

# Interview with Dave Abrahams



## Introduction

---

### Profile of Dave Abrahams

Dave (David) Abrahams is a well-known expert C++ programmer.

- In 1996, ever since he joined into the ISO C++ Standard Committee, he has been contributed to the evolution of C++.
- In 1998, together with Beman Dawes, he started up the Boost C++ Libraries which is a set of open-source C++ libraries.
- In 2001, he founded BoostPro Computing to render teaching, training and consulting about the Boost C++ Libraries.
- Every year since 2007, he held the BoostCon to create a place for programmers to meet and discuss with each other.

In C++98, he advocated the concept of exception safety and an exception guarantee. Thanks to the contribution of Dave, now we can use exception correctly.

In C++11, he designed rvalue-reference and noexcept. Additionally, he had proposed Concepts, but Concepts have been removed from the C++11 standardization effort.

The purpose of this interview:

In this interview, based on the theme "the Evolution of Languages", we asked Dave Abrahams, who is a leading-person of the Boost, semi-standard libraries of C++, mainly about the Library. Especially attracting our interest is the talking about the Boost's organization and right or wrong of the Boost's review system.



## About Dave Abrahams

---

Ryo Ezoe: Can you introduce yourself?

Dave Abrahams: I'm 46 years old, and live with my wife and son in Massachusetts, USA. I have been programming even longer than I've been playing guitar, but not as long as I've been riding bikes. My first programs were written in BASIC at teletypes on my school's PDP-8, a refrigerator-sized box with no disks and a magnetic tape drive, so that should give you some idea. Things were simpler back then.

These days, I am one of three principal partners in BoostPro Computing, a training and consulting company that serves up "all things Boost and advanced C++" to those who want what we've got to offer. It's really very open-ended: recently we've been doing a lot of work on C++ compiler development. I founded the company 10 years ago when I realized there was a need to bridge the gap between the open-source world and the more structured demands of corporate users.

By the way, I used to go by "David" in some circles so as to appear more professional and less casual, but all my friends called me "Dave." Eventually I realized I'd prefer to be everyone's friend, so it's "Dave," if you please.

Ryo Ezoë: What is your role in Boost?

Dave Abrahams: That's an interesting question. Up until the most recent BoostCon (in May), I was a "moderator," a catch-all official role established way back in 1998, when Boost was founded.

Shortly before this year's conference, though, I realized that Boost's system of governance had become almost completely ineffective. We had outgrown our casual practice of decision-making via consensus with the ultimate authority vested in the mailing list moderators. Among other problems, decisions often dropped on the floor because a given moderator might not have the time to reply or any opinion about how to proceed, and nobody felt they had the authority to decide that there was sufficient consensus to move forward. The Boost membership wanted to move and grow at a pace that our loose bureaucracy could no longer support.

With the agreement of the other moderators, we held a meeting of "invested parties" at BoostCon and formed a new governing body, the Boost Steering Committee<sup>1</sup>.



Dave Abrahams

So now I'm a steering committee member. What that means exactly is yet to be decided, but I have high hopes that we'll be able to keep up with Boost's natural pace of evolution.

<sup>1</sup> see <http://boost.2283326.n4.nabble.com/Boost-Steering-Committee-td3554964.html>

## Interview with Dave Abrahams

---

Ryo Ezoe: I didn't know Boost has steering committee movement recently. It's good to know. But as you say, it looks like it means nothing other than the fact moderators rename themselves, for now.

Dave Abrahams: I hope I didn't say that, because it's far from the case!

First of all, the steering committee currently has eight members, who are neither a strict superset nor a subset of the Boost moderators, with a plan to diversify by nominating more. We've appointed a chairman (Beman Dawes) and set out some principles by which we intend to operate. But the most important part is that we intend (and we're empowered to) actually make decisions decided that nobody was really willing to take responsibility for in the past. Of course there's never a guarantee that any new initiative (especially a volunteer initiative) will lead to more than "just talk," but the steering committee members are committed.

