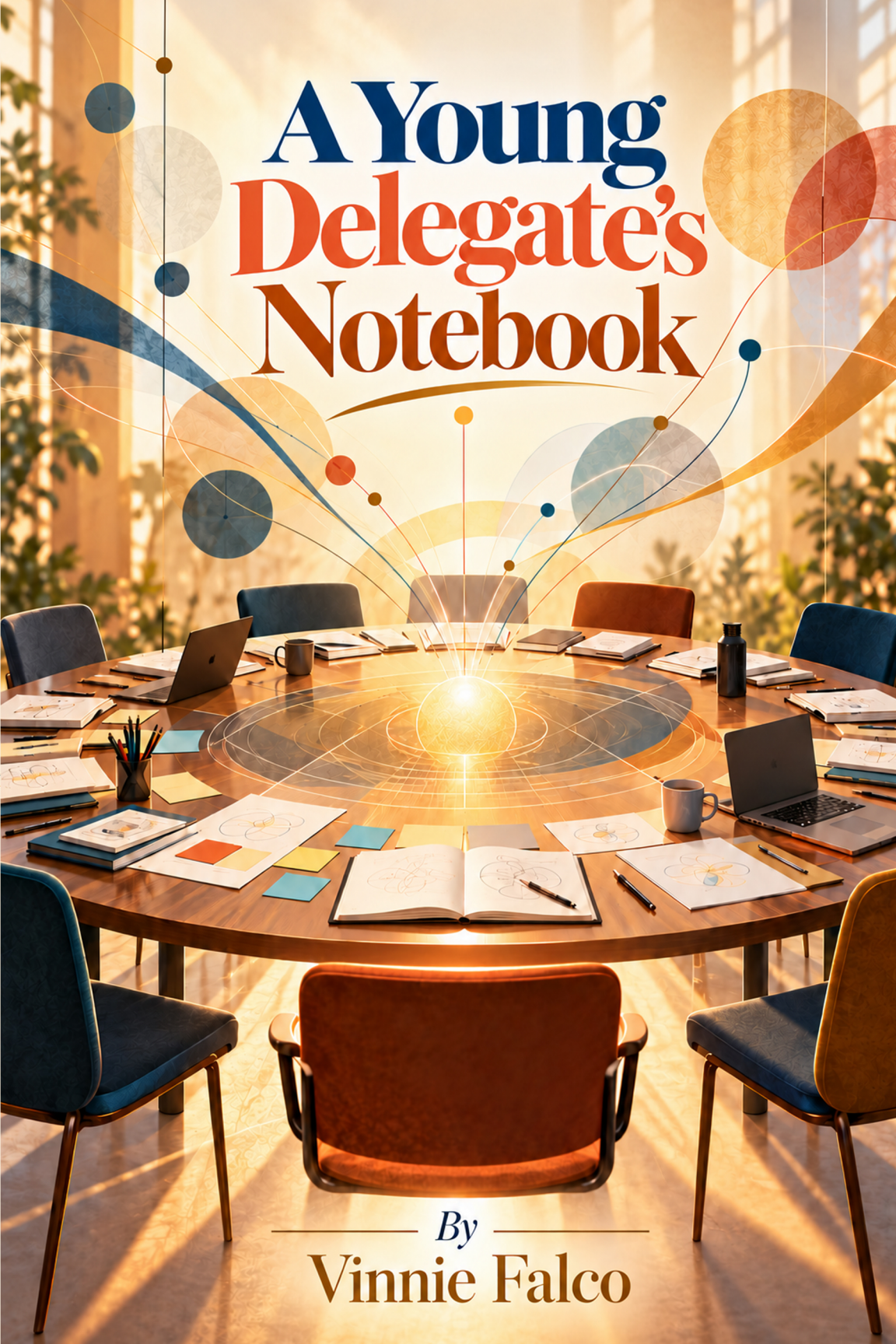

A Young Delegate's Notebook
By
Vinnie Falco

A Young Delegate's Notebook

A Young Delegate's Notebook

A Newcomer's Path into WG21 and the Standardization of C++

VINNIE FALCO

ALLIANCE PRESS

Copyright 2026 by Vinnie Falco

All rights reserved. This book or any portion thereof may not be reproduced or used in any manner whatsoever without the express written permission of the publisher except for the use of brief quotations in a book review.

ISBN 979-8-234-20924-5

Published by Alliance Press

To every engineer who found a flaw, imagined a fix, and walked into the room to argue for it. This book is for the next of you.

Acknowledgments

The author expresses appreciation to those who assisted with the creation of this book: Robert Beeston, Harold Bott, Can Çağrı, Nina Dinka Ranns, Alli Ebenkamp, Mungo Gill, Lori Hughes, David Pannett, Deborah Williams, and Michael Wong.

Table of Contents

Preface	xi
Introduction	xiii
1. The C++ Standard	1
2. Meet the Committee	7
3. How Papers Work	13
4. How to Participate Remotely	19
5. How WG21 Is Structured	25
6. Getting There	31
7. In the Room	35
8. How Decisions Get Made	41
9. What Goes into a Proposal	49
10. Producing and Submitting Your Paper	55
11. Championing Your Paper	61
12. C++ Design Principles	67
13. Common Mistakes	73

Preface

This book is a practical guide for anyone who wants to help shape C++ but has never set foot in a committee meeting. The chapters build on one another, but you don't need to finish the book before you can contribute. Every paper number is a live link, and every named resource points somewhere real.

Búzios edition, November 2026. Officer names, fees, attendance figures, and paper revisions are stated as of September 2026. The current roster is at isocpp.org/std/the-committee. The standing documents and the Direction Group's papers are revised over time. Committee practice can change before either is updated. Watch for updates, and when this notebook and the current SD-4 disagree, SD-4 wins.

Introduction

> *I have not found among my possessions anything which I hold more dear than, or value so much as, the knowledge of the actions of great men, acquired by long experience in contemporary affairs, and a continual study of antiquity; which, having reflected upon it with great and prolonged diligence, I now send, digested into a little volume, to your Magnificence.*
>
> Niccolò Machiavelli, dedicatory letter, *The Prince*, translated by W. K. Marriott

I wrote this for you. Not for the experts who've sat in these rooms for twenty years. For you, the one who just found out that a group of volunteers decides what goes into C++ and wondered how to join them.

The group is real, and you can join it. It's called WG21, and it's the committee that owns the C++ standard. Around two hundred people take part in each meeting. They argue, they vote, and every three years they ship a new version of the language.

These volunteers are some of the most careful people you'll ever meet. They'll spend an hour on the placement of a single word. You'll come to understand why.

Machiavelli gave a prince the one thing he valued most: what he'd learned watching powerful people up close. This notebook works the same way, smaller and plainer. It's the worn, marked-up notebook I wish someone had pressed into my hands before my first meeting.

Who This Notebook Is For

You belong here, wherever you're starting from. Maybe you've never been to a meeting, or maybe you've been to a few and still feel lost in the room. Everyone there was new once, even the people who run the place.

You don't need to know every rule in C++. You don't need to have written a compiler. You need only curiosity and the patience to keep showing up.

You also bring something the committee needs. You use C++, and you know where it hurts. Most people who use the language will never attend a meeting. When you speak up, you speak for them.

What This Notebook Is Not

This isn't a textbook, a manual, or an encyclopedia. It's my notebook, and I take sides in it. I value stability over novelty, and I put the people who use C++ ahead of the people who design it.

When the evidence is thin, I say so. Where the committee has written a rule down, I link to it, so you can check me. Names, fees, and dates are as of 2026, and they'll drift. If a linked rule and I ever disagree, trust the rule.

What Each Chapter Covers

The notebook moves, in order, from "what is this thing?" to "how do I take part well?"

It starts with your bearings. Chapter 1 shows you why the standard matters and why every change you make is close to permanent. Chapter 2 introduces the committee and the many ways you can join in. Chapter 3 explains how papers work, because the paper is the unit of work.

Then it takes you inside. Chapter 4 shows you how to take part from your desk, long before you ever travel. Chapter 5 maps the rooms, so you know who decides what. Chapters 6 and 7 get you to your first meeting and help you survive it. Chapter 8 shows what a vote really means.

Next comes your own paper, when you have one worth writing. Chapter 9 lays out what a proposal must contain to earn the committee's time. Chapter 10 walks through the tools, the format, and the deadlines. Chapter 11 covers the long work of building support and surviving feedback.

Last come the hard edges. Chapter 12 names the deep design constraints that decide what can and can't ship. Chapter 13 collects the common mistakes, so you don't make them.

How to Read This Notebook

Read the chapters in order, because each one builds on the ones before it. Every chapter also ends somewhere safe to stop, so you can set the notebook down and pick it up again when you're ready.

Follow the links. Many of them lead to the committee's own documents, and learning to read those is half the job.

Keep Your Wits about You

One practice matters more than any fact in this notebook. The committee runs on persuasion, and persuasion can talk you out of your own judgment. Hold on to yours.

Someone you admire calls a design obviously right. Nobody objects, and your own doubt starts to feel rude. That's the moment to ask two questions. Has anyone actually built it and used it? And who does this help, the people at the table or the people who'll live with it?

Here's the whole notebook in four sentences, short enough to remember:

> **RULE OF THUMB**
>
> Keep your wits about you.
>
> Be skeptical.
>
> Demand evidence.
>
> Put the users first.

Welcome. I'm glad you're here. This notebook is yours now, so mark it up and argue with me in the margins.

Let's start with the standard itself, and the responsibility that comes with changing it.

1. The C++ Standard

Every C++ compiler in the world is chasing one document. It's the C++ standard, and you'll help shape it.

IN THIS CHAPTER

- Why standardization is a responsibility, not a game
- The one-sentence oath that fits the whole job
- Why a feature, once standardized, is close to forever, and who pays for it
- Why the standard, the compilers, and the language you use are three different things
- The forms the standard takes: the working draft, the published standard, technical specifications, and white papers
- How to read the standard without reading all of it
- Who turns the words into compilers, and what your feature costs them

1.1 Standardization Is a Responsibility

The work you're volunteering for is serious. Banks and hospitals run on C++. When you change the language, you change the ground they stand on.

So this isn't a game. It isn't a place to collect a credential or win an argument.

1.2 The Delegate's Oath

The duty fits in a single sentence. Some delegates call it the Delegate's Oath, and it goes like this:

RULE OF THUMB

I vow to do what is best for the language, to make no unnecessary proposals, and to put the needs of sixteen million users ahead of my own.

Each part guards against a real temptation. “Best for the language” outranks best for you or your employer. It’s also the hardest part to live by, because your employer’s need is usually a real user’s need, dressed in a paycheck.

If your employer pays for your time here, the trouble starts when it asks you for a vote you believe is bad for the language. The oath exists for that moment. Talk the balance through with your manager early, before you’re standing in the room with your hand half raised.

“No unnecessary proposals” puts the burden on you to show a change earns its place. The urge to get your idea adopted is strong, and it can fool you. Passing a proposal was never the goal. The goal is a better language for the people who use it.

The last part is the crux of it. You serve those users first, even when they’ll never know your name.

To understand why the oath is this strict, look at what a single feature costs.

1.3 Every Feature Is Forever

The hard rule: the standard almost never shrinks. Features get added. Removing one takes years. Even after it’s gone from the standard, it lives on in compilers and in the code that uses it.

C++ got a regular-expression library, `std::regex`, in 2011, and everyone soon learned it was slow. More than a decade later, it’s still slow.

It can’t be fixed, because a faster version would break programs already compiled with the old one. So the slow version stays. The mistake is frozen in place.

Whatever you add to the standard freezes the same way, mistakes and all. Proposing a feature costs you several years. Maintaining it costs everyone else forever.

1.4 The People Who Inherit Your Work

The cost is paid by the people who use the language. More than sixteen million developers write C++ today, by one recent estimate. Whatever you standardize, they inherit.

They didn't vote for your feature, and most will never read a proposal. They discover your work as a line in compiler release notes. By then it's permanent.

Users aren't the only ones who pay. Each compiler team is expected to build and maintain every feature, and every textbook, course, and tutorial has to cover it.

> **WATCH OUT**
>
> When you catch yourself saying "it's just one small addition," stop. That urge is how careful people do lasting damage. Small is not the same as free. Ask who will maintain it in 2040.

1.5 The Standard Is Not the Language

You might think "C++" and "the C++ standard" are the same thing. They're not. Three different things go by that name.

The standard is a document. It's a long, exact text that says what C++ means. It's words, not a program.

