Document	D0000
Date:	2026-01-07
Reply-to:	Vinnie Falco <vinnie.falco@gmail.com>
Audience:	WG21

Capturing Living Knowledge in WG21

Abstract

WG21's founding generation holds irreplaceable knowledge about C++ standardization—not just *what* decisions were made, but *how* to make good decisions. This knowledge exists in the minds of experts who built the committee's culture over thirty years. It has not been effectively transmitted to newer participants because WG21's documentation transmits *conclusions* rather than *judgment*.

This paper proposes a solution: an agentic workflow to capture living knowledge through structured interviews with WG21 experts. The window of opportunity is finite, and none of us are getting younger. But the knowledge is alive, the experts are willing, and the technology to capture and synthesize their insights now exists. We can preserve this institutional wisdom for future generations.

Call to action: Join the paper authors by volunteering to be interviewed.

1. The Knowledge Transmission Challenge

C++ is infrastructure. It runs your phone, your car, your bank, your hospital. The language that powers civilization's critical systems deserves a standards committee that can maintain it for generations. Preserving the committee's accumulated wisdom is essential to that mission.

1.1 What We Mean by "Living Knowledge"

Samo Burja's *Great Founder Theory* provides a useful framework for understanding knowledge transmission in institutions. A **tradition of knowledge** is a body of understanding that has been consecutively worked on by multiple generations. Such traditions can exist in three states:

- **Living tradition**: Knowledge successfully transferred with comprehension
- **Dead tradition**: External forms transferred (documents, procedures), but not the underlying understanding
- **Lost tradition**: Not transferred at all

WG21 has procedures, documents, and voting records—the external forms survive. But do newer participants truly comprehend *why* certain proposals succeed and others fail?

> *"A living tradition of knowledge is a tradition whose body of knowledge has been successfully transferred, i.e., passed on to people who comprehend it."*

The challenge facing WG21 is keeping its tradition *living*—ensuring that the deep understanding of C++ standardization passes to each new generation of participants, not just the paperwork.

1.2 Tacit Knowledge: What Can't Be Written Down

Much of what experts know is **tacit knowledge**—understanding that cannot easily be transmitted through written documentation. A master craftsman knows things about their trade that they've never articulated, perhaps never *could* articulate. They demonstrate judgment in novel situations that no rulebook could anticipate. WG21's founding-era experts developed this judgment through decades of direct engagement with the language's evolution—experience that cannot be conveyed through SD documents.

> "*Since it cannot be easily transferred via texts, tacit knowledge must be taught via direct practice and extensive interaction with a skilled practitioner. Traditional master-apprentice relationships are the gold standard for these training relationships... Otherwise, the knowledge simply isn't transferred, and with many crafts, is lost forever.*"

WG21 has no formal apprenticeship structure. When Howard Hinnant evaluates whether a library proposal has sufficient field experience, he's drawing on decades of tacit knowledge about what has worked and what has failed—knowledge he developed through implementation experience, not committee documents. When Bjarne Stroustrup assesses whether a language feature maintains coherence with C++'s design philosophy, he's applying judgment that cannot be reduced to a checklist.

This tacit knowledge is WG21's most valuable asset. It guides the committee toward good decisions and away from bad ones. And it lives primarily in the minds of people who have been doing this work for decades.

1.3 The Succession Problem

Every institution faces what Great Founder Theory calls the **succession problem**: how to transfer both authority and capability from one generation to the next. WG21 is no exception—as founding-era participants reduce their involvement, who will carry forward their accumulated wisdom?

> "*The succession problem has two components: power succession (handing off the reins of the institution, keeping it piloted) and skill succession (transferring the skill needed to pilot the institution well, keeping it a live player).*"

WG21 has handled power succession reasonably well—the formal mechanisms work. The Convener position has transitioned, chairs rotate, meetings happen, standards ship. But skill succession—transferring the tacit knowledge of good standardization—has been more challenging. The committee can continue operating indefinitely, but can it continue making *good decisions*?

> "*If power succession is successful but skill succession is not, then the institution remains piloted, but not a live player. Someone is at the controls, but they don't really know how to use them.*"

This is the risk WG21 faces: a committee that follows procedures correctly while gradually losing the judgment that made the procedures meaningful.

1.4 WG21 Transmits Conclusions, Not Judgment

WG21 has created substantial documentation. The question is what that documentation transmits.

Procedural Documents tell participants *how the committee operates*:

Document	Knowledge Transmitted
SD-3, SD-4, SD-5	Procedure, logistics
SD-7	Formatting, submission
How to Submit a Proposal	Process steps

Policy Documents record *what decisions the committee has made*:

Document	Knowledge Transmitted
SD-8	What breaking changes WG21 reserves
SD-9	Default positions on `[[nodiscard]]`, `noexcept`, etc.
SD-10	References to principles (briefly)

Direction Documents articulate *high-level philosophy*:

Document	Knowledge Transmitted
P2000	Vision, priorities
P0592	Feature roadmaps

The pattern is clear: **WG21 transmits conclusions but not judgment**.

Consider SD-9, which says things like "use `[[nodiscard]]` for functions where ignoring the return value is always a bug." This is a *conclusion*—a design decision. It doesn't teach:

- How to recognize such functions
- How to make this judgment in ambiguous cases
- The underlying principle (error handling philosophy, RAII patterns)

SD-10 comes closest to real knowledge transfer by referencing "Design and Evolution of C++" principles. But the references are brief, newcomers may not have read D&E, and there's no explanation of how to apply principles to novel cases.

P2000 articulates philosophy—valuable for understanding committee goals. But it doesn't teach *how to evaluate* whether a proposal meets those goals.

What's missing:

- **Judgment training**: How to *think* about API design, not just what conclusions to reach
- **Generating principles**: The deep reasoning that allows extending the tradition faithfully
- **Failure case studies**: Why certain historical features failed, and how to recognize similar patterns
- **Verification mechanisms**: How to check whether understanding is genuine

When a new vocabulary type comes to LEWG, the question isn't just "does this follow SD-9?" but "does this belong in the standard at all?" That judgment requires understanding the *generating principles* behind C++ standardization.

> *"Someone who understands the generating principles of a tradition will be able to verify or check their knowledge, but, more importantly, they will also be able to extend it while remaining faithful to the original body of knowledge."*

Without these principles, newer participants can only imitate past decisions—they cannot reason about genuinely novel cases. The generating principles of C++ standardization—why certain things belong and others don't—exist in the minds of founding-era participants, undocumented.