Ryo Ezoe: What is your job at BoostPro?

Dave Abrahams: Officially, I'm the founder and CEO. In a small company like ours, though, it's hard to divide roles up so that people have well-defined jobs. Along with my partners, I teach, consult, do software development, make purchasing decisions, work on establishing contracts, do way more IT stuff than I'd like, and generally maintain BoostPro's mission and public face. We are a distributed company, with associates teaching and programming all over the world, on projects ranging from library design to high-performance computing.

Our vision for BoostPro is to use our expertise to support and advance a community of engineers to solve problems and make new discoveries. Naturally much of the work we do is Boost-related, but we don't limit ourselves to one specific domain. I'm especially excited about opportunities to expand our training programs—with C++0x compilers just around the corner there are going to be rich new layers to discover and teach about.

One of the best parts of my job is the opportunities I have to travel and meet programmers in person in different working environments. In fact, this is part of what inspired me to create the first BoostCon in 2007. I wanted to create a place to foster connections in person and spark ideas in a way that just doesn't happen without that face-to-face interaction.



### Boost

---

Ryo Ezoe: What is the aim of Boost? How do you describe the aim and purpose of the Boost?

Dave Abrahams: Another interesting question. The official and original aim of Boost is listed on our website:

Boost provides free peer-reviewed portable C++ source libraries.

We emphasize libraries that work well with the C++ Standard Library. Boost libraries are intended to be widely useful, and usable across a broad spectrum of applications. The Boost license encourages both commercial and non-commercial use.

We aim to establish "existing practice" and provide reference implementations so that Boost libraries are suitable for eventual standardization.

First of all, I should emphasize that these founding aims are still relevant today. That said, at our first steering committee meeting we realized that as Boost has grown, its mission may have evolved. Rather than trying to "herd" Boost into maintaining its singular mission, I think the steering committee should strive to acknowledge and support the Boost that exists today. We're planning to have a discussion aimed at revising and updating (or, if we decide it really hasn't changed, reinforcing) Boost's mission statement.

Ryo Ezoe: What is the difference between Boost and other libraries?

Dave Abrahams: To start with, Boost isn't *a* library, it's a collection of libraries. So the libraries in Boost are all part of this federation of related software, which pick up resources, practices, conventions, etc., from one another. Other libraries don't always have a similar advantage.

But the most important distinction between Boost libraries and others is that everything in Boost goes through a rigorous peer review process. With few exceptions, this process seems to yield less software that's obviously the product of one person's quirky vision and more that's high-quality and broadly useful. It also ensures that the author of any Boost library is prepared to respond to community input.

Ryo Ezoe: What do you think about the idea of Boost 2.0? (I admit this is practically a silly question. It's not me who come up this particular question) By Boost 2.0 I mean abandoning some libraries that were useful in C++03 but not anymore in C++0x (because C++0x Standard supports the functionality directly), and at the same time, dropping the support of some old compilers, so we can take full advantage of C++0x's new features.

Dave Abrahams: Well, nothing is completely out-of-the-question, but Boost's structure complicates things a bit. Support for toolsets and language features is ultimately up to the library maintainers, so we can't *prevent* anyone from supporting an older compiler if they want to. That said, I think C++03 will continue to be relevant for quite a few years and most Boost libraries will probably not officially drop support for quite a few years thereafter.

## Interview with Dave Abrahams

---

I expect most newly-accepted libraries will support C++03, at least to some extent, but I also expect to see a few submissions, like Matt Calabrese's concept support library, that are C++0x-only.

Ryo Ezoe: Do you think adoption of some Boost libraries into the Standard restrict the future evolution of the Boost?

Dave Abrahams: Maybe a bit; it might be hard to justify removing a feature that was already standardized, for example. That said, it's hard for Boost to make that kind of change anyway, because it breaks users' code. Regardless, Boost libraries can continue to grow as **extensions** to the standard.