The second is a compiler, a program that tries to follow that document. The big compilers don't follow it perfectly, and they don't all agree. So the C++ you can actually use is whatever your compiler accepts.

The third is the living language: C++ as it's used in all the code already out there. That code leans on compiler quirks and extensions as well as the text. It doesn't change when the standard does, so anything you add has to work with it.

Keep three things distinct: the standard is the text, the compilers try to follow it, and the living language is what real code relies on.

1.6 The Forms the Standard Takes

The standard comes in a few forms, and each name tells you how settled a feature is. Behind every C++ release is a living document called the *working draft*, and it changes all the time. You can read the current one online, and its source lives on GitHub.

Every three years, the committee freezes the draft and hands it to ISO to publish. That published version is an *International Standard*, or IS. C++23 is one. C++26 finished its technical work in March 2026. Before ISO publishes any version, member countries have to approve its final text in a vote.

Two lighter documents let people test a feature before the standard commits to it. A *Technical Specification*, or TS, is the experimental tryout, and a *white paper* does the same job with far less paperwork. The committee's own practices guide says the standards bodies have recommended the white paper since 2023.

So when someone says a feature is "in a TS," it isn't in the standard yet, and it may never be. You'll also run into the odd *Technical Report*, or TR, an older kind the other two have mostly replaced.

1.7 How to Read the Standard

The standard looks scary, and it is dense. But it's organized, and a few habits make it readable. You don't have to read it front to back.

Most of it is *normative text*, the parts that state actual rules. Those are the words people will spend an hour on, because a single one can change what a program means. Mixed in are notes and examples, which never add a rule. They only help you understand the rules around them.

Each part has a *stable section label* in brackets, like [container.requirements]. Those labels stay fixed between drafts. Section numbers shift, so point to the label, not the number.

> **TRY THIS TODAY**
>
> Open the working draft and search for a feature you use every day. Find its bracketed label, then read one paragraph of its rules, slowly. That's the kind of text you'd be changing, and millions of people depend on it.

1.8 From Standard to Compiler

Publishing the standard doesn't put a feature in your hands. Someone has to build it into a compiler, sometimes before the standard is out and sometimes years after. Until they do, it stays words on a page.

The bulk of that building falls to *the three implementers*: GCC, Clang, and MSVC. Most of the C++ in the world runs through one of them. If they won't build a feature, the standard can't force them. So a feature isn't real until it ships in a compiler you can download.

Picture the day your proposal is adopted. On each of those three teams, someone reads it and opens a ticket. They'll build it and chase down its bugs. Then they'll keep it working for as long as anyone writes C++.

> **THE SHORT VERSION**
>
> Changing C++ is a responsibility, not a hobby. Whatever you add stays, and sixteen million people inherit it. The standard is a text, and it comes as a working draft, a published standard, and lighter tryouts. Cite its bracketed labels, since section numbers shift. A feature isn't real until a compiler ships it.

If all this feels heavy, good. Keep that feeling. It's what will stop you the next time an idea seems too small to cost anything.

Next, let's meet the committee that does the work.

2. Meet the Committee

You can join this. The entrance is wider than it looks.

IN THIS CHAPTER

- Why nobody has to give you permission
- What the committee's long official name means
- How the committee meets
- What a national body is, and who actually votes
- The ways in: as a guest, through your own country, or through the United States, paid or free
- What being accredited means, and the trap that leaves regulars without a vote
- The roles you can play, and which ones need no membership
- What to expect on your first day
- Why showing up matters more than being brilliant
- Meeting committee members at conferences, and where good ideas start

2.1 You Don't Need Permission

No one grants you permission to help. For the first steps, you won't find a gatekeeper. You can read the proposals, review them, and talk ideas over in public today. If you want to contribute, you start.

The gates after that are a lot lower than the rumors make them sound. Their rules are posted, and the people at them want you through. You can attend a meeting as a guest for free. The June 2026 meeting in Brno had twenty-five first-time guests, so newcomers are normal here.

You also don't have to be a wizard. Plenty of useful work isn't writing code at all. Taking notes and reviewing other people's proposals both count.

2.2 The Nesting: ISO, JTC1, SC22, WG21

The committee has a long official name, and it tells a story. It's ISO/IEC JTC1/SC22/WG21. Read it left to right, and it nests from the biggest body down to the smallest.

ISO is the International Organization for Standardization, which publishes standards for almost everything. With its sister body, the International Electrotechnical Commission (IEC), it runs JTC1, the joint committee for information technology. That's why the name says ISO/IEC.

JTC1 holds SC22, the subcommittee for programming languages. WG21, Working Group 21, is the part of SC22 that owns C++.

Being at the bottom of the nest means that, on paper, WG21 only recommends. It agrees on what the standard should say, and the bodies above it run the official vote.

2.3 How the Committee Meets

The committee works in two modes. Most of the time it splits into *subgroups*, the smaller teams that each handle one slice of the work. At the start and end of every meeting, the whole committee gathers as one.

That full gathering is the *plenary*, where the whole committee makes its final decisions. The subgroups do the building. The plenary signs off.

This happens at a real scale. The Brno meeting drew about two hundred people from twenty-eight nations, a little over half in the room and the rest online. And the nations matter as much as the people, because the official members are countries.

2.4 What a National Body Is

Each country takes part through its *national body*, or NB, which is the country's standards organization. The national body holds the membership and appoints the experts who attend.

Inside each national body sits a small *mirror committee* that shadows WG21 at home. That's the group you actually join. It decides who represents your country and what your country thinks of a draft.

Within WG21, people vote, not countries. In the plenary, every expert appointed by a national body votes as an individual.

Countries get their own vote later, when a draft goes out for a formal ballot. There, and only there, it's one country, one vote, and a delegation of twenty counts the same as a delegation of one.

2.5 The Easiest Start: Attending as a Guest

The lowest-cost way in is to attend as a *guest*. A guest takes part in a meeting without joining a national body. It's free.

To come as a guest, email the convener, the person who runs WG21. Tell them you'd like to attend, and name the country where you live or work. ISO strongly encourages at least a week's notice.

As a guest you can do almost everything. You join the subgroups, argue, and take part in their polls. The one thing you can't do is vote in the plenary, where final decisions are made.

Guest status covers your first meeting (and potentially a second, if you ask the convener). To keep coming after that, or to get the plenary vote, you need a seat through a national body.

2.6 Getting Further In: Finding a Seat

Start at home. If your country has a mirror committee, join it. Email the national body chairs, say where you live or work, and ask how to join. Some national bodies run one committee for every programming language, so say you want C++. Rules and fees vary by country, and some charge nothing.

If your country has no mirror committee, you still have a way in. You don't have to be American to join through the United States. Its national body, INCITS, is open to anyone, with or without a company behind you. Many committee members take part through it.

You could start a mirror committee at home instead, but that takes about a year. Get a seat first, and build the committee once you know you'll stay.

INCITS charges a yearly fee per committee. In the 2026 cycle that's $2,703 at the full rate, $1,530 for small companies, and $510 for academics.

If that fee is a barrier, a free path exists. The Beman Project offers C++ developers free seats under its INCITS membership. You'd sit as an *alternate*, a backup to Beman's main representative. Travel is still on you, but the seat gets you into the rooms and the discussions.

Before you join, check who holds the seat. In some countries it's you. In the United States it's whoever pays the fee, which may be your employer. If your employer pays, the seat may not follow you to your next job.

2.7 Getting a Vote: The ISO Global Directory

A vote at the plenary counts only when you're accredited. *Accredited* means your name sits in the ISO *global directory*, the official roster of voting experts. Your national body puts it there.

> **WATCH OUT**
>
> You can go to meeting after meeting and never get your name into the global directory. Then the one vote that matters to you comes up, and you have no vote to cast.

A seat that gets you into the rooms doesn't always put your name on that roster. Ask your national body to list you, and check that it did. A guest needs a week's notice. A vote needs months, so ask long before the meeting where you want one.

A seat asks two things of you: read the proposals before you vote, and stay active in your mirror committee. You vote as yourself, but you're there on your national body's word.

A vote is only one way to matter. Many roles need no vote at all.

2.8 The Roles You Can Play

You can help in more ways than you'd guess. Here are the main ones:

- Note-taker, also called the scribe. Keeping a session's minutes is a clear way to be useful on day one.
- Reviewer. Read a proposal that's coming up and tell its author what you think.
- Author. Write up an idea and submit it for the committee to consider.

- Implementer. Build part of a feature in a compiler and report what you learn.
- Champion. Present a proposal and push it through the rooms. A *champion* can be the author, or someone else who believes in the idea.

The first two need no membership, just your time and attention.

2.9 Your First Day: Orientation and Introductions

Your first meeting starts before the work does. At 6 p.m. on the Sunday before, there's a newcomer orientation, which usually meets in the main hotel's lobby. It covers how the week works and answers your questions.

When the week opens, first-time attendees introduce themselves to the whole committee. It's brief and friendly, not a test.

2.10 Showing Up Is the Secret

The single biggest thing you can do is keep showing up. An old line says "eighty percent of success is showing up." In this committee, it's close to true.

Showing up builds two things: familiarity and trust. People learn your name and your judgment over many meetings. The committee runs on relationships, and relationships need time.

The reverse is true too. Stop coming, and the room quietly loses what you brought. Soon it forgets you brought anything. Presence matters here more than brilliance.

> **RULE OF THUMB**
>
> Show up, take notes, and read the proposals. Do those three things for three meetings, and you'll be a known quantity.

2.11 Conferences: Meeting the Committee Off Duty

Long before your first meeting, you can meet the committee at a conference. Many committee members go, so it's an easy place to talk with them in a relaxed setting.

Conferences are also where ideas get aired. A lot of C++ ideas start with an everyday problem. A product hits a limit in the language, a bug keeps coming back, or people keep writing the same code by hand. Sometimes a programmer just needs sockets, finds the standard library has none, and asks why not.

So if you want an idea worth chasing, look at what your code and your users can't do, not at what would be clever.

Two conferences stand out. C++Now is small and expert-heavy, held each spring in Aspen, Colorado. It's where people push and refine new ideas. CppCon is the big one, where ideas meet the wider community.

You don't have to fly to the United States, either. isocpp.org keeps a world-wide list, with ACCU and Meeting C++ among the European ones. Going to any of them is a low-pressure way to start.

> **THE SHORT VERSION**
>
> You don't need anyone's permission to start. Within WG21, which sits inside ISO/IEC JTC1/SC22, people vote, not countries. Countries vote later, on the formal ballot. You can come to your first meeting free, as a guest, with a week's notice. Guests do everything but vote at the plenary.
>
> To keep coming, you need a seat through a national body. Start at home if you can. Otherwise INCITS takes anyone for a fee, and the Beman Project offers free seats if that's too steep. A plenary vote also needs your name in the global directory, and that takes months, so ask early.
>
> Plenty of the work needs no vote at all. Whatever you do, keep showing up.

The people at the entrance are waving you in. Next up: what everyone in those rooms is actually reading.

3. How Papers Work

Open the committee's paper archive for the first time and you get a page of codes. Hundreds of rows, mostly a letter, four digits, an R, and a number. All the committee's work flows through one kind of object: the paper. Learn how papers work, and the rest of the process makes sense.

IN THIS CHAPTER

- The rule that there's no proposal without a paper
- The kinds of papers, and what each one wants from you
- What P-, D-, and N-numbers mean, and what a revision tells you
- Where every paper lives, and the quickest way to find one
- How mailings and deadlines decide what a meeting can act on
- The long path from idea to published standard
- Design review versus wording review
- Defect reports, and how bugs in the standard get fixed
- A five-question checklist for reading any paper
- The two documents to read before almost anything else

3.1 Nothing Happens Without a Paper

One rule explains most of the committee: no paper, no proposal. The rulebook, SD-4, puts it bluntly: "if a proposal doesn't have a paper, it doesn't exist." A *paper* is a written proposal or report, and it's the unit of all the work.

A paper needs three things before a room can act on it. The paper has to exist, it has to arrive on time, and someone has to be there to present it. Miss one and the room moves on.

Talk still matters. A hallway conversation can expose a flaw, improve a design, or change a mind. But the room only votes on what it can read, so if you want to move the committee, write it down.

3.2 What a Paper Is and Its Types

Papers sort first by what they want from you. An *ask-paper* proposes something and asks for a poll. An *inform-paper* puts facts on the record and asks for nothing. Those aren't official labels, just useful ones.