1.5 Case Study: The `[[nodiscard]]` Policy Papers

SD-9: Library Evolution Policies represents a well-intentioned attempt to capture policy knowledge. Yet the document dates from mid-2024 and contains only one item. The supporting papers reveal the challenge of transmitting judgment even when people consciously try.

The foundational paper P2267R1: Library Evolution Policies captures *process*—how policies are adopted—while P3201R1: LEWG [[nodiscard]] policy captures only *outcome*—what the policy says. Neither transmits the judgment needed to apply the policy to novel situations.

Two rationale papers attempt deeper knowledge transfer, with instructive differences in success:

P3162R0 (Neațu/Sankel) does reasonably well at conveying philosophy. The three guiding principles—minimize complexity, focus on the 90% case, center on outcomes—are genuinely transferable mental models. A reader could apply "minimize complexity" to reject a proposal that marks every getter `[[nodiscard]]`. The "90% case" principle teaches prioritization: catch the severe/common bugs, accept that edge cases slip through. "Center on outcomes" reframes the question from "what does the standard say" to "what diagnostics do users actually see"—a useful perspective shift.

The concrete examples (`.empty()` vs `.clear()` confusion, `async` synchronous trap, allocation leaks) give pattern-matching anchors. Someone could recognize analogous situations: a `reset()` that returns the old value might warrant `[[nodiscard]]` under the "common mistake" criterion.

However, the paper doesn't fully explain *why* these three principles were chosen over others, or how to resolve conflicts between them.

P3122R1 (Wakely) is weaker on tacit knowledge despite being more operationally precise. The core insight—"implementors know best, stop micromanaging"—is valuable but presented as assertion rather than derived from principles. The paper gives excellent *what* (the proposed wording categories) but limited *why* for each category. Why are comparison operators special? Why does the const/non-const distinction matter? The examples illustrate but don't teach the underlying reasoning.

The wording itself is a checklist, not a framework. A reader could apply it to existing standard library functions but would struggle with genuinely novel patterns. If someone invents a new abstraction that doesn't fit the bullets, they have no tools to reason about it.

What's missing from both:

Neither paper addresses the deeper question: what is `[[nodiscard]]` *for*? Is it a bug-catching mechanism, a documentation tool, an API design signal, or a way to encode programmer intent? The answer affects everything. If it's purely about catching bugs, frequency data matters. If it's about intent, consistency matters more.

Neither discusses the cost model for false positives. When does an unwanted warning become actively harmful? This omission means readers can't calibrate the tradeoff.

Neither provides negative examples—cases where `[[nodiscard]]` was added and later regretted, or where reasonable people disagreed. Learning from mistakes transfers tacit knowledge better than success stories.

Assessment: P3162R0 transfers approximately 60% of the tacit knowledge needed to reason independently about new cases. P3122R1 transfers approximately 30%, functioning more as a reference specification than a teaching document. Neither fully equips a reader to handle genuinely novel situations or to argue persuasively for a position the papers don't explicitly cover.

This case study illustrates why documentation alone—even good-faith rationale documentation—cannot fully capture the judgment that experienced committee members apply. The missing 40-70% is precisely what interviews could capture.

---

2. The Knowledge Exists

The good news is that the knowledge we need to capture demonstrably *exists*. Long-time committee participants possess deep understanding of WG21's principles and history. Their reflections reveal exactly the kind of tacit knowledge that should be preserved.

> **Note** The isocpp reflector is private, and quotes are provided by permission from the authors.

2.1 Bjarne Stroustrup on Coherence

Source: `lib/2016/11/1321.php` (November 9, 2016) **Context:** Post to the library reflector about C++17 and committee direction

> *That said, I worry about the future of C++. Those of you who saw my CppCon keynote or have participated in some of the WG21 evening sessions know that I'm concerned about our lack of direction.*
>
> *My main concern is that we are delaying decisions about major issues while moving quite fast with minor proposals that do not appear to be strongly connected to the rest of the language. The effect is to complicate the language with many minor conveniences (each helping someone but most not affecting the majority of C++ developers) while repeatedly rehashing major proposals (complicating parts, rejecting other parts, and failing to see connections between proposals brought forward in separate papers). This does not lead to a simpler and more coherent language as seen by developers. In fact, many developers fear the work of the committee as adding complexity without benefits. I find it far easier to allay such fears presenting major features delivering major benefits than by listing lots of little features – however beloved by C++ language and library experts.*
>
> *Please consider the possibility of a moratorium on new minor proposals until we have decided a set of major features that we should aim to include in C++20. This would avoid the problem we had with many significant proposals "almost ready" for C++17 while our efforts diffused on minor proposals and debates about novel alternatives to the major proposals. Ideally, we would have a vote on that set of features at the next meeting (Kona), so that we would have 3 years to integrate them into the standard and try them out.*

> *...We should aim at a coherent set of features and be able to lay out a (not too detailed) roadmap for ourselves and for the C++ community. The community wants to know that we are making significant progress and where we are going. It is important to them.*

The founder of C++ sees when coherence is being lost—when proposals don't connect to the language's underlying design philosophy. Note that Bjarne has to *ask* for a moratorium, *suggest* focusing on major features, *propose* a roadmap. If skill succession had worked, the committee would already be doing this. The fact that the founder has to intervene to remind the committee of its purpose is diagnostic. This judgment, honed over four decades, can be captured through structured conversation.

2.2 Howard Hinnant on Field Experience

Source: `lib/2016/07/0664.php` (July 22, 2016) **Context:** Discussion about std::variant lacking field experience before standardization

> *I think we (collectively) should either have an all-out push to publish an open-source C++17 variant implementation 2 weeks ahead of the nearest national body comment deadline, or we should all write strongly worded national body comments to remove variant from the CD for lack of field experience. This comment of mine is actually far more liberal than I am comfortable with. A more conservative viewpoint (with which I would not disagree) is to pull variant regardless, for lack of field experience.*
>
> *The last time we went down this road, the proposers managed to write an implementation of their proposal mere hours prior to the vote, and giddily announced that their proposal was implemented at the vote. That announcement swayed the committee to vote in favor. The result was just yet another example of our collective failure. An implementation needs positive field experience prior to standardization.*
>
> *I should quit asking: 'Has it been implemented?' The correct question is: What has been the field experience? Is there positive feedback from anyone outside your immediate family or people who could have a perceived conflict of interest (such as employees of your company)? Having your Mom and your direct reports say the proposal is wonderful is nice, but not sufficient.*
>
> *Doing otherwise is an affront to proposals such as Filesystem which has gone through years of scrutiny of an actual (widely-used) implementation.*

Howard is articulating a **generating principle**—the deep reasoning that should guide standardization decisions. He's not just saying "we need implementations"; he's explaining *why* and *what kind* of validation matters.