Ryo Ezoe: C++0x adopted some libraries that was originally developed and evolved in the Boost. Although I basically like this inclusion, however, I fear it may restrict the future evolution of the Boost Library. We can't easily change the standard library. Because it may break millions of existing C++ code. However, since Boost is not a Standard, it sometimes broke the compatibility to improve designs, interfaces, and implementations.

Dave Abrahams: We do try to avoid doing that where possible. Boost is popular enough that once a library has been in Boost for even a few months, breaking changes are still quite painful for users. When breaking changes are necessary, they usually come early in a library's life-cycle, well before anything has been proposed for standardization.

Ryo Ezoe: But what if, in the future, somebody invented a superior alternative design for such libraries, but to do so he must break the compatibility.

Dave Abrahams: I'm having trouble imagining the scenario. Technically speaking, practically any change at all to a C++ library breaks backward compatibility for some hypothetical program:

```
// main.cpp
#include <some_library.hpp>
using namespace some_library;

int f(void const*);
int x = f("test");
int main() {}
```

I can break this program by adding a function called `f` in `some_library.hpp`, inside its own namespace:

```
// some_library.hpp
namespace some_library { void f(char const*); } // main.cpp
```

Interviewer's note: the type of string literal "test" is `char const[5]`. So, function overload resolution will pick `void f(char const*)` over `int f(void const*)`.

There are countless similar examples. But aside from the unavoidable opportunities for name collision caused by using directives and macros, it's hard to imagine anything that could force Boost to break backward compatibility, since we can always use new names and/or namespaces to avoid changing what's there.

## Interview with Dave Abrahams

---

Ryo Ezoe: Should we just go ahead and break the compatibility with the Standard? Or offer the option to change the behavior by using `#ifdef` or template parameter? Or simply accept it as a different libraries(like `regex` and `xpressive`) or different versions(boost Phoenix) ?

Dave Abrahams: I think all options are on the table, and as you point out, we've been taking different approaches in different scenarios. Even if there was one single correct approach—an idea of which I'm not at all convinced—I don't think we've faced enough of these situations yet to be able to pick one.

Ryo Ezoe: Do you think the peer-review process is the best way to accept a library?

Dave Abrahams: Maybe it's not the best way for some other organization, but for Boost it's absolutely essential. It's part of who we are: without peer review, Boost would cease to be Boost. That said, there's always room for improvement in how we implement the process.

Ryo Ezoe: Boost allows anybody who subscribe the Boost ML(which anybody can join) to review the proposed library. Although this is more "open", but I think it has some problems. One problem is that nobody has a responsibility to review it.

For example, there was a particular proposed library that offers very interesting features for me. But it was rejected because there weren't enough reviews.

Actually, I didn't review it. Why didn't I review it then?

Because, I thought my skill is insufficient to evaluate the true value of that library. I fear my review would be treated like a spam. I think, many people feel the same way like I do.

"Why my review is needed when there are so many better programmers than me. He may post a totally superior review."

As a result, some libraries can't get enough reviews.

Dave Abrahams: Improvement in the open-source ecosystem depends on input from users who make requests based on their own needs. If we don't get enough reviews from volunteers, it's can be an indication that the necessary community to support the library's healthy evolution doesn't exist. But it is true that some potential users don't ever speak up.

I've met so many extremely bright programmers who are humble enough to say that they are “not qualified” to produce a review. The truth is, in an open-source, peer-review community, the prospective users of a library are the people most qualified to review a proposed feature. I know it can seem intimidating to jump in and put your opinions out there, but by participating in the review process you are making an essential contribution to the community. Moreover, by doing so you are developing your own skills and professional identity. The process of choosing my words and speaking up in public has always been the most powerful way to discover what's really important to me and what I believe.

## Interview with Dave Abrahams

---

Ryo Ezoe: Another problem is that review process is so easy to turn into a flame war.

There are many possible methods to design and implement a library. Most of the time, there is no single best way to solve the problem. Each methods has pros and cons. But people often believe that his method is better than opponent's. So he insist his method and reject other's. Thus flame war begin.