Here's a quiet truth: the inform-paper often gets read more. It asks for no time on a crowded schedule, and nobody has to pick a side to read it. So it slips in where an ask-paper waits in line.

Papers also split by job. Some argue for a design, and others nail down exact standard wording. A separate kind, the *standing document*, holds the committee's own rules instead of proposing anything.

3.3 The P-Number System

A paper's number starts with a letter that tells you what sort of document it is. A *P-number* marks a proposal, like P2300. A *D-number* marks a draft that hasn't been published yet. You'll often see one during a meeting week, when an author revises a paper overnight.

An *N-number* marks an official ISO document, like the working draft itself, an agenda, or minutes. Before 2015 every paper got one, so an older N-number is often a proposal.

After the number comes a *revision*, like R0 for the first version, then R1, R2, and up. R0 is brand new and untested in the room. A paper at R5 has been reworked five times.

> **RULE OF THUMB**
>
> A paper's revision number tells you its history, not its quality. A fifth revision means someone kept working, not that they got it right. Judge what's on the page.

Once you have a number, the paper is one link away.

3.4 Finding Any Paper

The papers all live in one official archive, open-std.org, in a plain list by date. Every paper link in this notebook points there.

You don't hunt for papers by hand. The quickest route is wg21.link, an unofficial shortcut that Mara Bos has run for years. Type a paper number after the slash, and it sends you to the copy on open-std.org. For P2300 revision 10, that's `wg21.link/p2300r10`.

Leave off the revision, as in `wg21.link/p2300`, and you land on the newest one.

3.5 The Mailing System and Deadlines

Papers arrive in batches called *the mailing*, most months of the year. The biggest ones land just before and just after each meeting. In 2025 there were nine in all. Counting revisions, that's eight hundred or more papers a year.

Timing decides what the committee can act on. Each meeting has a *pre-meeting mailing deadline*, weeks ahead of it. Miss it and your paper sits out until the next round.

There's one exception. A late update to a paper that made the deadline can still be heard.

3.6 The Paper Lifecycle

A paper that makes it follows a route called *the paper lifecycle*. It starts as an idea and ends in the published standard. Most new-feature proposals stop somewhere along the way.

The ones that finish pass through these stages, roughly in this order:

1. You submit the paper to a mailing.
2. A small group may incubate the idea and shape it first.
3. A design group decides whether the direction is right, then approves the design.
4. You write the exact standard wording, and a wording group reviews, fixes, and approves it.

5. The plenary, the full committee, votes it into the working draft (the living text that becomes the next standard).
6. The national bodies review the whole draft and send comments, and the committee answers every one. C++26 drew 411 of them.
7. The national bodies vote to approve it, and ISO publishes the new standard.

Any stage can be the end. A paper can stall, get sent back, or quietly die partway through. The whole journey takes years.

The committee keeps a longer, stage-by-stage version of this path, and it's worth reading once.

When a paper catches your eye, work out which stage it's at. That tells you what kind of help it can still use.

3.7 Design Review versus Wording Review

Two of those stages, design and wording, ask very different questions. *Design review* asks: is this the right feature, built the right way? *Wording review* asks: is the exact text precise and complete?

The two happen in order. Once the design is approved, the author's proposed text goes to a wording group. Wording review assumes the answer is yes and hunts for gaps in the spec text.

That order matters to you. If you doubt the design, say so while the paper is still in design review. Bring the same doubt to the wording group later, and it usually treats the question as settled.

3.8 Defect Reports and the Issues Lists

The standard ships with bugs like anything else. Anyone can report one as an *issue*, and the committee decides whether it needs a fix. When the full committee approves a fix to the published standard, the issue becomes a *defect report*. A fix to brand-new draft text doesn't get that label.

These bugs are tracked in public on two *issues lists*, one for the core language and one for the library. The wording groups work through them, and the fixes the plenary approves go into the working draft.

Because they're public, the issues lists are open to you today. Read a few and you'll learn how the wording groups think. Find a real bug, and the library list spells out how to file one. It even says "your chances of a successful issue greatly increase if you attempt wording."

3.9 Reading a Paper Critically

You can't read eight hundred papers a year, so don't try. Read the ones in the areas you care about, and skim the titles of the rest. Read deeply only where you'll vote.

Always read the newest revision. An old one makes you raise a concern the author already fixed. When a paper comes back, compare it against the last version to see what changed. Many papers list those changes near the top, in a revision history.

Start with the header. It gives the paper's number, its date, who to reply to, and its audience. The audience line names the groups it's aimed at, a clue to its stage.

Before you form an opinion on any paper, run it through a few questions:

- What problem does it solve, and does it say so?
- Does it weigh other options, or pretend none exist?
- Has anyone built it, or is it only an idea?
- What old code breaks, and how do users move forward?
- Is the wording clear on the hard cases?

The gaps are the red flags. Empty answers, like no implementation and no migration plan, are objections someone will raise in the room. A strong paper closes them before you ask. If a paper doesn't, tell the author.

> **TRY THIS TODAY**
>
> Open the newest mailing on open-std.org and pick one paper on something you use every day. Ask what it fixes, what it breaks, and whether anyone has built it. Write down one gap. That's how every review starts.

Two documents will save you more than any single paper.

3.10 The Documents Everyone Should Read: SD-4 and P2000

Before almost anything else, read the rulebook and the roadmap. The rulebook is SD-4, "WG21 Practices and Procedures," the practices everyone is expected to know. It covers polls, consensus, deadlines, ballots, and how to escalate a disagreement.

The roadmap is P2000, "Direction for ISO C++." A small group of experienced members called the *Direction Group* writes it to recommend where the committee should head over the long run. Papers often lean on it to make their case, so know what it says before you weigh theirs.

> **THE SHORT VERSION**
>
> No paper, no proposal. Every paper lives on open-std.org, and its number tells you its kind and history. New papers arrive in mailings, and a paper that misses the deadline waits a meeting. Most new-feature proposals never finish the years-long journey to the standard. Read the newest revision critically. Read SD-4 and P2000 first.

Go back to that page of codes. It reads differently now. Each row tells you what kind of paper it is and how many times it's been reworked. You can pull up the newest one and know what to ask of it.

Sooner or later you'll want to say something about what you've read, and that's where taking part begins.

4. How to Participate Remotely

You don't have to fly anywhere to start. Most of the committee's life happens online, between meetings. All you need is a desk and a browser.

IN THIS CHAPTER

- What you can do today, with no plane ticket at all
- The sites you'll work from, and which ones are official
- Where to float an idea before it becomes a paper
- Following meetings from afar, and open-source ways to help
- The one step that opens the rest
- The reflector: who gets in, what you can't quote, and why courtesy is the rule
- The wiki, the chat, telecons, and voting from home

4.1 You Can Take Part from Your Desk

Plenty of useful work happens with no plane ticket. You can read, review, and discuss from home, on your own schedule. None of it needs a meeting badge.

The lowest-effort start is reviewing a proposal. The volunteer-run wg21.org hosts a public paper-review list for exactly this. You read a paper that's coming up and post what you found, even if it's one point. A free login with a GitHub or Google account is all it takes.

Each paper gets a one-week review window, and afterward the reviews go into a credited summary that's attached to the paper on wg21.org. And giving reviews earns you reviews of your own.

Much of what follows is open to you today. The rest waits on one step: attending one meeting as a guest, in person or by video. After that, the reflector's closed lists open up.

4.2 The Official Sites, and One Unofficial One

Chapter 3 showed you how to find a paper, and these are the sites you'll work from. isocpp.org is the public-facing site, home to the standing documents and committee news. Start there for anything official, like the meetings page that tells you how to email the convener.

open-std.org is the official archive of every paper and draft, and it's where your own paper will land someday. It's sorted by date, with no search. It's the source of truth, even if it's bare.

wg21.org, home of that review list, is the unofficial one. It's where you'll do most of your browsing: filter the mailing by group, search the full text, and download each group's report. The papers themselves still live on open-std.org. Don't confuse it with the wg21.link shortcut from chapter 3.

4.3 Floating Ideas: std-proposals

Before you write a formal proposal, float the idea first. std-proposals is a public list for early-stage ideas. You describe your idea and see what people think.

This can save you months. People often point you to earlier papers on the same problem, and they raise the first objections while changing course is still cheap. Take a warm reply as encouragement, not a verdict. The real test is still the bar in chapter 9.

> **TRY THIS TODAY**
>
> Subscribe to std-proposals today and read it for a week before you post. You'll see what a warm reception looks like, and a cold one, before your own idea gets either.

4.4 Following Along from Anywhere

Every meeting gets written up. After each one, the results come out fast. Several delegates post public trip reports, like Herb Sutter's blog. The official minutes come out as an N-paper, like N5040 for the March 2026 meeting, so the record is there to read.

Start with a trip report, since it tells you what mattered and why. Check the minutes when you need the official version.

If you'd rather ask than read, the #include <C++> Discord is open to anyone. It's friendly and full of people who'll help a newcomer. It's a gentle place to learn the culture. Bring your beginner's questions there first.

4.5 Open-Source Help: The Beman Project

You can help with code without ever attending. The Beman Project builds open-source implementations of proposed standard library features, so people can try them for real.

You can help write and test them, or use them in your own code and report what breaks. A bug report from real code is the kind of evidence a proposal needs before the vote.

4.6 One Meeting Opens the Rest

The one step is attending one meeting as a guest, in person or by video. Email the convener first, the way chapter 2 described, and say you're coming.

Meetings are hybrid, so even this step works from your desk. You won't catch everything from afar. The hallway talk, where a lot gets settled, is hard to join remotely.

The other catch if you're far away is time zones. A meeting on another continent can run through your night. Pick the sessions worth losing sleep over, and read the rest later.

So put your effort where distance doesn't count. The rulebook from chapter 3, SD-4, prefers a concern raised on the committee's email lists before a meeting over one raised in the room. A careful review that lands weeks early often shapes a paper more than a remark from the floor.

4.7 The Reflector

The committee's main online home is email. Its mailing lists are called *the reflector*, and most discussion between meetings happens there. Each subgroup (smaller group) keeps its own list.

You can join the lists once you've attended one meeting as a guest, in person or by video. Before that, many topic lists are readable in public, and the person who runs a topic group can add an expert directly. Once you're in, start with the list for the papers you care about.

One rule matters from day one. Posts on the closed lists are private, so you can't quote them in public. You can share poll questions and their numbers freely, but a person's words need that person's consent. In practice, almost nobody asks, and almost nobody says yes.

Be courteous on the lists, especially when you're sure you're right. Email strips the tone out of an argument and leaves only the words. The people reading yours will run your sessions and review your papers.

4.8 The Wiki and Mattermost

The committee runs its meetings off *the wiki*. It holds the agendas, schedules, video-call links, and poll pages for each meeting. If you attend, you'll live in it that week.

It's password-protected, so ask the convener how a guest gets in. Don't edit it unless someone asks you to.

When things move fast, people use chat. Mattermost, at chat.isocpp.org, is the committee's own real-time chat, and during meetings it keeps the parallel rooms in touch. Like the wiki, it isn't public, so ask the convener about it too. Keep it open whenever a meeting is running.

The courtesy rule from the reflector holds in every chat, too. Chat is faster than email and even easier to misread.

> **WATCH OUT**
>
> A sharp reply on the lists feels good for a minute and costs you a reader for a decade.

4.9 Telecons between Meetings

The big meetings happen three times a year. Between them, subgroups hold online meetings called *telecons*, about thirty a month across the various groups.

The wiki lists the schedule and the links. Telecons take guests too, with the same heads-up to the convener. Pick a subgroup you care about and sit in. Once you're there, you may be asked to take the minutes.

4.10 Voting from Afar: Electronic Polls

Not every vote waits for a meeting. Some groups, the library side especially, run polls online between meetings. They use these to settle questions without burning floor time.

The rules are in P2195R2, "Electronic Straw Polls." A straw poll is a quick vote that steers a group's work, and chapter 8 shows you how to read one. Each round and its results are published as papers. The May 2025 round, for example, is P3712R0, and its outcomes are in P3713R0.

You can vote in these yourself once you've joined a national body, as chapter 2 described. Otherwise, you qualify for about a year after attending a week-long meeting in person. Vote only on what you've followed. That May poll paper asks anyone who missed the discussion to sit out unless they understand the topic.

Twenty-three people took part in that round, and it sent a `std::format` paper on toward C++29. None of them needed a plane ticket.