Source: `lib/2016/07/0694.php` (July 25, 2016) **Context:** Continued discussion about variant

> *It is now my understanding that we (the committee) have made significant design changes with respect to all existing variants in the field (most notably boost::variant). We /think/ these are good changes, and I /hope/ that we are right. We won't /know/ if these are good design changes until we have field experience with an implementation that implements these design changes.*
>
> *Currently we are on track to standardize something where the only field experience is with something with significant design changes relative to our spec. I won't bother to go down the list of libraries where this strategy has failed us in the past. It is hard enough to get something right /with/ a little field experience.*

> *The reason we have the TS system is to handle exactly this situation.*
>
> *Now I can only blame myself. I did not get myself to Oulu (this time it was a wedding instead of a funeral — a much better reason!). And if I had, I wasn't really following variant so closely as to know that there wasn't at least one implementation tracking at least the major design changes over the years. Perhaps I could have asked that question in full committee prior to the vote (if I had been there). Perhaps the answer would not have swayed any vote but my own.*

"I won't bother to go down the list of libraries where this strategy has failed us in the past." This is **tacit knowledge**—Howard knows which library standardizations failed and why, but he doesn't enumerate them. The knowledge lives in his head. When he retires or reduces participation, that list goes with him.

Also note Howard's self-blame: "I can only blame myself. I did not get myself to Oulu." The system depends on specific experts being physically present to ask the right questions. When they're absent, the questions don't get asked. This is not a robust institution.

2.3 Nico Josuttis on Integration Time

Source: `lib/2016/05/0304.php` (May 17, 2016) **Context:** Discussion about string_view integration in C++17

> *It seems what I feared has happened: The train model has failed, giving not enough time for integration. (No serious software project would add features with wide impact without giving time for integration and maturity. We gave less than a week for integration.)*
>
> *If we continue, c++17 will de-facto become a beta release for the next major release (if we don't make the same mistakes again...)*
>
> *Maybe it's time to feature freeze c++18 NOW with some additional language support such as concepts.*
>
> *Just my thoughts, but I really don't know now which story to tell about C++17...*

Nico observes that process is outpacing judgment. The train model (fixed release schedule) creates pressure to ship features before they're integrated—before experts can evaluate how they interact with the rest of the standard. "I really don't know now which story to tell about C++17"—the expert has lost the narrative. He can't explain what the committee is doing in terms that make sense.

Source: `lib/2016/05/0311.php` (May 18, 2016) **Context:** Discussion about whether to standardize string_view given unresolved design questions

> *I fear we have to decide now between one of three options regarding string_view. And in essence, the key question is WHICH MAJOR PURPOSE DOES string_view HAVE?*
>
> *EITHER:*
>
> *1 Decide that the major purpose of string_view is to represent non-null-terminated strings...*
>
> *OR:*
>
> *2 Decide that the major purpose of string_view is to become THE standard type for interfaces who want to process string-like objects...*
>
> *OR:*

> *3 If we can't decide, skip standardizing string_view at all. - Yes, we rarely do that, but*
>
> - auto_ptr and async() should be lessons learned... 😉*

Nico explicitly names **specific past failures**—`auto_ptr` and `async()`—as cautionary examples. This is the oral tradition in action: an expert invoking historical precedent to guide current decisions.

But notice the "😉"—he's half-joking, suggesting that invoking these lessons is unusual, perhaps even quaint. The committee "rarely" decides not to standardize something. The process has momentum; the default is to ship. There's no formal mechanism for error correction. The knowledge of *why* those features failed isn't documented anywhere.

2.4 Alisdair Meredith on Cultural Knowledge

Source: `lib/2016/09/1034.php` (September 9, 2016) **Context:** Asked why `bad_optional_access` inherits from `logic_error` but `bad_variant_access` doesn't

> *Lack of cultural knowledge.*
>
> *For C++98 there was a conscious effort to design a coherent exception hierarchy, with a distinction between runtime and logic errors in the library, and more primitive exceptions deriving directly from std::exception carrying the minimal payload - just a vtable pointer, with 'what' returning a string literal in each case.*
>
> *For C++11, we added a few new library exceptions for types added in TR1, and they followed the idiom of deriving directly from std::exception, rather than the more appropriate-looking std::runtime_error. I queried this at the time, I think I filed an NB comment, but the conclusion was that runtime/logic_error were failed experiments, and we should stop doing that. The exception to the rule being when you really did have more information to return in the exception object, when the new system_error type would be appropriate.*
>
> *For the Library Fundamentals TS, there does not appear to have been a consistent policy applied, and I am not sure if LEWG are aware of the history, or have an intentional exception model in mind when they review libraries.*
>
> *It would certainly be worth filing an NB comment on the topic again, to achieve some clarity on how the group feels...*

This is precisely the kind of knowledge that needs to be captured. An LWG expert explicitly states that LEWG "may not be aware of the history"—the design decisions that should inform their reviews exist only in the minds of founding-era participants. The exception hierarchy decision was made, recorded nowhere, and now LEWG makes new decisions without knowing the precedent. Alisdair holds this knowledge—and can share it.

2.5 The Pattern

These quotes demonstrate that **the knowledge is alive in experts' minds**:

Expert	Knowledge Held
Bjarne Stroustrup	Design coherence judgment, connection between proposals and philosophy

Expert	Knowledge Held
Howard Hinnant	List of library standardization failures, field experience principle
Nico Josuttis	Integration requirements, specific failure case studies
Alisdair Meredith	Exception hierarchy history, design decisions not documented

A participant can attend every meeting, follow every procedure, vote on every poll—and still lack genuine understanding. They know *what* the committee does without knowing *why*.

> "*Students of a tradition can appear to possess understanding of a tradition's body of knowledge despite actually lacking it. This is counterfeit understanding.*"

The risk facing WG21 is not that knowledge is lost entirely—it's that it remains locked in the minds of a few experts while newer participants develop counterfeit understanding: knowing the procedures without grasping the principles that make them meaningful.

---

3. Proposed Solution: Agentic Knowledge Capture

The knowledge exists. The experts are available. What's needed is a systematic method to extract, preserve, and transmit their tacit understanding.

