Combined WG21 Principles

Synthesized from expert interviews with Dave Abrahams, Doug Gregor, Howard Hinnant, Matheus Izvekov, and Sean Parent. Sorted by confidence and impact.

Highest Impact Principles

1. Standardize Existing Practice, Not Invent

Standards bodies should only standardize designs proven through extensive real-world use; invention and experimentation must happen outside the standardization process. Standardizing unproven features risks discovering they're unimplementable or impractical—the STL template features standardized before any compiler implemented them remain "a black eye for the committee."

(Dave Abrahams, Doug Gregor, Howard Hinnant)

---

2. Require Positive Field Experience Before Standardization

Proposals must demonstrate successful real-world usage with positive feedback from independent users before standardization; implementation alone is insufficient. Having an implementation proves something *can* work; having field experience proves it *does* work in practice.

(Howard Hinnant, Dave Abrahams, Doug Gregor)

---

3. Document Decision Rationale to Ensure Consistency

Standards bodies must document the rationale for each decision and maintain explicit design principles; without this, similar questions get inconsistent answers and languages accrete contradictions. C++'s accumulated "warts" stem from lack of a framework ensuring consistency across decisions.

(Sean Parent)

---

4. Make the Impossible Possible, Not the Easy Easier

Only standardize features that enable what was previously impossible/impractical, or that significantly reduce difficulty of hard tasks; reject proposals that merely add convenience for already-easy operations. Every standard addition incurs maintenance cost and cognitive load.

(Howard Hinnant)

---

5. One Person Can Derail Years of Collaborative Work

In committee processes, even after years of work and formal approval, a single influential person withdrawing support can kill a proposal. The concepts proposal was merged into the draft standard, then undone by one email. This risk makes major standardization investments precarious.

(Dave Abrahams, Doug Gregor)

---

6. Political Sensitivity Is Required for Technical Success

Technical excellence alone is insufficient for standardization success. Major proposals require political navigation—understanding stakeholder concerns, managing egos, and sometimes apologizing even when you've done nothing wrong.

(Dave Abrahams, Doug Gregor)

---

7. Proving Grounds Are Essential for Language Evolution

Languages need external proving grounds where ideas can be tested extensively before standardization. Boost provided this for C++; without it, "they're missing the experimentation to build up what is the best practice before we get into the language."

(Dave Abrahams, Doug Gregor, Sean Parent)

---

8. Backward Compatibility Is a Sacred Constraint

Any C++ proposal must preserve backward compatibility with existing code. This is not negotiable—breaking existing code is almost never acceptable, even for significant improvements. This constraint shapes what's possible in C++ evolution.

(Dave Abrahams, Doug Gregor)

---

9. Correctness Over Performance, Always

Correctness must always take precedence over performance; optimizations that sacrifice correctness (like compilers removing "UB" code that would otherwise work) invert the proper priority. Hardware has defined behavior; standards saying "undefined" enables dangerous optimizations.

(Howard Hinnant)

---

10. Community Collaboration Beats Solo Design

Software design for standards benefits enormously from collaborative community effort. Individual brilliance is insufficient; you need diverse perspectives finding weaknesses and suggesting improvements. Single-author proposals without community input are a red flag.

(Dave Abrahams, Doug Gregor)

High Impact Principles

11. The Committee Lacks Executive Power

WG21 is a volunteer organization without executive authority; no one can compel work to be done. Important but contentious proposals may be abandoned when champions lose interest—even obviously needed features can languish for years.

(Howard Hinnant)

---

12. High Vote Counts Do Not Indicate Understanding

Near-unanimous EWG votes can mask widespread misunderstanding of a paper's implications. Voters sometimes follow the lead of trusted experts on topics they don't fully understand. Problems surface later in CWG or during implementation.

(Matheus Izvekov)

---

13. Implementation Experience Reveals Design Flaws

Testimony from library implementers about actual implementation experience should heavily influence votes; negative implementation experience is strong evidence of design problems. Implementers discover practical issues that paper authors miss.

(Matheus Izvekov)

---

14. Libraries Enable Domain Focus

Libraries allow programmers to leverage expert implementations of common components, freeing them to focus their cognitive resources on their actual domain expertise. A music notation expert shouldn't spend time implementing binary search—they'll likely get it wrong.

(Dave Abrahams)

---

15. Legacy Accumulation Kills Languages

Languages that cannot shed legacy features eventually become too complex for anyone to master. C++'s inability to agree on evolution principles means it only grows, never simplifies. The standard is now thousands of pages long.

(Dave Abrahams)

---

16. Deliberate Instability Periods Enable Better Designs

New languages or major features benefit from explicit instability periods where breaking changes are expected. Swift's deliberate instability allowed the team to discover what worked before committing. C++ locks features immediately upon standardization.

(Dave Abrahams, Doug Gregor)

---

17. Simplicity Is Essential for Committee Acceptance

When presenting to a committee, complex or unfamiliar techniques will cause resistance even if they're sound. Reframing proposals in simple, familiar terms—making the unfamiliar feel familiar—is often necessary for acceptance.

(Dave Abrahams, Doug Gregor)

---

18. Political Deadlock Kills Good Proposals

When competing proposals from prominent figures reach deadlock, the committee often rejects both rather than choosing. The Concepts C++ rejection cost C++ a decade and lost Doug Gregor to Swift. Technical merit alone doesn't determine outcomes.

(Sean Parent)

---

19. Require Regularity for New Types

All new standard types should be regular (copyable, equality-comparable, with copies being equal) unless there is explicit, documented justification for deviation. Irregular types propagate complexity through all code that touches them.

