The Trouble with Checked Exceptions

A Conversation with Anders Hejlsberg, Part II

by Bill Venners with Bruce Eckel

August 18, 2003

Summary
Anders Hejlsberg, the lead C# architect, talks with Bruce Eckel and Bill Venners about versionability and scalability issues with checked exceptions.

ADVERTISEMENT

Anders Hejlsberg, a distinguished engineer at Microsoft, led the team that designed the C# (pronounced C Sharp) programming language. Hejlsberg first vaulted onto the software world stage in the early eighties by creating a Pascal compiler for MS-DOS and CP/M. A very young company called Borland soon hired Hejlsberg and bought his compiler, which was thereafter marketed as Turbo Pascal. At Borland, Hejlsberg continued to develop Turbo Pascal and eventually led the team that designed Turbo Pascal's replacement: Delphi. In 1996, after 13 years with Borland, Hejlsberg joined Microsoft, where he initially worked as an architect of Visual J++ and the Windows Foundation Classes (WFC). Subsequently, Hejlsberg was chief designer of C# and a key participant in the creation of the .NET framework. Currently, Anders Hejlsberg leads the continued development of the C# programming language.

On July 30, 2003, Bruce Eckel, author of *Thinking in C++* and *Thinking in Java*, and Bill Venners, editor-in-chief of Artima.com, met with Anders Hejlsberg in his office at Microsoft in Redmond, Washington. In this interview, which will be published in multiple installments on Artima.com and on an audio CD-ROM to be released this fall by Bruce Eckel, Anders Hejlsberg discusses many design choices of the C# language and the .NET framework.

- In Part I: The C# Design Process, Hejlsberg discusses the process used by the team that designed C#, and the relative merits of usability studies and good taste in language design.
- In this second installment, Hejlsberg discusses versionability and scalability issues with checked exceptions.

Remaining Neutral on Checked Exceptions

Bruce Eckel: C# doesn't have checked exceptions. How did you decide whether or not to put checked exceptions into C#?

Anders Hejlsberg: I see two big issues with checked exceptions: scalability and versionability. I know you've written some about checked exceptions too, and you tend to agree with our line of thinking.

Bruce Eckel: I used to think that checked exceptions were really great.

Anders Hejlsberg: Exactly. Frankly, they look really great up front, and there's nothing wrong with the idea. I completely agree that checked exceptions are a wonderful feature. It's just that particular implementations can be problematic. By implementing checked exceptions the way it's done in Java, for example, I think you just take one set of problems and trade them for another set of problems. In the end it's not clear to me that you actually make life any easier. You just make it different.

Bruce Eckel: Was there a lot of disagreement in the C# design team about checked excpetions?

Anders Hejlsberg: No, I think there was fairly broad agreement in our design group.

C# is basically silent on the checked exceptions issue. Once a better solution is known —and trust me we continue to think about it—we can go back and actually put something in place. I'm a strong believer that if you don't have anything right to say, or anything that moves the art forward, then you'd better just be completely silent and neutral, as opposed to trying to lay out a framework.

If you ask beginning programmers to write a calendar control, they often think to themselves, "Oh, I'm going to write the world's best calendar control! It's going to be polymorphic with respect to the kind of calendar. It will have displayers, and mungers, and this, that, and the other." They need to ship a calendar application in two months. They put all this infrastructure into place in the control, and then spend two days writing a crappy calendar application on top of it. They'll think, "In the next version of the application, I'm going to do so much more."

Once they start thinking about how they're actually going to implement all of these other concretizations of their abstract design, however, it turns out that their design is completely wrong. And now they've painted themself into a corner, and they have to throw the whole thing out. I have seen that over and over. I'm a strong believer in being minimalistic. Unless you actually are going to solve the general problem, don't try and put in place a framework for solving a specific one, because you don't know what that framework should look like.

Bruce Eckel: The Extreme Programmers say, "Do the simplest thing that could possibly work."

Anders Hejlsberg: Yeah, well, Einstein said that, "Do the simplest thing possible, but no simpler." The concern I have about checked exceptions is the handcuffs they put on programmers. You see programmers picking up new APIs that have all these throws clauses, and then you see how convoluted their code gets, and you realize the checked exceptions aren't helping them any. It is sort of these dictatorial API designers telling you how to do your exception handling. They should not be doing that.

Versioning with Checked Exceptions

Bill Venners: You mentioned scalability and versioning concerns with respect to checked exceptions. Could you clarify what you mean by those two issues?

Anders Hejlsberg: Let's start with versioning, because the issues are pretty easy to see there. Let's say I create a method `foo` that declares it throws exceptions `A`, `B`, and `C`. In version two of `foo`, I want to add a bunch of features, and now `foo` might throw exception `D`. It is a breaking change for me to add `D` to the throws clause of that method, because existing caller of that method will almost certainly not handle that exception.

Adding a new exception to a throws clause in a new version breaks client code. It's like adding a method to an interface. After you publish an interface, it is for all practical purposes immutable, because any implementation of it might have the methods that you want to add in the next version. So you've got to create a new interface instead. Similarly with exceptions, you would either have to create a whole new method called `foo2` that throws more exceptions, or you would have to catch exception `D` in the new `foo`, and transform the `D` into an `A`, `B`, or `C`.