3.1 The Interview-Based Approach

We propose an **agentic workflow** to capture living knowledge through structured interviews with WG21 experts:

1. **Structured interviews**: Conduct recorded conversations with experienced committee members, guided by questions designed to elicit tacit knowledge
2. **AI-assisted transcription**: Use modern speech-to-text to create accurate transcripts
3. **Knowledge synthesis**: Apply AI tools to identify patterns, extract principles, and organize insights
4. **Expert review**: Have interviewees review synthesized content for accuracy
5. **Publication**: Make captured knowledge available to the committee and broader C++ community

3.2 Why Interviews Work

Written documentation struggles to capture tacit knowledge because experts often can't articulate what they know until prompted by specific situations. Interviews solve this by:

- **Providing concrete scenarios**: "When you saw proposal X, what made you concerned?"
- **Allowing follow-up**: "Can you give another example of that pattern?"
- **Capturing stories**: Narratives about specific decisions and their outcomes
- **Revealing judgment**: How experts approach novel situations

Left to natural processes, knowledge degrades with each generation. WG21 cannot assume that informal mentorship and meeting attendance will preserve its institutional wisdom.

> "*Errors in transmission from one generation to the next are almost guaranteed and thus require proactive measures to correct them and maintain the fidelity of a tradition.*"

Interviews are precisely such a proactive measure—a deliberate intervention to capture knowledge before transmission errors compound into institutional amnesia.

3.3 Interview Guide

Effective interviews elicit tacit knowledge that experts may not realize they possess. The key is drawing out *stories*—specific narratives about decisions, problems, and outcomes. Learning from mistakes transfers tacit knowledge better than success stories, though both have value.

Interview Technique

Start with a general question, then use the response to drill down into a relatable story:

1. **Open broadly**: "What do you look for when evaluating a library proposal?"
2. **Identify a thread**: Listen for hints of specific experiences
3. **Request the story**: "Can you tell me about a time when that principle mattered?"
4. **Explore the details**: What happened? What did you learn? What would you do differently?

Example Questions

For failure stories (highest value for knowledge transfer):

- "Tell me about a time you had a very difficult decision on a proposal. What made it hard? How did it turn out?"
- "What's a feature that got standardized that you wish hadn't? What went wrong in the process?"
- "Tell me about a proposal that looked good on paper but failed in practice. What did we miss?"

For success stories:

- "What's a proposal that went through the process well? What made it work?"
- "Tell me about a time the committee correctly rejected something. How did people know?"

Topic Areas to Explore

Generating Principles

- What coordination failures justify standardization?
- When is an external library sufficient?
- How do we evaluate vocabulary necessity vs. utility convenience?
- What perpetual costs does standardization impose?

Design Philosophy

- What is the relationship between performance and interface standardization?
- When do we prioritize theoretical consistency vs. practical utility?
- How do we balance backwards compatibility against fixing mistakes?

Evaluation Frameworks

- What questions should every paper answer?
- What evidence demonstrates genuine demand?
- How do we distinguish proposer-driven from user-driven proposals?

Case Studies

- Which standardized features proved problematic, and why?
- Which rejected proposals were correctly rejected?
- What patterns indicate a proposal needs more work?

3.4 How AI Enables This Now

Modern AI capabilities make this project feasible in ways that weren't possible even a few years ago:

- **High-quality transcription**: Accurate speech-to-text for technical conversations
- **Synthesis across interviews**: Identifying common themes and principles from multiple sources
- **Structured extraction**: Converting narrative knowledge into organized frameworks
- **Accessibility**: Making captured knowledge searchable and navigable

We have developed an agentic knowledge extraction framework (WG21_CAPTURE_RULE.md) that processes interview transcripts and produces structured output distinguishing:

- **Principles**: Distilled, actionable rules that can be applied to evaluate new proposals
- **Experiences**: Supporting stories that illustrate and validate the principles

The output is dual-purpose: human-readable markdown for expert review and verification, with embedded metadata enabling downstream agentic processing. Each principle includes "When to Apply" conditions and "Red Flags" for violations; each experience links back to the principles it supports.

The technology to preserve WG21's institutional wisdom exists today.

3.5 Does This Devalue Expert Knowledge?

If AI can help capture and synthesize knowledge, does expertise become less valuable?

The economics of generative AI fundamentally invert traditional cost functions. Before LLMs, production was expensive—transcribing interviews, synthesizing themes, drafting documents required skilled human time. Judgment was comparatively cheap—a quick review, an approval, a correction. After generative AI, this inverts: production becomes nearly free (AI transcribes, synthesizes, and drafts at marginal cost), while judgment becomes the scarce resource (requiring irreducible human attention).

Generative AI doesn't devalue expertise—it *reveals* that judgment was always the valuable part, while production was just the tax experts paid.

In the generative AI economy, **human attention becomes the scarcest and most valuable resource**. Everything else scales; attention doesn't. The institution that captures and allocates expert attention most efficiently wins. Agentic knowledge capture is, fundamentally, an *attention allocation system*—routing the right questions to the right experts, filtering noise from signal, and ensuring that when a human reviews synthesized output, it matters.

This reframes any concern about displacement:

- **Comparative advantage shifts**: Experts focus on judgment rather than production. Howard Hinnant's value lies in knowing which library proposals lack sufficient field experience, not in typing out his reasoning. The AI handles transcription and synthesis; the expert provides the irreplaceable judgment.

- **Capability expansion**: More people can contribute meaningfully. An expert who might never write a paper can share insights through a one-hour interview. The total knowledge captured increases even as individual time requirements decrease.

The economics are clear: judgment is now the bottleneck owned by experts. This project doesn't diminish their role—it amplifies it.

---

4. Experimental Evidence

To demonstrate what captured knowledge looks like, we interviewed a few prominent members of the standardization committee, who possess deep knowledge of the language and extensive experience. Our methodology consisted of audio-only interviews with AI-assisted captured transcriptions. These transcriptions were processed using an agentic workflow to create a synthesis of distilled knowledge. We present the distilled knowledge below. The full transcripts, synthesized knowledge files, and the agentic extraction rule are available in the GitHub repository.

