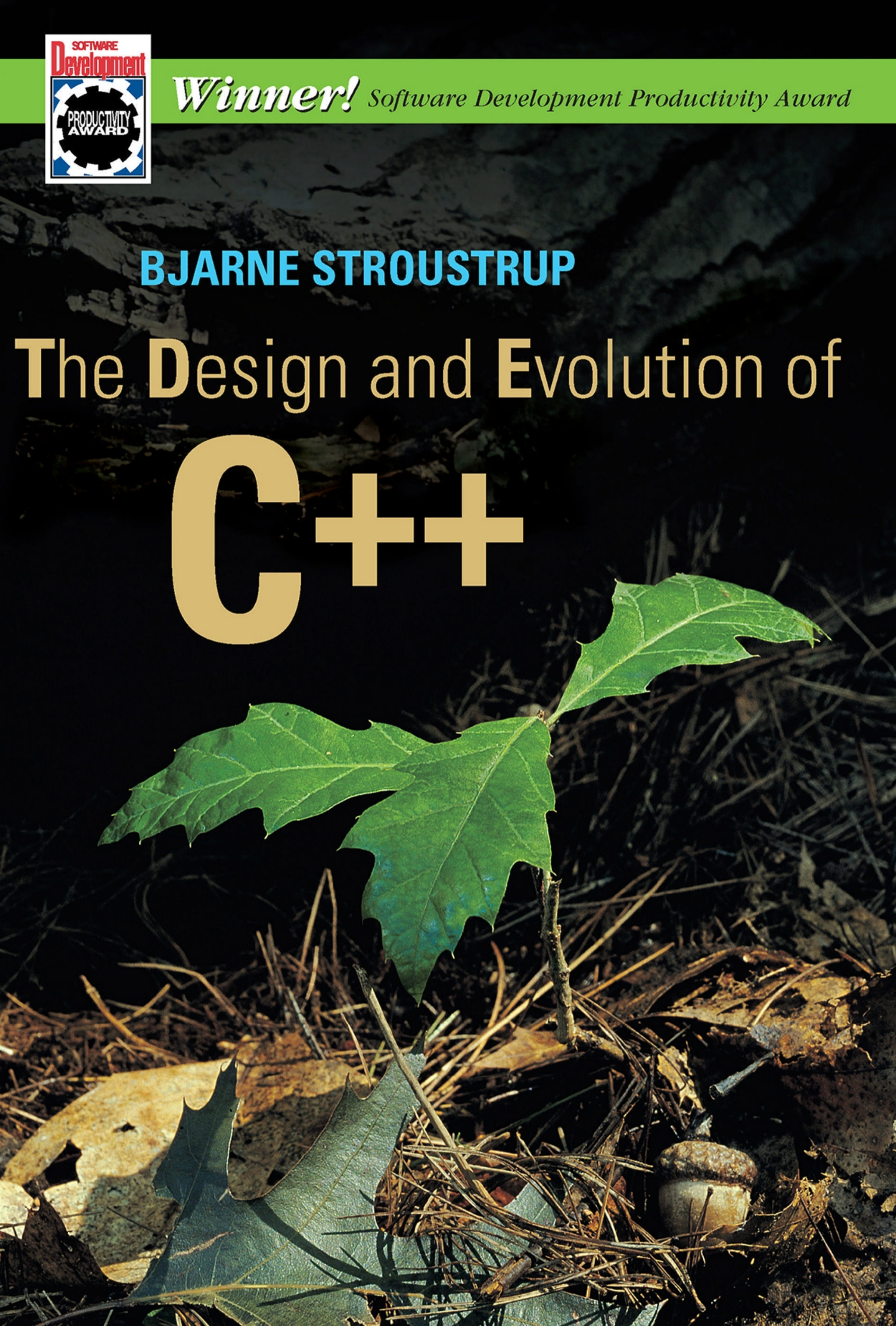

SOFTWARE Development PRODUCTIVITY AWARD

Winner! *Software Development Productivity Award*

BJARNE STROUSTRUP

The Design and Evolution of C++

The Design and Evolution of C++

Bjarne Stroustrup

AT&T Bell Laboratories
Murray Hill, New Jersey

ADDISON-WESLEY

Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sidney • Tokyo • Singapore • Mexico City

6

Standardization

Don't you try to outweird me,
I get stranger things than you
free with my breakfast cereal.
– Zaphod Beeblebrox

What is a standard? — aims of the C++ standards effort — how does the committee operate? — who is on the committee? — language clarifications — name lookup rules — lifetime of temporaries — criteria for language extension — list of proposed extensions — keyword arguments — an exponentiation operator — restricted pointers — character sets.

