC repo
  - No additional official releases
- Linux distributions’ system compiler chosen from planned GCC releases

LLVM Development Model

Contribution process

- Adopting more structured governance: Maintainers, Area Teams, Project Council.
- Very high rate of change. In previous 30 days:
  - LLVM: 3306 commits by 671 authors.
  - gcc: 793 commits by 129 authors.

Focus on maintaining a stable main branch

- Low bar for reverting patches to fix regressions in main.
- Downstream users pull and test main branch regularly.

LLVM Development Model

New feature work in main done in parallel with stabilization work in the release branch.

Predictable release cycle (6 months)

- RC/ Release Stabilization/X.1.0 Release Phase: 1 Month
- Bug Fix Updates (X.1.n) every 2 weeks for 3 Months
- Current proposal for 19.1.n to distribute bug fix updates for ~ 5 months.

LLVM Release Model - Recent Changes

Making releases cheap.

Simplifying backport process.

Continuous Integration.

LLVM Release Model - Challenges for Distributors

Two different deliverables in the same build:

- Tools (clang, lld, etc).
- Libraries (libLLVM.so, libclang.so, libclang-cpp.so)

Introducing new versions of tools is easy.

Introducing new versions of libraries is very difficult:

- ABI changes between major versions

Need to distribute multiple versions due library dependencies

Contrasting compiler usage models

Linux distributions and vendor compilers

Multi-purpose systems with long release horizons and long maintenance horizons

Single-purpose containers with rapid release cycles

Comparative Analysis

GCC

Strengths

- Stability
- Long-term support
- Reliably compile entire Linux distro

Weaknesses

- Slower innovation
- Rigid development process

LLVM

Strengths

- Flexibility
- Rapid innovation
- Modular design

Weaknesses

- Lack of LTS
- ABI and API stability

Impact on adoption

Long-term support

Stable ABI

Backward compatibility

Shared maintenance reduces duplication

Unified maintenance ensures consistency across all distributions

Woe the Linux System Compiler

Red Hat, SUSE, and Canonical have invested heavily in the GNU Toolchain release management design and process.

It is easier to change the toolchain release and maintenance process than the Linux distribution release and maintenance process.

How to make LLVM more effective as an alternative system compiler?

GCC niches

- Compiler of last resort (language extensions and corner cases)
- Long-term support

How can LLVM implement Long-Term Support and unified maintenance without stifling innovation?

LLVM Barriers to LTS Release

Too many releases at once.

Developer bandwidth for backports and fixes.

Not everyone is motivated to work on LTS releases.

Aligning schedule with downstream users.

Most distributors will still need newer versions.

Fast rate of change in main makes backporting more difficult.

LLVM hybrid model?

Choosing an LTS release:

- Every other release?
- Every 4th release?

How long are the supported?

What does ‘support’ look like?

Tools only LTS release?

Questions?

Join us at the Roundtable
to continue the conversation!
Thursday 2:15 PM

david.edelsohn@gmail.com