Dave Abrahams: I think we've been quite successful in keeping the tone of discourse professional and in avoiding flamewars. Sure, people sometimes get irritated in these discussions, but I don't recall a single review that I would characterize as a flamewar. And unfortunately, there are trolls in any corner of cyberspace. But in general the Boost community is one of the most generous and civil environments I've worked in.

Ryo Ezoe: Do you have something you can say to Japanese Boost users? Like, "Do contribute more".

Dave Abrahams: Yes! Do contribute more, please!

Ryo Ezoe: There are many Japanese Boost users. Unfortunately, most japanese users don't participate in the Boost mailing list because of the language barrier.

I wish all Japanese programmers learning English. Whether we like it or not, if we want to join an international software development, we have to use English.

Dave Abrahams: As you say, the need to learn English is a practical reality in today's technical world. By contrast, it's quite unfair that native English speakers like me can go anywhere and expect everyone else to adapt to our language. When I spoke in China last year, I got enthusiastic applause just for being able to say "hello" properly in Mandarin (I also made a point of learning how to say "I don't speak any Chinese" in Chinese).

Ryo Ezoe: But I think current situation that most Japanese programmer don't understand English can't be changed soon.

Dave Abrahams: Why not? Do you think the incentives are not clear enough?



### C++0x

---

Ryo Ezoe: Do you have any idea using C++0x's new feature to evolve the Boost?

Do you have any idea about improving existing libraries and new libraries by using C++0x's newly introduced features?

I mean, the question is not when can we use C++0x in Boost, But how can we use C++0x in Boost.

For example, rvalue reference allows us to support native Move Semantics. Variadic Templates reduce the usage of Preprocessor metaprogramming and we can actually support arbitrary parameters rather than current limited numbers determined by predefined macro.

Also, there was a interesting presentation at the BoostCon 2010("Instantiation Must Go"). A total redesign of MPL by using the decltype and its unevaluated operand.

Dave Abrahams: That presentation by Matt Calabrese was actually a consequence of a workshop I held at BoostCon 2009, where the idea was to experiment with the use of C++0x features in Boost. We broke up into groups working on different ways to apply these features. I proposed the MPL redesign with decltype and led a small group working on it.

The most interesting result in Matt's presentation was that the compile-time speedups we had expected to see (due to avoiding the instantiation of class template bodies) didn't materialize. Matt showed that the way we had been measuring compile-time complexity may have been flawed and that our use of overloading in MPL to avoid instantiations may not have paid off. This issue bears more careful investigation if and when there's an MPL rewrite.

I think there are probably still speedups available due to variadic templates, but we'll need more measurements to be sure of that.

So, yeah, I have ideas about how to use C++0x in Boost. However, I really haven't been able to explore language features as fully as I'd like to, because I have been devoting so much of my Boost time to the Ryppl project.

Ryo Ezoe: What do you think about the influence of Boost to the C++0x core language evolution? There are some libraries that influenced the C++0x core language evolution. For example, Boost.Lambda for native lambda expression. Boost.Foreach for native range-based for.

Dave Abrahams: That's true.

Aside from the influence of our libraries, the Boost community has also made major direct contributions to the proposal and design of core language features. Jaakko Järvi and Thorsten Ottosen shepherded the features you mentioned above through the standardization process, I worked on rvalue references with Howard Hinnant and Peter Dimov, on noexcept with Doug Gregor and Rani Sharoni. Doug did most of the work on variadic templates. I'm sure I'm forgetting a few others.

## Interview with Dave Abrahams

---

Boost.Parameter is another library I'd really like to see translated into a core language feature, or into a suite of features. The quality of interface one can produce with that library is amazing, and proves the feasibility of a core language feature, but the syntax one must use to achieve those results using a library interface is unfortunate.

Ryo Ezoe: So you want the named parameter in C++ core language.

Dave Abrahams: Yes. But Boost.Parameter does so much more than named parameters. It handles defaults better than C++ does today. It has deduced parameters. I also want these things for class template parameters, of course.