> ⚠ **Important Disclaimer**: The summaries and extracted principles presented below are **AI-generated syntheses** and should not be taken as absolute reflections of the interviewees' views. AI transcription can introduce errors, and AI synthesis can misinterpret meaning, lose nuance, or emphasize points differently than the speaker intended.
>
> **A production-ready process must include human review at each stage:**
>
> 1. **Transcript review**: Interviewees should review transcripts to correct transcription errors and clarify ambiguous passages
> 2. **Knowledge review**: Interviewees should review the synthesized principles and experiences to verify they accurately represent their views and intended meaning
> 3. **Evaluation review**: Any proposal evaluations generated from the knowledge base should be reviewed by humans before being acted upon
>
> The outputs presented here are **preliminary demonstrations** of what the workflow can produce. They have not yet undergone the full interviewee review process that a mature methodology would require. We present them to illustrate the potential of the approach, with the understanding that AI-assisted synthesis requires human verification to ensure fidelity to the source.

4.1 Matheus Izvekov

Matheus Izvekov brings a rare perspective to WG21: deep compiler implementation expertise in templates, overload resolution, and partial ordering—areas where most original experts have retired or become inactive. His experience reveals a gap between how papers pass through EWG (often without deep technical understanding from voters) and the hard realities discovered later in CWG or during implementation.

His central insight is that **high vote counts in EWG do not necessarily indicate understanding**. His first paper achieved near-consensus, yet later revealed that voters hadn't truly grasped the implications—a pattern he believes can be a problem, based on his personal experience with a small sample size. This disconnect stems from the separation of experts (concentrated in CWG) from the design phase (EWG), compounded by simultaneous scheduling that prevents cross-pollination. Importantly, this is not a flaw in the committee per se—it's unreasonable to expect everyone to be an expert in everything, especially given C++'s complexity.

Voters often follow the lead of trusted experts on topics they don't fully understand, which is a reasonable trust mechanism.

Matheus also articulates a tension between library and language features: foundational operations like `std::move` pay unnecessary compile-time costs as templates when they could be cheaper and better-diagnosed as language primitives. He warns that the committee's preference for library solutions may be counterproductive when language features would provide a more polished user experience.

Key Insights:

- Near-unanimous EWG votes can mask widespread misunderstanding of a paper's implications; voters sometimes follow trusted experts on topics they don't fully understand—a reasonable trust mechanism, but one that can obscure whether genuine comprehension backs the vote
- Features that bypass EWG review risk incomplete specifications that only surface during implementation—potentially years later
- The simultaneous scheduling of CWG and EWG prevents experts from contributing to evolution discussions, causing papers to arrive at CWG in suboptimal shape
- The committee should throttle feature adoption based on available implementation capacity; approving features faster than they can be implemented wastes resources
- Rationale discussed orally in study groups is often lost because it's not recorded in papers; however, requiring authors to document every objection may be counterproductive—papers typically address concerns with measurable consensus impact

References: Full transcript · Synthesized knowledge

4.2 Howard Hinnant

Howard Hinnant brings nearly three decades of WG21 experience, beginning as MetroWorks' sole standard library implementer in 1998 and later serving as LWG chair starting in 2005. His perspective is grounded in implementation reality: he has personally shipped standard library code and experienced the consequences of committee decisions at the vendor level.

His most powerful insight is the **standardization threshold principle**: the standard should make the impossible possible or the hard easy, but not the easy easier. This filters out "convenience" proposals that add maintenance burden without solving real problems. He couples this with an unwavering requirement for **positive field experience**—proposals without real-world validation are gambling with the standard's long-term quality.

Howard also articulates the fundamental structural limitation of WG21: it's a volunteer organization lacking executive power. No one can compel work to be done, which means important but contentious proposals (like his hash improvements) can simply be abandoned when participants tire of arguing. This explains why obvious needs sometimes go unaddressed for years.

Key Insights:

- Only standardize features that enable what was previously impossible/impractical, or that significantly reduce difficulty of hard tasks; reject proposals that merely add convenience
- Proposals must demonstrate successful real-world usage with positive feedback from independent users before standardization; implementation alone is insufficient

- WG21 is a volunteer organization without executive authority; no one can compel work to be done, so important proposals may be abandoned when champions lose interest
- Every proposal must clearly answer: what specific problem does this solve, and without this proposal, how hard is the problem to solve?
- ABI stability concerns should trigger cost/benefit analysis, not automatic rejection; some ABI breaks are worth the transition cost

References: Full transcript · Synthesized knowledge

The Power of Stories

When asked about `string_view`'s reputation as "dangerous" (it can dangle), Howard offered this:

> *"Any good tool is dangerous. I mean, I'm a big fan of pocket knives and kitchen knives... they can really help in the kitchen or they can chop your fingers off. You just have to know how to use them."*

This is the kind of insight that no policy document could capture. The knife analogy is visual, memorable, and instantly teachable—a newcomer hearing it once will remember it forever. It conveys not just a position (dangerous tools are acceptable) but a *philosophy* (C++ trusts its users; power matters more than protection from misuse).

The storytelling format of interviews yields surprisingly sharp insights precisely because experts think in stories. When Howard reaches for the knife analogy, he's accessing decades of accumulated judgment compressed into a vivid image. Written specifications capture *what*; stories capture *why* and *how to think*.

4.3 Dave Abrahams (From Existing Documentary Footage)

Dave Abrahams was present at Boost's founding, instrumental in establishing its peer review culture, and deeply involved in standardization efforts—most notably bringing exception safety guarantees to the C++ standard library. His interview was not conducted by the paper authors; it was recorded as part of a separate documentary project on Boost's history. This demonstrates a powerful extension of the knowledge capture workflow: **existing video content can be processed through the same agentic pipeline**.

YouTube alone contains thousands of hours of conference talks, panel discussions, and interviews with WG21 participants. Much of this content has never been systematically processed for knowledge extraction. The same workflow that synthesizes knowledge from purpose-conducted interviews can be applied to this vast archive—transcribe the audio, run the capture rule, and produce structured knowledge files.

Dave's central contribution is articulating the **philosophy of libraries as infrastructure**. Libraries enable domain experts to focus on their actual expertise rather than reinventing fundamental components. "Every time [programmers] have to go on an excursion to build an algorithm that is standard or a data structure... they're doing something that there's economic pressure on them not to give it the attention it deserves." This principle—that libraries free cognitive resources—explains why standardization matters: it shifts work from thousands of individual programmers to a few experts who can give components the attention they deserve.

He also provides a foundational account of how consensus-based collaboration can achieve great things. His exception safety work succeeded despite him not being a committee member with voting rights—he achieved it entirely through persuasion and education. "What the committee demonstrated for me at that time was, contrary to what a lot of people say about design by committee... you can actually accomplish great things by consensus."