(Sean Parent)

---

20. Ecumenical Leadership Creates Respect

Effective technical leadership means moderating discussions so all arguments are heard, remaining non-partisan toward solutions, and creating an environment of mutual respect. Beman Dawes's leadership style enabled productive collaboration and attracted high-caliber contributors.

(Dave Abrahams)

Significant Principles

21. The Standard Library Should Exemplify Good Practice

Standard library implementations should be readable exemplars of good practice, not baroque monuments to optimization. When `std::pair` requires 2000 lines of code, the language has failed pedagogically.

(Sean Parent)

---

22. Meta-Programming Is Implementation Detail, Not Interface

Meta-programming should be an implementation technique for library authors, not an exposed interface promoted to users. Stepanov himself says it was "a hack so he could implement generic programming"—not the goal.

(Sean Parent)

---

23. Offer Both Checked and Unchecked API Layers

Design APIs with dual layers—an unchecked low-level layer for trusted contexts and performance-critical paths, and a checked layer for adversarial inputs and safety-critical paths. Different users have different needs.

(Howard Hinnant)

---

24. Dangerous Tools Are Fine If Used Correctly

Features that can be misused are not inherently bad; the standard should provide powerful tools and trust users to learn correct usage. String_view can dangle, but that doesn't make it bad—it makes it a tool requiring skill.

(Howard Hinnant)

---

25. Evaluate ABI Breaks by Cost/Benefit, Not Automatic Veto

ABI stability concerns should trigger cost/benefit analysis, not automatic rejection. The SSO string transition was painful but worthwhile. Claims of "ABI break" should prompt analysis, not shutdown.

(Howard Hinnant)

---

26. Discovery Over Invention

The best technical work uncovers existing truths rather than creating arbitrary constructs. Frame questions as "What is the right answer?" rather than "Whose proposal should win?" This transforms arguments from adversarial to collaborative.

(Dave Abrahams)

---

27. Expertise Is Siloed Between CWG and EWG

Simultaneous scheduling of CWG and EWG prevents experts from contributing to evolution discussions, causing papers to arrive at CWG in suboptimal shape. Papers that pass EWG with near-consensus may still have fundamental issues.

(Matheus Izvekov)

---

28. Vocabulary Types Justify Standardization

Standard library additions are justified when they establish vocabulary types that enable interoperability between libraries; mere usefulness is insufficient justification. Standardization's value is coordination, not just provision of implementation.

(Matheus Izvekov)

29. Implementation Capacity Must Constrain Feature Adoption

The committee should throttle feature adoption based on available implementation capacity. Approving features faster than they can be implemented wastes committee resources and creates user expectations that can't be met.

(Matheus Izvekov)

---

30. Knowledge Is Lost in Committee Transitions

Rationale and evidence discussed orally in study groups and EWG sessions is often lost because it's not recorded in papers. Without recorded rationale, the committee cannot learn from past decisions.

(Matheus Izvekov)

---

31. Permissive Licensing Maximizes Adoption

If the goal is widespread adoption, licensing must impose minimal friction. The Boost Software License's permissiveness directly served Boost's mission. Every restriction reduces adoption; anything that makes lawyers nervous reduces use.

(Dave Abrahams, Sean Parent)

---

32. In-Person Collaboration Is Irreplaceable

Online collaboration can accomplish intellectual work, but in-person interaction is necessary for building the deep relationships and trust that sustain long-term collaboration. Schedule face-to-face meetings even when online would be "more efficient."

(Dave Abrahams)

---

33. Minimize Wording Changes in Proposals

When proposing changes to the standard, minimize the scope of wording changes. Small, focused changes are easier to review, less likely to introduce inconsistencies, and more likely to be accepted.

(Dave Abrahams)

---

34. Consensus Can Achieve Great Things

Consensus-based processes, when properly facilitated, can produce excellent technical outcomes—not diluted compromises but genuine discoveries of the best approach. Dave's exception safety work succeeded entirely through persuasion, without formal voting authority.

(Dave Abrahams)

---

35. Mentorship Is Informal but Essential

WG21 knowledge transfer happens through informal mentorship relationships, not formal onboarding. Newcomers must seek out mentors or learn through trial and error. Without deliberate mentorship, institutional knowledge is lost.

(Howard Hinnant)

---

36. Templates Lack Meaningful Type Checking

The compiler cannot verify template correctness for all instantiations; concepts help at call sites but don't provide definition-side checking. Users should minimize template usage accordingly—templates are expensive to compile and hard to debug.

(Matheus Izvekov)

---

37. Algorithms Are Central

Algorithms—explicit, understandable computations—should be the organizing principle of software design. Prefer explicit algorithms over implicit message-passing; code should be readable as a coherent computation, not a configuration of interacting objects.

(Dave Abrahams)

---

38. Libraries Preserve Languages During Stagnation

When language standardization stalls, high-quality open-source libraries can sustain a language ecosystem and seed future evolution. Boost kept C++ relevant during the 13-year gap between C++98 and C++11.

(Sean Parent)

---

39. Foundational Operations Should Be Language Features

Ubiquitous foundational operations (std::move, std::forward, potentially atomics) pay unnecessary compile-time costs as library templates; they should be language features for better performance and diagnostics.

(Matheus Izvekov)

---

40. Infrastructure Is Undervalued

Libraries are infrastructure with no direct financial incentive for creation or maintenance, yet they enable all other development. This structural underinvestment must be addressed—critical libraries maintained by single volunteers represent a market failure.

(Dave Abrahams)