> **THE SHORT VERSION**
>
> From home, starting today, you can review papers on wg21.org, float ideas on std-proposals, read the trip reports, and help test the Beman Project's libraries. The official sites are isocpp.org and open-std.org, and wg21.org is the unofficial one.
>
> The rest takes one step: attending one meeting as a guest, in person or by video. That opens the reflector's closed lists, whose posts stay private. Share poll questions and numbers freely, but ask before you quote anyone. Stay courteous on the lists and in chat.

> Ask the convener about the wiki and Mattermost. Telecons and electronic polls keep the work going between meetings. A concern posted early on the lists often counts for more than one raised in the room.

Distance costs you the hallway, not the work. Pick one paper tonight, read it, and post what you found. Next, let's map the rooms where all this work gets sorted.

5. How WG21 Is Structured

WG21 looks like a wall of acronyms at first. EWG, LEWG, and fifteen study groups blur together. This chapter turns that wall into a clear map.

IN THIS CHAPTER

- The rooms: two for design, two for wording, and the study groups that feed them
- How a paper climbs from one room to the next
- The train model, and where it came from
- Why features almost never come out once they're in
- Who runs the committee, and who appoints them
- Who sets the committee's priorities
- How much power a chair really has, and the rule they schedule by
- The standing documents, where the committee keeps its rules
- Why the committee runs on volunteers, and the implementer veto

5.1 One Committee, Many Rooms

WG21 does its work in subgroups (the smaller groups from chapter 2), and people call them *the rooms*. Each room owns one slice of the job, and every one of them is open to you, even as a guest. The rooms meet at the same time, in parallel, all week.

That parallel timetable is the *schedule grid*. Six or seven rooms run at once, so you can't attend them all.

The rooms split into two jobs. Some decide what to build (design), and others get the exact words right (wording).

5.2 The Evolution Rooms: EWG and LEWG

Two rooms do the design review from chapter 3, deciding whether a feature is worth building. *EWG*, the Evolution Working Group, handles the core language. It judges proposed language features.

LEWG, the Library Evolution Working Group, does the same for the standard library. It decides which library features the committee wants. If you have strong views on the tools you call every day, this room suits you.

5.3 The Wording Rooms: CWG and LWG

Two more rooms guard the exact standard text. The paper's author drafts the wording, and these rooms review it, fix it, and approve it. *CWG*, the Core Working Group, does this for the language. It reads every line for holes and for anything that could mean two things.

LWG, the Library Working Group, does the same for the library. These rooms ask whether the text says what the design rooms approved, not whether the feature is wanted. If you like finding the one case a sentence forgot, this is your kind of work.

5.4 Study Groups

Many new ideas start in a *study group*, or SG. A study group focuses on one area, like concurrency or networking, and shapes raw ideas.

They're numbered SG1 through SG23. Fifteen are active as of 2026, and the rest sit dormant until new papers wake them up. The committee page lists them all.

When an idea is ready, it graduates to a design room. The convener (the person who runs WG21) creates and closes study groups as needs change.

> **TRY THIS TODAY**
>
> Open the committee page on isocpp.org and read down the list of study groups. Find the one closest to the code you already write, and note who chairs it. Then look for that group's name on the audience line of a paper you care about.

5.5 How a Paper Moves Between Groups

A paper follows the lifecycle from chapter 3, and now you can name the rooms. A study group often shapes it first. Then a design room approves it, a wording room checks its text, and the plenary (the full committee) adopts it.

Each handoff is a vote to send it onward. The same vote can send it back for another revision instead, and a paper can face the same room more than once.

Some papers touch more than one room. A feature that changes the language and the library visits both evolution rooms, sometimes together in a joint session.

5.6 The Train Model

C++ ships on a fixed schedule, and that rule is *the train model*. Every three years a new standard leaves the station with whatever features are ready. C++14, 17, 20, 23, and 26 all rode it.

Herb Sutter pitched it at the committee's August 2011 meeting, and the minutes, N3316, record his case. Before the train, C++11 slipped for years. It's been the model since 2012, and P1000 lays out the current timetable. The schedule traded "now or never" for "catch the next train."

The model has a safety valve: pull what isn't ready. The committee did exactly that in 2019 with P1823, taking contracts out of C++20. A redesigned version caught a later train. In February 2025, the committee adopted P2900, which put contracts into C++26. The valve works, but pulling a big feature is never cheap.

5.7 How Features Get Removed

Taking a feature out is far harder than putting one in. Once a feature ships, real programs depend on it, and removing it breaks them. So the standard almost only grows, as chapter 1 warned.

Pulling is different. It takes a feature out before it ships, the way contracts left C++20. Removing something that already shipped usually starts

with *deprecation*, a formal notice in the standard that the feature may be removed in a later version. The actual cut comes years after that.

Take `auto_ptr`, an early smart pointer. It was deprecated in C++11 and removed in C++17, six years later. Exported templates were in the very first standard, but almost no compiler ever built them. C++11 cut them.

Cutting a feature from the text doesn't cut it from the world. GCC's standard library still ships `auto_ptr`, and you may meet it in old code you inherit.

5.8 The People Who Run It

A handful of officers keep the committee running. The convener appoints the chairs, creates study groups, sets the schedule, and judges whether the full committee agrees. SC22, WG21's parent committee, appoints the convener for terms of up to three years, with no limit on renewals. Two vice-conveners and a secretary help run things.

Two *project editors* keep the working draft. They're the people who apply every approved change to the text of the standard. It's quiet, exacting work that holds the whole document together.

Each room has its own small crew. A *chair* runs the session, and someone from the room takes the minutes. It might be you.

5.9 Who Sets Priorities: The Direction Group

A small senior group, the Direction Group from chapter 3, steers the committee's focus. It writes down where C++ should go in P2000 and what the next standard should put first in P5000. Membership is by invitation, not election.

The Direction Group leaves the votes on individual papers to the rooms. It sets the big-picture priorities that chairs lean on when scheduling. Framing your work to match those priorities helps it get attention.

5.10 Chair Power

The chair of a room holds more power than it looks. The chair decides which papers make the agenda, when they come up, how the polls are worded, and when the room has agreed. None of that is a vote, yet all of it shapes outcomes.

The strongest power is the calendar. A chair who never schedules a paper can stall it for years, without ever voting against it. So getting on the agenda is its own battle.

The calendar does have rules, though. SD-4 tells chairs what comes first: work tied to the Direction Group's priorities, work aimed at the next standard, and work that can make progress at this meeting. Everything else gets time as time allows.

You can watch the queue yourself. Every paper gets a tracking issue in a public list on GitHub. It shows you what's waiting and what's moving.

When one of those papers is yours, don't just watch it. Ask the chair for a slot.

5.11 The Rules: Standing Documents

The committee's own rules live in standing documents (the numbered SD files from chapter 3). Eight are current, and the numbering has gaps because older ones were retired. They run from study-group setup to how to write papers. The main one is SD-4, the practices and procedures.

Each design room also writes down how it judges papers. SD-9 holds LEWG's library policies, and SD-10 holds EWG's language principles. Read the one for your room before you write for it.

New standing documents need the full committee's agreement, and none of them expires on its own. So old practices tend to stay.

5.12 The Limits of Power: Volunteers and the Implementer Veto

Here's the truth under all the structure: nobody can be ordered to do anything. On paper, WG21 only recommends, as chapter 2 said, and the countries vote to make the standard official. To WG21, everyone is a volunteer, even the people an employer sends.

RULE OF THUMB

A proposal lives only as long as someone keeps showing up for it. Nobody can be assigned to do that for you. If it's your idea, the someone is you.

The compiler teams hold a quiet *implementer veto*. If GCC, Clang, and MSVC (the three implementers from chapter 1) won't build a feature, the standard can't make them. That's what stranded the exported templates from section 5.7.

It's a veto over what you can use, not over what gets adopted. The committee can standardize a feature nobody has built yet. Then the standard says one thing, your compiler does another, and you live in the gap.

This isn't only history. In January 2026, eighteen implementers wrote P3962 asking the committee to slow down, because features pile up faster than they can build them. The people who turn the standard real are stretched thin.

THE SHORT VERSION

EWG and LEWG decide what to build, CWG and LWG check the exact words, study groups feed them, and the plenary adopts. The train model ships C++ every three years with whatever's ready, and shipped features almost never come back out.

The convener runs the committee, the Direction Group sets priorities, and the chairs control the calendar. The rules live in eight standing documents, SD-4 above all.

WG21 runs on volunteers. On paper, it only recommends, and the countries make the standard official. And since nobody can make the compiler teams build a feature, they decide what you can actually use.

Now the acronyms have a shape. Next, let's get you to an actual meeting.

6. Getting There

Going to your first meeting is a leap. You're picturing the bill, a week away from work, and a room full of strangers. You can get ready for every one of them, and this chapter is your running start.

IN THIS CHAPTER

- What a six-day meeting week looks like
- Your homework before you fly
- What the trip costs in money and time, and who usually pays
- What to book, in what order, and when to land
- What to pack, including patience
- Why your first meeting will overwhelm you, and why that's fine

6.1 What a Meeting Week Looks Like

The committee meets in person three times a year, and the upcoming meetings page lists the dates and places. Each one is a *meeting week*, running Monday through Saturday. Days are long, roughly 8:30 a.m. to 5:30 p.m., with optional evening sessions on top. Saturday wraps up by 2 p.m.

During the week, the rooms run in parallel all day, six or seven at once. A short plenary opens the week on Monday morning, and everything builds toward the closing plenary on Saturday, where adopted wording enters the working draft. By Thursday, you'll feel every hour of it.

6.2 Preparing Before You Go

The best preparation is reading. Aim it at your *home room*, the one room that's your base all week. Read the papers on its agenda before you arrive. Try to track every room and you'll absorb none.

TRY THIS TODAY

Pick the one room you'd follow if you flew tomorrow. Open the latest mailing on wg21.org, filter it by group, and find three papers for that room. Start the first one tonight. Arriving with opinions about three papers beats arriving with none about thirty.

Two more steps round it out. Read SD-4 so the polls and rules don't surprise you. Then get onto the meeting wiki, which holds each room's agenda and the detailed schedule. It's password-protected, so ask the convener how a guest gets in.

Preparation is free. The trip itself is not.

6.3 The Cost and How to Pay for It

Attending has no registration fee. What costs money is the trip: flights, a week of hotel nights, and meals, which together can run into the thousands. Each meeting has a logistics paper, like N5057 for spring 2027, that tells you what to expect before you book. SD-5, "Meeting Information," covers the rest.

The bigger cost is often the week itself, away from your job. If no employer is sending you, that week may come out of your vacation.

If you keep coming, there's one more cost that nobody puts on the bill: three weeks a year far from home. Even the evenings belong to the meeting. Count that before you count the airfare.

People usually cover the expenses one of two ways. Many are sponsored by an employer who values the work, and others pay their own way.

Your first meeting has no membership fee, because you come as a guest. Coming back takes a seat through a national body, so ask early. Start with your own country's, since some charge nothing. If the fee is too steep, take the free seat from chapter 2. The trip is still yours to cover.

If the trip is too much this year, come by video. Meetings are hybrid, and a third or more of attendees join online. As chapter 4 warned, you'll miss the hallways, but you'll be in the rooms.

6.4 Booking the Trip

Start as soon as you've picked a meeting. ISO asks a guest for at least a week's notice, but a visa can take much longer. Go in this order:

- Email the convener to say you're coming as a guest, in person or by video, and name the country where you live or work.
- Read the meeting's logistics paper. It's usually out months ahead, with the venue, hotels, and directions from the airport.
- If you need a visa, don't wait. The paper may name a host contact who can send an official invitation, as N5057 does.
- Book your own hotel. If the paper names a main one, it's a good pick, since the Sunday orientation usually starts in its lobby.
- Land by Sunday afternoon for the 6 p.m. orientation, or a day earlier if you're crossing many time zones. Fly home after Saturday's closing plenary, not before.

Once the trip is booked, packing is quick.

6.5 What to Bring

Pack light, but pack a laptop. You'll read papers, follow along, and maybe take notes, all on screen. Bring its charger and a plug adapter for the host country. Download your home room's papers before you leave, and the flight becomes reading time.

Bring patience too. Inside a room, decisions come slowly and the debates run long. That's the work, not a sign something's wrong.

6.6 Bracing for Your First Meeting

Your first meeting will overwhelm you. The rooms all run at once, hundreds of papers are in play, and everyone seems to know each other. That's normal, and it passes.

> **WATCH OUT**
>
> Don't grade your first meeting by what you got done. Go in to win something, and six rooms will scatter you. You'll fly home feeling you failed. Your first meeting is for watching and learning. The payoff is your second one, when you walk in knowing the room.