Bill Venners: But aren't you breaking their code in that case anyway, even in a language without checked exceptions? If the new version of `foo` is going to throw a new exception that clients should think about handling, isn't their code broken just by the fact that they didn't expect that exception when they wrote the code?

Anders Hejlsberg: No, because in a lot of cases, people don't care. They're not going to handle any of these exceptions. There's a bottom level exception handler around their message loop. That handler is just going to bring up a dialog that says what went wrong and continue. The programmers protect their code by writing try finally's everywhere, so they'll back out correctly if an exception occurs, but they're not actually interested in handling the exceptions.

The throws clause, at least the way it's implemented in Java, doesn't necessarily force you to handle the exceptions, but if you don't handle them, it forces you to acknowledge precisely which exceptions might pass through. It requires you to either catch declared exceptions or put them in your own throws clause. To work around this requirement, people do ridiculous things. For example, they decorate every method with, "`throws Exception`." That just completely defeats the feature, and you just made the programmer write more gobbledy gunk. That doesn't help anybody.

Bill Venners: So you think the more common case is that callers don't explicitly handle exceptions in deference to a general catch clause further up the call stack?

Anders Hejlsberg: It is funny how people think that the important thing about exceptions is handling them. That is not the important thing about exceptions. In a well-written application there's a ratio of ten to one, in my opinion, of try finally to try catch. Or in C#, `using` statements, which are like try finally.

Bill Venners: What's in the finally?

Anders Hejlsberg: In the finally, you protect yourself against the exceptions, but you don't actually handle them. Error handling you put somewhere else. Surely in any kind of event-driven application like any kind of modern UI, you typically put an exception handler around your main message pump, and you just handle exceptions as they fall out that way. But you make sure you protect yourself all the way out by deallocating any resources you've grabbed, and so forth. You clean up after yourself, so you're always in a consistent state. You don't want a program where in 100 different places you handle exceptions and pop up error dialogs. What if you want to change the way you put up that dialog box? That's just terrible. The exception handling should be centralized, and you should just protect yourself as the exceptions propagate out to the handler.

The Scalability of Checked Exceptions

Bill Venners: What is the scalability issue with checked exceptions?

Anders Hejlsberg: The scalability issue is somewhat related to the versionability issue. In the small, checked exceptions are very enticing. With a little example, you can show that you've actually checked that you caught the `FileNotFoundException`, and isn't that great? Well, that's fine when you're just calling one API. The trouble begins

when you start building big systems where you're talking to four or five different subsystems. Each subsystem throws four to ten exceptions. Now, each time you walk up the ladder of aggregation, you have this exponential hierarchy below you of exceptions you have to deal with. You end up having to declare 40 exceptions that you might throw. And once you aggregate that with another subsystem you've got 80 exceptions in your throws clause. It just balloons out of control.

In the large, checked exceptions become such an irritation that people completely circumvent the feature. They either say, "`throws Exception`," everywhere; or—and I can't tell you how many times I've seen this—they say, "`try, da da da da da, catch curly curly`." They think, "Oh I'll come back and deal with these empty catch clauses later," and then of course they never do. In those situations, checked exceptions have actually degraded the quality of the system in the large.

And so, when you take all of these issues, to me it just seems more thinking is needed before we put some kind of checked exceptions mechanism in place for C#. But that said, there's certainly tremendous value in knowing what exceptions can get thrown, and having some sort of tool that checks. I don't think we can construct hard and fast rules down to, it is either a compiler error or not. But I think we can certainly do a lot with analysis tools that detect suspicious code, including uncaught exceptions, and points out those potential holes to you.

Next Week

Come back Monday, September 1 for part VII of a conversation with Elliotte Rusty Harold about the design of the XOM API. I am now staggering the publication of several interviews at once, to give the reader variety. The next installment of this interview with Anders Hejlsberg will appear on Monday, September 8. If you'd like to receive a brief weekly email announcing new articles at Artima.com, please subscribe to the Artima Newsletter.

Talk Back!

Have an opinion about the design principles presented in this article? Discuss this article in the News & Ideas Forum topic, The Trouble with Checked Exceptions.

Resources

Deep Inside C#: An Interview with Microsoft Chief Architect Anders Hejlsberg:
http://windows.oreilly.com/news/hejlsberg_0800.html

A Comparative Overview of C#:
http://genamics.com/developer/csharp_comparative.htm

Microsoft Visual C#:
http://msdn.microsoft.com/vcsharp/

Anders Hejlsberg was not the first Artima interviewee to mention taste. Jim Waldo made almost an identical comment about building a team of tasteful programmers in his interview:
http://www.artima.com/intv/waldo10.html

And an entire portion of Ken Arnold's interview was devoted to design taste - Taste and Aesthetics:

Sponsored Links

Google Search

Web Artima.com

Copyright © 1996-2019 Artima, Inc. All Rights Reserved. - Privacy Policy - Terms of Use