Interviewer's note: "deduced parameters" in Boost.Parameter is a feature that can be passed in any position without supplying an explicit parameter name under the environment supports "named parameter".

Ryo Ezoe: When that happens, I can imagine people abuse it to construct ESEL not even remotely looks like a function call on top of that.

Dave Abrahams: As you might imagine, I don't have a problem with that. EDSLs, used responsibly, are a **good** thing. And I think EDSL **design** (as opposed to implementation) is actually quite easy to do well. I've seen very few bad ones.

Ryo Ezoe: Do you have any other feature you want to have in the post C++0x standard? (My preference is an alternative feature for #include.)

Dave Abrahams: Yes! Modules are my number one desired feature to make C++ programming better for everybody. The #include model is a terrible drag on the language, and modules would lift that burden. I also still really want to see concepts adopted.

Ryo Ezoe: How can we teach C++ and Boost?

Dave Abrahams: Good question. We at BoostPro have spent a lot of time designing courses to do that well.

Ryo Ezoe: I feel C++ and Boost doesn't have the good education. There are peoples who says "C++ and Boost are too difficult to use."

C++ details and Boost implementation details are indeed difficult. But I believe we can use it without knowing these complex details. It should be.

How can we effectively teach how to use C++ and Boost without requiring to understand the deep details?

Dave Abrahams: Any library that requires its users to understand its **implementation details** is a failure. Or, to put it another way, if you need to understand an implementation detail to use the library, it isn't an implementation detail.

That said, there are complexities of any interesting design (be it a language, a library, or a pocket knife) that **aren't** implementation details. The easiest-to-use designs hide those complexities from users except in unusual cases, and limit the injury users can inflict on themselves when they encounter those complexities unexpectedly.

For the most part, Boost's library designs succeed in hiding complexity and limiting damage. C++ tries to do the same, but the constraints of its C-compatibility legacy make that a bigger challenge. The rest is up to how we teach.

Here are a few basic principles I try to follow when teaching anything:

- Show how to get started quickly so students get an immediate experience of power and competence. Avoid details assiduously at first.
- Develop and present a simple "mental model" that guides understanding even in circumstances we don't have time to cover in class. This is what prepares students to deal with those complexities you mentioned.
- Seek out and develop "big picture" themes—common threads (such as, "the benefits of value semantics") that run through many different topics and can help students to draw together the things they learn into a coherent and powerful worldview.

Ryo Ezoe: I personally like learning these details. Your book is great in that sense.

Dave Abrahams: It's interesting that you should say so. The book ("C++ Template Metaprogramming," which I wrote with Aleksey Gurtovoy) actually presents very few implementation details, instead focusing on the high-level ideas behind template metaprogramming. In fact, we go out of our way to base the book on the MPL so that you **don't** have to learn the most complex metaprogramming tricks—those are hidden inside the library. Some people have even complained that we don't show enough of the low-level "tricks."

Ryo Ezoe: But I think requiring all users to be familiar with complex metaprogramming tricks is not a good idea.

Dave Abrahams: I agree! Fortunately, very few Boost libraries require users to be familiar with metaprogramming at all, much less the "tricky" aspects of it. The exceptions are mostly the libraries for library writers (e.g. Boost.MPL).

Ryo Ezoe: What do you think about the future of DSEL on C++? DSEL implementation built on C++'s template were developed for years. Yet, not all people seriously using these DSEL libraries which use (more like abuse) Expression Template technique, such as Boost.Lambda, Boost.Phoenix, Boost.Xpressive and Boost.Spirit.

Dave Abrahams: For the reader: "DSEL" stands for "Domain Specific Embedded Language." As I've discovered since writing that book, "Embedded Domain Specific Language" or "EDSL" is actually the more common usage. Both terms refer to a mini-language implemented as a library within a host language.

Ryo Ezoe: The problem is, code which use these DSEL libraries, don't even look like a sane C++ code. So many people avoid using these libraries. If a library is not widely used (especially among beginners and average programmers), isn't it worthless no matter how good it is? How can we solve this problem?