Set your goal low on purpose. Stick with your home room for now, learn its rhythm, and let the rest wash over you.

You're not the odd one out. About two dozen new guests show up at every meeting, just as you will. The Sunday orientation exists because everyone starts here.

> **THE SHORT VERSION**
>
> A meeting week runs from a short plenary Monday to the closing one Saturday, with the rooms running side by side in between. Prepare by reading your home room's papers and SD-4. The week away from work and home often costs more than the airfare.
>
> Tell the convener you're coming, check the logistics paper, and land in time for Sunday's orientation. Bring a laptop and patience. Expect to be overwhelmed, and aim to watch, not to win.

You're as ready as anyone is for a first meeting. Forgive yourself now for everything you'll miss. Next, let's learn how to act once you're in the room.

7. In the Room

The first time in the room, you'll worry about doing something wrong. Don't. Once you know the unwritten rules, and a few written ones, you can relax and pay attention to the work.

IN THIS CHAPTER

- What the opening plenary is for, and your one moment in it
- What happens inside a session, start to finish
- When to leave your home room and follow a paper
- The small courtesies that keep a room running
- How to speak so the scribe and remote attendees can follow
- What stays in the room, and how everyone is expected to behave
- How to take minutes, and why it teaches you the room
- Where relationships form, in the hallway and over meals
- Why you should get out of your lane and stay skeptical

7.1 The Opening Plenary

The week opens with the whole committee in one room. This is the *opening plenary*. The officers say hello, first-time attendees introduce themselves, the agenda and the updated working drafts get approved, and then everyone scatters to the subgroups.

Don't expect drama here. The opening plenary is short and mostly logistics. The one part that's yours is the introduction. You stand, say your name and where you're from, and sit back down. The room nods and moves on. The work starts in the rooms right after.

7.2 Inside a Session

Most sessions run on the same loop. You slip into a seat near the back. The chair, who runs the room, calls up a paper, and its author presents. Then hands go up, and the chair keeps a queue of who speaks next.

Questions and pushback follow, and the author answers. Often the discussion ends in a quick poll of the room, which chapter 8 explains. Then the next paper comes up. Wording rooms run the loop on the text itself. After a few sessions, you'll see each step coming before the chair says a word.

7.3 Cycling Between Rooms

Keep your home room, but follow the paper. The room you picked in chapter 6 is still your base. When a paper you care about comes up somewhere else, go with it. Each room's agenda on the wiki shows when that paper is up.

Before you go, read that room's papers too. Walk in cold, and you'll spend the session decoding instead of listening. The discussion moves fast and won't recap for you.

7.4 Meeting Etiquette

The room runs on small courtesies. Don't dominate the discussion, and don't reopen a point the room has moved past. Keep your laptop for notes, not for distractions that pull your focus.

Listen more than you talk at first. Spend your early sessions learning how the room thinks.

7.5 How to Speak

When your turn in the queue comes, make one point, and keep it to about two minutes. The room has a lot to get through, and short is respected.

Two habits help everyone follow you. Say your name before you start, because the scribe and the remote attendees need it. Use the microphone, because without it the online side of the room can't hear you. Every room has one somewhere, so find it before you raise your hand.

RULE OF THUMB

You'll earn your voice faster by using it sparingly. Save it for the point nobody else in the room is making.

7.6 Confidentiality and the Code of Conduct

What happens in the room mostly stays in the room, and that protects you. You can float a half-formed idea, or change your mind midweek, and never see it quoted online. That's the *confidentiality rule*.

You don't live-tweet or blog from the sessions, record them, or photograph other people's screens. A screenshot of the slides for your own notes is fine.

Quoting works like the reflector rule from chapter 4, with the same exceptions. Poll questions and their numbers are free to share once the meeting is over, as the March 2026 minutes, N5040, spell out.

For everything else, paraphrase. When you write up the week afterward, describe what was said, never who said it.

WG21 follows two codes together: the ISO Code of Ethics and Conduct and the IEC Code of Conduct. They cover how you treat people in meetings and on social media. The short version is to be decent and assume good faith.

A slide at the start of every meeting sums them up. In a room drawn from dozens of countries, one line is worth taping to your laptop: "be aware that jokes and humor may not translate." If something goes wrong, SD-4 points you to ISO's process for reporting a concern.

7.7 Note-Taking

Every session has to be minuted, and anyone can be asked to do it. SD-4 says everyone present "should be prepared to take minutes." So you might be handed the keyboard on day one. Say yes.

The minute taker is called the scribe, and no job gets you inside a room faster. Writing an argument down is different from hearing it, because you have to understand it well enough to sum it up. A week of it teaches you what your room is wrestling with. People also remember who helped.

To volunteer, put your hand up when the chair asks for a scribe. The notes usually go straight onto the room's page in the meeting wiki. Don't try for a transcript. Get the gist of each point, and get every poll's exact wording and numbers right, since those are what people quote later.

The scribe also gets a small superpower. Everyone else waits their turn to speak, but you can usually skip the queue for one reason. If you missed a point, ask the speaker to repeat it, because the minutes have to be right. That lets you ask the question everyone else was too proud to ask.

7.8 Where Relationships Form

The hallway is not break time. A lot of the real work happens there, between sessions. People sort out disagreements over coffee that would stall a formal room.

Meals work the same way. Lunch and dinner groups form on their own, so join one. Introducing yourself over a meal is normal and expected, not pushy.

> **WATCH OUT**
>
> By midweek, the people at your lunch table will feel like friends. That's the good part, and it's also the trap. Liking people makes it easy to agree with them. Start nodding along, and you lose the one thing you walked in with: a fresh eye.

7.9 Get Out of Your Lane

Once your home room feels familiar, spend some time in rooms outside your specialty. The Direction Group's P2000 puts it plainly: "try to spend at least one day each meeting in a WG that isn't 'your own.'" You'll build trust across groups and see how the whole machine fits together.

Two habits earn respect fast. Serve before you push: scribe a session, or tell an author what you think of their paper. And when someone says "that's just how it works," ask how other groups like the Internet Engineering Task Force or the World Wide Web Consortium handle the same thing.

Stay skeptical, and question the room's assumptions, including your own.

THE SHORT VERSION

Keep a home room, but follow the papers you care about when they move, and read them before you walk in. Listen more than you talk at first. When you do speak, say your name, use the microphone, and make one short point.

Keep the room off the public record: describe what was said, never who said it. Be decent to everyone. Say yes when you're asked to take minutes. Get to know people over meals, and spend a day outside your own room. Through it all, question the room's assumptions and your own.

You'll pick up the room's rhythm faster than you expect. Then it pulls at you: sessions back to back, evenings that run late, poll after poll. A packed week passes for progress, even when nothing you care about moved. On the flight home, ask what moved. Next time, ask whether you helped move it.

The next mystery is how all this talk turns into decisions.

8. How Decisions Get Made

Voting here doesn't work the way you'd expect. No majority rules, and no simple count wins. This chapter shows you how the committee really decides, so you can read a room and vote with a clear head.

IN THIS CHAPTER

- What consensus means, in ISO's own definition, and what it isn't
- The five-point straw poll, who may vote, and how strongly
- The 2:1 guideline, why the count isn't everything, and how a small minority can block
- Straw polls versus formal votes
- When to vote, when to abstain, and why neutral should be rare
- Direction versus design approval, and why neither is a promise to ship
- What to do when you can't live with a decision, and the deadline for saying so
- What actually happens at the closing plenary
- How old bugs in the standard get fixed, and how you can help
- The two national ballots, and which one changes the text
- What a good ballot comment looks like

8.1 The Culture of Consensus

The committee decides by *consensus*, which isn't what most people think. It doesn't mean everyone agrees, and it doesn't mean the majority wins. It means no sustained opposition is left standing.

SD-4, the practices guide from chapter 3, quotes ISO's definition: "general agreement, characterized by the absence of sustained opposition to substantial issues." The test is whether people can live with a decision.

A few mild objections don't block it. A small group of firm, reasoned objections can.

> **RULE OF THUMB**
>
> Consensus is social pressure with a polite name. Keep your wits about you, and don't let the room's mood stand in for your own judgment.

8.2 Straw Polls and the Five-Point Scale

To read the room, the chair takes a *straw poll*. In the usual kind, you pick one of five positions: strongly favor, weakly favor, neutral, weakly against, or strongly against. Where the room clearly agrees, the chair may just ask whether anyone objects.

Pick the strength as carefully as the side. Weakly against means you'd rather not. Save strongly against for something you truly can't live with. Those are the hands that sink proposals.

In a subgroup, everyone in the room can vote, guests included. The chair weighs the discussion and the numbers, then decides whether there's consensus. No fixed number settles it, so the chair judges.

8.3 Reading Poll Results

A rough rule guides the call. A proposal normally advances when those in favor outnumber those against by more than two to one. This is the *2:1 guideline*, and the word *normally* is where the chair's judgment lives. Neutrals stay out of the ratio, but a mostly neutral room is a warning sign.

ISO's definition has a second half, and it's why the ratio comes last. Consensus also takes a process that hears everyone out and tries to reconcile the arguments. So the chair may first ask the against voters why, and may poll again.

But the count isn't everything. A small group can stop a lot. Because the bar is no sustained opposition, a block of strong-against votes can sink a proposal even when far more people favor it. That's built into the system, not a loophole.

This really happens. In a January 2022 online poll, one large library proposal for C++23 drew 37 in favor and 17 against, 11 of them strongly.

It still failed for "sustained strong opposition." More than two to one wasn't enough.

That power has a limit, though. SD-4 says consensus is "not a tyranny of the majority," and it's "not a tyranny of the minority" either. Once objectors have been heard, they can't just keep saying no.

8.4 Straw Poll versus Formal Vote

A straw poll is only a temperature check. A *formal vote* is the binding kind, taken at plenary and on the national ballots. One steers the work, the other commits it.

Still, take straw polls seriously. The committee treats them as honest predictors of the formal vote to come. How you vote in a subgroup signals where you'll land later.

Whichever vote it is, you face the same question: should you vote at all?

8.5 When to Vote and When to Abstain

Vote your honest position when you have one. If you favor something even weakly, say so. Neutral means you read the paper, weighed the issues, and truly land in the middle, which should be rare.

If you vote against, be ready to say why, and to name what would change your mind.

The chair calls for strongly favor, and a forest of hands goes up. Weakly favor, and more follow. When the chair calls for strongly against, yours is the only hand in the air. Heads turn. The chair asks whether you'd like to say why.

You give your reason in two sentences and name the change that would win you over. The chair thanks you and moves on. Your hand didn't stop anything, but your concern is on the record now. That's the poll doing its job.

When the topic is outside what you know, *abstain*. Abstaining means not voting at all, and it's the honest move on a paper you haven't read. Hearing a summary isn't the same as knowing the issue.

WATCH OUT

Voting with the room on a paper you haven't read feels safe, and raising your hand feels like belonging. But the chair reads every hand as judgment, so a borrowed vote is noise that still counts. Read it or don't vote.

The Direction Group (the senior group from chapter 5) makes the matching point about voting against. Its paper, P2000, says "I didn't have time to read the document" is no reason to oppose a paper that came in on time. So don't oppose what you haven't read, and don't support it either.

Whatever you vote, ask for evidence. Don't vote yes on enthusiasm or a friend's nod. Be moved by proof, and vote your conviction, not the room's momentum.

8.6 Two Gates: Direction versus Design

A proposal passes through two different gates. *Direction approval* is early: does the committee want this kind of thing at all? *Design approval* is later: is this specific design the right one?

Here's the trap. Passing either gate is not a promise to ship. A warm direction poll can sit for years with nothing delivered.

Read encouragement for what it is. "We'd like to see more work" means keep going, and nothing more. Expansion statements, P1306R1, won approval for C++20 in 2019 from EWG, the language design room. They missed C++20 anyway and were adopted for C++26 in 2025, six years later.

8.7 When You Can't Live with It

Can't live with where the room is headed? Say so early: on the reflector before the meeting if you can, or during the discussion, before the poll. A lone objector is weaker than one who's found an ally on the merits.

If you still think a serious mistake is being made, there's a short *escalation path* with a hard deadline. Tell the EWG and LEWG chairs as early as you can. Post the concern on the subgroup's reflector by 5 p.m. the evening before the closing plenary, and raise it with your national body.

At plenary the convener may ask your delegation's chair whether the objection is personal or national. It's only national if the delegation has already talked it through.