6.1 What is a Standard?

There is much confusion in the minds of programmers about what a standard is and what it ought to be. One ideal for a standard is to completely specify exactly which programs are legal and exactly what the meaning of every such program is. For C and C++ at least, that is not the whole story. In fact, it can't and shouldn't be the ideal for languages designed to exploit the diverse world of hardware architectures and gadgets. For such languages, it is essential to have some behavior implementation-dependent. Thus, a standard is often described as "a contract between the programmer and the implementer." It describes not only what is "legal" source text, but also what a programmer can rely on in general and what behavior is implementation-dependent. For example, in C and C++ one can declare variables of type `int`, but the standard doesn't specify how large an `int` is, only that it has at least 16 bits.

It is possible to have long and somewhat learned debates about what the standard really is and what terminology can best be employed to express it. However, the key points are to sharply distinguish what is and what is not a valid program, and further to specify what behavior should be the same in all implementations and what is implementation-dependent. Exactly how those distinctions are drawn is important, but not very interesting to practical programmers. Most committee members focus on the more language-technical aspects of standardization so the main burden of tackling the thorny issues of what the standard standardizes falls on the committee's project editor. Fortunately, our original project editor Jonathan Shopiro has an interest in such matters. Jonathan has now retired as editor in favor of Andrew Koenig, but Jonathan is still a member of the committee.

Another interesting (that is, very difficult) question is to which extent an implementation with features not specified in the standard is acceptable. It seems unreasonable to ban all such extensions. After all, some extensions are necessary to important sub-sections of the C++ community. For example, some machines have hardware that supports specific concurrency mechanisms, special addressing constraints, or special vector hardware. We can't burden every C++ user with features to support all these incompatible special-purpose extensions. They will be incompatible and will often impose a cost even on non-users. However, it would be unfortunate to discourage implementers serving such communities from trying to be perfectly conforming except for their essential extensions. On the other hand, I was once presented with an ''extension'' that allowed access to private members of a class from every function in the program; that is, the implementer had not bothered to implement access control. I didn't consider that a reasonable extension. Wordsmithing the standard to allow the former and not the latter is a nontrivial task.

An important point is to ensure that nonstandard extensions are detectable; otherwise, a programmer might wake up some morning and find significant code dependent on a supplier's unique extensions and thus without the option to change suppliers with reasonable ease. As a naive student, I remember being surprised and pleased to find that the Fortran on our university mainframe was an ''extended Fortran'' with some neat features. My surprise turned to dismay when I realized that this implied that my programs would be useless except on CDC6000 series machines.

Thus, 100% portability of standards-conforming programs is not in general an achievable or desirable ideal for C++. A program that conforms to a standard is not necessarily 100% portable because it may display implementation-dependent behavior. Actually, most do. For example, a perfectly legal C or C++ program may change its meaning if it happens to depend on the results of the built-in remainder operator `%` applied to a negative number.

Further, real programs tend to have dependencies on libraries providing services not offered on every system. For example, a Microsoft Windows program is unlikely to run unchanged under X, and a program using the Borland foundation classes will not trivially be ported to run under MacApp. Portability of real programs comes from design that encapsulates implementation and environment dependencies, not just from adherence to a few simple rules in a standards document.

Knowing what a standard doesn't guarantee is at least as important as knowing what it does promise.

6.1.1 Implementation Details

Every week, there seems to be a new request for standardizing things like the virtual table layout, the type-safe linkage name encoding scheme, or the debugger. However, these are quality-of-implementation issues or implementation details that are beyond the scope of the standard. Users would like libraries compiled with one compiler to work with code compiled with another, would like binaries to be transferable from one machine architecture to another, and would like debuggers to be independent of the implementation used to compile the code being examined.

However, standardization of instruction sets, operating-system interfaces, debugger formats, calling sequences, and object layouts is far beyond the ability of the standards group for a programming language that is merely one little cog in a much bigger system. Such universal standardization probably isn't even desirable because it would stifle progress in machine architectures and operating systems. If a user needs total independence from hardware the system/environment must be built as an interpreter with its own standard environment for applications. That approach has its own problems; in particular, specialized hardware becomes hard to exploit and local style guides cannot be followed. If those problems are overcome by interfacing to code written in another language that allows nonportable code, such as C++, the problem recurs.

For a language suitable for serious systems work, we must live with the fact that every now and again a naive user posts a message to the net: ''I moved my object code from my Mac to my SPARC and now it won't work.'' Like portability, interoperability is a matter of design and understanding of the constraints imposed by the environments. I often meet C programmers who are unaware that code compiled with two different C compilers for the same system is not guaranteed to link and in fact is unlikely to do so – yet express horror that C++ doesn't guarantee such interoperability. As usual, we have a major task in educating users.