Dave Abrahams: beginners. For example, take the MPL, which, though not very accessible to most programmers, is used in the implementation of countless very popular Boost libraries.

But MPL is not an EDSL library, so let's consider those for a moment. Blitz++, a numerical computing library, was probably the first C++ library to provide an EDSL. It was widely adopted among computational physicists and others doing high-performance numerical computing, because it could beat FORTRAN on

## Interview with Dave Abrahams

---

some computations while providing a much more expressive and economical interface. Blitz++ is still in active use today. Even though the users of Blitz++ amount to no more than a tiny fraction of all C++ users, the library and others like it have allowed them to do work that would have been completely impractical otherwise—work that could impact the lives of ordinary people everywhere. So even an EDSL serving just a few experts can be a worthwhile endeavor.

That said, I think you are pointing at a real issue, which is that usage of an EDSL will look unfamiliar to many people. To make the discussion concrete, let's take an example of EDSL usage from the documentation for Boost.MSM (the Meta State Machine Library) by Christophe Henry:

```
BOOST_MSM_EUML_TRANSITION_TABLE((
  Stopped + play [some_guard] / (some_action , start_playback) == Playing ,
  Stopped + open_close/ open_drawer                          == Open   ,
  Stopped + stop                                             == Stopped ,
  Open   + open_close / close_drawer                         == Empty  ,
  Empty  + open_close / open_drawer                          == Open   ,
  Empty  + cd_detected [good_disk_format] / store_cd_info    == Stopped
),transition_table)
```

This little code fragment represents a state machine. If your choice is between that fragment and 50 lines of nested switch statements, which would you choose? Now quadruple the complexity of the state machine, and imagine 24 lines of code that looks like the above, compared with 200 lines of nested switch statements. Does that change your answer?

Ultimately, are trade-offs here, as with **any other abstraction**, be it a programming language, an ordinary library interface, or an EDSL:

you have to ask yourself whether the correctness, maintainability, readability, and encapsulation benefits are worth the costs of requiring future maintainers to learn the “language” of the abstraction. My personal experience is that avoiding abstraction leads (surprisingly quickly!) to nightmarishly complex software that doesn't scale, so my resistance to introducing new abstractions—provided they are well-documented—into a codebase is extremely low.

Ryo Ezoe: Do you get the idea of new idioms, techniques etc... from other languages or libraries recently?

Dave Abrahams: Other languages, definitely. This isn't my idea, but one of the most interesting things that happened at the last BoostCon was that some people who know Haskell got together with some who know Boost.Proto and discovered how to reformulate Proto in terms of monads---in C++! As for me, I've been thinking a lot about the new multicore reality and what that will mean for immutability (as in pure functional programming languages) and for the “semi-immutability” we get from value semantics. I've also been lucky enough to be involved in some of the discussions about how to integrate transactional memory with C++, which is a real cross-disciplinary effort.



### Interviewer's Postscript

---

"Our next interviewee should be Dave Abrahams." That was the consensus after we completed our 1st "grimoire". Dave has made many contributions to the ISO C++ standardization effort and, moreover, is a leading figure in Boost. We were sure it would be a very interesting interview.

Takatoshi Kondo, who wrote about Boost.Serialization in this "grimoire", had also participated in Boost-Con 2010, so he approached Dave about this interview. Dave agreed, and that's how this interview came to be.

Kondo-san's words about Dave-san's character were as follows: "His tone of argument was very incisive, without compromise". For example, there was a BoostCon 2010 presentation, "Demystifying C++ Exceptions," which proposed redefining the three levels of exception-safety guarantee and adding a "Minimal Guarantee". Dave-san objected the idea and put forward an in-depth, valuable argument on the meaning of that guarantee. That interaction is typical of Dave-san's uncompromising stance.

According to Kondo-san, Dave-san is also known as a devoted husband. He brings his wife and child with him to BoostCon, and is known to have left early from C++ committee meetings to care for his family when his wife was ill.