You can still raise a hand at plenary and force a count. But SD-4 says an objection that skipped these steps, new or repeated, "should not be given weight." After the deadline, the concern waits for a later round: a new reflector thread or a national ballot comment, backed by at least a draft paper.

8.8 The Plenary: Where Features Enter the Draft

The week ends with the *closing plenary* on Saturday. Each subgroup reports, and the committee polls on motions to adopt finished wording. A feature enters the working draft the moment its motion passes.

Plenary usually runs on silence. The convener (the person who runs WG21) asks if anyone objects, and if no one does, the motion passes. Raising a hand to object takes no explanation.

An objection forces a three-way poll: favor, against, or neutral. Only experts in the global directory vote, so guests watch. The convener then judges whether there's consensus. Plenary is where objections are counted, not where they're born.

Passing the subgroups isn't the final word. A proposal can clear every room and still fail at plenary. The type `std::byte` did exactly that in November 2016, failing with 24 in favor and 15 against, after a late fight over its name.

It came back at the next meeting and passed 44 to 2, name unchanged. What did change was the paper trail. In between, P0583R0, "`std::byte` is the correct name," answered the objection in writing. A loss at plenary isn't final. Your answer to an objection is a paper.

8.9 Processing Issues

Not every decision is about new features. The wording rooms, CWG and LWG, work steadily through the core and library issues lists, the standard's bug trackers. They settle each defect's exact fix, then move the ready ones at plenary in one batch.

It's quiet work, and you can help without a vote. File a clear issue with a proposed fix, as chapter 3 suggested, and this is where it gets decided.

8.10 The Countries Vote

Before the standard is final, the countries vote, and each one gets a single vote. They usually see the draft twice, and the first round is where you can help.

First it goes out as a *committee draft* for comments. That's the round meant for changing the text. That round gave C++26 its 411 comments. A good comment works like a small paper: it says what's wrong, where, and what text would fix it. Drafting one for your national body takes no seniority.

Then the cleaned-up draft goes out for approval as a *DIS* (Draft International Standard). For C++26, that ballot was scheduled to run from July 31 to October 9, 2026. Passing takes a strong majority. Two-thirds of the voting countries must approve, and no more than a quarter of the votes can be *no*.

A country with concerns usually votes *yes* and attaches comments. A *no* is blunt. SD-4 says it means the country doesn't want the work "to progress at all in any form." Every comment gets considered whatever the vote, so a *no* buys nothing extra, except the risk of sinking the whole draft.

If the DIS passes and the text needs no technical changes, the draft goes straight to publication. Technical changes trigger one last, shorter vote on a near-final *FDIS* (Final Draft International Standard). The committee aims to make the DIS its last ballot, so a design change this late needs nearly everyone on board.

> **THE SHORT VERSION**
>
> Consensus means no sustained opposition is left standing, not that the majority wins. In a straw poll, more than two to one in favor usually moves a proposal forward, but a firm minority can still sink it. Once heard, though, it can't just keep saying no.
>
> Vote your honest position, even as the only hand in the air. Ask for evidence, and abstain on papers you haven't read. Direction and design approval aren't promises to ship.

> Plenary is where objections are counted, not born, so raise yours by 5 p.m. the evening before. Then the countries vote, and the committee draft round is where a good comment can still change the text.

You can read a room now, and you don't have to follow it. Your best question in any room is "show me the evidence," and your best answer is a paper. Chapter 9 shows you how to build a good one.

9. What Goes into a Proposal

You have an idea you love, and you think it belongs in C++. The room won't care how you feel about it. It will ask what you can prove. This chapter is the cold shower and the toolkit: what a proposal must bring to earn the committee's time, and why the bar is so high.

IN THIS CHAPTER

- The low bar you control, the yes you don't, and the four arguments a real proposal makes
- The three pillars the committee asks for, and the one that's easiest to skip
- Writing an abstract a busy stranger can grasp
- Tony Tables, and why honest ones win
- Steel-manning your own idea, twice
- Implementation experience and the feature-test macro
- Writing wording that reads like law, and when to bring in a wording expert
- Teaching your feature with a short tutorial
- Scope, dependencies, and the high bar

9.1 What Belongs in a Paper

The low bar is all in your hands: a paper, on time, with someone to present it. Then you ask the chair for a slot on the agenda, and chapter 10 shows you how. But getting a yes is a different game.

A real proposal makes four arguments. It shows the problem is worth solving, lays out the design, weighs the alternatives, and proves the thing works. Miss one and a reviewer will find the hole. And do your homework: cite the earlier papers on the same ground, because the room remembers them.

You don't need coauthors, and plenty of papers have none. In the February 2026 mailing, more than half the papers had one author. The Direction Group, the small senior group that steers the committee, agrees.

The group's P4024R0 says a paper from "a single author citing feedback from diverse reviewers" is just as valid as a coauthored one. So thank your reviewers by name, and say what kind of C++ each one writes. That shows your design survived readers outside your own corner.

9.2 The Three Pillars

Three habits make those four arguments land. Call them the *three pillars*: lead with concrete examples, argue from clear principles, and show the alternatives you weighed. SD-4, the practices guide from chapter 3, asks every proposal for all three. It calls them the difference between "someone's cool idea" and "a real proposal."

A paper missing a pillar feels thin. Examples do the heavy lifting, and alternatives are where you'll be tempted to cheat, so each gets its own section below.

Principles are the pillar that's easiest to skip. Name the design rules your proposal follows. SD-4 wants your design to fit the language's "principles and design philosophy," and "explicitly not" its quirks. It even suggests citing Bjarne Stroustrup's book on how C++ was designed, which chapter 12 introduces.

9.3 The Abstract as Elevator Pitch

Most readers decide from the abstract alone. So write it as an elevator pitch a busy stranger can grasp. The first sentence should state your finding, not wind up to it.

Keep it honest, though. Promise only what the paper delivers, because reviewers punish hype. A title that names the result beats a title that names the topic.

9.4 Tony Tables

One of the committee's favorite tools is the *Tony Table*, a side-by-side of old code and new. You put the current way on the left and your way on the right. The reader sees the win without you claiming it.

The name honors Tony Van Eerd, who came up with the format. His tables of C++17 features show it at its best.

Make the table honest to earn trust. Include the cases where your design isn't shorter or clearer. A table that hides its weak spots gets caught and loses the room.

9.5 Show the Alternatives: The Steel Man

Don't dodge the alternatives. Attack your own idea first. A *steel man* is the strongest possible case against your proposal, stated fairly, which you then answer with evidence.

Build two of them. One argues that no standard feature is needed at all, that a library would do. The other argues that a rival design is better, and then you show why yours wins.

This is the hard part. You've lived with this design for months, and you love it. Now you write the best case against it, so fair a rival would sign it. Go easy on it, and someone in the room won't. They'll make that case out loud, with your paper on the screen.

Don't stack the deck. The Direction Group's P2000R5 says that showing only your design's strengths and only a rival's weaknesses "is not acceptable."

The strongest answer to both steel men is the same: evidence the room can check.

9.6 Implementation Experience

Nothing beats proof that your idea works. *Implementation experience* means you've built the feature and used it, ideally in more than one place. The committee wants that proof before it freezes anything forever.

No rule demands it, though. The submit-a-proposal page says there's no ISO requirement to build it first, only to win consensus. But it adds that "consensus is easier to achieve when there's a working implementation."

WATCH OUT

Optional in the rules, expected in the room. Show up with no code and you're asking everyone to take your word for it, and that's a hard sell. Build your feature anyway, even for your first paper.

The bar is real but not impossible. For a library, ship code that compiles, passes tests, and has real users. For a language feature, a rough compiler fork that shows it working is enough to start.

One small piece rides along with new features: the *feature-test macro*. It's a predefined name that lets code check whether a compiler has your feature yet. Including it lets people adopt the feature safely while support spreads. The rules and the full list live in SD-FeatureTest.

9.7 Writing Standard Wording

Standard text isn't prose, it's closer to law. Two small words matter more than they look: *shall* marks a hard requirement, and *should* marks advice. Mixing them up changes what implementers must do.

Point to the right place, too. Cite the stable section labels (the bracketed names from chapter 1) instead of page or section numbers. The numbers shift between drafts. The labels don't.

Don't write your first wording alone. Once the design is settled, SD-4 expects you to find a wording expert to help you draft it. The chairs will help you find one.

CWG or LWG (the wording rooms from chapter 5) then goes through your text with you, fixes what's wrong, and approves it. Have your expert on board before that review starts, not after the room finds the holes.

9.8 A Short Tutorial

A great proposal teaches its feature. Include a short tutorial that shows someone how to use it. P2000 asks for one early, because a tutorial keeps a feature meant for "ordinary programmers" from drifting into expert-only territory.

If you can't explain it in a few clear lines, the design may be too complex.

9.9 Scope and Dependencies

Keep each paper to one topic. If your title needs the word "and," split it in two. Big, bundled proposals stall, while small focused ones move.

For a large idea, ship it in stages. Land the useful piece first, let people use it, and build the rest on top. Some features even wait on another paper to land before they can.

9.10 The High Bar

The burden of proof sits on you, the proposer. The default is no, because a mistake is forever. Being useful isn't enough, so show why only the standard can deliver what you need.

> **RULE OF THUMB**
>
> Be convinced by overwhelming evidence and nothing else. Not by the room, not by a big name, and least of all by how much you love your own idea.

Language features are the hardest of all. A library lives on GitHub if the committee says no, but a language feature has nowhere else to go. So the committee guards the language even more tightly than the library.

Most new-feature proposals don't make it, and that's the system working. Big efforts have stalled or died, even ones that got as far as a TS (the experimental tryout from chapter 1).

The Networking TS came out in 2018 and still isn't in the standard. In 2021, the library design room's online polls leaned toward building networking on a different model for asynchronous code. That model, `std::execution`, didn't land until C++26.

The Transactional Memory TS never merged. A 2019 paper, P1875R0, said it "has not been widely implemented and used," and proposed a "lite" version instead.

One stalled over design, the other for lack of proof.

THE SHORT VERSION

The low bar is in your hands: a paper, on time, and someone to present it. Getting a yes is hard. Make the four arguments, and build them on the three pillars SD-4 asks for: examples, principles, and alternatives.

Show an honest Tony Table, answer two steel men, and prove it works. Once the design is settled, find a wording expert before CWG or LWG reviews your text. Keep to one topic, and expect the default answer to be no.

Keep loving your idea, but love it with clear eyes. Now let's actually write and submit the paper.

10. Producing and Submitting Your Paper

You know the bar your idea has to clear. Here's the practical path: the tools, the format, and the route from a first post to the committee's agenda.

IN THIS CHAPTER

- Why you float the idea before you write it, and the "no surprise competitors" rule
- A prototype that readers can run in the browser and study in a public repo
- The tools that format your paper, and the template to start from
- Making the paper easy to read, even on a phone between sessions
- Coauthors, revision after revision, and the patent rule
- How to submit, and the one click that beats the mailing deadline
- How to get on the agenda, and why asking the chair isn't pushy
- Why design approval leaves you with "the other 80%"

10.1 Float the Idea First

Before you write a full paper, float the idea. Post a short description to std-proposals (the public list from chapter 4), covering the problem it solves and the alternatives you weighed. Then see who bites. The official submit-a-proposal page makes this its first step, too.

This saves you from writing a paper nobody wants. You'll find prior art, hear objections early, and maybe meet a coauthor. Applause on a list is encouragement, and chapter 9's bar still stands.

Floating the idea also finds your competitors before the room does. Check for other papers on the same problem, and talk to their authors first. The

Direction Group even has a name for this, the "no surprise competitors" rule in P4024R0.

Best case, you and the other author merge your designs into one paper. If you can't agree on one design, write a joint paper that compares both and lays out the trade-offs. Chairs can ask for either one before they give your papers more time.

Alone or together, once the idea has interest, build something before you write it up. The strongest part of a paper is proof that it runs.

10.2 Prototyping: Compiler Explorer and GitHub

Let readers run your code, don't just describe it. Compiler Explorer, at godbolt.org, runs C++ in the browser across many compilers. Share a live link and the reader sees it work.

> **TRY THIS TODAY**
>
> Build a live Tony Table. Put the code people write now on Compiler Explorer, where anyone can run it. For a library, paste your prototype beside it. For a language feature, sketch the syntax in a comment for now. Save the link. It's often the first thing a reviewer clicks.

Put your implementation on GitHub. A public repo lets people read the code, file issues, and try it themselves. It also tracks your revisions as the design evolves.