Crucially, Dave also articulates why he eventually left C++: the language's accumulated legacy made it unable to serve his mission of "empowering programmers." The standard grew to thousands of pages; the committee couldn't agree on evolution principles; complexity accumulated without corresponding simplification. This cautionary perspective—from someone who helped build C++'s success—is precisely the kind of hard-won wisdom that should inform current standardization decisions.

Key Insights:

- Standardize only components that have demonstrated real-world success through actual use, not theoretical designs; the committee's straying from this principle for STL template features was "a black eye for the committee"
- Libraries provide more than code—they provide paradigms and conventions that eliminate entire categories of design decisions for users
- Consensus-based processes, when properly facilitated, can produce excellent technical outcomes; focus on education and understanding rather than political maneuvering
- Effective technical leadership means moderating discussions so all arguments are heard, remaining non-partisan toward solutions; Beman Dawes's ecumenical leadership style created a level of respect that attracted high-caliber contributors
- Languages that cannot shed legacy features eventually become too complex for anyone to master; C++ "has become complex and difficult to the point of impracticality"
- The best technical work uncovers existing truths rather than creating arbitrary constructs; approach problems as exploration toward the right answer, not competition between proposed solutions

References: Full transcript · Synthesized knowledge

4.4 Sean Parent (From Existing Documentary Footage)

Sean Parent, a senior principal scientist at Adobe who previously worked at Apple and Google, offers a unique perspective spanning three decades of C++ infrastructure work. Like Dave Abrahams, his interview comes from the same documentary project on Boost's history, further demonstrating how existing video content can feed the knowledge capture pipeline.

Parent's central contribution is framing C++ as a **Shakespearean tragedy**—a protagonist whose virtues have been undermined by accumulated complexity and institutional failures. In his "Tragedy of C++" keynote, he presents `std::pair` as exhibit A: "std::pair within the standard library is something close to 2000 lines of code. It's ridiculous... if your example for how to write a pair of two values is 2000 lines of code, the complexity is more than most reasonable developers can digest." When the canonical implementation of the simplest possible composite type is incomprehensible, the language has failed pedagogically.

He provides crucial historical context on Boost's role in keeping C++ alive. The 13-year gap between C++98 and C++11 could have killed the language through stagnation: "I think C++ in some sense would've greatly diminished during that time period just from stagnation." Boost filled this void, and many of its innovations became C++11 features.

Parent also drove the **licensing consolidation** that enabled commercial adoption. Early Boost had over 100 different licenses—one per contributor—creating enormous friction for enterprise legal review. He pushed Dave Abrahams to establish the unified Boost Software License, making Adobe the first major company to officially adopt Boost.

Most critically, Parent diagnoses the standards committee's systemic dysfunction: "Every decision that the standards committee tends to make tends to almost be made in isolation. And they don't then document the rationale for that decision. And so when a similar decision is made, they may come up with a different answer." This lack of documented design principles is, in his view, the root cause of C++'s accumulated inconsistencies and complexity.

Key Insights:

- High-quality open-source libraries can sustain a language during standardization stagnation and seed future evolution; Boost's role in the "lost decade" was existential, not merely convenient
- License fragmentation kills commercial adoption even for technically excellent libraries; consolidating to a single, commercially-friendly license removes friction that legal departments use to say "no"
- The standard library should be readable exemplars of good practice, not baroque monuments to optimization; when trivial types require thousands of lines, the language has failed
- Standards bodies must document rationale for decisions and maintain explicit design principles; without this, similar questions get inconsistent answers and languages accrete contradictions
- Meta-programming should be implementation technique for library authors, not exposed interface; Stepanov always viewed it as "a hack so he could implement generic programming"
- Political deadlock between prominent figures can kill technically sound proposals; the Concepts C++ rejection cost C++ a decade and lost Doug Gregor to Swift
- C++ has over-focused on micro-optimizations while leaving 99%+ of modern hardware performance (GPU, SIMD) inaccessible through standard means

References: Full transcript · Synthesized knowledge

Independent Validation of the Knowledge Capture Thesis

Sean Parent's critique of committee dysfunction is, in effect, an independent call for the solution this paper proposes. Without knowing the paper's thesis, he diagnoses exactly the problem that systematic knowledge capture addresses:

> "*The biggest problem with the standards committee is an unwillingness to specify what the goals are and basically what the rules are... every decision that the standards committee tends to make tends to almost be made in isolation. And they don't then document the rationale for that decision. And so when a similar decision is made, they may come up with a different answer.*"

This is a direct call for explicit knowledge capture. The "goals," "rules," and "rationale" he wants documented are precisely the Principles this workflow extracts. He even provides a concrete example of what a captured principle should look like:

> "*Even what I would consider basic things like saying all new types within the language should be regular, uh, doesn't happen and doesn't get held up.*"

"All new types should be regular" is exactly the form of actionable, verifiable principle that the capture rule produces. Parent is describing, from painful experience, the absence of the very artifact this paper proposes to create.

4.5 Dave Abrahams & Doug Gregor (Dinner Reunion)

The most candid interview in this collection comes from an unexpected format: a verite dinner conversation between Dave Abrahams, Doug Gregor, and their spouses, reuniting after years of estrangement following tensions at Apple. This documentary footage captures the kind of unguarded reflection that formal interviews rarely achieve.

The conversation centers on what may be C++ standardization's most instructive failure: the **original Concepts proposal**. Doug Gregor spent years implementing Concepts in GCC, traveling repeatedly to Texas A&M to negotiate a unified design with Bjarne Stroustrup, and shepherding hundreds of pages of proposals through the committee. The proposal was formally accepted and merged into the draft standard. Then, approximately one week before the Frankfurt meeting in 2009, Bjarne sent an email withdrawing his support unless incompatible changes were made. Years of work were undone.

Doug's reflection captures the damage: "I knew how much effort it took to get there. And I knew it could disappear on the whim of one person. One bad email chain. And it's hard to personally invest in something that can go away like that." This experience drove both him and Dave away from C++ language work entirely.

The core lesson they articulate is that **standards committees must standardize existing practice, not invent**: "Standards committees should not invent per se, they should take what is known to work well and enshrine it so everyone can build on it." Boost succeeded precisely because it provided a proving ground outside the committee where designs could be tested extensively before standardization. The Concepts proposal failed because—despite having an implementation—it lacked broad real-world validation from independent users.

They also describe carrying these lessons forward to Swift. Chris Lattner deliberately made Swift unstable for its first several years, allowing the team to discover what worked before committing. When asked about Swift's evolution process, Dave's answer was simple: "Boost did it. Let's just do it again." Doug notes that C++ now lacks what Boost once provided: "They're missing the experimentation to build up what is the best practice before we get into the language."