6.1.2 Reality Check

In addition to the many formal constraints on a standards committee, there is an informal and practical one: Many standards are simply ignored by their intended users. For example, the Pascal and Pascal2 standards are almost completely forgotten. For most Pascal programmers, ''Pascal'' means Borland's greatly extended Pascal dialect. The language defined by the Pascal standard didn't provide features users considered essential and the Pascal2 standard didn't appear until a different informal ''industry standard'' had established itself. Another cautionary observation is that on UNIX most work is still done in K&R C; ANSI C is struggling in that community. The reason seems to be that some users don't see the technical benefits of ANSI/ISO C compared to K&R C outweighing the short-term costs of a transition. Even an unchallenged standard can be slow finding its way into use. To become accepted, a standard must be timely and relevant to users' needs. In my opinion, delivering a good standard for a good language in a timely manner is essential. Trying to change C++ into a ''perfect'' language or to produce a standard that cannot be misread by anyone –

however devious or ill-educated – is far beyond the abilities of the committee (§3.13). In fact, it is beyond anyone working under the time constraint provided by a large user community (§7.1).

6.2 How does the Committee Operate?

There are actually several committees formed to standardize C++. The first and largest is the American National Standards Institute's ANSI-X3J16 committee. That committee is the responsibility of the Computer and Business Equipment Manufactures Association, CBEMA, and operates under its rules. In particular, this means one-company-one-vote voting and a person who doesn't work for a company counts as a company. A member can start voting at the second meeting attended. Officially, the most important committee is the International Standards Organization's ISO-WG-21. That committee operates under international rules and is the one that will finally make the result an international standard. In particular, this means one-country-one-vote voting. Other countries, including Britain, Denmark, France, Germany, Japan, Russia, and Sweden now have their own national committees for standardizing C++. These national committees send requests, recommendations, and representatives to the joint ANSI/ISO meetings.

Basically, we have decided not to accept anything that doesn't pass under both ANSI and ISO voting rules. This implies that the committee operates rather like a bicameral parliament with a "lower house" (ANSI) doing most of the arguing and an "upper house" (ISO) ratifying the decisions of the lower house provided they make sense and duly respect the interests of the international community.

On one occasion, this procedure led to the rejection of a proposal that would otherwise have passed by a small majority. Thus, I think the national representatives saved us from a mistake that could have caused dissension. I couldn't interpret that majority as reflecting a consensus and I therefore think that – independently of the technical merit of the proposal – the national representatives gave the committee an important reminder of their responsibilities under their charter. The issue in question was that of whether C++ should have a specific form of defined minimum translation limits. A significantly improved proposal was accepted at a later meeting.

The ANSI and ISO committees meet jointly three times a year. To avoid confusion I will refer to them using the singular *committee*. A meeting lasts a week out of which many hours are taken up with legally mandated procedural stuff. Yet more hours are taken up by the kind of confusion you might expect when 70 people try to understand what the issues really are. Some daytime hours and several evenings are taken up by technical sessions where major C++ issues, such as international character handling and run-time type identification, and issues relevant to standards work, such as formal methods and organizations of international standardization bodies, are presented and discussed. The rest of the time is mostly taken up by working group meetings and discussions based on the reports from those working groups.

The current working groups are:

- C compatibility
- Core language
- Editorial
- Environment
- Extensions
- International issues
- Libraries
- Syntax

Clearly, there is too much work for the committee to handle in only three weeks of meetings a year, so much of the actual work goes on between meetings. To aid communication, we use email a lot. Every meeting involves something like three inches of double-sided paper memos. These memos are sent in two packages: one arrives a couple of weeks before a meeting to help members prepare, and one a couple of weeks after to reflect work done between the first mailing and the end of the meeting.

6.2.1 Who is on the C++ Standards Committee?

The C++ committee consists of individuals of diverse interests, concerns, and backgrounds. Some represent themselves, some represent giant corporations. Some use PCs, some use UNIX boxes, some use mainframes, etc. Some use C++, some don't. Some want C++ to be more of an object-oriented language (according to a variety of definitions of ''object-oriented''), others would have been more comfortable had ANSI C been the end-point of C's evolution. Many have a background in C, some don't. Some have a background in standards work, many don't. Some have a computer science background, some don't. Some are programmers, some are not. Some are language lawyers, some are not. Some serve end-users, some are tools suppliers. Some are interested in large projects, some are not. Some are interested in C compatibility, some are not.

Except that all are officially unpaid volunteers (though most represent companies), it is hard to find a generalization that covers all. This is good; only a very diverse group could ensure that the diverse interests of the C++ community are represented. It does make constructive discussion difficult and slow at times. In particular, this very open process is vulnerable to disruption by individuals whose technical or personal level of maturity doesn't allow them to understand or respect the views of others. I also worry that the voice of C++ users (that is, programmers and designers of C++ applications) can be drowned by the voices of language lawyers, would-be language designers, standards bureaucrats, implementers, etc.

Usually about 70 people attend a meeting, and of those, about half attend almost all meetings. The number of voting, alternate, and observing members is more than 250. I'm an alternate member, meaning that I represent my company, but someone else from my company votes. Let me give you an idea about who is represented here by simply glancing over a list of members and copying out some of the better-known names chosen from the membership list in 1990: Amdahl, Apple, AT&T, Bellcore,