10.3 Writing the Paper: Tools and Template

You don't format a paper by hand. Most authors use mpark/wg21, which turns Markdown into a committee-styled document, or Bikeshed, which builds HTML specs. Newer tools like COWEL and plain LaTeX work too.

Start from a template, which gives you the front matter and the expected sections. The tools above ship their own, and many authors begin there. The submit-a-proposal page also has an *official template*, the "Template for Library Proposals." It was built for library features, but its questions make a good checklist for any paper.

10.4 Make It Accessible

Use enough color contrast, a monospace font for code, and clear headings. Don't lean on color alone to make a point.

A clear paper is one the busy delegate can skim on a phone between sessions. Accessible papers get read.

10.5 Coauthoring and Revision History

A coauthor splits the writing, can build while you draft, and brings reviewers you don't know yet. You don't need one, as chapter 9 showed. List everyone in the front matter, and name one person as the presenter.

Few papers survive unchanged. Expect to revise, again and again. Each revision (R0, R1, R2) answers the last round of feedback. Reviewers read what changed, so spell out your changes.

10.6 Patent Disclosure

ISO has a patent rule, and you must follow it. If you know of a patent that covers your proposal, disclose it before the work advances. The ISO patent policy spells out the duty.

10.7 Submitting the Paper: SD-7

SD-7 covers the mechanics: what goes in the header, which file formats work, and when the mailing deadlines fall. Read it once before your first submission.

Remember the basics from chapter 3. Your paper gets a P-number and a revision, and it has to land before the mailing deadline. SD-7 puts that deadline at 2 p.m. UTC on the Monday four weeks before the meeting.

A new paper that misses the deadline waits for the next round. A late update to a paper that made the deadline can still be heard.

You upload the paper at isocpp.org/papers, which needs an isocpp.org login. The system hands you a P-number, or bumps the R on a revision, so make sure your title page matches. A saved draft is only a working copy, so submit the paper itself before the deadline.

Once it's in a mailing, your paper gets an issue in the GitHub tracker from chapter 5. Watch it to see where you stand. Some chairs schedule from that tracker, and a label on the issue shows when your paper gets a slot.

10.8 Getting on the Agenda

A paper with no presenter goes nowhere. Email the chair (the person running the room) before the meeting to ask for agenda time. Confirm whether you'll present in person or by video.

> **WATCH OUT**
>
> Don't wait to be invited. A paper in the mailing isn't a paper on the agenda. If you don't ask the chair for a slot, you may find out on Monday that yours was never in the plan. Asking isn't pushy. Waiting can cost you a whole meeting.

The chair you want runs the room your paper goes to first. That's usually a study group or one of the evolution rooms from chapter 5. The committee page lists every chair. If you can't tell which one is yours, ask any of them, and they'll point you to the right one.

When your slot comes, be ready to present and to listen. You'll show the problem, the design, and the proof, then take questions. The room's reaction steers your next revision.

10.9 The Other 80%

Winning design approval (the "this design is right" gate from chapter 8) feels like victory. You'll think you're 80 percent done. The committee's own page on the life of a proposal grants you that, from a design point of view. Then it calls the rest "the other 80%."

After design comes the long part. A wording room polishes the exact text, and the closing plenary votes it into the working draft. Then the whole draft goes to the countries, first as a committee draft for comments, then as a Draft International Standard for approval.

Stay with your paper the whole way, and watch for ballot comments that touch it.

THE SHORT VERSION

Float the idea on std-proposals first, and talk to anyone working on the same problem. Then build something: put your code on Compiler Explorer and GitHub so people can run it. Write it up with mpark/wg21 or Bikeshed, and make it easy to read, even on a phone.

Name a presenter, expect to revise again and again, and disclose any patent you know of. Submit by the SD-7 deadline, then ask the chair for agenda time. Design approval will feel like 80 percent of the job. What's left is "the other 80%."

That's the route, from a short post on std-proposals to a slot on a chair's schedule. Get R0 into the next mailing before its deadline. Next comes the art of championing it.

11. Championing Your Paper

Your paper's in the mailing, with your name on the front and a P-number of its own. Enjoy that for a day. A paper doesn't pass itself. Getting to yes is a long, social campaign, and it's the part you're most likely to underestimate.

IN THIS CHAPTER

- Why most papers are won before they're presented, and how reviewing and coauthoring other people's papers earns you supporters
- Finding a champion, or being one for someone else
- Surviving the first gate: "do we want it at all?"
- What to do when a poll goes against you
- Back-pocket alternatives and the max-min solution
- Disagreement versus opposition, and when to withdraw
- How reputation compounds, and how the presenter is judged
- The long game, the headwinds, and protecting yourself

11.1 Before the Room: Presocialization

Most papers are won before they're presented. *Presocialization* means talking up your idea in hallways and on the lists, building support ahead of the room. A cold presentation to a surprised room usually stalls.

Once you're in the room, the five minutes before a poll settle what the hallway left open. The chair's summary, the first voice to speak, and the room's mood shape the result. Line up a few supporters who'll speak early, people who actually reviewed the paper and can answer hard questions.

Yes, that builds momentum. Momentum built on merit is fair game. Momentum built on numbers is what chapter 8 told you to resist.

Supporters aren't conjured the week before a poll. You earn them over years, starting with other people's papers. Send an author a careful review, then another on the next revision, and before long authors start asking you to look at their drafts. The people whose papers you review tend to review yours.

A careful review is rare, so authors remember who sent one. Keep at it, and they'll ask for more: a section, an implementation, and finally your name on the paper. People you've coauthored with will speak up for your paper in rooms you can't be in. That's where champions come from.

11.2 Finding and Being a Champion

Every paper needs a champion (the person who presents and pushes it, from chapter 2). Often that's you, the author, but it doesn't have to be. If you can't present it yourself, in person or by video, find someone who knows the work well enough to defend it.

You can also champion someone else's paper. Backing a good idea you didn't write builds trust and skill. It's one of the fastest ways to become useful.

11.3 The First Gate: "Do We Want It at All?"

The first discussion of your paper is the scary one. The room asks "do we want this at all?" before it touches any detail. A clear no here can kill the idea outright.

Frame your opening as solving a problem the committee already cares about. Tie it to the Direction Group's priorities from chapter 5, laid out in P5000R1. A paper that matches those priorities gets heard first. One that doesn't waits.

Even a wanted paper will take fire.

11.4 When the Poll Goes Against You

Most polls against you aren't a death sentence. They send the paper back for more work, and that's the normal path. You take the feedback, revise, and return with a stronger paper.

Know the one risky move: reopening a settled question. *Re-litigation* is re-debating something the room already decided, and it costs you credibility. Reopen only with genuinely new evidence. The same argument twice costs you twice.

11.5 Negotiating: Back-Pocket and Max-Min

Always have a fallback. A *back-pocket alternative* is a second design you're ready to offer if your first choice is rejected. Walking in with one keeps you from leaving empty-handed.

When the room is split, shrink the ask. The *max-min solution* is the smallest version everyone can accept. A small win that ships beats a big proposal that stalls.

11.6 Disagreement versus Opposition, and When to Withdraw

Learn to tell two things apart. Disagreement is a technical objection you can answer with evidence or a better design. Opposition is someone who doesn't move, however well you answer. The two need different responses.

Answer disagreement in full, before the poll and again in your next revision. That's how papers get better. Answer opposition once, on the record, then stop trying to win that person over. Watch the room instead. One holdout rarely sinks a paper. A small group that shares the objection is a different story.

Sometimes the right move is to stop. If the room clearly doesn't want it and won't budge, withdraw the paper with grace.

> **RULE OF THUMB**
>
> Knowing when to stop is a skill, not a defeat. A dead paper keeps eating the time and credibility your next one will need.

11.7 How Reputation Works

In a room of volunteers, reputation is what gets your paper read. People can't review every proposal, so they weigh who wrote it. A trusted name gets the benefit of the doubt, while a new one gets extra scrutiny.

You build it in small steps, and the first ones are the careful reviews you send other authors. Then you become the reliable expert in one narrow area, and you spend your credibility only on the battles that matter. Reputation compounds each time you turn out to be right.

11.8 The Presenter Is Judged Too

When you stand up to present, you're judged again, live. Are you reasonable, willing to concede a fair point, someone people want to work with? Those signals shape how your paper lands.

Don't be the obstacle. Causing needless delay, or escalating every dispute, marks you as hard to work with. The escalation path from chapter 8 exists, but SD-4 warns that using it topic after topic erodes your credibility.

Persistence does pay, though. *Procedural momentum* is the benefit a paper earns from many revisions and prior polls. By the fourth or fifth revision, the room often assumes it'll pass, so steady iteration works in your favor. When you're the one voting, remember that a revision count isn't evidence.

11.9 The Long Game

Set your clock to years, not months. The train runs every three years, and most features ride more than one. Patience is the job.

The famous features all took ages. Coroutines, modules, and the executors work that became `std::execution` each ran many years and many revisions. Stackful coroutines have waited over a decade. Their paper got to revision 24 in July 2026, and they still aren't in.

11.10 The Structural Headwinds

Some headwinds aren't about you. The first is the *Bandwidth Gap*: far more papers arrive than anyone can review. Chapter 3 counted eight hundred or more a year. Nobody tracks what delegates read, but a typical one gets through a few dozen, mostly for their own room.

> **WATCH OUT**
>
> A quiet inbox isn't a *no*. When nobody writes back about your paper, most people simply haven't read it yet. Give up over that, and you've lost to a crowded mailing, not to a bad idea.

Write for the reader who has two minutes: put your abstract and a Tony Table (chapter 9's before-and-after code) right up front.

Another is the *expert bubble*. Papers are often written by experts for experts, which makes it hard for newcomers and outside ideas to break in.

Two habits poke holes in it. Review other people's papers on wg21.org, where your careful read sits in public beside the paper. And give your own paper the short tutorial from chapter 9, written for ordinary programmers.

Resources tilt the field, too. Everyone in the room is a volunteer, as chapter 5 said, but some are paid by an employer to be there.

Proposals with that paid time and a funded implementation move faster than ones built on spare evenings. It's not a conspiracy, just the weight of who can afford to show up every cycle.

If you're on your own time, keep the paper small enough to finish. Then look for a compiler or library developer who wants the feature. For a library feature, the Beman Project from chapter 4 gives you a place to build it in the open.

All three headwinds wear people down, and you'll feel them too.

11.11 The Emotional Side

Having your paper picked apart stings, and that's normal. Don't take it personally. The room argues hard about every idea, and today it was yours.

Your slot ends, the poll goes up, and the numbers send you back. For an hour it feels like the room voted on you. That night you reread your notes, and two of the objections are fair. By breakfast you've sketched the fix. R1 has its first change.

Burnout is real here, and you won't be the only one who feels it. Good people leave because of it. In a 2020 pulse poll, over 60 percent of those who answered said they often felt exhausted by meetings and other demands. That was the year meetings moved online.

Set your limits before you need them. Keep to your home room and one more, cap your telecons, and read the papers you'll vote on instead of the

whole mailing. Let an evening session go when you're spent. And know when to step back for a meeting or two.

The committee runs for decades, and it needs you around for more than one standard. A heroic burst burns out first.

THE SHORT VERSION

Win your paper in the hallway before the room, and earn supporters years earlier by reviewing, then coauthoring, other people's papers. Find a champion or be one. Frame your idea around the Direction Group's priorities. A poll that sends you back is feedback for the next revision, but don't re-litigate without new evidence.

Keep a back-pocket alternative ready, and shrink to the max-min solution when the room splits. Answer disagreement in full and opposition once, and know when to withdraw. Reputation compounds, and so does procedural momentum. Think in years, mind the Bandwidth Gap and the expert bubble, and set your limits before you need them.

Your name stays on the front for years, so plan to last all of them. Some proposals die for reasons no campaign can fix. Next, the deep design rules that decide what's even possible.

12. C++ Design Principles

Not every no is politics. Some come from old rules the room rarely bends.

IN THIS CHAPTER

- Where the rules come from, and the book to read before you propose a language change
- Why the safest path is to standardize what already exists
- Backward compatibility, and why the committee is reluctant to ship a second version of a type
- ABI, the constraint you're least likely to see coming, and where the committee stands on breaking it
- The zero-overhead principle, and the famous feature that breaks it
- Freestanding and hosted environments, the two worlds C++ runs in

The oldest of these rules come from Bjarne Stroustrup. He wrote them down in 1994 in *The Design and Evolution of C++*, the book everyone calls D&E. EWG's own principles, SD-10, still start from his list. The book explains why each early feature is the way it is, and what was tried and thrown out.