Key Insights:

- Standards bodies should only standardize designs proven through extensive real-world use; invention and experimentation must happen outside the standardization process
- Having an implementation proves something *can* work; having field experience from independent users proves it *does* work—both are necessary, but field experience is harder and more important
- One influential person withdrawing support can kill years of collaborative work, even after formal approval; this political risk makes major standardization investments precarious
- Presenting complex techniques to committees requires reframing them in simple, familiar terms; Jeremy Siek saved the tuple proposal by saying "it's just a typo" and writing something "boring and simple"
- New languages or features benefit from deliberate instability periods where breaking changes are expected, allowing real-world learning before permanent commitment
- Languages need external proving grounds (like Boost was for C++) where ideas can be tested extensively before standardization; the committee structure itself cannot serve this function

References: Full transcript · Synthesized knowledge

4.6 Agentic Synthesis: Combining Knowledge Across Interviews

Individual knowledge files capture one expert's perspective. Greater value emerges when these files are combined agentically—processed together to identify principles that multiple experts independently corroborate.

The synthesis process calculates two scores for each principle:

- **Confidence**: How many independent experts articulated similar insights? A principle mentioned by three experts (e.g., "standardize existing practice, not invent") carries higher confidence than one from a single source.
- **Relevance**: How directly applicable is this principle to evaluating standardization proposals? Principles about evaluation criteria rank higher than general observations about committee dynamics.

From the five knowledge files produced in this experiment, agentic synthesis extracted **40 distinct principles**, sorted from highest to lowest confidence and relevance. The full output is available in combined-principles.md.

Top 3 Principles (highest confidence and impact):

- **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.
- **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.
- **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.

These three principles were independently articulated by multiple experts (Abrahams, Gregor, Hinnant, Parent), giving them the highest confidence scores in the synthesized output.

4.7 Application: Paper Preflight

What do we do with this distilled knowledge?

The answer is **human-agentic synthesis**: distilled knowledge from expert interviews combines with WG21 leadership guidance and stakeholder input, synthesized through an agentic process into tools that augment proposals with supplementary information and confidence signals. For example, committee leadership could work with captured knowledge to produce a paper testing framework—a "preflight check" that surfaces potential concerns before proposals consume committee time. Humans provide direction and make final decisions; AI performs synthesis and pattern-matching at scale.

To demonstrate what such a tool might look like, we developed a **Paper Tester** (WG21_PAPER_TESTER.md) that evaluates proposals against 13 categories and includes a "gate" mechanism requiring library proposals to justify why standardization is necessary.

> ⚠ **Important**: The Paper Tester presented here was generated from the **author's opinions**, not from captured expert knowledge. It represents one perspective on what makes a strong proposal, derived from observations about committee discussions. A legitimate, authoritative paper testing framework can only be produced once sufficient knowledge capture is achieved—far more interviews than the few

> conducted for this paper—combined with input from WG21 leadership and stakeholders. This example demonstrates the *form* such a tool could take, not its *content*.

The framework was applied to ten papers:

Paper	Score	Result	Summary
P1673R13	23/26	PASS	Gold standard proposal bringing the 40-year BLAS interface into C++, with cross-organizational authorship from NVIDIA, Intel, and national laboratories. Builds on `std::mdspan`. (evaluation)
P0214R9	22/26	PASS	Makes portable SIMD programming possible—currently impossible without standardization since vendor intrinsics are non-portable. 9 revisions, PhD-backed implementation. (evaluation)
P3179R6	22/26	PASS	Unifies two existing standard features (C++17 parallel algorithms + C++20 ranges) that currently cannot be used together. Only standardization can fix this internal fragmentation. (evaluation)
D3952R0	21/26	PASS	Textbook "make the impossible possible"—enables correct, portable pointer-in-range checking that currently requires undefined or unspecified behavior. (evaluation)
P3552R3	21/26	PASS	Framework-completing proposal: C++26 shipped `std::execution` without a usable coroutine type. This makes the framework ergonomically usable. (evaluation)
P0843R10	21/26	WARN	Useful container with extensive field experience, but standardization justification relies on "vocabulary type" assertion. Stronger argument (no standard container for freestanding) is underemphasized. (evaluation)
P0429R3	—	GATE	Failed because it relied on "widespread use of Boost.FlatMap" as justification, which proves ecosystem distribution works rather than demonstrating a coordination failure. (evaluation)
P0447R21	—	GATE	Despite exceptional documentation and 21 revisions, failed because no coordination failure was documented; plf::hive already exists as a successful third-party library. (evaluation)
P1255R11	—	GATE	Failed because the "before" code works today—`views::maybe` makes code more elegant but doesn't enable anything previously impossible. "Making the easy easier" fails the standardization threshold. (evaluation)
P2075R5	—	GATE	Failed because it cited multiple vendor implementations (Intel MKL, cuRAND, rocRAND) as evidence *for* standardization, when this demonstrates the ecosystem already delivers the functionality. (evaluation)

> ⚠ **Not Automatic Acceptance or Rejection**: We are not arguing that papers processed through automated evaluation should be automatically accepted or rejected. The paper tester is a possible *first step* in a paper's journey—a way to surface signals that humans must then review and judge. The value is efficiency: incoming papers must be read, and for best results, read by experts whose attention is scarce. Automated application of distilled knowledge synthesizes useful signals at low cost, freeing

> expert attention for cases that genuinely require it. The tool augments human judgment; it does not replace it.

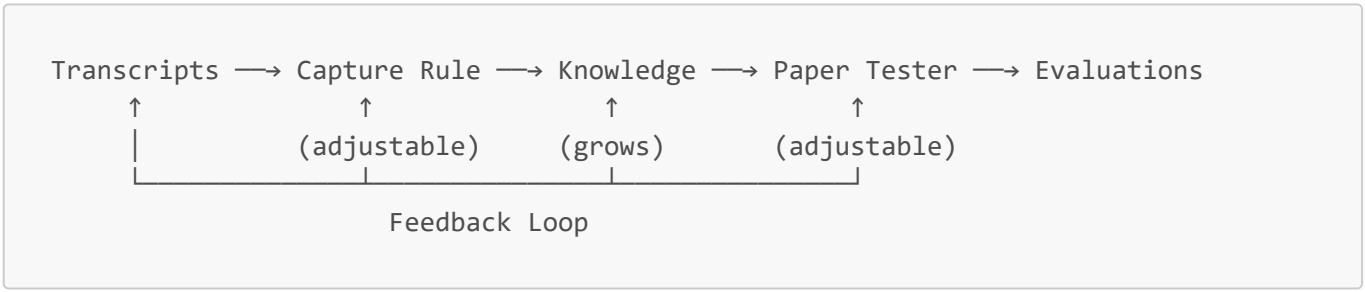
4.8 Reproducibility and Iteration

While the capture rule and paper tester reflect subjective judgments that have not passed through rigorous human review, the outputs they produce are reproducible. Anyone can re-prompt the capture rule on the provided transcripts or re-prompt the paper tester on the original papers and obtain similar results.

The repository preserves the complete processing chain:

Stage	Location	Description
Source data	`inputs/`	Original interview transcripts
Extraction rules	`rules/WG21_CAPTURE_RULE.md`	Prompts for knowledge synthesis
Synthesized knowledge	`knowledge/`	Distilled principles and experiences
Evaluation rules	`rules/WG21_PAPER_TESTER.md`	Prompts for paper evaluation
Generated evaluations	`outputs/`	Paper assessments

This architecture enables iterative refinement:



As the captured knowledge base grows through additional expert interviews, the system can be refined to produce better outputs. Adjustments to the capture rules regenerate synthesized knowledge with new assumptions; adjustments to the paper tester regenerate evaluations with updated criteria. The source data remains stable while the processing rules evolve.

Experimental Status

This methodology is experimental. The distilled knowledge, output formats, and evaluation process are all subject to refinement. Potential improvements include:

- Better categorization of principles (library-only vs. language-only)
- Calibrated confidence levels based on accumulated evaluations
- Reconciliation of potentially contradictory principles as the knowledge base grows

Feedback from practitioners will shape how this methodology evolves.

---

5. Conclusion: The Window Is Open

The knowledge we need to capture is alive in the minds of WG21's most experienced participants. Bjarne Stroustrup is 75. Herb Sutter is 60. The founding generation's members are not getting younger. None of us are.

But this is not an obituary—it's an opportunity. The experts are here, they're willing to share what they know, and we now have the tools to capture and preserve their insights. The window is open, but it won't stay open forever. WG21 is not immune to the forces that erode all institutions.

> "*Over time, functional institutions decay. As the landscape of founders and institutions changes, so does the landscape of society.*"

Decay is not inevitable if we act. WG21 can choose to be proactive—to capture its living knowledge while the founding generation remains available. C++ has earned a standards committee that can maintain its health for generations. The founding generation built something remarkable. Preserving their wisdom is how we honor that achievement—and how we ensure WG21 remains a living tradition of knowledge rather than a dead one.

What you can do:

If you are:

- An experienced WG21 participant willing to share your knowledge through an interview: please contact the paper authors. Your insights—about design principles, historical decisions, evaluation frameworks, failure cases—are exactly what we need to capture.
- Interested in conducting similar interviews within your organization or among other WG21 members: the authors would be happy to share the methodology and materials. The more institutions that capture their living knowledge, the better.
- A newer participant who would benefit from access to captured knowledge, your interest demonstrates the demand for this project.

The knowledge exists. The technology exists. The will exists. Let's capture living knowledge while we still can.

5.1 Beyond WG21

This process is not specific to WG21. The agentic interview workflow described here can be applied to any organization facing knowledge transmission challenges.

The job of the interviewer is not particularly difficult given a reasonable general script. People enjoy talking about themselves and their experiences—especially when asked thoughtful questions about work they're proud of or lessons they've learned the hard way. The interviewer's role is primarily to listen, follow interesting threads, and ask for stories. More is better than less; large language models are particularly good at separating signal from noise.

A company like Bloomberg could deploy this workflow across the organization to capture and preserve institutional knowledge from senior engineers, traders, and domain experts before they retire or move on. The same applies to any institution where tacit knowledge accumulates in the minds of experienced practitioners: law firms, hospitals, research labs, government agencies.

The combination of structured interviews, AI-assisted transcription, and knowledge synthesis creates a scalable approach to a problem that has plagued organizations for as long as organizations have existed: how do you keep hard-won wisdom from walking out the door?

Acknowledgements

Much thanks to Samo Burja for his wonderful *Great Founder Theory* framework, which provides the conceptual lens for understanding knowledge transmission and institutional health.

Thanks to Alisdair Meredith, Howard Hinnant, Bjarne Stroustrup, and Nico Josuttis for their candid reflections on committee knowledge transmission over the years.

Thanks to long-time committee participants whose conversations informed this analysis.

References

Great Founder Theory

- Burja, Samo. Great Founder Theory. 2020.

WG21 Structure and Participation

- ISO C++ Committee. The Committee. Description of WG21 structure and subgroups.
- ISO C++ Committee. Meetings and Participation. How to participate in WG21.
- ISO C++ Committee. How to Submit a Proposal. Guide for newcomers.

Standing Documents

- SD-3: Study Group Organizational Information. Formation and operation of Study Groups.
- SD-4: WG21 Practices and Procedures. Operational practices, consensus, proposal handling.
- SD-7: Mailing Procedures and How to Write Papers. Document formatting and submission.
- SD-8: Standard Library Compatibility. Breaking change policies.
- SD-9: Library Evolution Policies. Default design policies.
- SD-10: Language Evolution Principles. EWG design principles.

Direction Papers

- Stroustrup, Hinnant, Orr, Vandevoorde, Wong. P2000: Direction for ISO C++. Committee philosophy and priorities.
- Voutilainen, Ville. P0592: Bold Overall Plan. Feature roadmaps.

Committee Reports

- WG21. N4990: WG21 2023-2024 Convener's Report. Committee structure and workflow.
- WG21. N4999: WG21 Active Subgroups. Current study groups and working groups.

Process Documentation

- Bastien, JF; Revzin, Barry. P2138R4: Rules of Design <=> Wording engagement. Defines the separation between Evolution and Wording group responsibilities.

Policy Papers (Case Study)

- Levi, Inbal et al. P2267R1: Library Evolution Policies. Framework for LEWG policy adoption.
- Neațu, Darius; Sankel, David. P3162R0: LEWG [[nodiscard]] policy. Rationale paper with guiding principles.
- Wakely, Jonathan. P3122R1: [[nodiscard]] should be a little less aggressive. Implementation-focused policy proposal.
- Neațu, Darius; Sankel, David. P3201R1: LEWG [[nodiscard]] policy. Final policy wording.

Revision History

- R0 (2026-01-07): Reframed from diagnostic to solution-oriented; added agentic interview proposal