Read it before you write a language proposal. Many of the objections you'll hear are in it, with the answers.

12.1 Standardize Existing Practice

The committee's safest path is to *standardize existing practice*. That means blessing a design that already works in the field, instead of inventing something new at the table. A design with years of real use behind it clears the bar far faster. Think of it as chapter 9's implementation experience, at full scale.

The pattern shows up again and again. The fmt library became `std::format`, and range-v3 became `std::ranges`, each after proving itself in the wild. Designs invented at the table are a gamble.

Why so cautious? Because the standard can't easily fix its mistakes.

12.2 Backward Compatibility

C++ makes a hard promise: old code keeps working. This is *backward compatibility*, and it guards decades of existing programs. The committee breaks it rarely, and usually after long warning. Break working code, and you're arguing against everyone who owns it.

That promise means a flawed type usually can't be changed in place. A fix would have to ship beside it as a second type, and the committee is reluctant to do that. A second, "safer" vector would split the library, since functions written for one kind wouldn't accept the other.

So one type has to serve everyone. When the committee does add a second type, everyone lives with both. C++20 put `std::jthread` beside `std::thread` rather than change the old one. Now you have two thread types and have to pick.

> **RULE OF THUMB**
>
> The promise that old code keeps working blocks some good changes, maybe yours. It also protects all the code you've already written. Respect it on the days it says no.

12.3 ABI: The Invisible Constraint

The constraint you're least likely to see coming is *ABI*, the application binary interface. ABI is the binary contract between compiled pieces of a program: the memory layout of types and the way functions are called. Keeping it stable is backward compatibility one level down, for compiled programs instead of source code.

Remember `std::regex` from chapter 1, the slow library nobody can fix? ABI is why. A faster version would need new internals, and programs already built against the old one could crash or quietly misbehave. Their source code would compile fine, so the damage hides in the binaries.

Why not just rebuild? Because few programs come with the source for every library, plugin, and DLL they load. And ABI sits outside the standard's text. It's a promise the compiler teams make to their users. That's chapter 5's implementer veto at its sharpest: the committee can't force a break the implementers won't ship.

This is why some slow types never get faster. The hash map `std::unordered_map` is stuck twice over, because its own interface demands the slow design. So the committee keeps the slow versions, and the market routes around them.

The committee took this on at the *Prague ABI vote* in early 2020, arguing over two papers by Titus Winters. P1863R1 was "ABI - Now or Never," and P2028R0 asked "What is ABI, and What Should WG21 Do About It?"

Both sides had a real case, and Winters laid out both. Keeping the old ABI leaves speed on the table, perhaps 5 to 10 percent overall by his estimate. Breaking it means rebuilding the world, at a cost he put in engineer-millennia. Either way, somebody pays.

The committee chose not to break ABI across the library for C++23, while refusing to promise stability forever. In practice, that left existing binaries safe and the frozen types frozen.

That's still roughly where things stand. SD-10 makes a stable ABI the default and allows a break case by case, with the reasons written down. The ABI Review Group advises on proposals that touch layout or calling conventions.

> **WATCH OUT**
>
> If your proposal changes the layout of a type people already use, expect the ABI question. Answer it in your paper, where you have time to think. On your feet in the room, the same question is much harder.

12.4 The Zero-Overhead Principle

C++ lives by the *zero-overhead principle*. It has two halves: you don't pay for what you don't use, and what you do use runs as fast as hand-written code. Features that break it die in the room, or get switched off in the field.

Exceptions are the famous case: just turning them on makes programs bigger, even if nothing ever throws. Herb Sutter's P0709R4 counts them as one of two language features that break the rule. In a 2018 survey cited there, 52 percent of developers said exceptions were banned in part or all of their code.

The rule shapes every proposal. If your feature slows down code that doesn't use it, expect to defend that cost in writing. The committee protects the people who came for speed.

12.5 Freestanding versus Hosted

C++ runs in two very different worlds, and the standard names them. A *hosted* environment has an operating system and the full library behind it. A *freestanding* environment has neither, like a microcontroller, a kernel, or a bootloader.

This split shapes what a library feature can assume. Something that needs files or threads usually won't work freestanding, since there may be no operating system to provide them. A proposal that forgets the freestanding world will hear about it from the people who live there.

> **THE SHORT VERSION**
>
> The oldest of these rules come from Stroustrup's D&E, so read it before you write a language proposal. Your safest bet is a design that already works in the field. Old code keeps working, so one type has to serve everyone. When a second type does ship, like `std::jthread`, everyone lives with both.
>
> Watch for ABI, since changing a type's layout breaks programs already built, and nobody can rebuild them all. In 2020 the committee chose not to break it for C++23, without promising it never will.
>
> Don't make people pay for a feature they don't use. And remember the freestanding world, where there may be no operating system at all.

These constraints explain the quiet noes: an idea can be right and still impossible. Knowing them saves you years on a proposal the rules killed before you wrote it.

If you ever present a faster version of an old type, someone will ask what it does to programs compiled a decade ago. You'll have to say it breaks them. The no that follows protects sixteen million users who'll never read your paper. That's the Delegate's Oath from chapter 1, kept by someone else.

One chapter left: the common mistakes, all in one place.

13. Common Mistakes

You've made it far enough to be dangerous. This chapter lists the traps that catch smart newcomers, what each one costs, and how to step around them. Read it once now, and again before your first meeting.

IN THIS CHAPTER

- Showing up: why you announce yourself, and what changes after your first meeting
- Your first paper: why core language can wait, what evidence to bring, and who else is working on the problem
- In the room: when to abstain, what to keep private, when to object, how consensus works, and how much to talk
- After the poll: why you follow subgroup direction, and why no single yes is the finish line

13.1 Showing Up Without Announcing

The first trap is simply appearing. Chapter 2 said you can start without permission, and it's tempting to stretch that to the meeting itself. But guests give the convener at least a week's notice, and members register ahead. Show up cold, and the first thing the convener learns about you is that you didn't ask.

The same trap waits a meeting later. Guest status covers your first meeting only, so coming back needs a seat through a national body. Wait until the first meeting is over to ask, and the next one may come before your seat does.

13.2 Proposing Core Language Features Too Early

Your first proposal probably shouldn't change the core language. That's often where the itch is, and a language feature looks like the real prize.

But core changes face the highest bar of all, as chapter 9 said, and a cold first attempt usually fails. The room remembers how it first met you, too. Start with something smaller, and bring the big idea once people know your work.

13.3 Writing an Idea-Only Paper

An idea with no examples, alternatives, or proof wastes the room's time. Yours is so clear to you that examples feel like padding. But the room can only weigh what's on the page. A thin paper gets a vague poll, and you've spent a scarce agenda slot learning nothing.

No formal rule demands an implementation, but the room expects one, even on a first paper, as chapter 9 said.

13.4 Skipping the Search for Competing Papers

Someone may already be working on your problem. It's easy to skip the search when your idea seems new and the archive looks endless. If you don't check, you've set up a fight you didn't need.

The Direction Group asks for "no surprise competitors" in P4024R0. Search the mailings, write to the other authors, and say in your paper that you did. Talk first, and the rival may become your coauthor.

13.5 Voting on Things You Haven't Followed

Chapter 8 warned you off voting on a paper you never read. The version that catches careful people is subtler: you read R0 last spring, and the poll is on R3. You think you know it. But the design moved, and you'd be voting on a paper that no longer exists.

Either way, your vote counts the same as one from someone who read every revision, so it muddies the signal. Don't hide in neutral: it means you read the paper and landed in the middle. If you haven't read the revision being polled, abstain (the honest non-vote from chapter 8).

13.6 Quoting Reflectors or Notes Publicly

A great line from the reflector or a session begs to be posted. But the closed lists, the wiki, and the room are off the public record, and quoting them is a real breach. People speak freely because what they say stays private. Leak once, and you stop hearing the candid version.

Describe what was said, never who said it, as chapter 7 put it, and wait until the meeting is over to share a session's poll questions and numbers. Never post from inside a session, even in your own words.

13.7 Raising Concerns Too Late

Saving your objection for the closing plenary gains nothing. Waiting seems polite, and nobody wants to stop a room that's clearly happy. But by plenary the design is settled, and a surprise objection spends credibility you'll want for the next fight.

Raise it in the subgroup, with a paper, while the design can still bend. Past that, chapter 8's escalation path closes at 5 p.m. the evening before plenary.

13.8 Expecting Majority Rule

Counting hands and expecting the bigger number to win is wrong here. Nearly every vote you've ever cast worked that way, so it's a natural mistake. The committee runs on consensus, the absence of sustained opposition, not majority rule.

A determined minority can stall a paper, and a bare majority can't simply outvote it. Win the count and brush past the objections, and they'll be waiting at the next meeting. When you're the one objecting, make your case with evidence and expect to be heard. But being heard isn't a veto.

13.9 Talking Too Much Too Soon

At your first meetings, speaking constantly works against you. You've done the reading and have thoughts on all of it, so your hand goes up a fourth time before lunch. By Wednesday, nobody turns to look when it does.

You build trust by learning the room before you fill it with your voice. Listen first, and speak when you have something specific, like a worked example or a fact the room hasn't heard yet.

13.10 Ignoring Subgroup Direction

When a room gives you direction, ignoring it stalls your paper. You'll be sure the room was wrong, and quietly going your own way in the next revision will look like a shortcut. The people who asked for the change will read that revision first.

The design rooms exist to set direction, and working against it keeps the opposition alive. Follow the guidance, or reopen it in that same room with a paper and new evidence, but don't route around it.

13.11 Declaring Victory Too Soon

The first poll comes back warm, and people clap you on the shoulder in the hallway. After months of work, you'd swear it's the finish. It's closer to the start. Ease off now, and the next release leaves without your paper.

Count the gates still ahead. A warm poll means keep going. Design approval comes later, and it only starts what chapter 10 called "the other 80%." The wording rooms go over every word, plenary votes the text into the working draft, and the countries ballot the whole draft.

Budget for every stage. Even a published standard is only text until the compilers build it. Don't celebrate until it ships in a compiler you can download.

> **THE SHORT VERSION**
>
> Email the convener before you go, and line up a national body before your second meeting. Start smaller than the core language. Give every paper examples, the alternatives you weighed, and code, even the first one. Search for competing papers first.
>
> In the room, abstain on any revision you haven't read. Describe what was said, never who said it, and never post live. Object early, in the subgroup, with a paper. Answer every objection, because the bigger number doesn't win here. And say less than you want to.
>
> Follow the direction a room gives you, or reopen it there with new evidence. Treat every yes as one gate passed, with more ahead, all the way to the compilers.

One mistake isn't on the list, because it never feels like one: confusing being busy with making a difference.

Chapter 7 warned you about one packed week. Now stretch it to five years. Your calendar is full of telecons, and your paper is at R9. Every poll says keep going. And not one person writing C++ has used a line of it.

Chapter 2 called showing up the secret, and it is. It gets you in, but it isn't the work. Showing up and passing polls can feel like impact, but a feature that never ships helped nobody. Keep your eyes on the people who use C++.

Last, don't treat this notebook as scripture. The committee's rules and habits keep changing, sometimes from one meeting to the next, so parts of it will age. Read each new SD-4 and each trip report, and trust them over anything here.

I wrote this for you, back when you were only wondering how to join. This is where my notes end and yours begin. Mark it up with what you learn the hard way, and someday press it into a newcomer's hands.

By now you know the whole notebook in four sentences: keep your wits about you, be skeptical, demand evidence, and put the users first. And I'm still glad you're here. Now go do the work, and do it well.

Somewhere, a few hundred volunteers are deciding what C++ will be. You can be one of them. Every three years, the committee known as WG21 ships a new version of the standard, and its members will spend an hour on the placement of a single word, because once a change goes in, it's close to permanent.

A Young Delegate's Notebook is the guide its author wishes someone had pressed into his hands before his first meeting. It is an opinionated notebook, not a reference manual. And it asks you to keep one habit above all: stay skeptical, demand evidence, and put the users first.

Vinnie Falco is the founder and President of the C++ Alliance, a nonprofit dedicated to C++ education and advocacy. He is the author of Boost.Beast, an active WG21 participant, and a longtime advocate for the developers who use C++ every day.

This notebook was made possible by the support of the C++ Alliance and by the many Standards Committee members who generously shared their time, expertise, and candor.



ISBN 979-8-234-20925-2